Software Guide

ICPDAS LinCon-8000 SDK

Implement industry control with Linux Technique

(version 4.6)

Contents

1. Introduction

Recently, Linux has been adopted widely by many users because of the properties of stability, open source, and free charge. In the mean while, the development of linux is supported by many large international companies and the function in linux is not inferior to Windows so that linux OS is more and more popular and accepted. In the other hand, the hardware requirement that linux OS can works in embedded system smoothly is not high, just only 386 CPU or better and 8 MB RAM. Therefore except Win CE of Microsoft, Linux has been already another good choice in embedded OS.

The Linux OS demands less system resources from the embedded controller and is therefore the best fit for it because of the embedded controller has some limitations in system resources. It is for this reason that the LinCon-8000 embbeded controller has been published to be a new generation product from ICPDAS and the Embedded-Linux OS has been adopted into the LinCon-8000. The LinCon-8000's main purpose is to allow the numerous enthusiastic linux users to control their own embedded systems easily within the Linux Environment.

ICPDAS provides the library file － libi8k.a which includes all the functions from the I-7000/8000/87000 series modules which are used in the LinCon-8000 Embedded Controller. The libi8k.a is desiged specially for the I-7000/8000/87000 series modules on the Linux platform for use in the LinCon-8000. Users can easily develop applications in the LinCon-8000 by using either C or Java Language and the .NET applications will also be supported In the future. The various functions of the libi8k.a are divided into the sub-group functions for ease of use within the different applications. The powerful functions of the LinCon-8000 embedded controller are depicted in figure 1-1, which includes a **<u>VGA</u>, <u>USB</u>(Card Reader, Camera …), <u>Mouse</u>, <u>Keyboard</u>, <u>Compact flash card</u>, <u>Series ports</u>(RS-232, RS-485), Ethernet(Hub…) and <u>many I/O slots</u>** in the picture. Presently, HTTP、FTP、Telnet、SSH、SFTP Servers are built in and users can transfer files or use remote control with the LinCon-8000 more conveniently. In network communication, **wireless**、**Bluetooth** transfer and **Modem**、**GPRS**、**ADSL**、**Firewall** are also supported. Fig. 1-2 illustrates the outline of the LinCon-8000 with modules. Fig. 1-3 illustrates hardware architecture of the LinCon-8000.

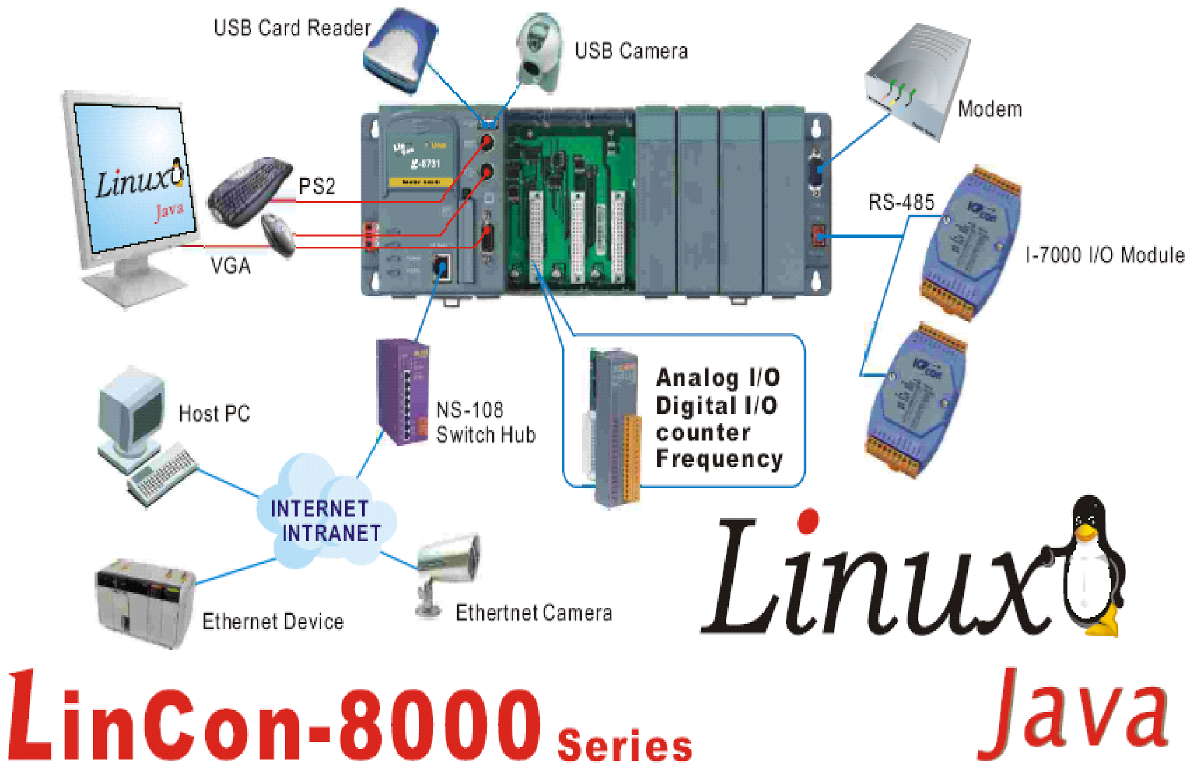

USB Card Reader
USB Camera
Modem
PS2
VGA
RS-485
I-7000 I/O Module
Host PC
NS-108
Switch Hub
Analog I/O
Digital I/O
counter
Frequency
INTERNET
INTRANET
Ethernet Device
Ethertnet Camera
Linux
LinCon-8000 Series
Java

Fig. 1-1

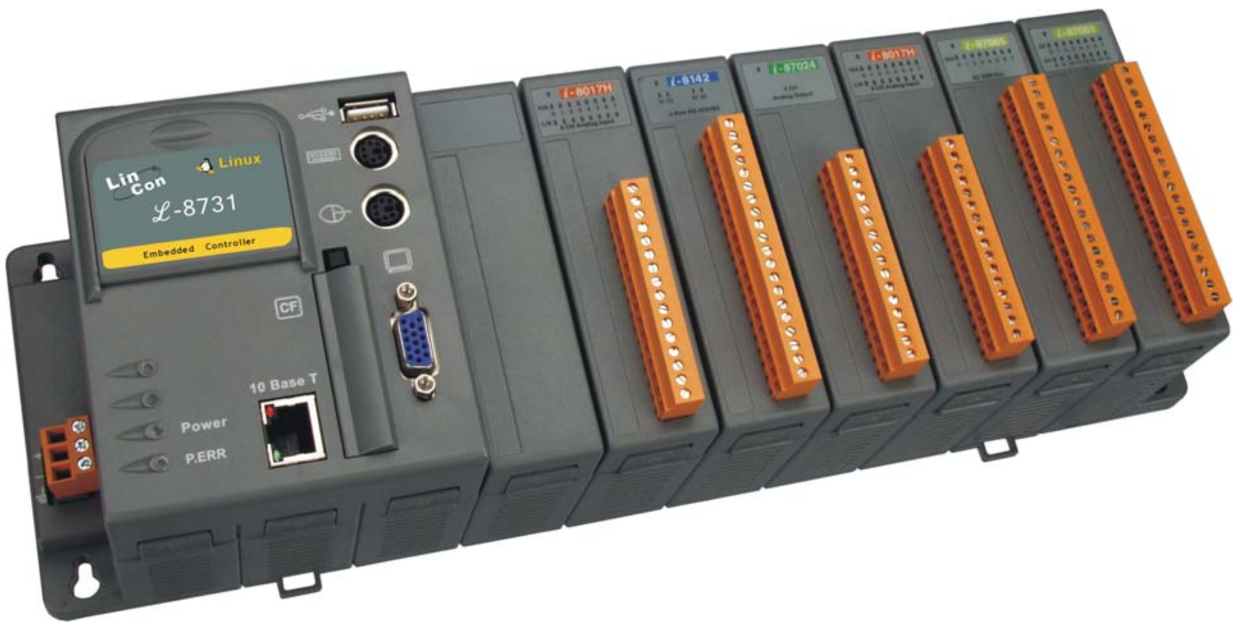


Fig. 1-2

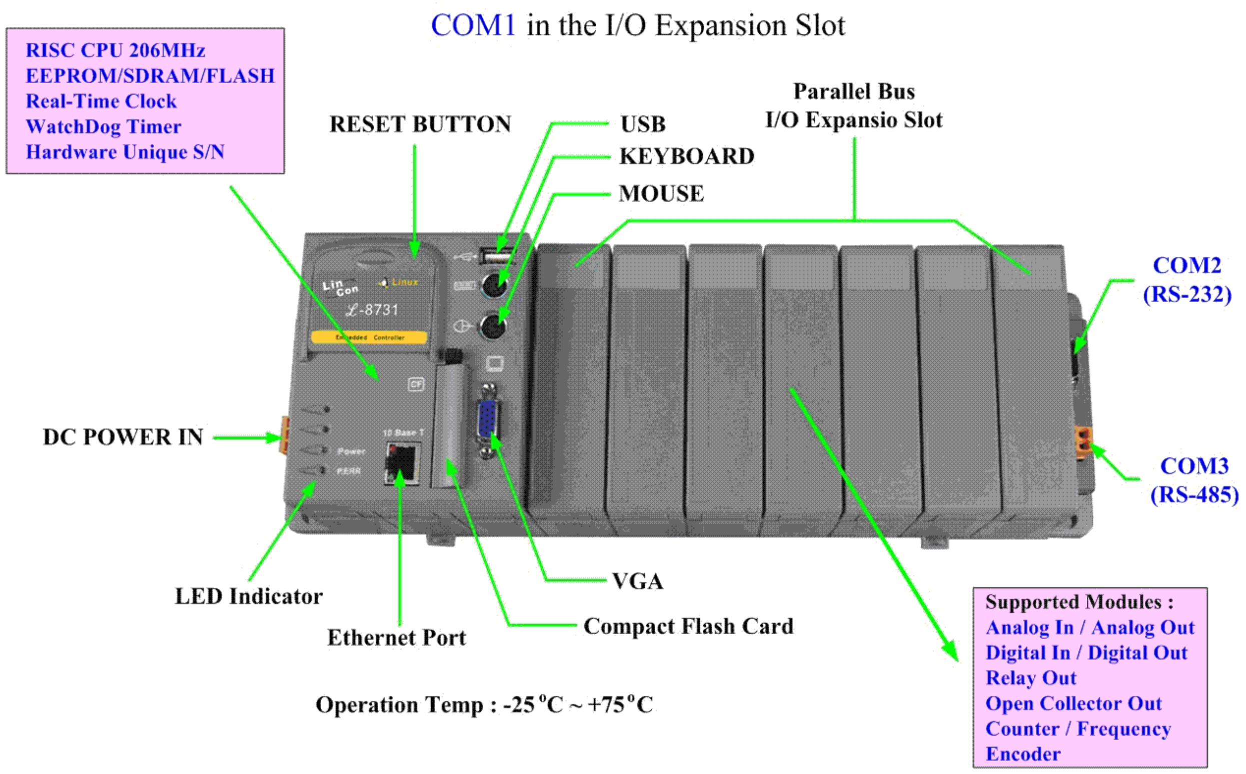

COM1 in the I/O Expansion Slot
RISC CPU 206MHz
EEPROM/SDRAM/FLASH
Real-Time Clock
WatchDog Timer
Hardware Unique S/N
RESET BUTTON
USB
KEYBOARD
MOUSE
Parallel Bus
I/O Expansio Slot
COM2
(RS-232)
DC POWER IN
COM3
(RS-485)
LED Indicator
VGA
Ethernet Port
Compact Flash Card
Operation Temp : -25°C ~ +75°C
Supported Modules :
Analog In / Analog Out
Digital In / Digital Out
Relay Out
Open Collector Out
Counter / Frequency
Encoder

Fig. 1-3

2. Installation of LinCon-8000 SDK

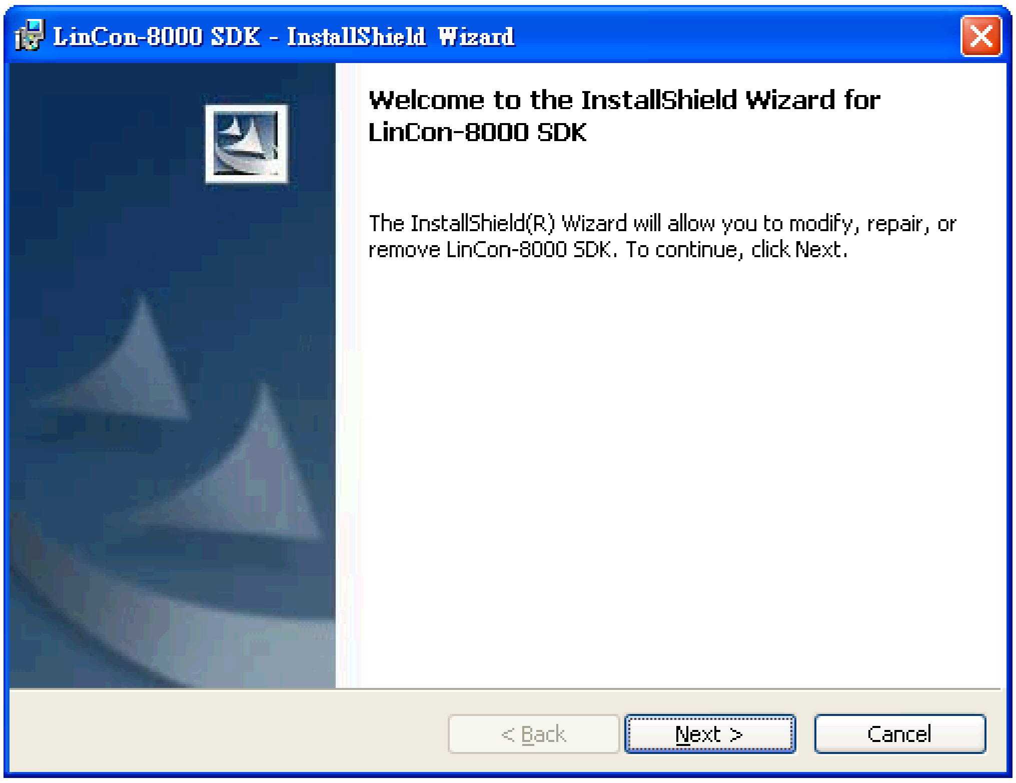
"LinCon-8000 SDK" consists of the following major items.

- LinConSDK library files
- LinConSDK include files
- Demo files
- GNU ToolChain

From ftp://ftp.icpdas.com/pub/cd/linconcd/napdos/linux/sdk/, users can download the latest version of LinCon-8000 SDK. Then follows below steps to install the development toolkit provided by ICPDAS for the application development of the LinCon-8000 embedded controller platform easily.

2.1 Quick Installation of LinCon-8000 SDK

1. Please insert the installation CD into your CD-ROM driver.
2. Run the "linconsdk_for_windows.exe" file under the folder \napdos\linux\SDK\. Then click on the "Next" button, refer to Fig. 2-1.

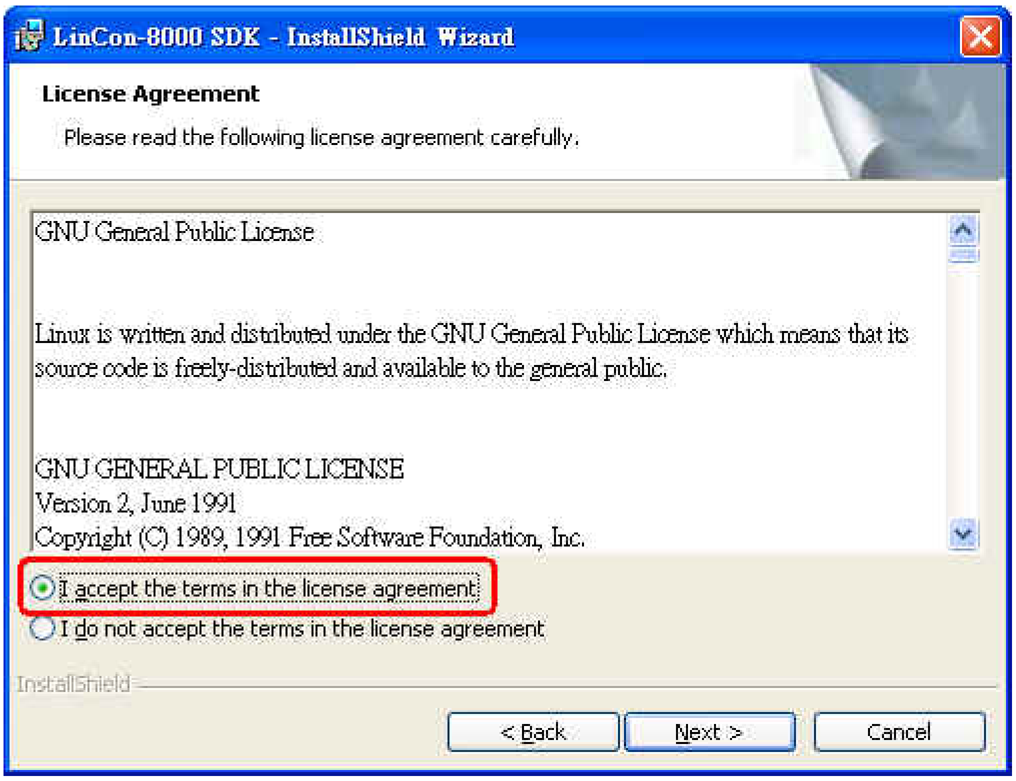


Welcome to the InstallShield Wizard for LinCon-8000 SDK

The InstallShield(R) Wizard will allow you to modify, repair, or remove LinCon-8000 SDK. To continue, click Next.

Fig. 2 -1

3. Choose the option of "I accept the terms in the license agreement" and click the "next" button, refer to Fig. 2-2 below.


LinCon-8000 SDK - InstallShield Wizard

License Agreement

Please read the following license agreement carefully.

GNU General Public License

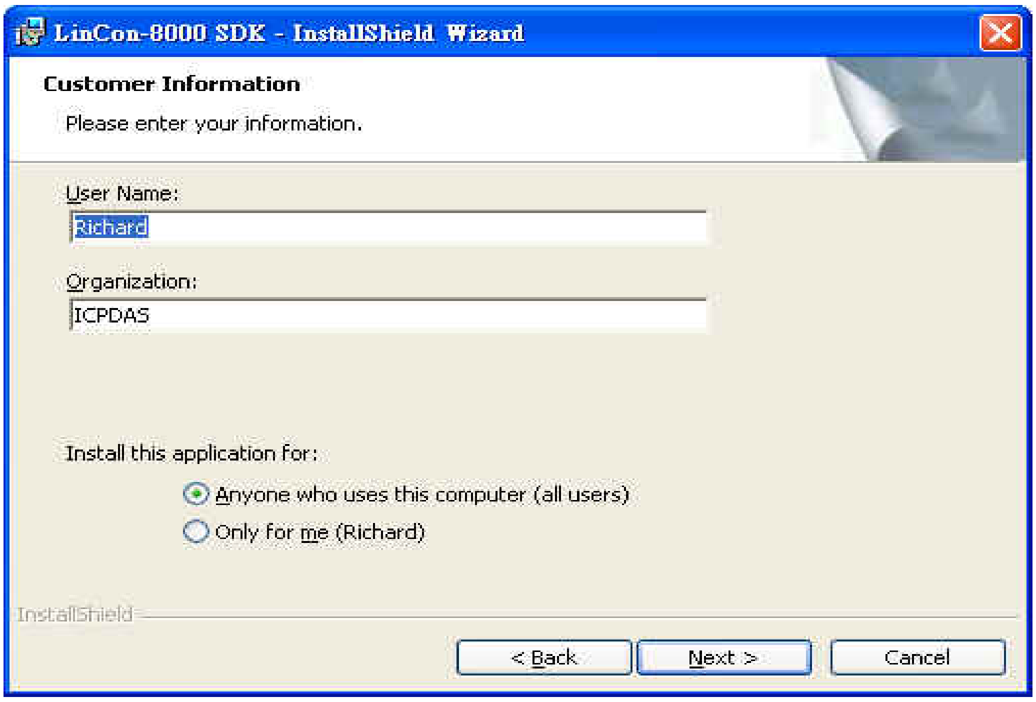
Linux is written and distributed under the GNU General Public License which means that its source code is freely-distributed and available to the general public.

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.

[✓ I accept the terms in the license agreement]
I do not accept the terms in the license agreement

Fig. 2-2

4. Input your user name and your organization's name, then to click on the "Next" buttion, refer to Fig 2-3.


LinCon-8000 SDK - InstallShield Wizard

Customer Information

Please enter your information.

User Name: Richard

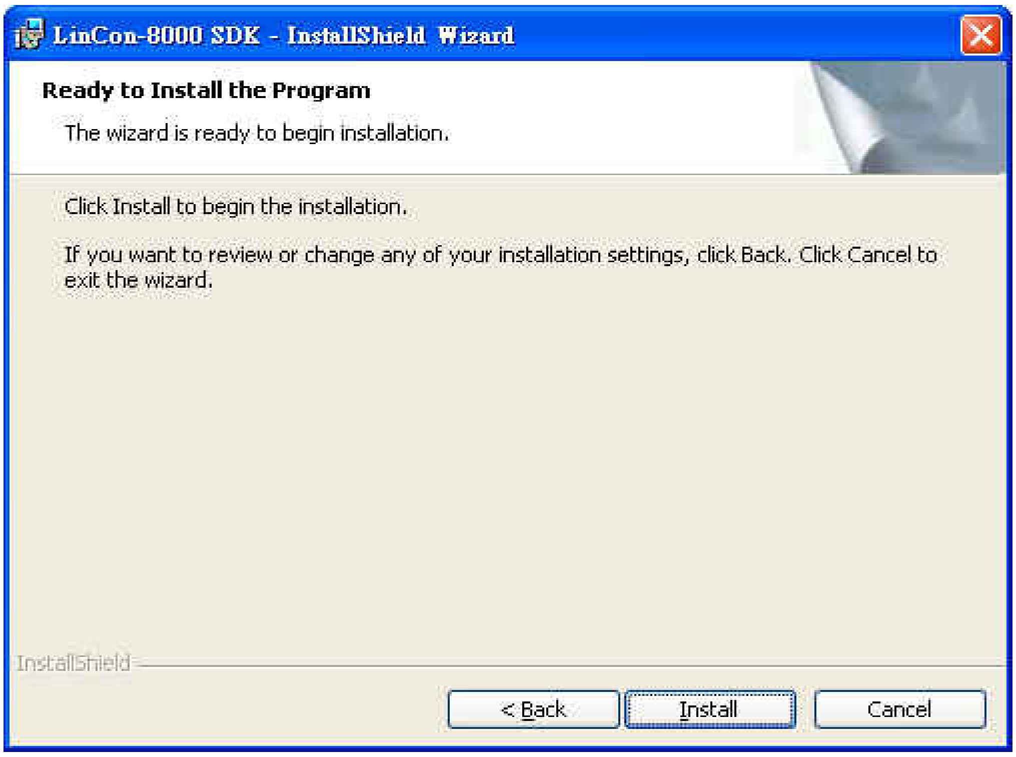
Organization: ICPDAS

Install this application for:

[✓ Anyone who uses this computer (all users)]
Only for me (Richard)

Fig. 2-3

5. Then click on the "Install" button to install the LinCon-8000 SDK, refer to Fig 2-4.


LinCon-8000 SDK - InstallShield Wizard

Ready to Install the Program

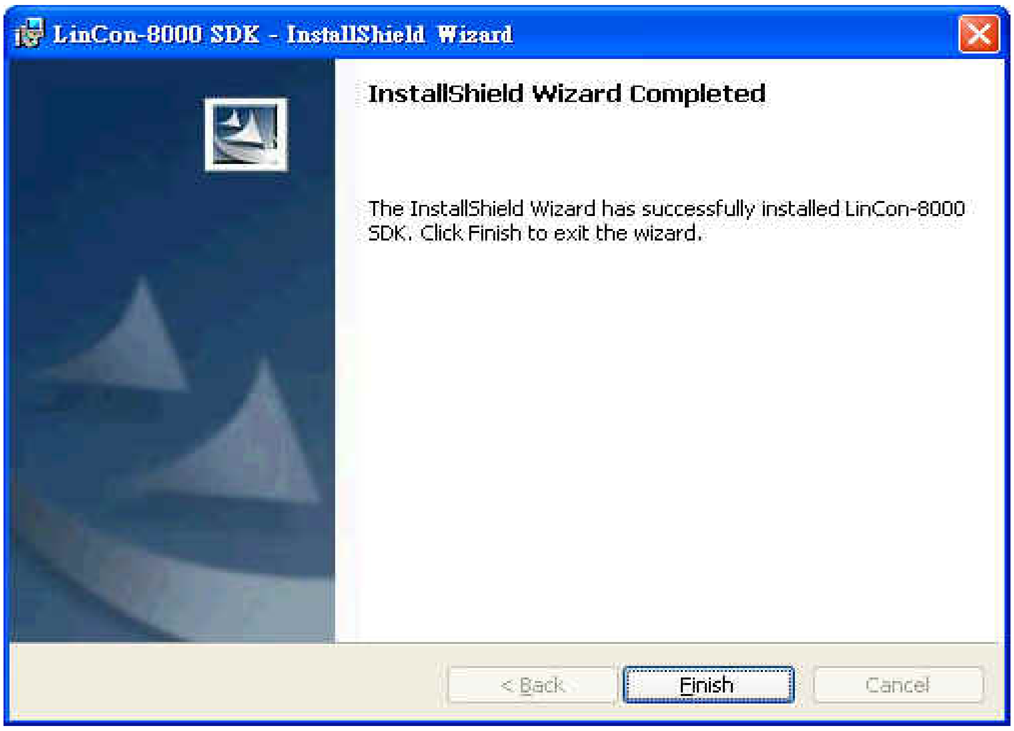
The wizard is ready to begin installation.

Click Install to begin the installation.

If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

Fig. 2-4

6. After successfully installing the software, please click on the "Finish" button to finish the development toolkit installation, refer to Fig. 2-5


LinCon-8000 SDK - InstallShield Wizard

InstallShield Wizard Completed

The InstallShield Wizard has successfully installed LinCon-8000 SDK. Click Finish to exit the wizard.

Fig. 2-5

7. Open the "**C:\cygwin\LinCon8k**" folder and see the content. Refer to Fig 2-6.

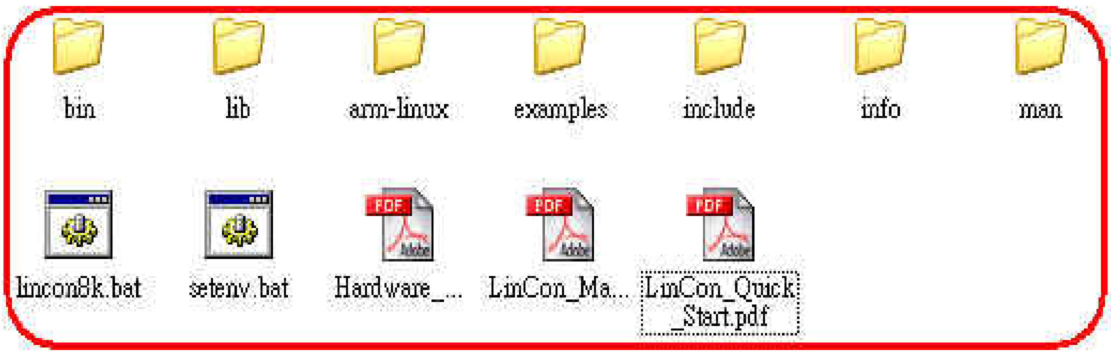
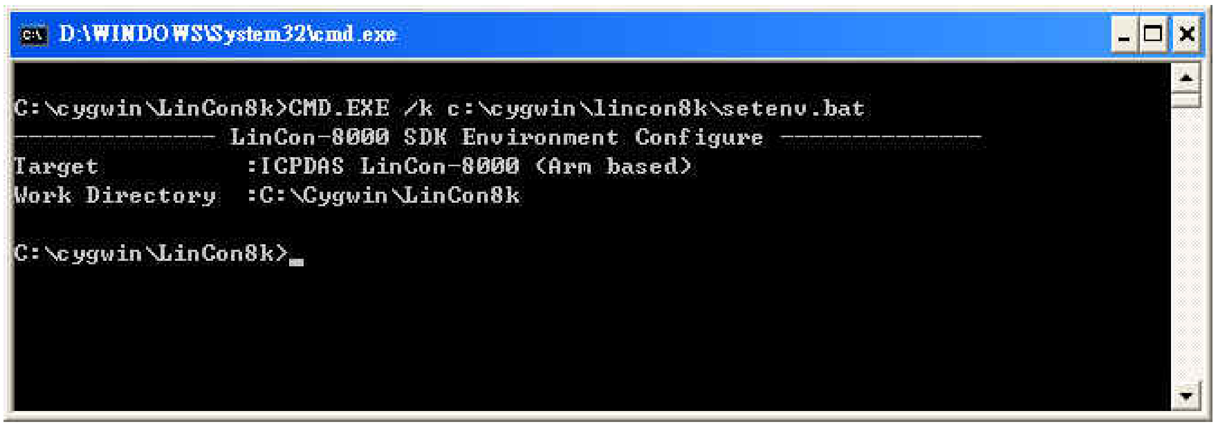


Fig. 2-6

8. Start using the "LinCon-8000 Build Environment" by double clicking the shortcut for the "**LinCon-8000 Build Environment**" on the desktop or by clicking through "Start" > "Programs" > "ICPDAS" > "LinCon-8000 SDK" > " LinCon-8000 Build Environment" icon. Then a special DOSBOX will be displayed in which we can compile applications for the LinCon-8000. refer to Fig. 2-7.



```
C:\cygwin\LinCon8k>CMD.EXE /k c:\cygwin\lincon8k\setenv.bat
--------------- LinCon-8000 SDK Environment Configure ---------------
Target            :ICPDAS LinCon-8000 (Arm based)
Work Directory    :C:\Cygwin\LinCon8k

C:\cygwin\LinCon8k>_
```

Fig. 2-7

Once your Installation is complete, you can find the files for the library and demo in the following paths.

The Libi8k.a path is "**C:\cygwin\LinCon8k\lib**".

The include files path is "**C:\cygwin\LinCon8k\include**"

The demo path is "**C:\cygwin\LinCon8k\examples**".

2.2 The LinCon-8000 SDK Introduction

In this section, we will discuss some techniques that are adopted in the LinCon-8000. Through our detailed explanations, users can learn how to use the LinCon-8000 easily. LinCon-8000 SDK is based on cygwin and it is also a Linux-like environment for Windows. It still provides a powerful GCC cross-compiler and an IDE (Integrated Development Environment) for developing LinCon-8000 applications quickly. Therefore after you have written your applications, you can compile them through the LinCon-8000 SDK into executable files that can be run in your LinCon-8000 embedded controller.

2.2.1 Introduction to Cygwin

What is Cygwn ? Cygwin is a collection of free software tools originally developed by Cygnus Solutions to allow various versions of Microsoft Windows to act somewhat like a UNIX system. That is Cygwin is a Linux-like environment for Windows. It consists of two parts：

(1) A DLL (cygwin1.dll) which acts as a Linux emulation layer providing substantial Linux API functionality.

(2) A collection of tools, which provide users with the Linux look and feel.

2.2.2 Introduction to Cross-Compilation

What is Cross-Compilation? Generally, compiling a program takes place by running the compiler on the build platform. The compiled program will run on the target platform. Usually these two processes are on the same platform; if they are different, the process is called cross-compilation. That is the process that can compile source code on one platform to the executable files on other platforms. For example, you can compile source code in a x86 windows platform into an executable file that can run on an arm-linux platform if you use the cross-compiler - "**arm-linux-gcc**".

So why do we use Cross-Compilation? In fact, Cross-Compilation is sometimes more involved and errors are easier to make than with normal compilation. Therefore it is often only employed if the target is not able to compile programs on its own or when we want to compile large programs that need more resources than the target can provide. For many embedded systems, cross-compilation is the only possible way.

2.2.3 Download the LinCon-8000 SDK

(1) For Windows system :

Users can download the latest version of LinCon-8000 SDK for Windows system－**linconsdk_for_windows.exe** from ftp://ftp.icpdas.com/pub/cd/linconcd/napdos/linux/sdk/linconsdk_for_windows.exe

(2) For Linux system :

Users can download the latest version of LinCon-8000 SDK for Linux system that include a cross compiler － **linconsdk_for_linux.tar.bz2** as below:

ftp://ftp.icpdas.com/pub/cd/linconcd/napdos/linux/sdk/linconsdk_for_linux.tar.bz2

3.The Architecture of LIBI8K.A in the LinCon-8000

The **libi8k.a** is a library file that is designed for I7000/8000/87000 applications running in the Lincon-8000 Embedded Controller using the Linux OS. Users can apply it to develop their own applications **with GNU C language**. In order to assist users to build their project quickly, we provide many demo programs. Based on these demo programs, users can easily understand how to use these functions and develop their own applications within a short period of time.

The relationships among the libi8k.a and user's applications are depicted as Fig. 3-1：

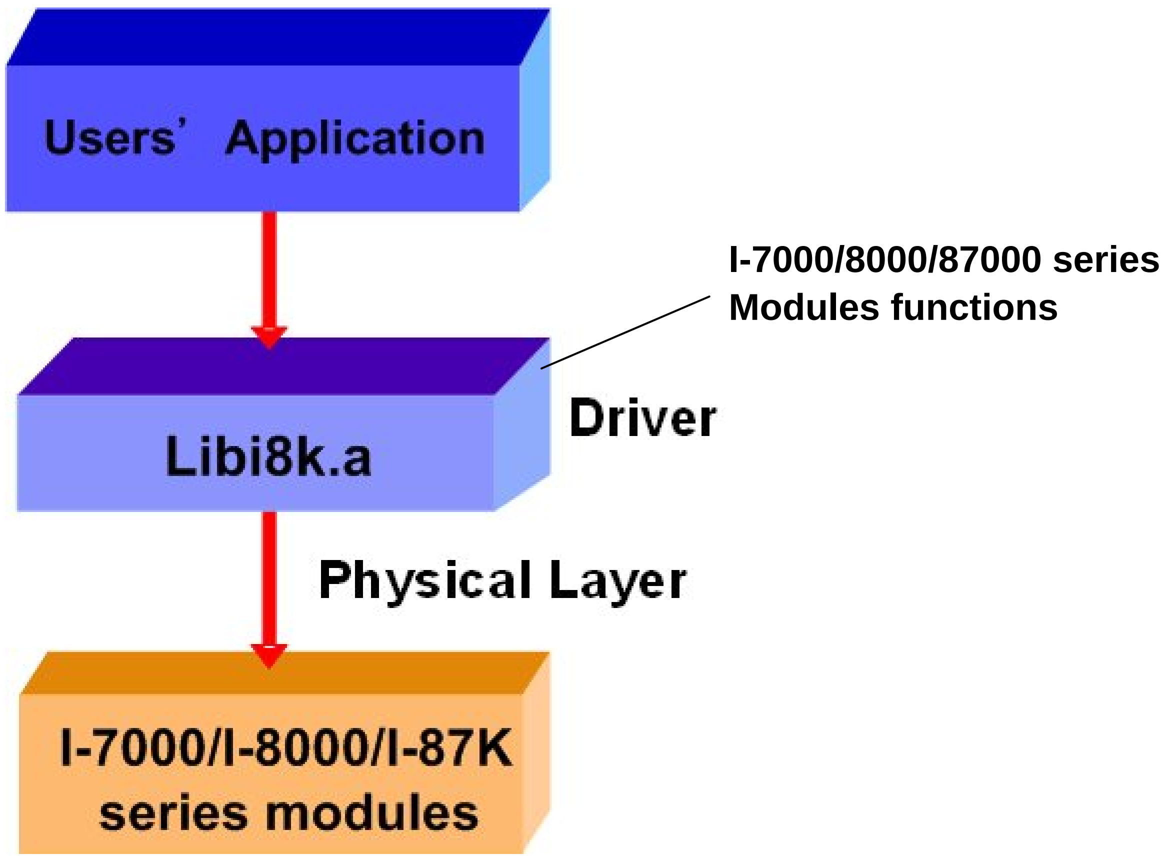


Fig. 3-1

Functions for Lincon-8000 Embedded Controller are divided into sub-groups for ease of use within the different applications：

1. System Information Functions
2. Digital Input/Output Functions
3. Watch Dog Timer Functions
4. EEPROM Read/Write Functions
5. Analog Input Functions
6. Analog Output Functions
7. 3-axis Encoder Functions
8. 2-axis Stepper/Servo Functions

The functions in the Libi8k.a are specially designed for LinCon-8000. Users can easily find the functions they need for their applications from the descriptions in chapter 6 and in the demo programs provided in chapter 7.

4. LinCon-8000 System Settings

In this section, we will introduce how to setup the LinCon-8000 configuration. Let users can use the LinCon-8000 more easily.

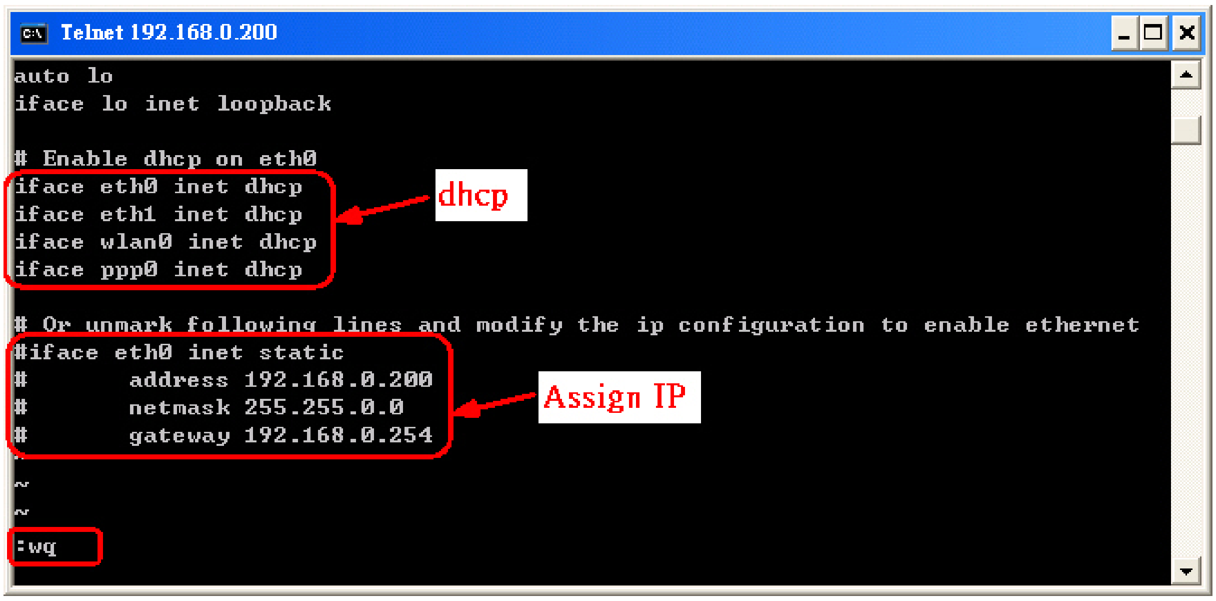
4.1 Settings for the LinCon-8000 Network

The LinCon-8000 network setting includes two ways. One is **DHCP** and the other is "**Assigned IP**". DHCP is the default setting after the LinCon-8000 is produced and this way is easy for users. However, if your network system is without DHCP server, then users need to configure the network setting by using "Assigned IP".

4.1.1 Setting the IP、Netmask and Gateway

(1) Using DHCP :

Boot up LinCon-8000 and click the " **start/xterm** " to open a " **command Prompt** ". Type in " **vi /etc/network/interfaces** " to open the network setting file. Remove " **#** " in the dhcp block and add " **#** " in the Assign IP block. Then type " **:wq** " to save the setting. Type " **ifup eth0** " to make the setting work. (Refer to the Fig 4-1)

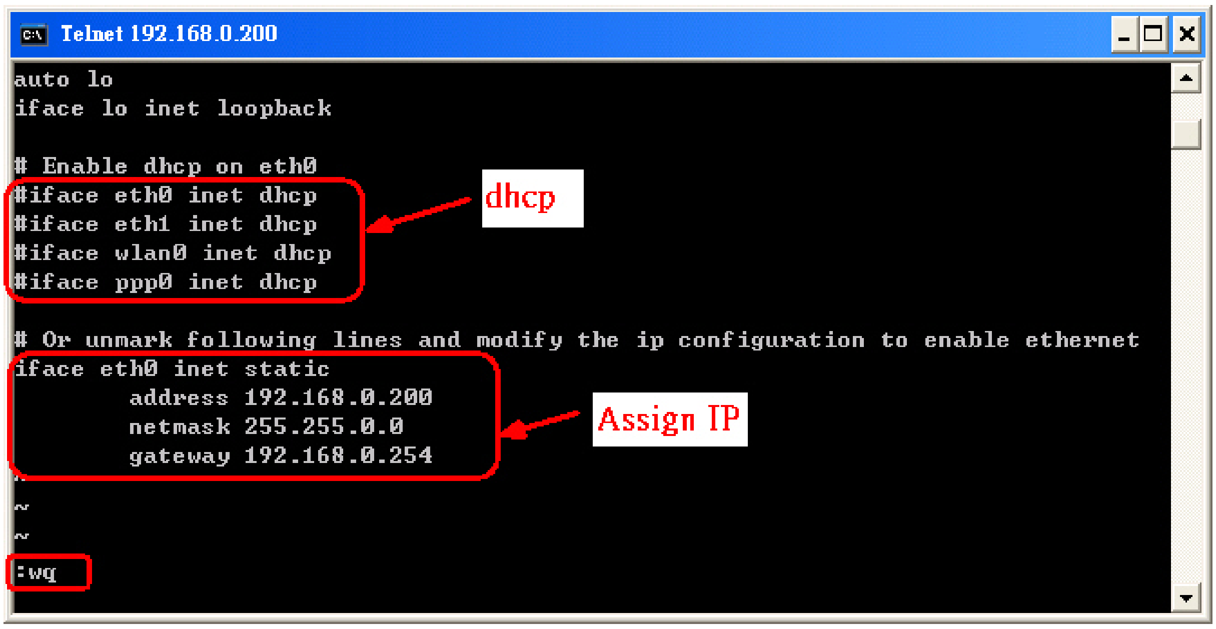


```
auto lo
iface lo inet loopback

# Enable dhcp on eth0
iface eth0 inet dhcp
iface eth1 inet dhcp
iface wlan0 inet dhcp
iface ppp0 inet dhcp

# Or unmark following lines and modify the ip configuration to enable ethernet
#iface eth0 inet static
#       address 192.168.0.200
#       netmask 255.255.0.0
#       gateway 192.168.0.254
~
~
~
:wq
```

Fig 4-1

(2) Using "Assigned IP" :

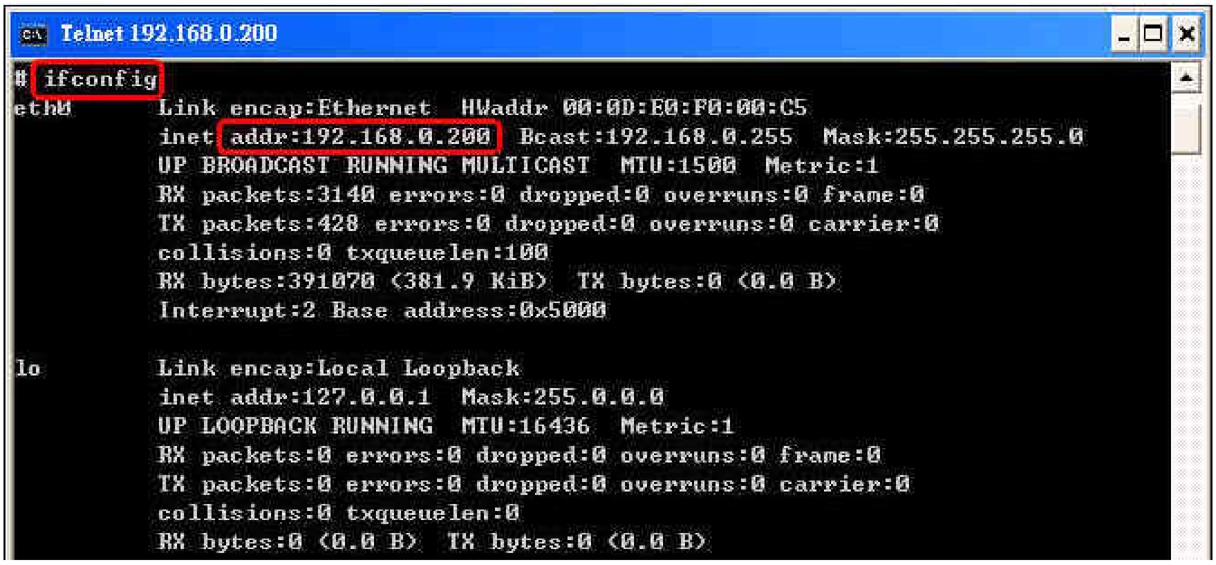
Boot up LinCon-8000 and click the " **start/xterm** " to open a "command line". Type in " **vi /etc/network/interfaces** " to open the network setting file. Remove " # " in the Assign IP block and add " # " in the dhcp block. Type ip、netmask and gateway you want in the Assign IP block. Then type " **:wq** " to save the setting. Type " **ifup eth0** " to make the setting work. (Refer to the Fig 4-2)



```
auto lo
iface lo inet loopback

# Enable dhcp on eth0
#iface eth0 inet dhcp
#iface eth1 inet dhcp
#iface wlan0 inet dhcp
#iface ppp0 inet dhcp

# Or unmark following lines and modify the ip configuration to enable ethernet
iface eth0 inet static
        address 192.168.0.200
        netmask 255.255.0.0
        gateway 192.168.0.254
~
~
~
:wq
```

Fig 4-2

After finish the LinCon network setting, users can type " **ifconfig** " to see the network setting. (Refer to the Fig 4-3)

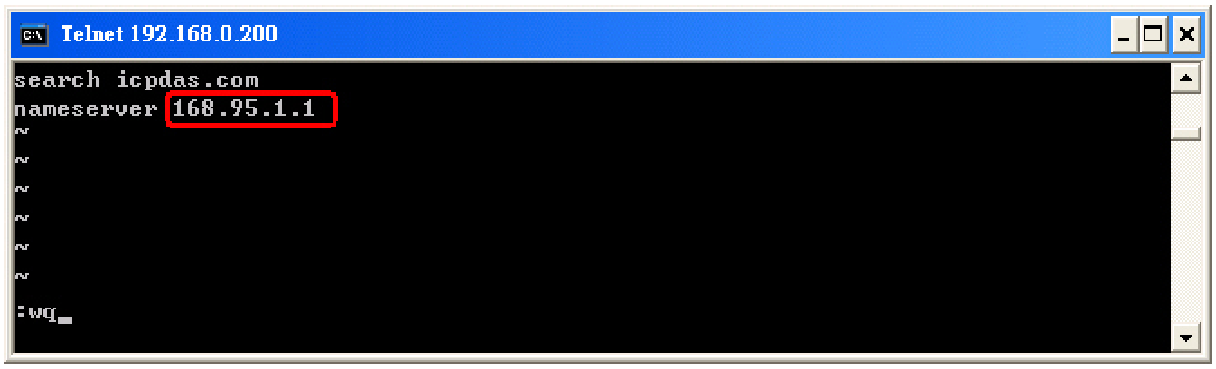


```
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0D:E0:F0:00:C5
          inet addr:192.168.0.200  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3140 errors:0 dropped:0 overruns:0 frame:0
          TX packets:428 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:391070 (381.9 KiB)  TX bytes:0 (0.0 B)
          Interrupt:2 Base address:0x5000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Fig 4-3

4.1.2 Setting of DNS

Boot up LinCon-8000 and click the **“ start/xterm ”** to open a “command line”. Type in **“ vi /etc/resolv.conf ”** to open the DNS setting file. Type “ DNS server ” in the “ **nameserver** “ field. Then type **“ :wq ”** to save the setting. Type **“ reboot “** to reboot the LinCon-8000 to make the setting work. (Refer to the Fig 4-4)



```
search icpdas.com
nameserver 168.95.1.1
~
~
~
~
~
~
:wq
```

Fig 4-4

4.2 CF Card Usage

The contents of CF Card in the LinCon-8000 is in the default path of **/mnt/hda**. Therefore, users can access the files of CF Card in the directory.

4.2.1 Mount CF Card

When you want to use the CF Card, you can insert the CF Card into the Slot of CF Card in the LinCon-8000. (Refer to Fig. 1-3) It will be auto-mounted in the LinCon-8000, and you can access the files of CF Card in the **/mnt/hda** directory.

4.2.2 Umount CF Card

Before you want to pull out the CF Card from the LinCon-8000, you need to type the “ **umount /mnt/hda** “ command first. Then you can pull out the CF Card safely to prevent the damage to CF Card.

4.3 USB Device Usage

Users need to mount the USB device to the LinCon-8000, before they can access the USB device. This is because it will not auto-mount the USB device in the LinCon-8000

4.3.1 Mount USB Device

The steps are as follows：

(1) Type “ **mkdir /mnt/usb** “ to build a usb directory.

(2) Type “ **mount /dev/sda1 /mnt/usb** “ to mount the USB device to the usb directory and type “ **ls /mnt/usb** ” to see the content of USB device.

4.3.2 Umount USB Device

Before users pull out the USB device from the LinCon-8000, users need to type the “ **umount /mnt/usb** “ command first. Then pull out the USB device to prevent any damage to usb device.

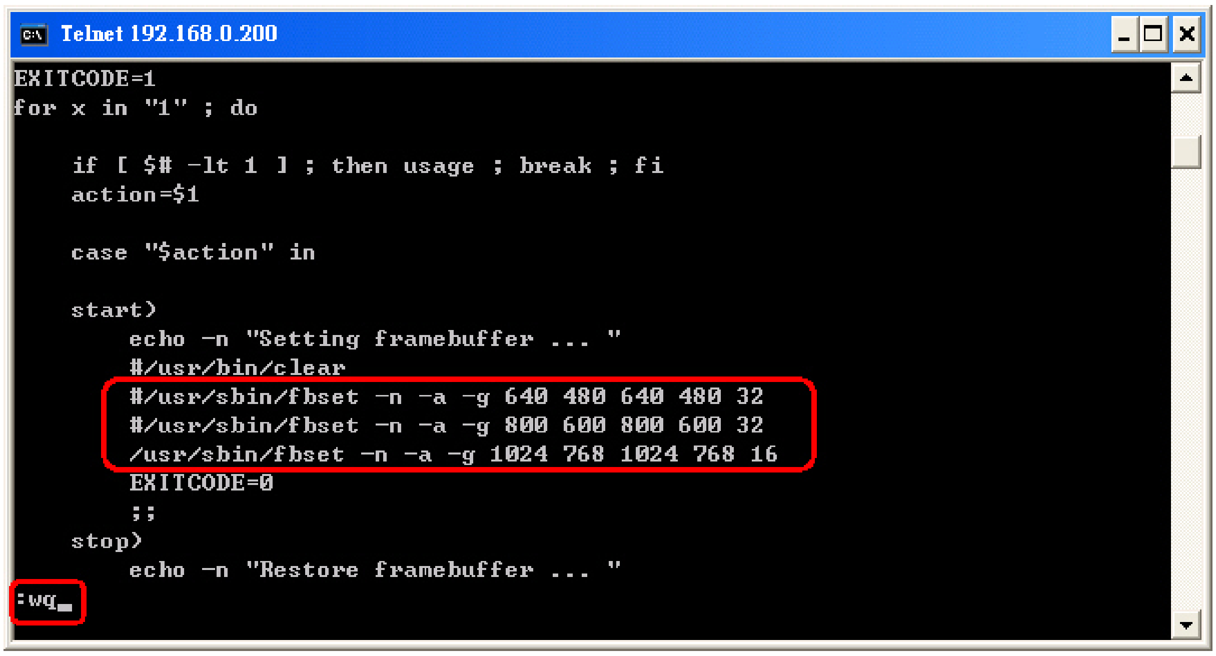
4.4 Adjust VGA Resolution

There are three modes -- **640x480**、**800x600**、**1024x768** supported in the LinCon VGA resolution and the default setting is 800x600. If users want to change the VGA resolution. Please follow below steps :

(1) Type **“ vi /etc/init.d/fbman “** to open resolution setting file.

(2) If users want to set the resolution to be 1024x768. First, Add **“ # ”** in the 800x600 column and then remove **“ # ”** in the 1024x768 column. Type **“ :wq “** to save the setting. (Refer to Fig 4-5)

(3) Type **“ reboot “** to reboot LinCon-8000.



```
EXITCODE=1
for x in "1" ; do

    if [ $# -lt 1 ] ; then usage ; break ; fi
    action=$1

    case "$action" in

    start)
        echo -n "Setting framebuffer ... "
        #/usr/bin/clear
        #/usr/sbin/fbset -n -a -g 640 480 640 480 32
        #/usr/sbin/fbset -n -a -g 800 600 800 600 32
        /usr/sbin/fbset -n -a -g 1024 768 1024 768 16
        EXITCODE=0
        ;;
    stop)
        echo -n "Restore framebuffer ... "
:wq
```

Fig 4-5

4.5 Running applications automatically at boot time

A “run level” determines which programs are executed at system startup. Run level 2 is the default run level of LinCon-8000.

The contents of run level are in the /etc/init.d directory that directory contains the scripts executed at boot time. These scripts are referenced by symbolic links in the /etc/rc2.d.

These links are named S<2-digit-number><original-name>. The numbers determine the order in which the scripts are run, from 00 to 99 — the lower number would earlier executed. Scripts named with an **S** are called with start, and named with a **K or x** are called with stop.

4.5.1 Making program run at boot time

Making program run at boot time, you should create a startup script placed in /etc/init.d directory that runs the required commands for executed automatically at boot time and be symbolically linked to /etc/rc2.d directory.

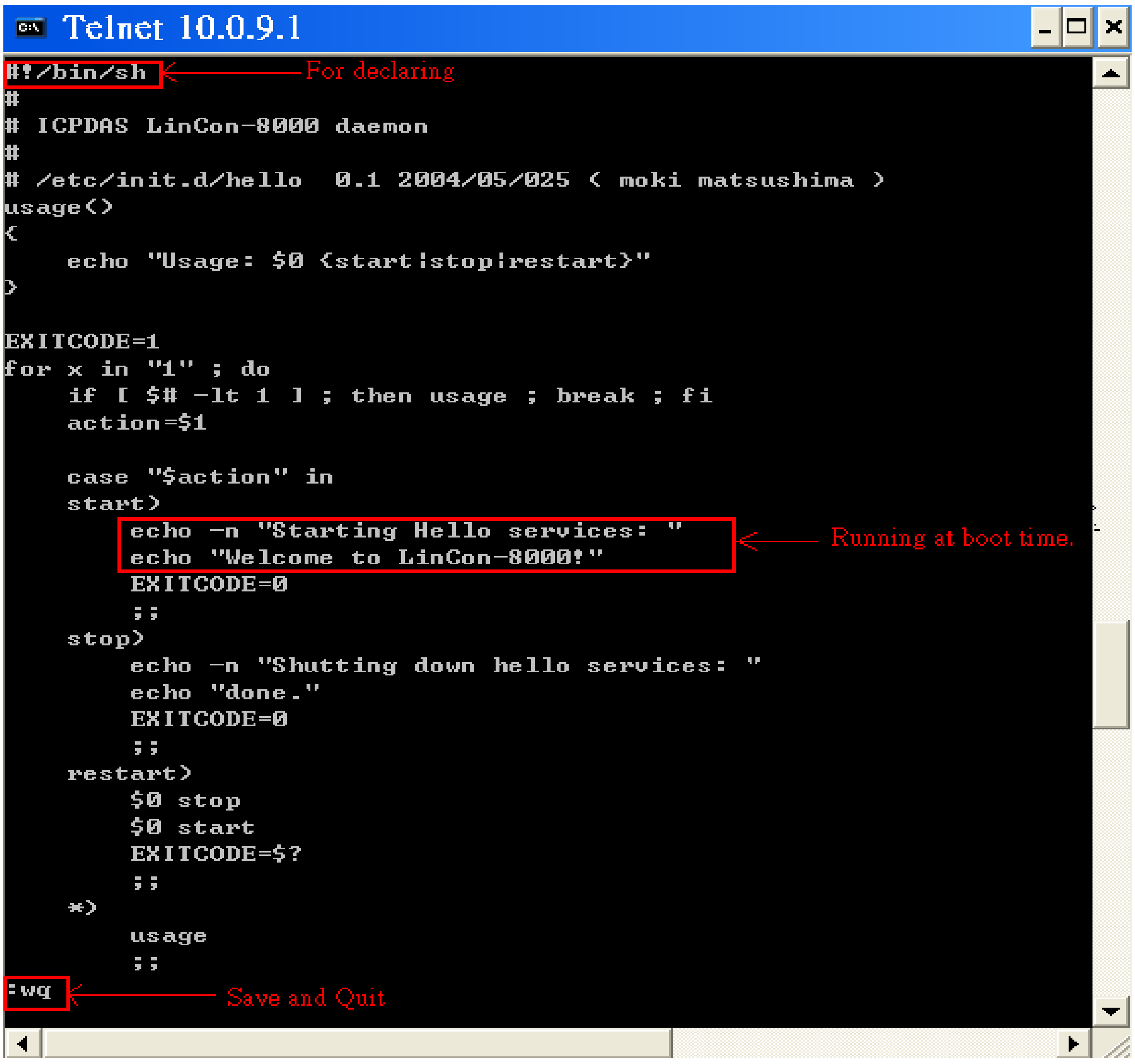
The steps are as follows :

(1) Type “ **vi /etc/init.d/hello** “ to edit a script that would like to executed program, filename is hello. Type “ **:wq** “ to save and quit the script. (Refer to the Fig 4-6)

(2) Type “ **chmod 755 /etc/init.d/hello** “ to change authority.

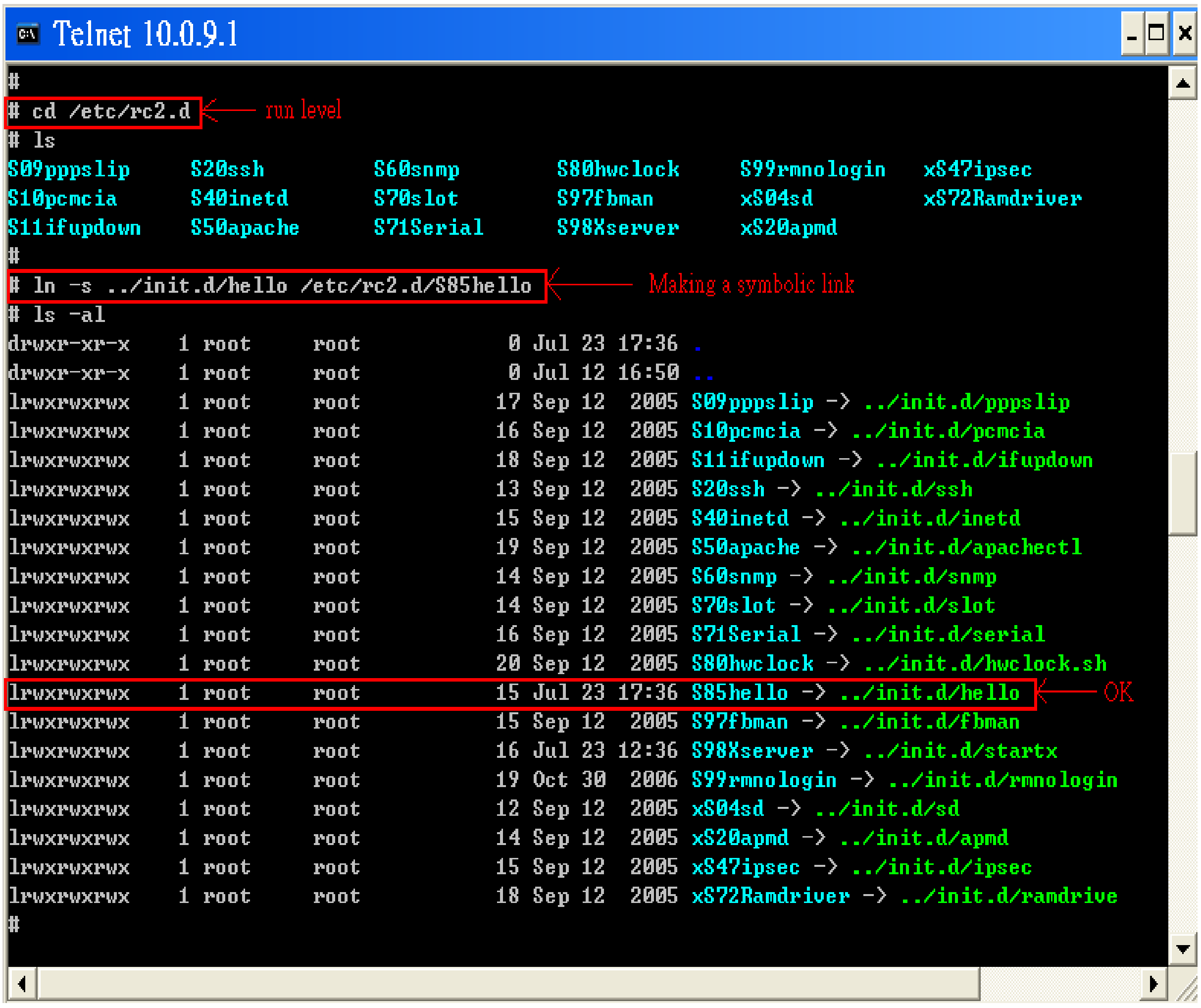
(3) Type “ **cd /etc/rc2.d** “ to into default run level.

(4) Type ” **ln -s ../init.d/hello /etc/rc2.d/S85hello** “ to make a symbolic link into the script file and it will be executed automatically at boot time. (Refer to the Fig 4-7)



```
#!/bin/sh
#
# ICPDAS LinCon-8000 daemon
#
# /etc/init.d/hello  0.1 2004/05/025 ( moki matsushima )
usage()
{
    echo "Usage: $0 {start|stop|restart}"
}

EXITCODE=1
for x in "1" ; do
    if [ $# -lt 1 ] ; then usage ; break ; fi
    action=$1

    case "$action" in
    start)
        echo -n "Starting Hello services: "
        echo "Welcome to LinCon-8000!"
        EXITCODE=0
        ;;
    stop)
        echo -n "Shutting down hello services: "
        echo "done."
        EXITCODE=0
        ;;
    restart)
        $0 stop
        $0 start
        EXITCODE=$?
        ;;
    *)
        usage
        ;;
:wq
```

Fig. 4-6



```
#
# cd /etc/rc2.d
# ls
S09pppslip     S20ssh         S60snmp        S80hwclock     S99rmnologin   xS47ipsec
S10pcmcia      S40inetd       S70slot        S97fbman       xS04sd         xS72Ramdriver
S11ifupdown    S50apache      S71Serial      S98Xserver     xS20apmd
#
# ln -s ../init.d/hello /etc/rc2.d/S85hello
# ls -al
drwxr-xr-x    1 root     root            0 Jul 23 17:36 .
drwxr-xr-x    1 root     root            0 Jul 12 16:50 ..
lrwxrwxrwx    1 root     root           17 Sep 12  2005 S09pppslip -> ../init.d/pppslip
lrwxrwxrwx    1 root     root           16 Sep 12  2005 S10pcmcia -> ../init.d/pcmcia
lrwxrwxrwx    1 root     root           18 Sep 12  2005 S11ifupdown -> ../init.d/ifupdown
lrwxrwxrwx    1 root     root           13 Sep 12  2005 S20ssh -> ../init.d/ssh
lrwxrwxrwx    1 root     root           15 Sep 12  2005 S40inetd -> ../init.d/inetd
lrwxrwxrwx    1 root     root           19 Sep 12  2005 S50apache -> ../init.d/apachectl
lrwxrwxrwx    1 root     root           14 Sep 12  2005 S60snmp -> ../init.d/snmp
lrwxrwxrwx    1 root     root           14 Sep 12  2005 S70slot -> ../init.d/slot
lrwxrwxrwx    1 root     root           16 Sep 12  2005 S71Serial -> ../init.d/serial
lrwxrwxrwx    1 root     root           20 Sep 12  2005 S80hwclock -> ../init.d/hwclock.sh
lrwxrwxrwx    1 root     root           15 Jul 23 17:36 S85hello -> ../init.d/hello
lrwxrwxrwx    1 root     root           15 Sep 12  2005 S97fbman -> ../init.d/fbman
lrwxrwxrwx    1 root     root           16 Jul 23 12:36 S98Xserver -> ../init.d/startx
lrwxrwxrwx    1 root     root           19 Oct 30  2006 S99rmnologin -> ../init.d/rmnologin
lrwxrwxrwx    1 root     root           12 Sep 12  2005 xS04sd -> ../init.d/sd
lrwxrwxrwx    1 root     root           14 Sep 12  2005 xS20apmd -> ../init.d/apmd
lrwxrwxrwx    1 root     root           15 Sep 12  2005 xS47ipsec -> ../init.d/ipsec
lrwxrwxrwx    1 root     root           18 Sep 12  2005 xS72Ramdriver -> ../init.d/ramdrive
#
```

Fig. 4-7

4.5.2 Disabling program run at boot time

The steps are as follows :

(1) Type “ **cd /etc/rc2.d** “ to into default run level.

(2) Type “ **mv S85hello xS85hello** “ to rename the S85hello symbolic link for turn off running program automatically at boot time.

5. Instructions for the LinCon-8000

In this section, some Linux instructions that are often used will be introduced. The use of these instructions in linux is very familiar with those in DOS and generally they are **used in lower case.**

5.1 Basic Linux Instructions

5.1.1 ls : list the file information －＞ (like dir in DOS)

Parameter：

(1) -l：list detailed information of file (Example：ls -l)

(2) -a：list all files including hidden files (Example：ls -a)

(3) -t：list the files that are arranged by time(from new to old)

5.1.2 cd directory : Change directory －＞ (like cd in DOS)

Parameter：

(1) **..**：move to the upper directory (Example：cd **..**)

(2) ~：move back to the root directory (Example：cd ~)

(3) /：divided sign (for examples：cd /root/i8k)

5.1.3 mkdir：create the subdirectory －＞ (like md in DOS)

mkdir –parameter subdirectory

(Example：mkdir owner)

5.1.4 rmdir：delete(remove) the subdirectory and it must be empty －＞ (like rd in DOS)

mkdir –parameter subdirectory

(Example：rmdir owner)

5.1.5 rm : delete file or directory －＞ (like del or deltree in DOS)

rm –parameter file (or directory)

Parameter：

(1) i：it will show the warning message when deleting (Example：rm -i test.exe)

(2) r：delete directory despite that it isn't empty (Example：rm –r Test)

(3) f：it will not show a warning message when deleting (Example：rm -f test.exe)

5.1.6 cp：copy file －＞ (like copy in DOS)

cp –parameter source file destination file

(Example：cp test.exe /root/Test/test.exe)

5.1.7 mv：move or rename file or directory －＞ (like move or ren in DOS)

mv –parameter source file (or directory) destination file (or directory)

(Example：mv test.exe test1.exe)

(Example：mv test.exe /root/Test)

5.1.8 pwd：show the current path

5.1.9 who：show the on-line users

5.1.10 chmod：change authority of file

chmod ??? file －＞ ??? means owner：group：all users

For example：

chmod 754 test.exe

7 5 4 －＞ <u>111</u>(read, write, execute) <u>101</u>(read, write, execute) <u>100</u>(read, write, execute)

The first number 7 ：**owner** can read and write and execute files

The second number 5：**group** can only read and execute files

The third number 4 ：**all users** can only read files

5.1.11 uname：show the version of linux

5.1.12 ps：show the procedures that execute now

5.1.13 ftp：transfer file

ftp IPAdress (Example：ftp 192.168.0.200 －＞ connet to ftp server)

！：exit FTP back to pc temporarily；**exit**：back to ftp

bin：transfer files in “binary” mode

get：download file from LinCon to PC (Ex：get /mnt/hda/test.exe c:/test.exe)

put：upload file from PC to LinCon (Ex：put c:/test.exe /mnt/hda/test.exe)

bye：exit FTP

5.1.14 telnet：connect to other PC

telnet IPAddress (Example：telnet 192.168.0.200－＞remote control LinCon-8000)

5.1.15 date：show date and time

5.1.16 netstat：show the state of network

Parameter：

(1) -a：list all states (Example：netstat -a)

5.1.17 ifconfig：show the ip and network mask (like ipconfig in DOS)

5.1.18 ping：check to see if the host in the network is alive

ping IPAddress (Example：ping 192.168.0.1)

5.1.19 clear：clear the screen

5.1.20 passwd：change the password

5.1.21 reboot：reboot the LinCon

5.2 General GCC Instructions

GCC is a cross-compiler provided by GNU and it can compile source code written by ANSI C or by Tranditional C into executable files. The executable file compiled by GCC can run in different OSs and in different Hardware systems. Therefore GCC is very popular within the Unix system which is a large part of why its popularity is growing so well. Furthermore it is free, and therefore can be downloaded via your network with ease.

First, Fig. 5-1 illustrates the compilation procedure within Linux：

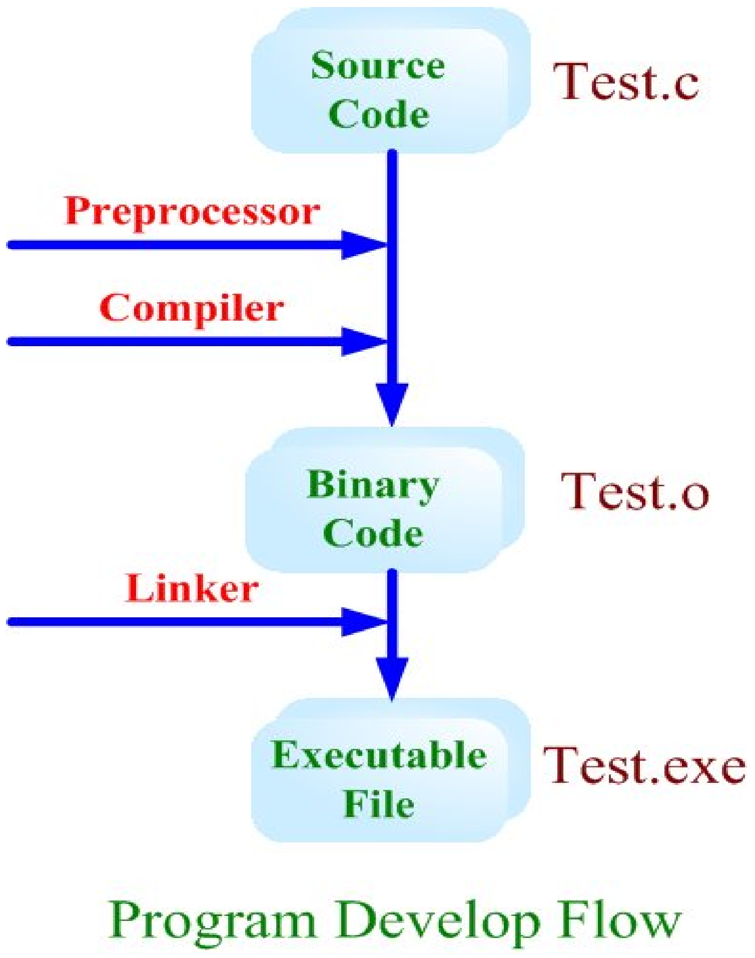


Fig. 5-1

Second, we will list some GCC instructions to let users compile *.c to *.exe smoothly and to explain the parameters for GCC in its compilation process.

5.2.1 Compile without linking the LinCon-8000 library

(1) Purpose： *. c   to   *. exe

Command：gcc   –o   target   source.c

Parameter：

-o target：assign the name of output file

source.c：source code of C

Example：gcc   –o   helloworld.exe   helloworld.c

Output File：helloworld.exe

5.2.2 Compile with linking the LinCon-8000 library (libi8k.a)

(1) Purpose：*. c to *. o

Command：arm-linux-gcc –lincludeDIR -lm –c –o target source.c library

Parameter：

-lincludeDir：the path of include files

-lm：include math library (libm.a)

-c：just compile *.c to *.o (object file)

-o target：assign the name of output file

source.c：source code of C

library：the path of library

Example：arm-linux-gcc –I. –I../include –lm –c –o test.o test.c ../lib/libi8k.a

Output File：test.o

(2) Purpose：*. o to *. exe

Command：arm-linux-gcc –lincludeDIR -lm –o target source.o library

Parameter：

-lincludeDir：the path of include files

-lm：include math library (libm.a)

-o target：assign the name of output file

source.o：object file

library：the path of library

Example：arm-linux-gcc –I. –I../include –lm –o test.exe test.o ../lib/libi8k.a

Output File：test.exe

(3) Purpose：*. c to *. exe

Command：arm-linux-gcc –lincludeDIR -lm –o target source.c library

Parameter：

-lincludeDir：the path of include files

-lm：include math library (libm.a)

-o target：assign the name of output file source.c：source code of C

library：the path of library

Example：arm-linux-gcc –I. –I../include –lm –o test.exe test.c ../lib/libi8k.a

Output File：test.exe

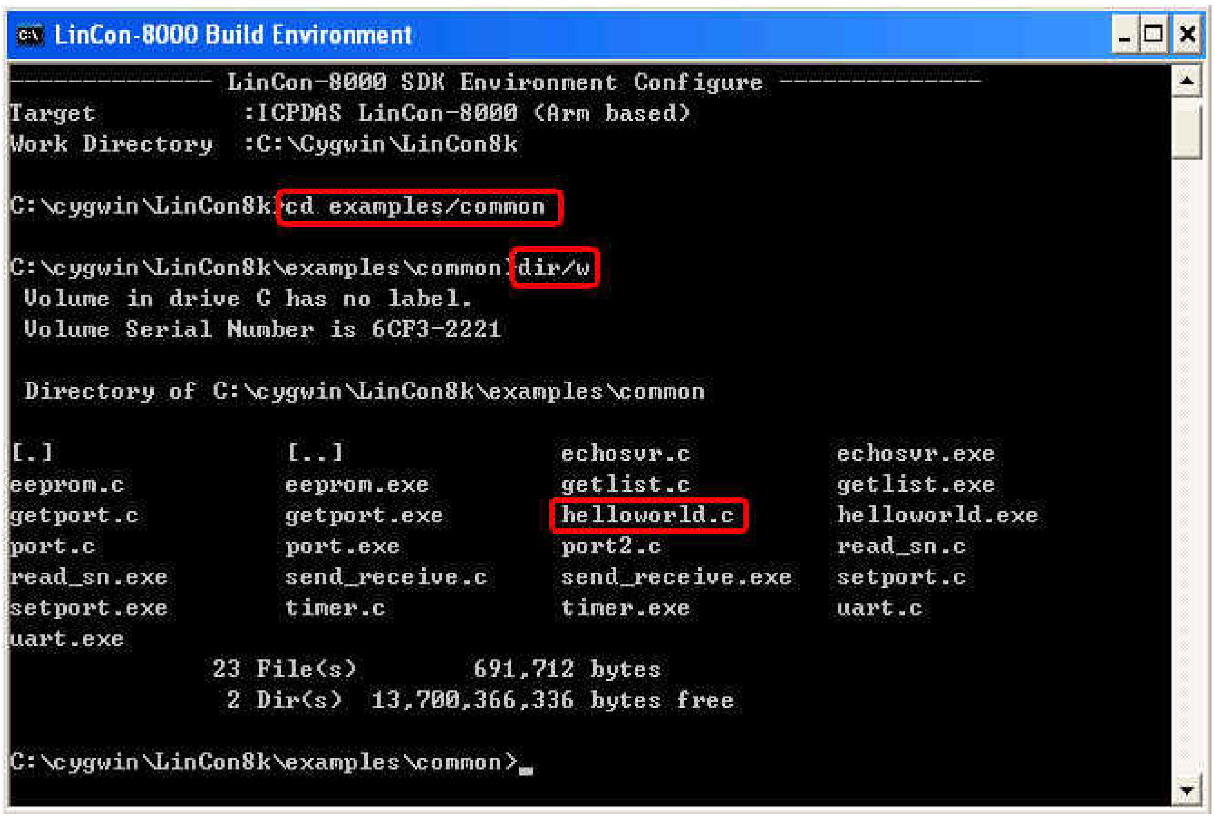
5.3 A Simple Example – Helloworld.c

In this section, we will introduce how to compile the helloworld.c to helloworld.exe and transfer the helloworld.exe to the LinCon-8000 by using FTP. Finally executes this file via the Telnet Server on the LinCon-8000. These steps can be accomplished in one pc without another monitor for the LinCon-8000. In this example, no ICPDAS modules are used. If you want to use the modules of ICPDAS to control your system, you can refer to demo in the chapter 7.

These processes can be divided into three steps and thet are given as below：

STEP 1 ：(Compile helloworld.c to helloworld.exe)

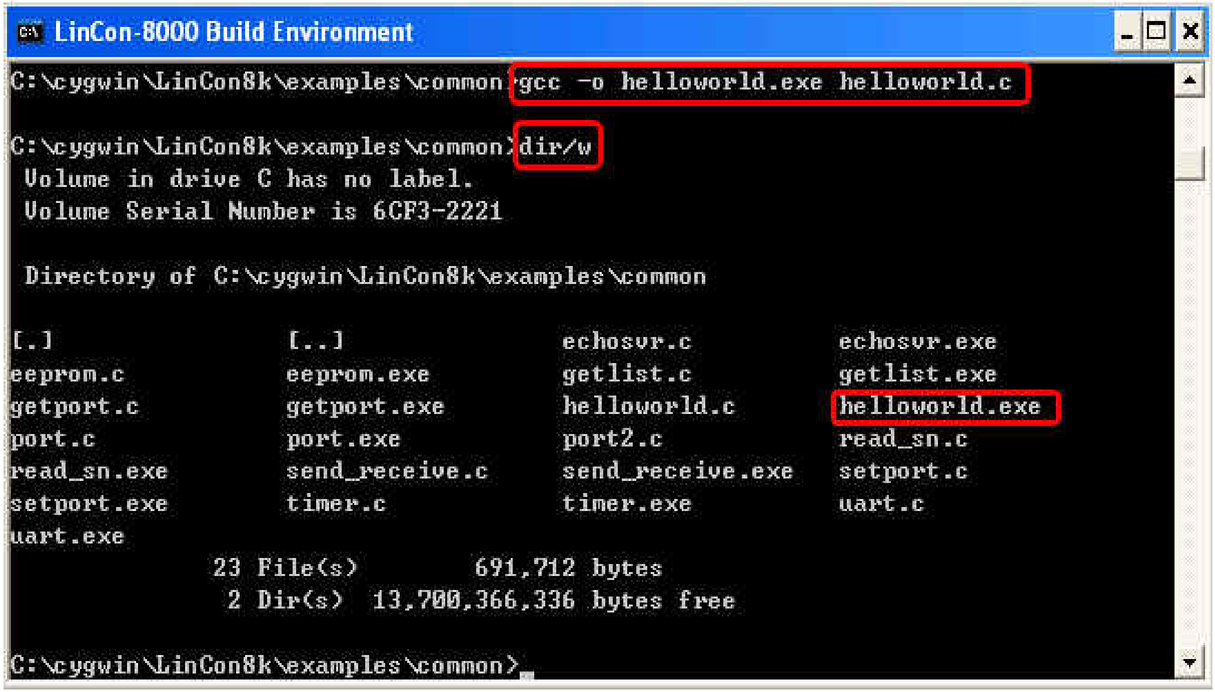
(1) Open LinCon-8000 SDK (refer to step 8 in section 2.1) and type “ **cd examples/common** ” to change the path to **C:/cygwin/LinCon8k/examples/common**. Type “**dir/w**” and you can see the helloworld.c file. (refer to Fig.5-2)



```
LinCon-8000 Build Environment
-------------- LinCon-8000 SDK Environment Configure --------------
Target          :ICPDAS LinCon-8000 (Arm based)
Work Directory  :C:\Cygwin\LinCon8k

C:\cygwin\LinCon8k>cd examples/common

C:\cygwin\LinCon8k\examples\common>dir/w
 Volume in drive C has no label.
 Volume Serial Number is 6CF3-2221

 Directory of C:\cygwin\LinCon8k\examples\common

[.]                 [..]                echosvr.c           echosvr.exe
eeprom.c            eeprom.exe          getlist.c           getlist.exe
getport.c           getport.exe         helloworld.c        helloworld.exe
port.c              port.exe            port2.c             read_sn.c
read_sn.exe         send_receive.c      send_receive.exe    setport.c
setport.exe         timer.c             timer.exe           uart.c
uart.exe
              23 File(s)        691,712 bytes
               2 Dir(s)  13,700,366,336 bytes free

C:\cygwin\LinCon8k\examples\common>_
```

Fig. 5-2

(2) Type in “**arm-linux-gcc –o helloworld.exe helloworld.c**“ to compile helloworld.c into helloworld.exe. Then type “**dir/w**“ to see the helloworld.exe file. (refer to Fig.5-3)



```
LinCon-8000 Build Environment
C:\cygwin\LinCon8k\examples\common>gcc -o helloworld.exe helloworld.c

C:\cygwin\LinCon8k\examples\common>dir/w
 Volume in drive C has no label.
 Volume Serial Number is 6CF3-2221

 Directory of C:\cygwin\LinCon8k\examples\common

[.]                 [..]                echosvr.c           echosvr.exe
eeprom.c            eeprom.exe          getlist.c           getlist.exe
getport.c           getport.exe         helloworld.c        helloworld.exe
port.c              port.exe            port2.c             read_sn.c
read_sn.exe         send_receive.c      send_receive.exe    setport.c
setport.exe         timer.c             timer.exe           uart.c
uart.exe
              23 File(s)        691,712 bytes
               2 Dir(s)  13,700,366,336 bytes free

C:\cygwin\LinCon8k\examples\common>_
```

Fig. 5-3

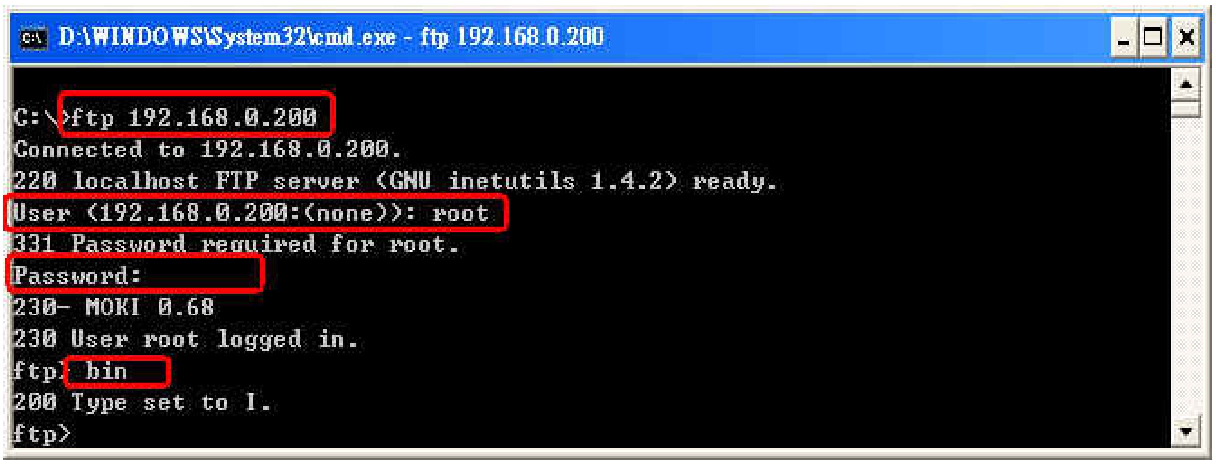
STEP 2 ：(Transfer helloworld.exe to the LinCon-8000)

There are two methods for transferring files to the LinCon-8000：

＜ **Method one** ＞ **By Using the “DOS Command Prompt”：**

(1) Open a “DOS Command Prompt” and type in the ftp IPAddress of the LinCon-8000 (Example：**ftp 192.168.0.200**) to connect to the FTP Server on the LinCon-8000. Then type the **User_Name** and **Password (“ root ” is the default value.)** to accomplish the connection from the PC to the LinCon-8000.

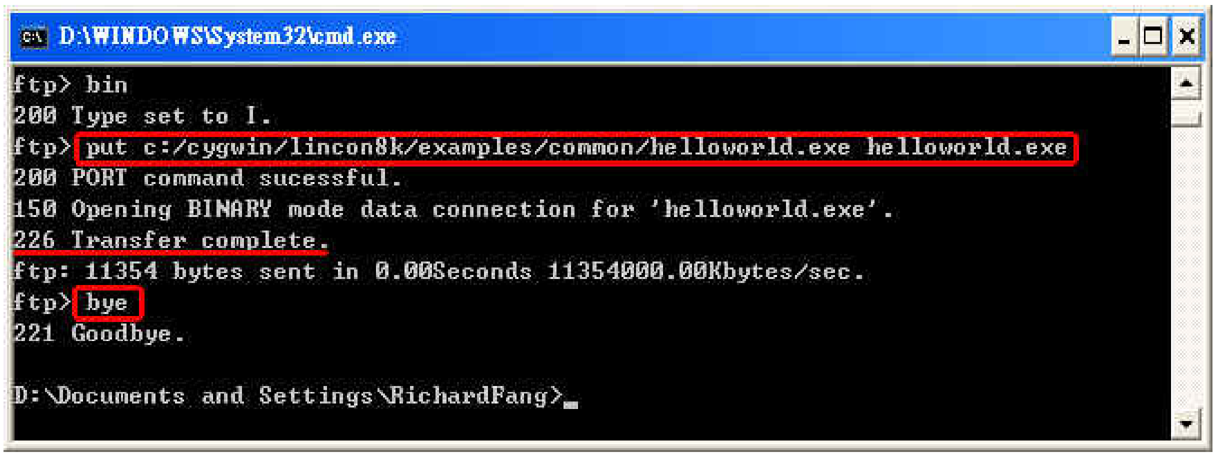
(2) Before transferring your files to the LinCon-8000, type in the “**bin**” command to make the file transfer to the LinCon-8000 **in binary mode**. (refer to Fig.5-4)



```
C:\>ftp 192.168.0.200
Connected to 192.168.0.200.
220 localhost FTP server (GNU inetutils 1.4.2) ready.
User (192.168.0.200:(none)): root
331 Password required for root.
Password:
230- MOXI 0.68
230 User root logged in.
ftp> bin
200 Type set to I.
ftp>
```

Fig.5-4

(3) Type in “ **put C:/cygwin/LinCon8k/examples/common/helloworld.exe helloworld.exe** ” to transfer helloworld.exe to the LinCon-8000. If it shows the message of “ **Transfer complete** ”, then the whole transferring process has been accomplished. If you need to disconnect from the LinCon-8000, type in the “ **bye** ” command to return to the PC console. (refer to Fig.5-5).

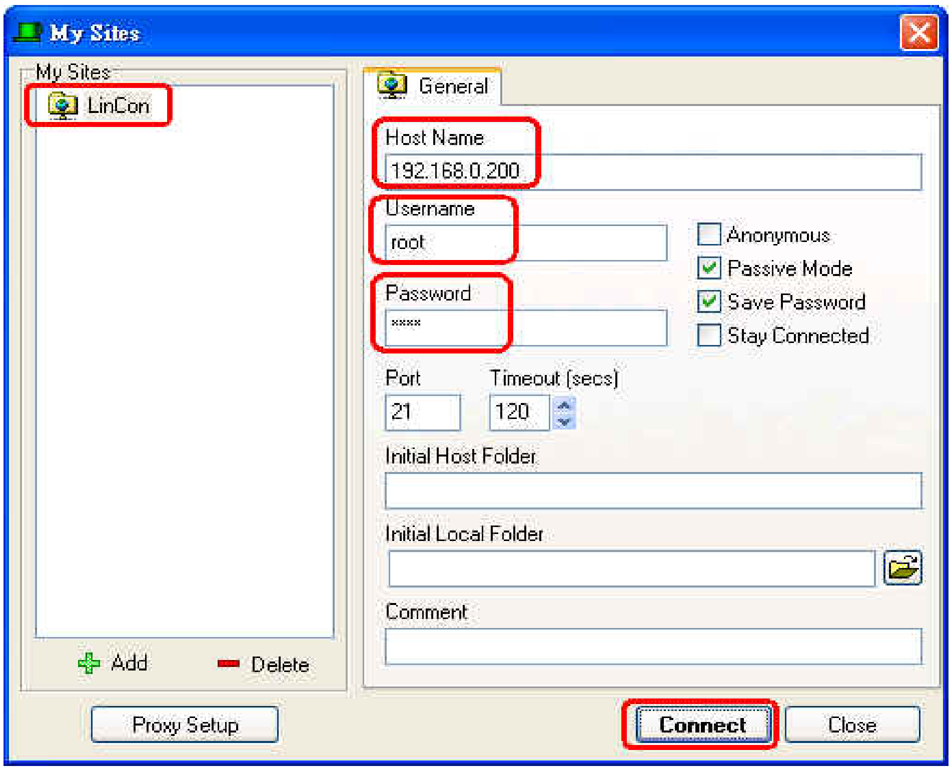


```
ftp> bin
200 Type set to I.
ftp> put c:/cygwin/lincon8k/examples/common/helloworld.exe helloworld.exe
200 PORT command sucessful.
150 Opening BINARY mode data connection for 'helloworld.exe'.
226 Transfer complete.
ftp: 11354 bytes sent in 0.00Seconds 11354000.00Kbytes/sec.
ftp> bye
221 Goodbye.

D:\Documents and Settings\RichardFang>_
```

Fig.5-5

＜ **Method two** ＞ **By Using FTP Software**：

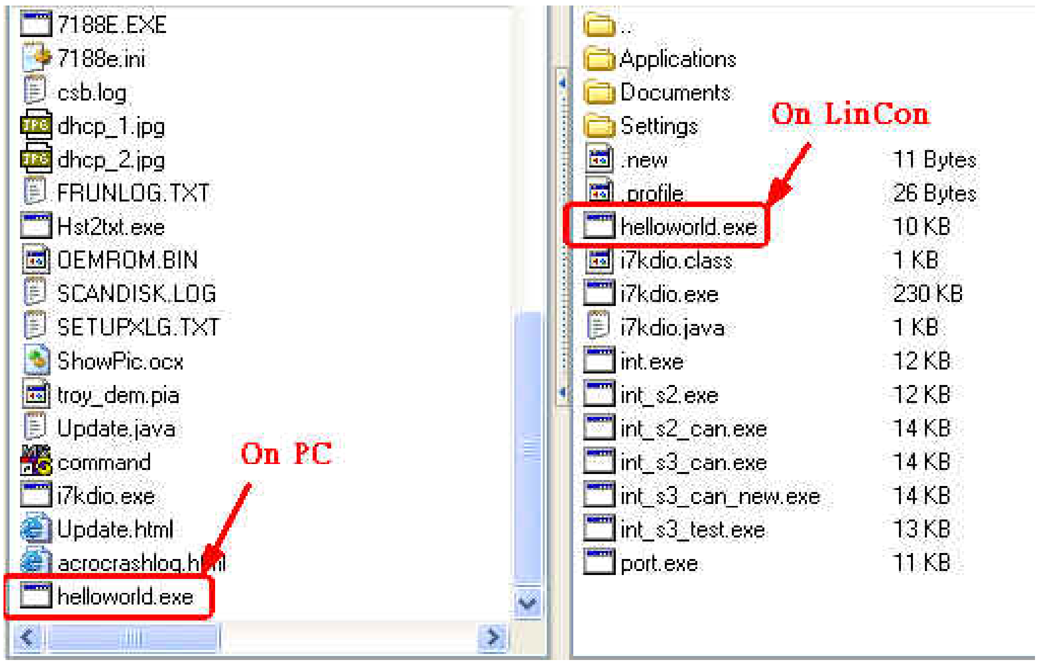
(1) Open the FTP Software and add a ftp site to the LinCon-8000. The **User_Name** and **Password** default value is **" root "**. Then click the **"Connect"** button to connect to the ftp server of the LinCon-8000. (refer to Fig.5-6).



Host Name: 192.168.0.200
Username: root
Password: ****
Anonymous [ ]
Passive Mode [✓]
Save Password [✓]
Stay Connected [ ]
Port: 21
Timeout (secs): 120
Initial Host Folder:
Initial Local Folder:
Comment:

Fig.5-6

(2) Upload the file - **Helloworld.exe** to the LinCon-8000. (refer to Fig.5-7).



```
7188E.EXE                ..
7188e.ini                Applications
csb.log                  Documents
dhcp_1.jpg               Settings
dhcp_2.jpg               .new                 11 Bytes
FRUNLOG.TXT              .profile             26 Bytes
Hst2txt.exe              helloworld.exe       10 KB
OEMROM.BIN               i7kdio.class         1 KB
SCANDISK.LOG             i7kdio.exe           230 KB
SETUPXLG.TXT             i7kdio.java          1 KB
ShowPic.ocx              int.exe              12 KB
troy_dem.pia             int_s2.exe           12 KB
Update.java              int_s2_can.exe       14 KB
command                  int_s3_can.exe       14 KB
i7kdio.exe               int_s3_can_new.exe   14 KB
Update.html              int_s3_test.exe      13 KB
acrocrashlog.html        port.exe             11 KB
helloworld.exe
```

Fig.5-7

(3) Choose helloworld.exe in the LinCon-8000 and click the right button of mouse to choose the “ **Permissions** ” option. Then type 777 into the Numeric textbox. (refer to Fig.5-8 and Fig.5-9).

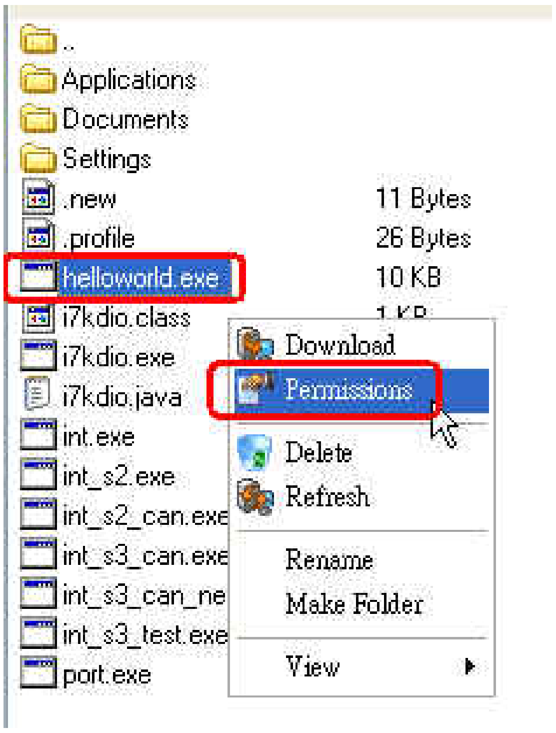
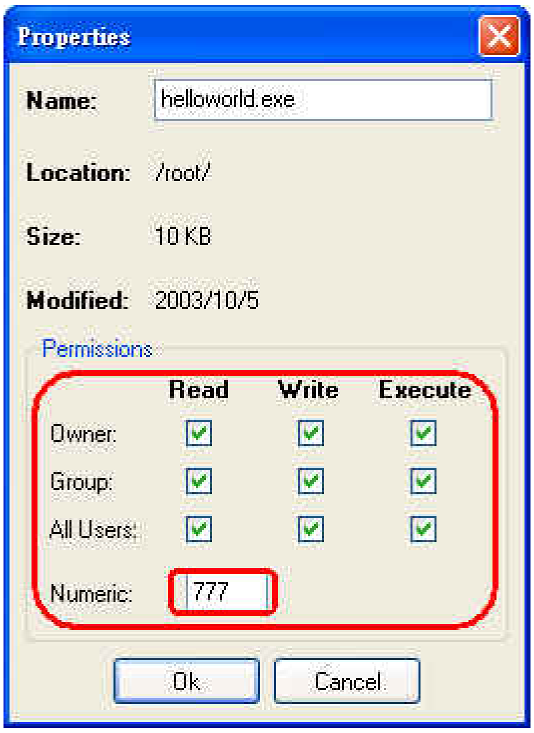


Fig.5-8



Name: helloworld.exe
Location: /root/
Size: 10 KB
Modified: 2003/10/5

Permissions

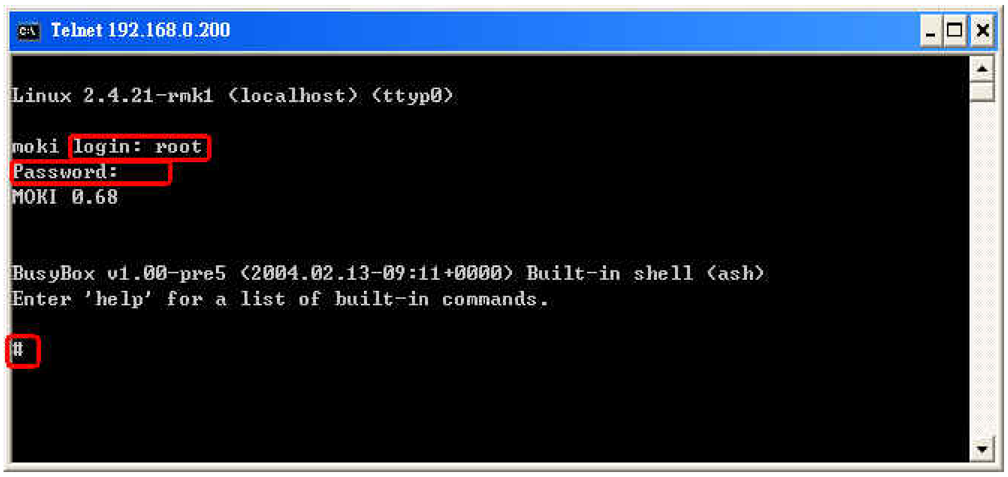
	Read	Write	Execute
Owner:	✓	✓	✓
Group:	✓	✓	✓
All Users:	✓	✓	✓

Numeric: 777

Fig.5-9

STEP 3 ：(Telnet to the LinCon-8000 and execute program)

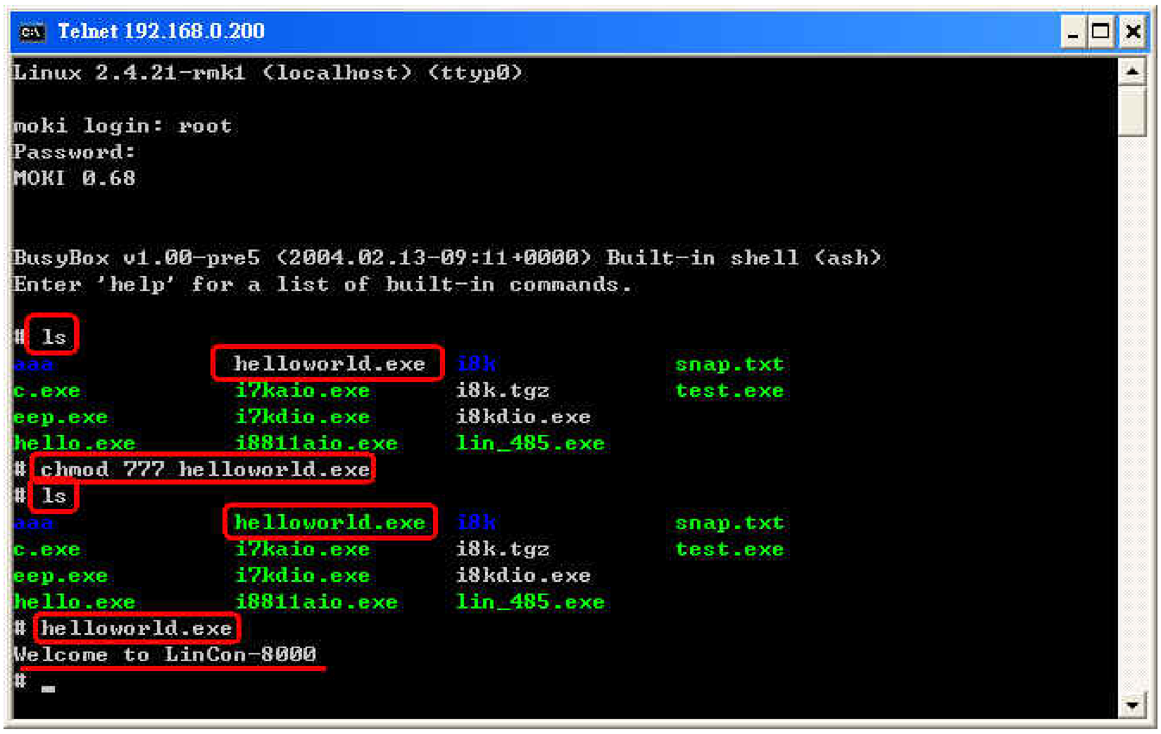
(1) Open a “ DOS Command Prompt ” and then type in the <u>telnet IPAddress of the LinCon-8000</u> (Example：**<u>telnet 192.168.0.200</u>**) to connect to the telnet server of the LinCon-8000. Then type the **User_Name** and **Password (“ <u>root</u> ” is the default value.)**. If it shows the **“ # “** prompt character, the process of connecting from your PC to the telnet server of the LinCon-8000 is finished. (refer to Fig.5-10)



```
Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.02.13-09:11+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

#
```

Fig.5-10

(2) Type in the “**ls**“ command in order to list all the files in /root and to see the <u>helloworld.exe</u> file. If its color is white and then type in the “**chmod 777 helloworld.exe**” command to change the authority of helloworld.exe and then type in the “**ls**“ command again to see “helloworld.exe” and its color will now be green. This means that the file is executable. Type in “**helloworld.exe**“ to execute the file and it will show “ Welcome to LinCon-8000 ”. Then all the steps from <u>**compile**</u>、 <u>**transfer**</u> to <u>**telnet to execute program**</u> will be completed. (refer to Fig.5-11)



```
Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.02.13-09:11+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# ls
aaa                 helloworld.exe  i8k             snap.txt
c.exe               i7kaio.exe      i8k.tgz         test.exe
eep.exe             i7kdio.exe      i8kdio.exe
hello.exe           i8811aio.exe    lin_485.exe
# chmod 777 helloworld.exe
# ls
aaa                 helloworld.exe  i8k             snap.txt
c.exe               i7kaio.exe      i8k.tgz         test.exe
eep.exe             i7kdio.exe      i8kdio.exe
hello.exe           i8811aio.exe    lin_485.exe
# helloworld.exe
Welcome to LinCon-8000
# _
```

Fig.5-11

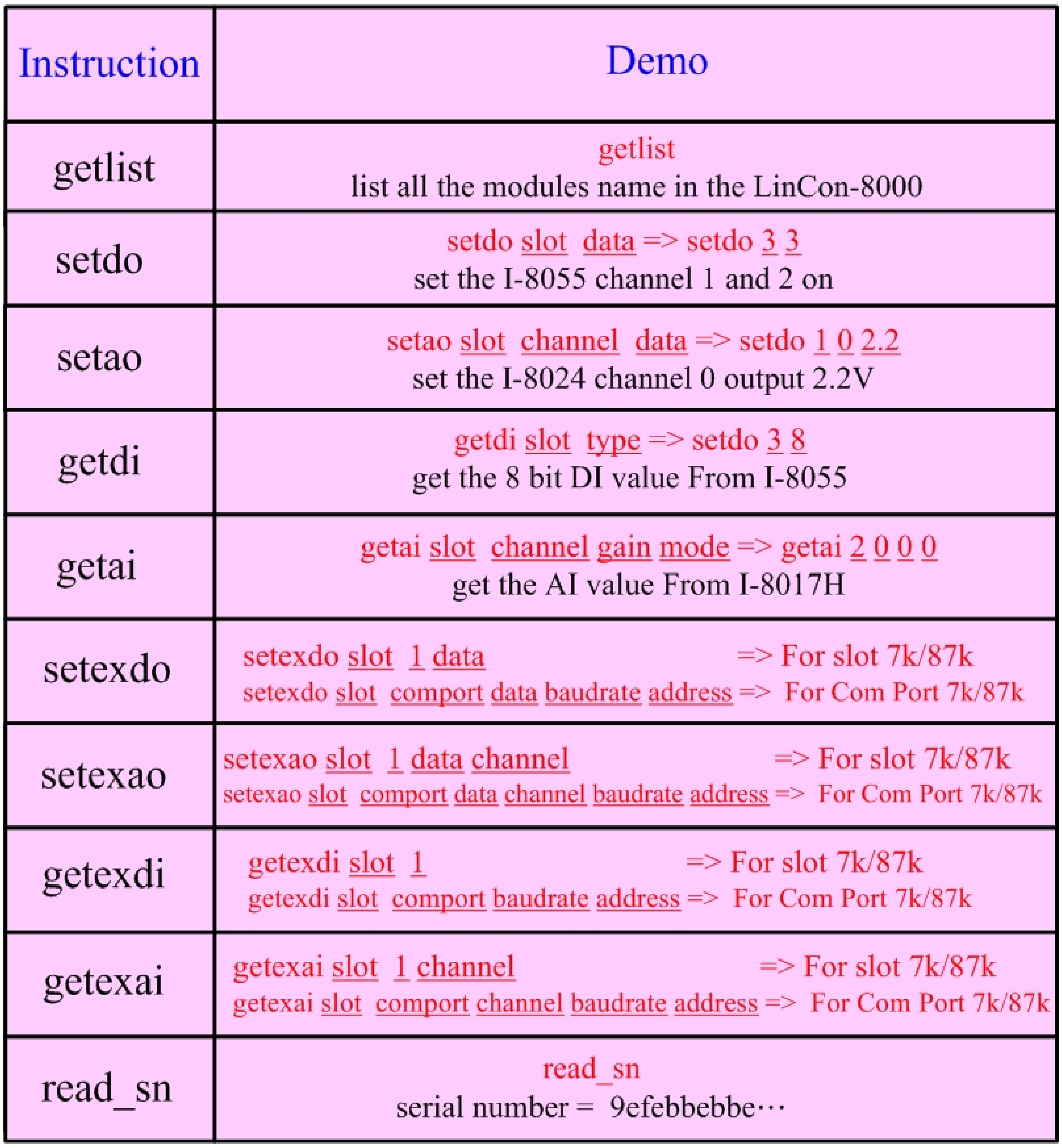
5.4 i-Talk Utility

The **i-Talk utility** provides **sixteen instructions** that make it convenient for users to access the modules and hardware in the LinCon-8000 and they are placed in the path — **/usr/local/bin**. Fig. 5-12 describes the functions of i-Talk utility.

Instruction	Function Description
getlist	List All Modules Name In The LinCon-8000
setdo	Set Digital Output Value To 8k Module
setao	Set Analog Output Value To 8k Module
getdi	Get Digital Input Value From 8k Module
getai	Get Analog Input Value From 8k Module
setexdo	Set Digital Output Value To 7k/87k Module
setexao	Set Analog Output Value To 7k/87k Module
getexdi	Get Digital Input Value From 7k/87k Module
getexai	Get Analog Input Value From 7k/87k Module
setport	Set Port Value By Offset To A Module
getport	Get Port Value By Offset From A Module
setsend	Send String from LinCon COM port
getreceive	Receive String from LinCon COM port
getsendreceive	Send/Receive String from LinCon COM port
read_sn	Get Hardware Serial Number of LinCon-8000
setLinConMAC	Set the MAC Address of LinCon-8000

Fig. 5-12

Fig. 5-13 lists the demo that show how to use the I-talk utility. In the demo, the **I-8024** (AO Module) 、**I-8017H** (AI Module) and **I-8055** (DIO Module) are all used and they are plugged into the slots 1 、2 and 3 of the LinCon seperately.



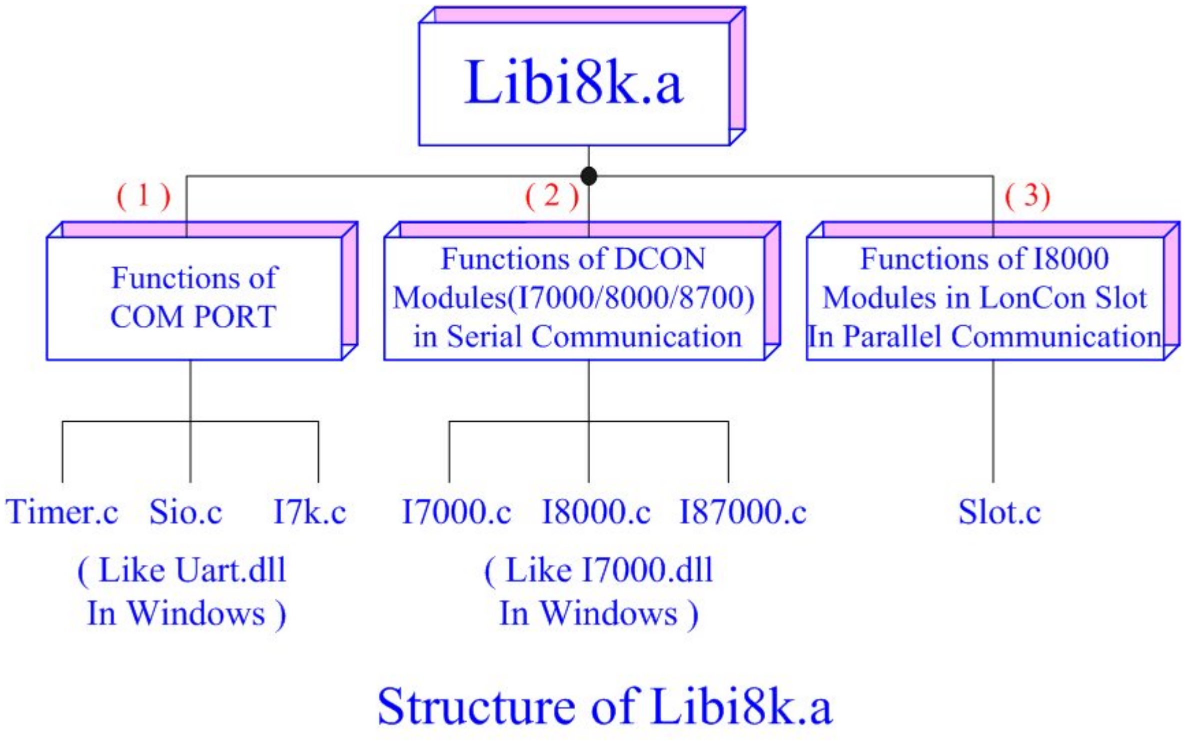
Instruction	Demo
getlist	getlist list all the modules name in the LinCon-8000
setdo	setdo slot data => setdo 3 3 set the I-8055 channel 1 and 2 on
setao	setao slot channel data => setdo 1 0 2.2 set the I-8024 channel 0 output 2.2V
getdi	getdi slot type => setdo 3 8 get the 8 bit DI value From I-8055
getai	getai slot channel gain mode => getai 2 0 0 0 get the AI value From I-8017H
setexdo	setexdo slot 1 data => For slot 7k/87k setexdo slot comport data baudrate address => For Com Port 7k/87k
setexao	setexao slot 1 data channel => For slot 7k/87k setexao slot comport data channel baudrate address => For Com Port 7k/87k
getexdi	getexdi slot 1 => For slot 7k/87k getexdi slot comport baudrate address => For Com Port 7k/87k
getexai	getexai slot 1 channel => For slot 7k/87k getexai slot comport channel baudrate address => For Com Port 7k/87k
read_sn	read_sn serial number = 9efebbebbe…

Fig. 5-13

Users can also type in the instructions name and it will show the instructions usage.

6. LIBI8K.A

In this section, we will focus on examples for the description of and application of the functions found in the Libi8k.a. The Libi8k.a functions can be clarified into 3 groups which are listed in Fig. 6-1



Libi8k.a
(1)
(2)
(3)
Functions of COM PORT
Functions of DCON Modules(I7000/8000/8700) in Serial Communication
Functions of I8000 Modules in LonCon Slot In Parallel Communication
Timer.c
Sio.c
I7k.c
I7000.c
I8000.c
I87000.c
Slot.c
(Like Uart.dll In Windows)
(Like I7000.dll In Windows)
Structure of Libi8k.a

Fig. 6-1

Functions (1) and (2) in the Libi8k.a are the same as with the DCON.DLL Driver (including Uart.dll and I7000.dll) as used in the DCON modules (I-7000 / I-8000 / I-87000 in serial communication). You can refer to the DCON.DLL Driver manual which includes the functions on how to use DCON modules. The DCON.DLL Driver has already been wrapped into the Libi8k.a. Functions (3) of the Libi8k.a consist of the most important functions as they are specially designed for I-8000 modules in the LinCon-8000 slots. They are different from functions (1) and (2) because the communication of I-8000 modules in the LinCon-8000 slots are parallel and not serial. Therefore ICPDAS rewrote I8000.c to Slot.c especially for I-8000 modules in the LinCon-8000 slots. Here we will introduce all the funcitions for slot.c and they can be divided into eight parts for ease of use.

1. System Information Functions;
2. Digital Input/Output Functions;

3. Watch Dog Timer Functions;
4. EEPROM Read/Write Functions;
5. Analog Input Functions;
6. Analog Output Functions;
7. 3-axis Encoder Functions;
8. 2-axis Stepper/Servo Functions;

When using the development tools to develop applications, the **msw.h** file must be included in front of the source program, and when building applications, **Libi8k.a** must be linked. If you want to control ICPDAS I/O remote modules like i7k, i8k and i87k **through COM 2 or COM 3 of the LinCon**, the functions are all the same with DCON DLL. And if you want to control **i8k modules** that are plugged in the slots of the LinCon, then the functions are different and they are described as follows :

6.1 System Information Functions

■ Open_Slot

Description:

This function is used to open and initiate a specifed slot in the LinCon-8000. The 8k or I-87k modules in the LinCon-8000 will use this function. For example, if you want to send or receive data from a specified slot, this function must be called first. Then the other functions can be used later.

Syntax:

```
[ C ]
int Open_Slot(int slot)
```

Parameter:

slot : [Input] Specify the slot number in which the I/O module is plugged into.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
Int slot=1;
Open_Slot(slot);
// The first slot in the LinCon-8000 will be open and initiated.
```

Remark:

■ Close_Slot

Description:

If you have used the function of Open_Slot() to open the specifed slot in the LinCon-8000, you need to use the Close_Slot() function to close the specifed slot in the LinCon-8000. The 8k or I-87k modules in the LinCon-8000 will use this function. For example, once you have finished sending or receiving data from a specified slot, this function would then need to be called.

Syntax:

```
[ C ]
void Close_Slot(int slot)
```

Parameter:

slot : [Input] Specify the slot number in which the I/O module is plugged into.

Return Value:

None

Example:

```
Int slot=1;
Close_Slot(slot);
// The first slot in the LinCon-8000 will be closed.
```

Remark:

■ Open_SlotAll

Description:

This function is used to open and initiate **all slots** in the LinCon-8000. For example, if you want to send or receive data from multiple slots, you can call this function to simplify your program. Then you can use the other functions later.

Syntax:

```
                         [ C ]
int Open_Slot(void)
```

Parameter:

None

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
Open_SlotAll();
// All slots in the LinCon-8000 will be open and initiated.
```

Remark:

■ Close_SlotAll

Description:

If you have used the function Open_SlotAll() to open all the slots in the LinCon-8000, you can use the Close_SlotAll() function to close all the slots in the LinCon-8000. For example, once you are finish sending or receiving data from many slots, this function can be called to close all the slots rapidly.

Syntax:

```
[ C ]
void Close_SlotAll(void)
```

Parameter:

None

Return Value:

None

Example:

```
Close_Slot();
// All slots in the LinCon-8000 will be closed.
```

Remark:

■ ChangeToSlot

Description:

This function is used to dedicate serial control to the specified slots for the control of the I-87k series. The serial bus in the LinCon-8000 backplane is for mapping through to COM1. For example, if you want to send or receive data from a specified slot, you need to call this function first. Then you can use the other series functions.

Syntax:

```
                         [ C ]
void ChangeToSlot(char slot)
```

Parameter:

slot :	[Input] Specify the slot number in which the I/O module is plugged into.

Return Value:

None

Example:

```
char slot=1;
ChangeToSlot (slot);
// The first slot is specified as COM1 port in LinCon-8000.
```

Remark:

■ Open_Com

Description:

This function is used to configure and open the COM port. It must be **called once before** sending/receiving command through COM port. For example, if you want to send or receive data from a specified COM port, you need to call this function first. Then you can use the other series functions.

Syntax:

```
[ C ]
WORD Open_Com(char port, DWORD baudrate, char cData, char cParity, char cStop)
```

Parameter:

port :	[Input] COM1, COM2, COM3..., COM255.
baudrate:	[Input] 1200/2400/4800/9600/19200/38400/57600/115200
cDate :	[Input] Data5Bit, Data6Bit, Dat7Bit, Data8Bit
cParity :	[Input] NonParity, OddParity, EvenParity
cStop :	[Input] OneStopBit, TwoStopBit

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
```

Remark:

■ Close_Com

Description:

This function is used to closes and releases the resources of the COM port computer rescourse. And it must be **called before exiting the application program**. The Open_Com will return error message if the program exit without calling Close_Com function.

Syntax:

```
[ C ]
BOOL Close_Com(char port)
```

Parameter:

port :	[Input] COM1,COM2, COM3...COM255.

Return Value:

None

Example:

```
Close_Com (COM3);
```

Remark:

■ Send_Receive_Cmd

Description:

This function is used to sends a command string to RS-485 network and receives the response from RS-485 network. If the wChkSum=1, this function automatically adds the two checksum bytes into the command string and also check the checksum status when receiving response from the modules. Note that the end of sending string is added [0x0D] to mean the termination of every command.

Syntax:

```
                         [ C ]
WORD Send_Receive_Cmd (char port, char szCmd[ ], char szResult[ ],
                       WORD wTimeOut, WORD wChksum, WORD *wT)
```

Parameter:

port :	[Input] 1=COM1, 2=COM2, 3=COM3..., 255=COM255.
szCmd:	[Input] Sending command string
szResult :	[Input] Receiving the response string from the modules
wTimeOut :	[Input] Communicating timeout setting, the unit=1ms
wChkSum :	[Input] 0=Disable, 1=Enable
*wT:	[Output] Total time of send/receive interval, unit=1 ms

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char m_port =1;
DWORD m_baudrate=115200;
WORD m_timeout=100;
WORD m_chksum=0;
WORD m_wT;
char m_szSend[40], m_szReceive[40];
int RetVal;
m_szSend[0] = '$';
m_szSend[1] = '0';
m_szSend[2] = '0';
```

```
m_szSend[3] = 'M';
m_szSend[4] = 0;
/* open device file */
Open_Slot(1);
RetValue = Open_Com(m_port, m_baudrate, Data8Bit, NonParity, OneStopBit);
if (RetValue >0) {
        printf("Open COM%d failed!\n", m_port);
        return FAILURE;
}
RetValue = Send_Receive_Cmd(m_port, m_szSend, m_szReceive, m_timeout,
                              m_chksum, &m_wT);
if (RetValue) {
        printf("Module at COM%d Address %d error !!!\n", m_port, m_szSend[2] );
        return FAILURE;
}
Close_Com (m_port);
```

Remark:

■ Send_Cmd

Description:

This function only sends a command string to DCON series modules. If the wChkSum=1, it automatically **adds the two checksum bytes to the command string.** And then the end of sending string is further added [0x0D] to mean the termination of the command (szCmd). And this command string cannot include space char within the command string. For example: “$01M 02 03” is user’s command string. However, the actual command send out is “$01M”.

Syntax:

```
[ C ]
WORD Send_Cmd (char port, char szCmd[ ], WORD wTimeOut, WORD wChksum)
```

Parameter:

port :	[Input] 1=COM1, 2=COM2, 3=COM3..., 255=COM255.
szCmd :	[Input] Sending command string
wTimeOut :	[Input] Communicating timeout setting, the unit=1ms
wChkSum :	[Input] 0=Disable, 1=Enable

Return Value:

None

Example:

```
char m_port=1;
char m_szSend[40] ;
DWORD m_baudrate=115200;
WORD m_timeout=100, m_chksum=0;
m_szSend[0] = '$';
m_szSend[1] = '0';
m_szSend[2] = '0';
m_szSend[3] = 'M';
Open_Slot(2);      // The module is plug in slot 2 and address is 0.
Open_Com(m_port, m_baudrate, Data8Bit, NonParity, OneStopBit);
Send _Cmd(m_port, m_szSend, m_timeout, m_chksum, &m_wT);
Close_Com (m_port);
```

Remark:

■ Receive_Cmd

Description:

This function is used to obtain the responses string from the modules in RS-485 network. And this function provides a response string without the last byte [0x0D].

Syntax:

```
                              [ C ]
WORD Receive_Cmd (char port, char szResult[ ], WORD wTimeOut,
                              WORD wChksum)
```

Parameter:

port :	[Input] 1=COM1, 2=COM2, 3=COM3..., 255=COM255.
szResult :	[Output] Sending command string
wTimeOut :	[Input] Communicating timeout setting, the unit=1ms
wChkSum :	[Input] 0=Disable, 1=Enable

Return Value:

None

Example:

```
char m_port=3;
char m_Send[40], m_szResult[40] ;
DWORD m_baudrate=115200;
WORD m_timeout=100, m_chksum=0;
m_szSend[0] = '$';
m_szSend[1] = '0';
m_szSend[2] = '1';
m_szSend[3] = 'M';
m_szSend[4] = 0;
Open_Com (m_port, m_baudrate, Data8Bit, NonParity, OneStopBit);
Send _Cmd (m_port, m_szSend, m_timeout, m_chksum);
Receive_Cmd (m_port, m_szResult, m_timeout, m_chksum);
Close_Com (m_port);
// Read the remote module:I-7016D , m_ szResult : “!017016D”
```

Remark:

■ Send_Binary

Description:

Send out the command string by fix length, which is controlled by the parameter “iLen”. The difference between this function and Send_cmd is that Send_Binary terminates the sending process by the string length “iLen” instead of the character "CR"(Carry return). Therefore, this function can send out command string with or without null character under the consideration of the command length. Besides, because of this function without any error checking mechanism (Checksum, CRC, LRC... etc.), users have to add the error checking information to the raw data by themselves if communication checking system is required. <u>Note that this function is usually applied to communicate with the other device, but not for ICPDAS DCON (I-7000/8000/87K) series modules.</u>

Syntax:

```
                    [ C ]
WORD Send_Binary (char port, char szCmd[ ], int iLen)
```

Parameter:

port :	[Input] 1=COM1, 2=COM2, 3=COM3..., 255=COM255.
szCmd :	[Input] Sending command string
iLen :	[Input] The length of command string.

Return Value:

None

Example:

```
int m_length=4;
char m_port=3, char m_szSend[40];
DWORD m_baudrate=115200;
m_szSend[0] = '0';
m_szSend[1] = '1';
m_szSend[2] = '2';
m_szSend[3] = '3';
Open_Com(m_port, m_baudrate, Data8Bit, NonParity, OneStopBit);
Send _Binary(m_port, m_szSend, m_length);
Close_Com (m_port);
```

Remark:

■ Receive_Binary

Description:

This function is applied to receive the fix length response. The length of the receiving response is controlled by the parameter "iLen". The difference between this function and Receive_cmd is that Receive_Binary terminates the receiving process by the string length "iLen" instead of the character "CR"(Carry return). Therefore, this function can be used to receive the response string data with or without null character under the consideration of receiving length. Besides, because of this function without any error checking mechanism (checksum, CRC, LRC... etc.), users have to remove from the error checking information from the raw data by themselves if communication checking system is used. <u>Note that this function is usually applied to communicate with the other device, but not for ICPDAS DCON (I-7000/8000/87K) series modules.</u>

Syntax:

```
[ C ]
WORD Receive_Binary (char cPort, char szResult[], WORD wTimeOut,
                     WORD wLen, WORD *wT)
```

Parameter:

port :	[Input] 1=COM1, 2=COM2, 3=COM3..., 255=COM255.
szResult :	[Input] Receiving the response string from the modules
wTimeOut :	[Input] Communicating timeout setting, the unit=1ms
wLen :	[Input] The length of command string.
*wT:	[Output] Total time of send/receive interval, unit=1 ms

Return Value:

None

Example:

```
int m_length=10;
char m_port=3;
char m_szSend[40];
char m_szReceive[40];
DWORD m_baudrate=115200;
WORD m_wt;
WORD m_timeout=10;
WORD m_wlength=10;
m_szSend[0] = '0';
m_szSend[1] = '1';
m_szSend[2] = '2';
m_szSend[3] = '3';
m_szSend[4] = '4';
m_szSend[5] = '5';
m_szSend[6] = '6';
m_szSend[7] = '7';
m_szSend[8] = '8';
m_szSend[9] = '9';
Open_Com(m_port, m_baudrate, Data8Bit, NonParity, OneStopBit);
Send _Binary(m_port, m_szSend, m_length);                    // send 10 character
Receive_Binary(char m_port, char m_szResult[], WORD m_timeout,
                WORD m_wlength, WORD &m_wt)                  // receive 10 character
Close_Com (m_port);
```

Remark:

■ sio_open

Description:

This function is used to open and initiate a specifed serial port in the LinCon-8000. The n-port modules in the LinCon-8000 will use this function. For example, if you want to send or receive data from a specified serial port, this function must be called first. Then the other functions can be used later.

Syntax:

```
[ C ]
int sio_open(int port)
```

Parameter:

port :	[Input] device name: /dev/ttyS2, /dev/ttyS3…/dev/ttyS29

Return Value:

This function returns int port descriptor for the port opened successfully.

ERR_PORT_OPEN is for Failure

Example:

```
#define   COM_M1   "/dev/ttyS2"          // Defined the first port of i-8144 in slot 1
char fd[5];
fd[0]=sio_open(COM_M1, B9600, DATA_BITS_8, NO_PARITY,ONE_STOP_BIT);
if (fd[0] == ERR_PORT_OPEN) {
      printf("open port_m failed!\n");
      return (-1);
}
// The i8114 is plug in slot 1 and the first port will be open and initiated.
```

Remark:

This function can be applied on modules: I-8114, I-8112, I-8142 and I-8144.

■ sio_close

Description:

If you have used the function of sio_slot() to open the specifed serial port in the LinCon-8000, you need to use the sio_close() function to close the specifed serial port in the LinCon-8000. The n-port modules in the LinCon-8000 will use this function. For example, once you have finished sending or receiving data from a specified serial port, this function would then need to be called.

Syntax:

```
                    [ C ]
int sio_close(int port)
```

Parameter:

port :	[Input] device name: /dev/ttyS2, /dev/ttyS3…/dev/ttyS29

Return Value:

None

Example:

```
#define   COM_M2   "/dev/ttyS3"     // Defined the second port of i-8144 in slot 1
char fd[5];
fd[0]=sio_open(COM_M2, B9600, DATA_BITS_8, NO_PARITY,ONE_STOP_BIT);
sio_close (fd[0]);
// The second port of i8144 in slot 1 will be closed.
```

Remark:

This function can be applied on modules: I-8114, I-8112, I-8142 and I-8144.

■ GetModuleType

Description:

This function is used to retrieve which type of 8000 series I/O module is plugged into a specific I/O slot in the LinCon-8000. This function performs a supporting task in the collection of information related to the system's hardware configurations.

Syntax:

```
[ C ]
int GetModuleType(char slot)
```

Parameter:

slot : [Input] Specify the slot number in which the I/O module is plugged into.

Return Value:

Module Type: it is defined in the **IdTable[]** of slot.c.

Type	Value
_PARALLEL	0x80
_AI	0xA0
_AO	0xA1
_DI8	0xB0
_DI16	0xB1
_DI32	0xB2
_DO6	0xC0
_DO8	0xC1
_DO16	0xC2
_DO32	0xC3
_DI4DO4	0xD0
_DI8DO8	0xD1
_DI16DO16	0xD2
_MOTION	0xE2

Example:

```
int slot=1;
int moduleType;
Open_Slot(slot);
printf("GetModuleType= 0x%X \n", GetModuleType(slot));
Close_Slot(slot);
// The I-8057 card is plugged in slot 1 and has a return Value : 0xC2
```

Remark:

■ GetNameOfModule

Description:

This function is used to retrieve the name of an 8000 series I/O module, which is plugged into a specific I/O slot in the LinCon-8000. This function supports the collection of system hardware configurations.

Syntax:

```
[ C ]
int GetNameOfModule(char slot)
```

Parameter:

slot:	[Input] Specify the slot number where the I/O module is plugged into.

Return Value:

I/O module ID. For Example, the I-8017 will return 8017.

Example:

```
char slot=1;
int moduleName;
Open_Slot(slot);
moduleID=GetNameOfModule(slot);
Close_Slot(Slot);
// The I-8017 card plugged in slot 1 of LinCon-8000
// Returned Value: moduleName=” 8017 ”
```

Remark:

■ Read_SN

Description:

This function is used to retrieves the hardware serial identification number on the LinCon-8000 main controller. This function supports the control of hardware versions by reading the serial ID chip

Syntax:

```
[ C ]
void Read_SN(unsigned char serial_num[])
```

Parameter:

serial_num : [Output] Receive the serial ID number.

Return Value:

None

Example:

```
int slot ;
unsigned char serial_num[8];
Open_Slot(0);
Read_SN(serial_num);
printf("SN=%x%x%x%x%x%x%x%x\n",serial_num[0],serial_num[1], serial_ num[2]
        ,serial_num[3],serial_num[4],serial_num[5],serial_num[6],serial_num[7]);
```

Remark:

6.2 Digital Input/Output Functions

6.2.1 For I-8000 modules via parallel port

■ DO_8

Description:

This function is used to output 8-bit data to a digital output module. The 0~7 bits of output data are mapped into the 0~7 channels of digital module output respectively.

Syntax:

```
[ C ]
void DO_8(int slot, unsigned char data)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.
data :	[Input] output data.

Return Value:

None

Example:

```
int slot=1;
unsigned char data=3;
DO_8(slot, data);
// The I-8064 card is plugged in slot 1 of LinCon-8000 and can turn on channel 0
// and 1.
```

Remark:

This function can be applied on modules: I-8060, I-8064, I-8065, I-8066 I-8068 and I-8069.

■ DO_16

Description:

This function is used to output 16-bit data to a digital output module. The 0~15 bits of output data are mapped into the 0~15 channels of digital output modules respectively.

Syntax:

```
[ C ]
void DO_16(int slot, unsigned int data)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.
data :	[Input] output data.

Return Value:

None

Example:

```
int slot=1;
unsigned int data=3;
DO_16(slot, data);
// The I-8057 card is plugged in slot 1 of LinCon-8000 and can turn on channel 0     //
and 1.
```

Remark:

This function can be applied on modules: I-8037, I-8056, I-8057.

■ DO_32

Description:

Output the 32-bit data to a digital output module. The 0~31 bits of output data are mapped into the 0~31 channels of digital output modules respectively.

Syntax:

```
[ C ]
void DO_32(int slot, unsigned int data)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.
data :	[Input] output data.

Return Value:

None

Example:

```
int slot=1;
unsigned int data=3;
DO_32(slot, data);
// The I-8041 card is plugged in slot 1 of LinCon-8000 and can turn on channel 0
// and 1.
```

Remark:

This function can be applied on module: I-8041.

■ DI_8

Description:

Obtains 8-bit input data from a digital input module. The 0~7 bits of input data correspond to the 0~7 channels of digital input modules respectively.

Syntax:

[C]

```
unsigned char DI_8(int slot)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.

Return Value:

Input data

Example:

```
int slot=1;
unsigned char data;
data=DI_8(slot);
// The I-8058 card is plugged in slot 1 of LinCon-8000 and has inputs in
// channel 0 and 1.
// Returned value: data=0xfC
```

Remark:

This function can be applied on modules: I-8048, I-8052, I-8058.

■ DI_16

Description:

This function is used to obtain 16-bit input data from a digital input module. The 0 ~15 bits of input data correspond to the 0~15 channels of digital module's input respectively.

Syntax:

```
[ C ]
unsigned int DI_16(int slot)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.

Return Value:

Input data

Example:

```
int slot=1;
unsigned int data;
data=DI_16(slot);
// The I-8053 card is plugged in slot 1 of LinCon-8000 and has inputs in
// channel 0 and 1.
// Returned value: data=0xfffC
```

Remark:

This function can be applied on modules: I-8051, I-8053.

■ DI_32

Description:

This function is used to obtain 32-bit input data from a digital input module. The 0~31 bits of input data correspond to the 0~31 channels of digital input module respectively.

Syntax:

```
[ C ]
unsigned long DI_32(int slot)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.

Return Value:

Input data

Example:

```
int slot=1;
unsigned long data;
data=DI_32(slot);
// The I-8040 card plugged is in slot 1 of LinCon-8000 and has inputs in
// channels 0 and 1.
// Returned value: data=0xfffffffC
```

Remark:

This function can be applied on module: I-8040.

■ DIO_DO_8

Description:

This function is used to output 8-bit data to DIO modules. These modules run 8 digital input channels and 8 digital output channels simultaneously. The 0~7 bits of output data are mapped onto the 0~7 output channels for their specific DIO modules respectively.

Syntax:

```
[ C ]
void DIO_DO_8(int slot, unsigned char data)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.
data :	[Input] output data.

Return Value:

None

Example:

```
int slot=1;
unsigned char data=3;
DIO_DO_8(slot, data);
// The I-8054 card is plugged in slot 1 of LinCon-8000 and can turn on channels 0
// and 1.
// It not only outputs a value, but also shows 16LEDs.
```

Remark:

This function can be applied in modules: I-8054, I-8055, and I-8063.

■ DIO_DO_16

Description:

This function is used to output 16-bits of data to DIO modules, which have 16 digital input and 16 digital output channels running simultaneously. The 0~15 bits of output data are mapped onto the 0~15 output channels for their specific DIO modules respectively.

Syntax:

```
[ C ]
void DIO_DO_16(int slot, unsigned int data)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.
data :	[Input] output data.

Return Value:

None

Example:

```
int slot=1;
unsigned int data=3;
DIO_DO_16(slot, data);
// The I-8042 card is plugged in slot 1 of LinCon-8000 and can turn on the
// channels 0 and 1.
// It not only outputs a value, but also shows 32LEDs.
```

Remark:

This function can be applied on modules: I-8042, I-8050

■ DIO_DI_8

Description:

This function is used to obtain 8-bit data from DIO modules. These modules run 8 digital input and 8 digital output channels simultaneously. The 0~7 bits of intput data, are mapped onto the 0~7 iutput channels for their specific DIO modules respectively.

Syntax:

```
[ C ]
Unsigned char DIO_DI_8(int slot)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.

Return Value:

Input data

Example:

```
int slot=1;
unsigned char data;
data=DIO_DI_8(slot);
// The I-8054 card is plugged in slot 1 of LinCon-8000 and has inputs in
// channel 0 and 1.
// Returned value: data=0xfC
```

Remark:

This function can be applied in modules: I-8054, I-8055, and I-8063.

■ DIO_DI_16

Description:

This function is used to obtain 16-bit data from DIO modules. These modules run 16 digital input and 16 digital output channels simultaneously. The 0~15 bits of iutput data are mapped onto the 0~15 iutput channels for their specific DIO modules respectively.

Syntax:

```
[ C ]
Unsigned char DIO_DI_16(int slot)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.

Return Value:

Input data

Example:

```
int slot=1;
unsigned char data;
data=DIO_DI_16(slot);
// The I-8042 card is plugged in slot 1 of LinCon-8000 and has inputs in
// channel 0 and 1.
// Returned value: data=0xfffC
```

Remark:

This function can be applied in modules: I-8042.

■ DO_8_RB、DO_16_RB、DO_32_RB
DIO_DO_8_RB、DIO_DO_16_RB

Description:

This function is used to **Readback** all channel status from a Digital Output module.

Syntax:

```
[ C ]
unsigned char DO_8_RB(int slot)
unsigned int DO_16_RB(int slot)
unsigned long DO_32_RB(int slot)
unsigned char DIO_DO_8_RB(int slot)
unsigned int DIO_DO_16_RB(int slot)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.

Return Value:

all DO channel status

Example:

```
int slot=1;
Open_Slot(slot);
printf("%u",DO_32_RB(slot));
Close_Slot(slot);
// The I-8041 module is plugged in slot 1 of LinCon-8000 and return all DO channel
   status.
```

Remark:

These functions can be applied on modules:

DO 8 channel：I-8060, I-8064, I-8065, I-8066, I-8068, and I-8069.

DO 16 channel：I-8037, I-8056, and I-8057

DO 32 channel：I-8041

■ DO_8_BW、DO_16_ BW、DO_32_ BW
DIO_DO_8_ BW、DIO_DO_16_ BW

Description:

This function is used to output **assigned single channel** status (ON / OFF) of a Digital Output module.

Syntax:

```
[ C ]
void DO_8_BW(int slot, int bit, int data)
void DO_16_BW (int slot, int bit, int data)
void DO_32_BW (int slot, int bit, int data)
void DIO_DO_8_BW (int slot, int bit, int data)
void DIO_DO_16_BW (int slot, int bit, int data)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.
bit :	[Input] channel of module.
data :	[Input] channel status [on(1) / off(0)].

Return Value:

None

Example:

```
int slot=1, bit=0, data=1;
Open_Slot(slot);
DO_32_BW(slot, bit, data);
Close_Slot(slot);
// The I-8041 module is plugged in slot 1 of LinCon-8000 and just turn on channel 0 of
   I-8041.
```

Remark:

These functions can be applied on modules:

DO 8 channel：I-8060, I-8064, I-8065, I-8066 I-8068, and I-8069.

DO 16 channel：I-8037, I-8056, I-8057

DO 32 channel：I-8041

■ DI_8_BW、DI_16_ BW、DI_32_ BW

Description:

This function is used to **Readback assigned single channel** status (ON / OFF) from a Digital Input module.

Syntax:

```
[ C ]
int DI_8_BW(int slot, int bit)
int DI_16_BW (int slot, int bit)
int DI_32_BW (int slot, int bit)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.
bit :	[Input] channel of module.

Return Value:

None

Example:

```
int slot=1, bit=0;
Open_Slot(slot);
printf("DI channel %d = %d\n", bit, DI_32_BW(slot, bit));
Close_Slot(slot);
// The I-8040 module is plugged in slot 1 of LinCon-8000 and return channel 0
// status. ( 0：ON；1：OFF ).
```

Remark:

These functions can be applied on modules:

DI 8 channel：I-8048, I-8052, and I-8058.

DI 16 channel：I-8051, I-8053

DI 32 channel：I-8040

■ UDIO_WriteConfig_16

Description:

This function is used to configure the channel of the universal DIO module which is digital input or digital output mode. The universal DIO module can be up to 16 digital input or digital output channels running simultaneously.

Syntax:

```
[ C ]
unsigned short UDIO_WriteConfig_16(int slot, unsigned short config)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.
config :	[Input] channel status.[DO : 1 / DI : 0]

Return Value:

None

Example:

```
int slot=1;
unsigned short config=0xffff;
UDIO_WriteConfig_16(slot, config);
// The I-8050 card is plugged in slot 1 of LinCon-8000.
// WriteConfig: 0xffff (ch 0~ch15 is DO mode)
```

Remark:

This function can be applied on modules: I-8050.

■ UDIO_ReadConfig_16

Description:

This function is used to read the channels configuration of the universal DIO module which is digital input or digital output mode.

Syntax:

```
[ C ]
unsigned short UDIO_ReadConfig_16(int slot)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.

Return Value:

None

Example:

```
int slot=1;
unsigned int ret;
unsigned short config=0x0000;
UDIO_WriteConfig_16(slot, config);
ret=UDIO_ReadConfig_16(slot);
printf("Read the I/O Type is : 0x%04lx \n\r",ret);
// The I-8050 card is plugged in slot 1 of LinCon-8000.
// WriteConfig: 0x0000 (ch 0~ch15 is DI mode)
// Read the I/O Type is: 0x0000
```

Remark:

This function can be applied on modules: I-8050.

■ UDIO_DO16

Description:

This function is used to output 0~15 bits data to a universal DIO module according to the channel configuration. The 0~15 bits of output data are mapped onto the 0~15 output channels for their specific universal DIO modules respectively.

Syntax:

```
[ C ]
void UDIO_DO16(int slot, unsigned short config)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.
config :	[Input] output data.

Return Value:

None

Example:

```
int slot=1;
unsigned int data;
unsigned short config =0x00ff;
UDIO_WriteConfig_16(slot, config);
scanf("%d:",&data);
UDIO_DO16(slot, data);
printf("DO(Ch0~Ch7) of I-8050 in Slot %d = 0x%x\n\r",slot, data);
// The I-8050 card is plugged in slot 1 of LinCon-8000
// WriteConfig: 0x00ff (ch 0~ch7 is DO mode and ch8~ch15 is DI mode)
// Input DO value: 255
// DO(Ch0~Ch7) of I-8050 in Slot 1 = 0xff
```

Remark:

This function can be applied on modules: I-8050.

■ UDIO_DI16

Description:

This function is used to input 0~15 bits data to a universal DIO module according to the channel configuration. The 0~15 bits of input data are mapped onto the 0~15 input channels for their specific universal DIO modules respectively.

Syntax:

```
[ C ]
unsigned short UDIO_DI16(int slot)
```

Parameter:

slot :	[Input] the slot number where the I/O module is plugged into.

Return Value:

None

Example:

```
int slot=1;
unsigned int data;
unsigned short config =0xff00;
UDIO_WriteConfig_16(slot, config);
data=UDIO_DI16(slot);
printf("DI(Ch0~Ch7) of I-8055 in Slot %d = 0x%x\n\r",slot, data);
scanf("%d:",&data);
UDIO_DO16(slot, data);
printf("DO(Ch8~Ch15) of I-8050 in Slot %d = 0x%x\n\r",slot, data);
// The I-8050 card is plugged in slot 1 of LinCon-8000.
// WriteConfig: 0x00ff (ch 0~ch7 is DI mode and ch8~ch15 is DO mode)
// DI(Ch0~Ch7) of I-8055 in Slot 1 = 0xfbff
// Input DO value: 255
// DO(Ch8~Ch15) of I-8050 in Slot 1 = 0xff
```

Remark:

This function can be applied on modules: I-8050.

6.2.2 For I-7000/I-8000/I-87000 modules via serial port

6.2.2.1 I-7000 series modules

■ DigitalOut

Description:

This function is used to output the value of the digital output module for I-7000 series modules.

Syntax:

```
[ C ]
WORD DigitalOut(WORD wBuf[], float fBuf[], char szSend[], char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7011/12/14/42/43/44/50/60/63/65/66/67/80
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Input] 16-bit digital output data
wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7050;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[5] = 0x0f;                    // 8 DO Channels On
wBuf[6] = 0;
DigitalOut(wBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalBitOut

Description:

This function is used to set digital output value of the channel No. of I-7000 series modules. The output value is “0” or “1”.

Syntax:

```
[ C ]
WORD DigitalBitOut(WORD wBuf[ ], float fBuf[ ], char szSend[ ], char szReceive[ ])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7042/43/44/50/60/63/65/66/67
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	Not used
wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7] :	[Input] The digital output channel No.
wBuf[8] :	[Input] Logic value(0 or 1)
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Ouptut] Result string receiving from I-7000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7065;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[6] = 0;
wBuf[7] = 0x08;                    //RL4 relay On
wBuf[8] = 1;
DigitalBitOut (wBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalOutReadBack

Description:

This function is used to read back the digital output value of I-7000 series modules.

Syntax:

```
[ C ]
WORD DigitalOutReadBack(WORD wBuf[ ], float fBuf[ ],char szSend[ ],
                        char szReceive[ ])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7042/43/44/50/60/63/65/66/67/80
wBuf[3] :	[Input] 0=Checksum disable; 1=Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Output] 16-bit digital output data read back
wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD DO;
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7050;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[6] = 0;
DigitalOutReadBack (wBuf, fBuf, szSend, szReceive);
DO=wBuf[5];
Close_Com(COM3);
```

Remark:

■ DigitalOut_7016

Description:

This function is used to set the digital output value of the specified channel No. of I-7016 module. If the parameter of wBuf[7] is "0", it means to output the digital value through Bit0 and Bit1 digital output channels. If wBuf[7] is "1", it means to output the digital value through Bit2 and Bit3 digital output channels.

Syntax:

[C]

```
WORD DigitalOut_7016(WORD wBuf[], float fBuf[], char szSend[],
                     char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7016
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable

wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Input] 2-bit digital output data in decimal format
wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7] :	[Input] 0 : Bit0, Bit1 output 1 : Bit2, Bit3 output
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7016;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[5] = 1;
wBuf[6] = 0;
wBuf[7] = 1;      // Set the Bit2, Bit3 digital output
DigitalOut_7016(wBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalIn

Description:

This function is used to obtain the digital input value from I-7000 series modules.

Syntax:

```
[ C ]
WORD DigitalIn(WORD wBuf[], float fBuf[], char szSend[], char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7041/44/50/52/53/55/58/60/63/65
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Output] 16-bit digital output data
wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD DI;
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
```

```
    Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
    wBuf[0] = m_port;
    wBuf[1] = m_address;
    wBuf[2] = 0x7050;
    wBuf[3] = m_checksum;
    wBuf[4] = m_timeout;
    wBuf[6] = 0;
    DigitalIn(wBuf, fBuf, szSend, szReceive);
    DI=wBuf[5];
    Close_Com(COM3);
```

Remark:

■ DigitalInLatch

Description:

This function is used to obtain the latch value of the high or low latch mode of digital input module.

Syntax:

```
[ C ]
WORD DigitalInLatch(WORD wBuf[], float fBuf[], char szSend[], char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7041/44/50/52/53/55/58/60/63/65/66
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Input] 0: low Latch mode ; 1:high Latch mode
wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7] :	[Output] Latch value
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port ;
wBuf[1] = m_address ;
wBuf[2] = 0x7050;
wBuf[3] = m_checksum ;
wBuf[4] = m_timeout ;
wBuf[5] = 1;          // Set the high Latch mode
wBuf[6] = 0;
wBuf[7] = 0x03;     // Set the Latch value
DigitalInLatch(wBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ ClearDigitalInLatch

Description:

This function is used to clear the latch status of digital input module when latch function has been enable.

Syntax:

```
                         [ C ]
WORD ClearDigitalInLatch(WORD wBuf[], float fBuf[],char szSend[],
                         char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7011/12/14/42/43/44/50/55/58/60/63/65/66/67
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	Not used.
wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7050;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[6] = 0;
ClearDigitalInLatch(wBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalInCounterRead

Description:

This function is used to obtain the counter event value of the channel number of digital input module.

Syntax:

```
[ C ]
WORD DigitalInCounterRead(WORD wBuf[], float fBuf[], char szSend[],
                          char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7041/44/50/51/52/53/55/58/60/63/65
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Input] The digital input Channel No.
wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7] :	[Output] Counter value of the digital input channel No.
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD DI_counter;
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7050;
wBuf[3] = m_checksum;
wBuf[4] = 100;
wBuf[5] = 0;            // Set the digital input channel No.
wBuf[6] = 0;
DigitalInCounterRead(wBuf, fBuf, szSend, szReceive);
DI_counter=wBuf[7];
Close_Com(COM3);
```

Remark:

■ ClearDigitalInCounter

Description:

This function is used to clear the counter value of the channel number of digital input module.

Syntax:

```
                        [ C ]
WORD ClearDigitalInCounter(WORD wBuf[], float fBuf[],char szSend[],
                                 char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7041/44/50/51/52/53/55/58/60/63/65
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Input] The digital input channel No.
wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7050;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[5] = 0;            // Set the digital input channel No.
wBuf[6] = 0;
ClearDigitalInCounter(wBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ ReadEventCounter

Description:

This function is used to obtain the value of event counter of I-7000 series modules. This function only supports I-7011, I-7012, I-7014 and I-7016 modules.

Syntax:

```
                              [ C ]
WORD ReadEventCounter(WORD wBuf[], float fBuf[],char szSend[],
                              char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7011/12/14/16
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	Not used
wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7] :	[Output] The value of event counter
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD Counter;
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7012;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[6] = 0;
ReadEventCounter (wBuf, fBuf, szSend, szReceive);
Counter=wBuf[7];
Close_Com(COM3);
```

Remark:

■ ClearEventCounter

Description:

This function is used to clear the value of event counter of I-7000 series modules. This function only supports I-7011, I-7012, I-7014 and I-7016 modules.

Syntax:

```
[ C ]
WORD ClearEventCounter(WORD wBuf[], float fBuf[], char szSend[],
                       char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7011/12/14/16
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	Not used
wBuf[6] :	[Input] 0 → no save to szSend &szReceive
	1 → Save to szSend &szReceive
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7012;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[6] = 0;
ClearEventCounter (wBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

6.2.2.2 I-8000 series modules

■ DigitalOut_8K

Description:

This function is used to set the digital output value of digital output module for I-8000 series modules.

Syntax:

```
[ C ]
WORD DigitalOut_8K(DWORD dwBuf[], float fBuf[],char szSend[],char szReceive[])
```

Parameter:

dwBuf:	WORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8041/42/54/55/56/57/60/63/64/65/66/67/68/77
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] 16-bit digital output data
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-8000 main unit.
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8041;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 10;                    // digital output
dwBuf[6] = 0;
dwBuf[7] = m_slot;
DigitalOut_8K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalBitOut_8K

Description:

This function is used to set the digital value of the digital output channel No. of I-8000 series modules. Theoutput value is “0” or “1”.

Syntax:

```
                    [ C ]
WORD DigitalBitOut_8K(DWORD dwBuf[], float fBuf[],char szSend[],
                      char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8041/42/54/55/56/57/60/63/64/65/66/67/68/77
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] 16-bit digital output data
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-8000 main unit.
dwBuf[8] :	[Input] The output channel No.
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8041;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 10;                  // digital output
dwBuf[6] = 0;
dwBuf[7] = m_slot;
dwBuf[8] = 3;
DigitalBitOut_8K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalIn_8K

Description:

This function is used to obtain the digital input value from I-8000 series modules.

Syntax:

```
[ C ]
WORD DigitalIn_8K(DWORD dwBuf[], float fBuf[], char szSend[], char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8040/42/51/52/54/55/58/63/77
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Output] 16-bit digital output data
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-8000 main unit.
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD DI;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8040;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 10;                    // digital output
dwBuf[6] = 0;
dwBuf[7] = m_slot;
DigitalIn_8K(dwBuf, fBuf, szSend, szReceive);
DI=dwBuf[5];
Close_Com(COM3);
```

Remark:

■ DigitalInCounterRead_8K

Description:

This function is used to output 8-bit data to a digital output module. The 0~7 bits of output data are mapped into the 0~7 channels of digital module output respectively.

Syntax:

```
[ C ]
WORD DigitalInCounterRead_8K(DWORD dwBuf[], float fBuf[], char szSend[],
char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] Channel No.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-8000 main unit.
dwBuf[8] :	[Output] DigitalIn counter value
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD DI_counter;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8040;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 0;
dwBuf[6] = 0;
dwBuf[7] = m_slot;
DigitalInCounterRead_8K(dwBuf, fBuf, szSend, szReceive);
DI_counter=dwBuf[8];
Close_Com(COM3);
```

Remark:

■ ClearDigitalInCounter_8K

Description:

This function is used to clear the counter value of the digital input channel No. of I-8000 series modules.

Syntax:

```
[ C ]
WORD ClearDigitalInCounter_8K(DWORD dwBuf[], float fBuf[],char szSend[],
                                char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] Channel No.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-8000 main unit.
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8040;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 0;
dwBuf[6] = 0;
dwBuf[7] = m_slot;
ClearDigitalInCounter_8K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalInLatch_8K

Description:

This function is used to obtain the digital input latch value of the high or low latch mode of I-8000 series modules.

Syntax:

```
[ C ]
WORD DigitalInLatch_8K(DWORD dwBuf[], float fBuf[], char szSend[],
                                      char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] 0→ select to latch low 1→ select to latch high
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-8000 main unit.
dwBuf[8] :	[Output] Latched data
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
```

```
DWORD DI_latch;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8040;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 0;
dwBuf[6] = 0;
dwBuf[7] = m_slot;
DigitalInLatch_8K(dwBuf, fBuf, szSend, szReceive);
DI_latch=dwBuf[8];
Close_Com(COM3);
```

Remark:

■ ClearDigitalInLatch_8K

Description:

This function is used to clean the latch status of digital input module when latch function has been enabled.

Syntax:

```
[ C ]
WORD ClearDigitalInLatch_8K(DWORD dwBuf[], float fBuf[], char szSend[],
                            char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	Not used.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-8000 main unit.
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8040;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 0;
dwBuf[6] = 0;
dwBuf[7] = m_slot;
ClearDigitalInLatch_8K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

6.2.2.3 I-87000 series modules

■ DigitalOut_87K

Description:

This function is used to set the digital output value of the digital output module for I-87000 series modules.

Syntax:

```
[ C ]
WORD DigitalOut_87K(DWORD dwBuf[], float fBuf[], char szSend[],char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87041/54/55/57/58/60/63/64/66/68
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input]16-bit digital output data.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87054;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 3;
dwBuf[6] = 0;
DigitalOut_87K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalOutReadBack_87K

Description:

This function is used to read back the digital output value of the digital output module for I-87000 series modules.

Syntax:

```
[ C ]
WORD DigitalOutReadBack_87K(DWORD dwBuf[], float fBuf[], char szSend[],
                            char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87041/54/55/57/58/60/63/64/66/68
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Output]16-bit digital output data.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD DO;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87054;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[6] = 0;
DigitalOutReadBack_87K (dwBuf, fBuf, szSend, szReceive);
DO=dwBuf[5] ;
Close_Com(COM3);
```

Remark:

■ DigitalBitOut_87K

Description:

This function is used to set the digital output value of the specific digital output channel No. of the digital output module for I-87000 series modules. The output value is only for "0" or "1".

Syntax:

```
                              [ C ]
WORD DigitalBitOut_87K(DWORD dwBuf[], float fBuf[], char szSend[],
                              char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87041/54/55/57/58/60/63/64/66/68
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input]1-bit digital output data.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] The digital output channel No.
dwBuf[8] :	[Input] Data to output(0 or 1)
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87054;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 0;
dwBuf[7] = 1;
dwBuf[8] = 1;
DigitalBitOut_87K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalIn_87K

Description:

This function is used to obtain the digital input value from I-87000 series modules.

Syntax:

```
[ C ]
WORD DigitalIn_87K(DWORD dwBuf[], float fBuf[],char szSend[], char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Output]16-bit digitalintput data.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD DI;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87054;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[6] = 0;
DigitalIn_87K(dwBuf, fBuf, szSend, szReceive);
DI=dwBuf[5];
Close_Com(COM3);
```

Remark:

■ DigitalInLatch_87K

Description:

This function is used to obtain the digital input latch value of the high or low latch mode of I-87000 series modules.

Syntax:

```
[ C ]
WORD DigitalInLatch_87K(DWORD dwBuf[], float fBuf[], char szSend[],
                        char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] 0:low latch mode, 1:high latch mode
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Output] Latch value
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD DI_latch;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87051;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 0;
DigitalInLatch_87K(dwBuf, fBuf, szSend, szReceive);
DI_latch=dwBuf[7];
Close_Com(COM3);
```

Remark:

■ ClearDigitalInLatch_87K

Description:

This function is used to output 8-bit data to a digital output module. The 0~7 bits of output data are mapped into the 0~7 channels of digital module output respectively.

Syntax:

```
[ C ]
WORD ClearDigitalInLatch_87K(DWORD dwBuf[], float fBuf[], char szSend[],
                             char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	Not used
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87051;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[6] = 0;
ClearDigitalInLatch_87K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalInCounterRead_87K

Description:

This function is used to obtain the counter value of the digital input channel No. of I-87000 series modules.

Syntax:

```
[ C ]
WORD DigitalInCounterRead_87K(DWORD dwBuf[], float fBuf[], char szSend[],
char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] The digital input channel No.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Output] Counter value of the digital input channel No.
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD DI_counter;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87051;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 0;
DigitalInCounterRead_87K (dwBuf, fBuf, szSend, szReceive);
DI_counter=dwBuf[7];
Close_Com(COM3);
```

Remark:

■ ClearDigitalInCounter_87K

Description:

This function is used to clear the counter value of the digital input channel No. of I-87000 series modules.

Syntax:

```
[ C ]
WORD ClearDigitalInCounter_87K(DWORD dwBuf[], float fBuf[], char szSend[],
char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] The digital input channel No.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87051;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 0;
ClearDigitalInCounter_87K (dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

6.3 Watch Dog Timer Functions

■ EnableWDT

■ DisableWDT

Description:

This function can be used to enable the watch dog timer (WDT) and users need to reset WDT in the assigned time set by users. Or LinCon will reset automatically.

Syntax:

```
[C]
void EnableWDT(unsigned int msecond)
void DisableWDT(void)
```

Parameter:

msecond:	LinCon will reset in the assigned time if users don’t reset WDT. The unit is mini-second.

Return Value:

None

Example:

```
EnableWDT(10000);   //Enable WDT interval 10000ms=10s
while (getchar()==10)
{
      printf("Refresh WDT\n");
      EnableWDT(10000);   //Refresh WDT 10s
}
printf("Disable WDT\n");
DisableWDT();
```

Remark:

■ WatchDogSWEven

Description:

This function is used to read the LinCon Reset Condition and users can reinstall the initial value according to the Reset Condition.

Syntax:

```
[C]
unsigned int WatchDogSWEven (void)
```

Parameter:

None

Return Value:

Just see the last number of the return value – **RCSR** (Reset Controller Status Register). For example : RCSR is “20009a4”, so just see the last number “4”. 4 is **0100** in bits and it means :

Bit 0 : Hardware Reset (Like : Power Off, Reset Button)

Bit 1 : Software Reset (Like : Type “Reboot” in command prompt)

Bit 2 : WDT Reset (Like : Use “EnableWDT(1000)”)

Bit 3 : Sleep Mode Reset (Not supported in the LinCon)

Example:

```
printf("RCRS = %x\n", WatchDogSWEven() );
```

Remark:

■ ClearWDTSWEven

Description:

This function is used to clear RCSR value.

Syntax:

```
[C]
void ClearWDTSWEven (unsigned int rcsr)
```

Parameter:

rcsr : Clear bits of RCSR. Refer to the following parameter setting：

1	clear bit 0
2	clear bit 1
4	clear bit 2
8	clear bit 3
F	clear bit 0 ~ bit 3

Return Value:

None

Example:

ClearWDTSWEven(0xF) ; //Used to clear bit 0 ~ bit 3 of RCRS to be zero.

Remark:

6.4 EEPROM Read/Write Functions

■ Enable_EEP

Description:

This function is used to make EEPROM able to read or write. It must be used before using Read_EEP or Write_EEP. This EEPROM is divided into 256 blocks (0 to 255), and each block is 64 bytes in length from offset 0 to 63.

Syntax:

```
[ C ]
void Enable_EEP(void)
```

Parameter:

None

Return Value:

None

Example:

```
Enable_EEP();
// After using this function, you can use Write_EEP or Read_EEP to write or read
// data of EEPROM.
```

Remark:

■ Disable_EEP

Description:

This function is used to make EEPROM unable to read or write. You need to use this function after using Read_EEP or Write_EEP. Then it will protect you from modifying your EEPROM data carelessly.

Syntax:

```
[ C ]
void Disable_EEP(void)
```

Parameter:

None

Return Value:

None

Example:

```
Disable_EEP();
// After using this function, you will not use Write_EEP or Read_EEP to write or
// read data of EEPROM.
```

Remark:

■ Read_EEP

Description:

This function will read one byte data from the EEPROM. There is a 16K-byte EEPROM in the main control unit in the LinCon-8000 system. This EEPROM is divided into 256 blocks (0 to 255), and each block is 64 bytes in length from offset 0 to 63. This EEPROM with its accessing APIs provides another mechanism for storing critical data inside non-volatile memory.

Syntax:

```
[ C ]
unsigned char Read_EEP(int block, int offset)
```

Parameter:

block :	[Input] the block number of EEPROM.
offset:	[Input] the offset within the block.

Return Value:

Data read from the EEPROM.

Example:

```
int block, offset;
unsigned char data;
data= ReadEEP(block, offset);
// Returned value: data= read an 8-bit value from the EEPROM (block & offset)
```

Remark:

■ Write_EEP

Description:

To write one byte of data to the EEPROM. There is a 16K-byte EEPROM in the main control unit of the LinCon-8000 system. This EEPROM is divided into 256 blocks (0 to 255), and each block is 64 bytes in length from the offset of 0 to 63. This EEPROM with its accessing APIs, provides another mechanism for storing critical data inside non-volatile memory.

Syntax:

```
                                   [ C ]
void Write_EEP(int block, int offset, unsigned char data)
```

Parameter:

block :	[Input] the block number of EEPROM.
offset:	[Input] the offset within the block.
Data:	[Input] data to write to EEPROM.

Return Value:

None

Example:

```
int block, offset;
unsigned char data=10;
WriteEEP(block, offset, data);
// Writes a 10 value output to the EEPROM (block & offset) location
```

Remark:

6.5 Analog Input Functions

6.5.1 For I-8000 modules via parallel port

■ I8017_Init

Description:

This function is used to initialize the I-8017H modules (Analog input module) into the specified slot. Users must execute this function before trying to use other functions within the I-8017H modules.

Syntax:

```
[ C ]
int I8017_Init(int slot)
```

Parameter:

slot :	[Input] specified slot of the LinCon-8000 system (Range: 1 to 7)

Return Value:

The version of library

Example:

```
int slot=1,ver;
ver=I8017_Init(slot);
// The I-8017H card is plugged in slot 1 of LinCon-8000 and initializes the module
// I-8017H.
```

Remark:

This function can be applied on module: I-8017H and I-8017HS.

■ I8017_SetLed

Description:

Turns the I-8017H modules LED’s on/off. They can be used to act as an alarm.

Syntax:

```
[ C ]
void I8017_SetLed(int slot, unsigned int led)
```

Parameter:

slot :	[Input] specified slot of the LinCon-8000 system (Range: 1 to 7)
led :	[Input] range from 0 to 0xffff

Return Value:

None

Example:

```
int slot=1;
unsigned int led=0x0001;
I8017_SetLed (slot, led);
// There will be a L/A-LED light on channel 0 of the I-8017H card which is plugged in slot 1 on the Lincon-8000.
```

Remark:

This function can be applied on module: I-8017H and I-8017HS.

■ I8017_SetChannelGainMode

Description:

This function is used to configure the range and mode of the analog input channel for the I-8017H modules in the specified slot before using the ADC (analog to digital converter).

Syntax:

```
[ C ]
void I8017_SetChannelGainMode (int slot, int ch, int gain, int mode)
```

Parameter:

slot :	[Input] Specify the slot in the LinCon-8000 system (Range: 1 to 7)
ch :	[Input] Specify the I-8017H channel (Range: 0 to 7) Specify the I-8017HS channel (Range: 0 to 15)
gain :	[Input] input range: **0: +/- 10.0V,** **1: +/- 5.0V,** **2: +/- 2.5V,** **3: +/- 1.25V,** **4: +/- 20mA.**
mode :	[Input] **0: normal mode** (polling)

Return Value:

None

Example:

```
int slot=1,ch=0,gain=0;
I8017_SetChannelGainMode (slot, ch, gain,0);
// The I-8017H card is plugged in slot 1 of Lincon-8000, and the range of the data //
value from channel 0 for I-8017H will be -10 ~ +10 V.
```

Remark:

This function can be applied on module: I-8017H and I-8017HS.

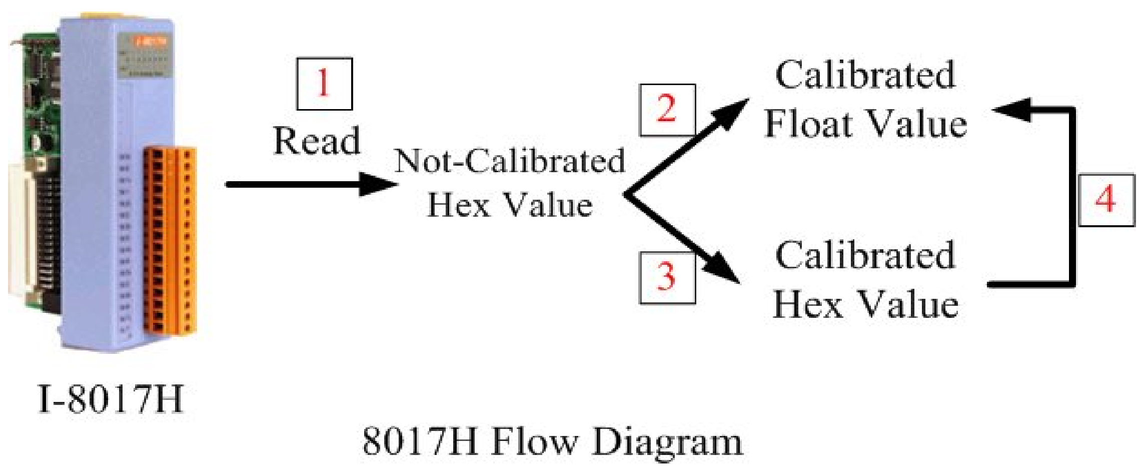


Fig.6-2

Function of [1]

■ **I8017_GetCurAdChannel_Hex**

Description:

Obtains the non-calibrated analog input value in the Hex format from the analog input I-8017H modules. Please refer to Fig. 6-2

Syntax:

```
[ C ]
int I8017_GetCurAdChannel_Hex (int slot)
```

Parameter:

slot : [Input] specified slot of the LinCon-8000 system (Range: 1 to 7)

Return Value:

The analog input value in Hex format.

Example:

```
int slot=1,ch=0,gain=4;
int data;
I8017_SetChannelGainMode (slot, ch, gain,0);
data= I8017_GetCurAdChannel_Hex (slot);
// The I-8017H card is plugged into slot 1 of LinCon-8000 and the range of the data
// value from channel 0 in I-8017H is +/- 20mA
```

■ I8017_AD_POLLING

Description:

This function is used to get the analog input non-calibrated hex values of the specified channel from an analog input module and can convert it to the value according to the slot configuration, the gain and the data number.

Syntax:

```
                              [ C ]
int I8017_AD_POLLING(int slot, int ch, int gain, unsigned int datacount,
                      int *DataPtr)
```

Parameter:

slot :	[Input] Specified slot in the LinCon-8000 system (Range: 1 to 7)
ch :	[Input] Specified channel for I-8017H (Range: 0 to 7) Specified channel for I-8017HS (Range: 0 to 15)
gain :	[Input] Input range: **0: +/- 10.0V,** **1: +/- 5.0V,** **2: +/- 2.5V,** **3: +/- 1.25V,** **4: +/- 20mA.**
datacount :	[Input] Range from 1 to 8192, total ADCs number
*DataPtr :	[Output] The starting address of data array[ ] and the array size must be equal to or bigger than the datacount.

Return Value:

0 : indicates success.

Not 0 : indicates failure.

Example:

```
int slot=1, ch=0, gain=0, data[10];
unsigned int datacount = 10;
I8017_AD_POLLING(slot, ch, gain, datacount, data);
// You gain ten not-calibrated hex values via channel 0 in the I-8017H module.
```

Function of [2]

■ HEX_TO_FLOAT_Cal

Description:

This function is used to convert the data from not-calibrated hex to calibrated float values based on the configuration of the slot, gain. (Voltage or current). Please refer to the Fig. 6-2.

Syntax:

```
[ C ]
float HEX_TO_FLOAT_Cal(int HexValue, int slot, int gain)
```

Parameter:

HexValue :	[Input] spedified not-calibrated HexValue before converting
slot :	[Input] specified slot of the LinCon-8000 system (Range: 1 to 7)
gain :	[Input] Input range:

0:	+/- 10.0V,
1:	+/- 5.0V,
2:	+/- 2.5V,
3:	+/- 1.25V,
4:	+/- 20mA.

Return Value:

The Calibrated Float Value.

Example:

```
int slot=1, ch=0, gain=0, hdata;
float fdata;
I8017_SetChannelGainMode (slot, ch, gain,0);
hdata = I8017_GetCurAdChannel_Hex (slot);
fdata = HEX_TO_FLOAT_Cal(hdata, slot, gain);
// You can convert not-calibrated Hex Value to calibrated Float Value
```

Remark:

This function can be applied on module: I-8017H and I-8017HS.

■ ARRAY_HEX_TO_FLOAT_Cal

Description:

This function is used to convert the data from non-calibrated hex values to calibrated float values in the array mode based on the slot's configuration. (Voltage or current). Please refer to Fig. 6-2.

Syntax:

```
                              [ C ]
void ARRAY_HEX_TO_FLOAT_cal(int *HexValue, float *FloatValue, int slot,
                              int gain,int len)
```

Parameter:

*HexValue :	[Input]	data array in not-calibrated Hex type before converting
*FloatValue :	[Output]	Converted data array in calibrated float type
slot :	[Input]	specified slot of the LinCon-8000 system (Range: 1 to 7)
gain :	[Input]	Input range:
len :	[input]	ADC data length

Return Value:

None

Example:

```
int slot=1, ch=0, gain=0, datacount=10, hdata[10];
float fdata[10];
I8017_SetChannelGainMode (slot, ch, gain,0);
I8017_AD_POLLING(slot, ch, gain, datacount, data);
ARRAY_HEX_TO_FLOAT_Cal(data, fdata, slot, gain, len);
// You can convert ten not-calibrated Hex values to ten calibrated Float values
```

Remark:

This function can be applied on module: I-8017H and I-8017HS.

Function of [3]

■ I8017_Hex_Cal

Description:

This function is used to convert the data from non-calibrated hex values to calibrated hex values. (Voltage or current). Please refer to Fig. 6-2.

Syntax:

```
                              [ C ]
int I8017_Hex_Cal(int data)
```

Parameter:

data :	[Input] specified not-calibrated hex value

Return Value:

The Calibrated Hex Value.

Example:

```
int slot=1, ch=0, gain=0, hdata;
int hdata_cal;
I8017_SetChannelGainMode (slot, ch, gain,0);
hdata = I8017_GetCurAdChannel_Hex (slot);
hdata_cal = I8017_Hex_Cal (hdata);
// You can convert not-calibrated Hex Value to calibrated Hex Value
```

Remark:

This function can be applied on module: I-8017H and I-8017HS.

■ I8017_Hex_Cal_Slot_Gain

Description:

This function is used to convert the data from non-calibrated hex values to calibrated hex values based on the configuration of the slot, gain. (Voltage or current).. (Voltage or current). Please refer to the Fig. 6-2.

Syntax:

```
[ C ]
int I8017_Hex_Cal_Slot_Gain(int slot, int gain, int data)
```

Parameter:

slot :	[Input]	specified slot of the LinCon-8000 system (Range: 1 to 7)
gain :	[Input]	Input range:
data :	[Input]	specified not-calibrated hex value

Return Value:

The Calibrated Hex Value.

Example:

```
int slot=1, ch=0, gain=0, hdata;
int hdata_cal;
I8017_SetChannelGainMode (slot, ch, gain,0);
hdata = I8017_GetCurAdChannel_Hex (slot);
hdata_cal = I8017_Hex_Cal_Slot_Gain (slot, gain, hdata);
// You can convert not-calibrated Hex Value to calibrated Hex Value according to
// the gain of slot you choose.
```

Remark:

This function can be applied on module: I-8017H and I-8017HS.

Function of [4]

■ CalHEX_TO_FLOAT

Description:

This function is used to convert the data from calibrated hex values to calibrated float values based on the configuration of the gain. (Voltage or current). Please refer to Fig. 6-2.

Syntax:

```
[ C ]
float CalHex_TO_FLOAT(int HexValue,int gain)
```

Parameter:

HexValue :	[Input] spedified not-calibrated HexValue before converting
gain :	[Input] Input range:

0:	+/- 10.0V,
1:	+/- 5.0V,
2:	+/- 2.5V,
3:	+/- 1.25V,
4:	+/- 20mA.

Return Value:

The Calibrated Float Value.

Example:

```
int slot=1, ch=0, gain=0, hdata, hdata_cal;
float fdata;
I8017_SetChannelGainMode (slot, ch, gain,0);
hdata = I8017_GetCurAdChannel_Hex (slot);
hdata_cal = I8017_HEX_Cal(hdata);
fdata = CalHex_TO_FLOAT(hdata_cal, gain);
// You can convert calibrated Hex Value to calibrated Float Value
```

Remark:

This function can be applied on module: I-8017H and I-8017HS.

■ ARRAY_CalHEX_TO_FLOAT

Description:

This function is used to convert the data from calibrated hex values to calibrated float values in the array mode based on the configuration of the gain. (Voltage or current). Please refer to the Fig. 6-2.

Syntax:

```
[ C ]
void ARRAY_CalHex_TO_FLOAT(int *HexValue, float *FloatValue, int gain, int len)
```

Parameter:

*HexValue :	[Input]	data array in calibrated Hex format
*FloatValue :	[Output]	Converted data array in calibrated float format
gain :	[Input]	Input range:
len :	[input]	ADC data length

Return Value:

The Calibrated Float Value.

Example:

```
int slot=1, ch=0, gain=0, hdata_cal[10];
float fdata[10];
fdata = ARRAY_CalHex_TO_FLOAT (hdata_cal, fdata, gain, len);
// You can convert ten calibrated Hex Values to ten calibrated Float Values.
```

Remark:

This function can be applied on module: I-8017H and I-8017HS.

Function of [1] + [3]

■ I8017_GetCurAdChannel_Hex_Cal

Description:

Obtain the calibrated analog input values in the Hex format directly from the analog input modules, i8017H . This function is a combination of the "I8017_GetCurAdChannel_Hex" function and the "I8017_Hex_Cal". Please refer to Fig. 6-2.

Syntax:

```
[ C ]
int I8017_GetCurAdChannel_Hex_Cal(int slot)
```

Parameter:

slot :	[Input] specified slot of the LinCon-8000 system (Range: 1 to 7)

Return Value:

The analog input value in Calibrated Hex format.

Example:

```
int slot=1,ch=0,gain=0, data;
I8017_SetChannelGainMode (slot, ch, gain,0);
data = I8017_GetCurAdChannel_Hex_Cal (slot);
// The I-8017H card is plugged into slot 1 of LinCon-8000 and the range of the
// data value from channel 0 in I-8017H is 0x0000 ~ 0x3fff.
```

Remark:

This function can be applied on module: I-8017H and I-8017HS.

■ I8017_AD_POLLING_Cal

Description:

This function is used to get the analog input calibrated hex values in the array mode from an analog input module and can convert according to the slot configuration value, the gain and the data number.

Syntax:

```
[ C ]
int I8017_AD_POLLING_Cal(int slot, int ch, int gain, unsigned int datacount,
                         int *DataPtr)
```

Parameter:

slot :	[Input] Specified slot in the LinCon-8000 system (Range: 1 to 7)
ch :	[Input] Specified channel for I-8017H (Range: 0 to 7) Specified channel for I-8017HS (Range: 0 to 15)
gain :	[Input] Input range: **0: +/- 10.0V,** **1: +/- 5.0V,** **2: +/- 2.5V,** **3: +/- 1.25V,** **4: +/- 20mA.**
datacount :	[Input] Range from 1 to 8192, total ADCs number
*DataPtr :	[Output] The starting address of data array[ ] and the array size must be equal to or bigger than the datacount.

Return Value:

0 : indicates success.

Not 0 : indicates failure.

Example:

```
int slot=1, ch=0, gain=0, data[10];
unsigned int datacount = 10;
I8017_AD_POLLING_Cal(slot, ch, gain, datacount, data);
// You gain ten calibrated hex values via channel 0 in the I-8017H module.
```

Function of [1]+[2]

■ I8017_GetCurAdChannel_Float_Cal

Description:

Obtains the calibrated analog input value in the Float format directly from the analog i8017H input modules. This function is a combination of the "I8017_GetCurAdChannel_Hex" function and the "Hex_TO_FLOAT_Cal" function. Please refer to Fig. 6-2.

Syntax:

```
[ C ]
int I8017_GetCurAdChannel_Float_Cal(int slot)
```

Parameter:

slot :	[Input] specified slot of the LinCon-8000 system (Range: 1 to 7)

Return Value:

The analog input value in Calibrated Float format.

Example:

```
int slot=1,ch=0,gain=0;
float data;
I8017_SetChannelGainMode (slot, ch, gain,0);
data = I8017_GetCurAdChannel_Float_Cal (slot);
// The I-8017H card is plugged into slot 1 of LinCon-8000 and the range of the
// data value from channel 0 in I-8017H is –10V ~ +10V.
```

Remark:

This function can be applied on module: I-8017H and I-8017HS .

6.5.2 For I-7000/I-8000/I-87000 modules via serial port

6.5.2.1 I-7000 series modules

■ AnalogIn

Description:

This function is used to obtain input value form I-7000 series modules.

Syntax:

```
[ C ]
WORD AnalogIn (WORD wBuf[], float fBuf[],char szSend[],char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7005/11/12/13/14/15/16/17/18/19/33
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Input] Channel number for multi-channel
wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Output] Analog input value return
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Note : “wBuf[6]” is the debug setting. If this parameter is set as “1”, user can get whole command string and result string from szSend[] and szReceive[] respectively.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
```

```
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7016;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[5] = 0;
wBuf[6] = 1;
AnalogIn (wBuf, fBuf, szSend, szReceive);     // szSend="#02"
AI = fBuf[0];                                  // AI = 1.9
Close_Com(COM3);
```

Remark:

■ AnalogInHex

Description:

This function is used to obtain the analog input value in “Hexadecimal” form I-7000 series modules.

Syntax:

```
[ C ]
WORD AnalogInHex (WORD wBuf[], float fBuf[],char szSend[],char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7005/11/12/13/14/15/16/17/18/19/33
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Input] Channel number for multi-channel
wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7] :	[Ouput] The analog input value in “Hexadecimal “ format
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Note : Users have to use DCON utility to set up the analog input configuration of the module in hex format.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7012;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[5] = 0;
wBuf[6] = 1;
AnalogInHex (wBuf, fBuf, szSend, szReceive);
AI = wBuf[7];                          // Hex format
Close_Com(COM3);
```

Remark:

■ AnalogInFsr

Description:

This function is used to obtain the analog input value in "FSR" format form I-7000 series modules. The "FSR" means "Percent" format.

Syntax:

[C]

```
WORD AnalogInFsr (WORD wBuf[], float fBuf[],char szSend[],char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7005/11/12/13/14/15/16/17/18/19/33
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Input] Channel number for multi-channel

wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Float Input/Output argument table.
fBuf[0] :	[Output] Analog input value return
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Note : Users have to use DCON utility to set up the analog input configuration of the module in hex format.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7012;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[5] = 0;
wBuf[6] = 1;
AnalogInFsr (wBuf, fBuf, szSend, szReceive);
AI = wBuf[7];
Close_Com(COM3);
```

Remark:

■ AnalogInAll

Description:

This function is used to obtain the analog input value of all channels form I-7000 series modules.

Syntax:

```
[ C ]
WORD AnalogInAll (WORD wBuf[], float fBuf[],char szSend[],char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7005/15/16/17/18/19/33
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Float Input/Output argument table.
fBuf[0] :	[Output] Analog input value return of channel_0
fBuf[1] :	[Output] Analog input value return of channel_1
fBuf[2] :	[Output] Analog input value return of channel_2
fBuf[3] :	[Output] Analog input value return of channel_3
fBuf[4] :	[Output] Analog input value return of channel_4
fBuf[5] :	[Output] Analog input value return of channel_5
fBuf[6] :	[Output] Analog input value return of channel_6
fBuf[7] :	[Output] Analog input value return of channel_7
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Note : Users have to use DCON utility to set up the analog input configuration of the module in hex format.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float AI[12];
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7017;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[6] = 1;
AnalogInAll (wBuf, fBuf, szSend, szReceive);
AI[0] = fBuf[0];
AI[0] = fBuf[1];
AI[0] = fBuf[2];
AI[0] = fBuf[3];
AI[0] = fBuf[4];
AI[0] = fBuf[5];
AI[0] = fBuf[6];
AI[0] = fBuf[7];
Close_Com(COM3);
```

Remark:

■ ThermocoupleOpen_7011

Description:

This function is used to detect the thermocouple state of I-7011 modules for the supporting type “J, K, T, E, R, S, B, N, C” is open or close. If the response value is “0”, thermocouple I-7011 is working in close state. If the response value is “1”, thermocouple I-7011 is working in open state. For more information please refer to user manual.

Syntax:

```
                              [ C ]
WORD ThermocoupleOpen_7011(WORD wBuf[], float fBuf[],char szSend[],
                              char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7011
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Output] response value 0 → the thermocouple is close response value 1 → the thermocouple is open
wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
WORD state;
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7011;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[5] = 0;
wBuf[6] = 1;
ThermocoupleOpen_7011(wBuf, fBuf, szSend, szReceive);
state = wBuf[5];
Close_Com(COM3);
```

Remark:

■ SetLedDisplay

Description:

This function is used to configure LED display for specified channel of I-7000 analog input serial modules.

Syntax:

[C]

```
WORD SetLedDisplay (WORD wBuf[], float fBuf[],char szSend[], char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7013/16/33
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond

wBuf[5] :	[Input] Set display channel
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7033;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[5] = 1;                    // Set channel 1 display
wBuf[6] = 1;
SetLedDisplay (wBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ GetLedDisplay

Description:

This function is used to get the current setting of the specified channel for LED display channel for specified channel of I-7000 analog input serial modules.

Syntax:

```
[ C ]
WORD GetLedDisplay (WORD wBuf[], float fBuf[],char szSend[], char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7013/16/33
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Output] Current channel for LED display 0 = channel_0 1 = channel_1
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf :	Not used
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
WORD led;
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
```

```
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7033;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[6] = 1;
GetLedDisplay (wBuf, fBuf, szSend, szReceive);
Led = wBuf[5];
Close_Com(COM3);
```

Remark:

6.5.2.2 I-8000 series modules

■ AnalogIn_8K

Description:

This function is used to obtain input value form I-8000 analog input series modules.

Syntax:

```
[ C ]
WORD AnalogIn_8K(DWORD dwBuf[], float fBuf[],char szSend[], char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8017
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] Channel number of analog input module
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7] :	[Input] Slot number.
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Output] Analog input value
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
AnalogIn_8K (dwBuf, fBuf, szSend, szReceive);
AI = fBuf[0];
Close_Com(COM3);
```

Remark:

■ AnalogInHex_8K

Description:

This function is used to obtain input value in “Hexadecimal” form I-8000 analog input series modules.

Syntax:

[C]

```
WORD AnalogInHex_8K(DWORD dwBuf[], float fBuf[],char szSend[],char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8017
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] Channel number of analog input module
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive

1 → Save to szSend & szReceive

dwBuf[7] :	[Input] Slot number.
dwBuf[8] :	[Output] The analog input value in Hex format.
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
DWORD AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
AnalogInHex_8K (dwBuf, fBuf, szSend, szReceive);
AI = dwBuf[8];
Close_Com(COM3);
```

Remark:

■ AnalogInFsr_8K

Description:

This function is used to obtain input value in “FSR” form I-8000 analog input series modules. The “FSR” means “Percent” format.

Syntax:

```
                              [ C ]
WORD AnalogInFsr_8K(DWORD dwBuf[], float fBuf[],char szSend[],
                    char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8017HW
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] Channel number of analog input module
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7] :	[Input] Slot number.
fBuf :	Float input/Output argument table.
fBuf[0] :	[Output] The analog input value.
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
AnalogInFsr_8K (dwBuf, fBuf, szSend, szReceive);
AI = fBuf[0];
Close_Com(COM3);
```

Remark:

■ AnalogInAll_8K

Description:

This function is used to obtain input value of all channels form I-8000 analog input series modules.

Syntax:

```
[ C ]
WORD AnalogInAll_8K(DWORD dwBuf[], float fBuf[], char szSend[],char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8017HW
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7] :	[Input] Slot number.
fBuf :	Float input/Output argument table.
fBuf[0] :	[Output] Analog input value of channel 0.
fBuf[1] :	[Output] Analog input value of channel 1.
fBuf[2] :	[Output] Analog input value of channel 2.
fBuf[3] :	[Output] Analog input value of channel 3.
fBuf[4] :	[Output] Analog input value of channel 4.
fBuf[5] :	[Output] Analog input value of channel 5.
fBuf[6] :	[Output] Analog input value of channel 6.
fBuf[7] :	[Output] Analog input value of channel 7.
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float AI[12];
float fBuf[12];
```

```
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[6] = 1;
dwBuf[7] = 1;
AnalogInAll_8K (dwBuf, fBuf, szSend, szReceive);
AI[0] = fBuf[0];
AI[1] = fBuf[1];
AI[2] = fBuf[2];
AI[3] = fBuf[3];
AI[4] = fBuf[4];
AI[5] = fBuf[5];
AI[6] = fBuf[6];
AI[7] = fBuf[7];
Close_Com(COM3);
```

Remark:

6.5.8.1 I-87000 series modules

■ AnalogIn_87K

Description:

This function is used to obtain input value form I-87000 series analog input modules.

Syntax:

```
[ C ]
WORD AnalogIn_87K(DWORD dwBuf[], float fBuf[],char szSend[],char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87013/17/18
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] Channel number for multi-channel
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive
	1 → Save to szSend & szReceive
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Output] The analog input value return
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
```

```
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
AnalogIn_87K(dwBuf, fBuf, szSend, szReceive);
AI = fBuf[0];
Close_Com(COM3);
```

Remark:

■ AnalogInHex_87K

Description:

This function is used to obtain input value in “Hexadecimal” form I-87000 series analog input modules.

Syntax:

```
[ C ]
WORD AnalogInHex_87K(DWORD dwBuf[], float fBuf[],char szSend[],
                     char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87013/17/18
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] Channel number for multi-channel
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive

dwBuf[7] :	[Output] The analog input value in “Hex” format.
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
DWORD AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
AnalogInHex_87K(dwBuf, fBuf, szSend, szReceive);
AI = dwBuf[8];
Close_Com(COM3);
```

Remark:

■ AnalogInFsr_87K

Description:

This function is used to obtain input value in “FSR” form I-87000 series analog input modules.

Syntax:

```
[ C ]
WORD AnalogInFsr_87K(DWORD dwBuf[], float fBuf[],char szSend[],
                     char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87013/17/18
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] Channel number for multi-channel
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive
	1 → Save to szSend & szReceive
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Output] The analog input value
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
DWORD AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
AnalogInHex_87K(dwBuf, fBuf, szSend, szReceive);
AI = fBuf[0];
Close_Com(COM3);
```

Remark:

■ AnalogInAll_87K

Description:

This function is used to obtain input value of all channels form I-87000 series analog input modules.

Syntax:

```
[ C ]
WORD AnalogInAll_87K(DWORD dwBuf[], float fBuf[],char szSend[],
                     char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87013/17/18
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Output] Analog input value of channel 0
fBuf[1] :	[Output] Analog input value of channel 1
fBuf[2] :	[Output] Analog input value of channel 2
fBuf[3] :	[Output] Analog input value of channel 3
fBuf[4] :	[Output] Analog input value of channel 4
fBuf[5] :	[Output] Analog input value of channel 5
fBuf[6] :	[Output] Analog input value of channel 6
fBuf[7] :	[Output] Analog input value of channel 7
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
DWORD AI;
float fBuf[12];
```

```
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[6] = 1;
AnalogInAll_87K(dwBuf, fBuf, szSend, szReceive);
AI[0] = dwBuf[0];
AI[1] = dwBuf[1];
AI[2] = dwBuf[2];
AI[3] = dwBuf[3];
AI[4] = dwBuf[4];
AI[5] = dwBuf[5];
AI[6] = dwBuf[6];
AI[7] = dwBuf[7];
Close_Com(COM3);
```

Remark:

6.6 Analog Output Functions

6.6.1 For I-8000 modules via parallel port

■ I8024_Initial

Description:

This function is used to initialize the I-8024 module in the specified slot. You must implement this function before you try to use the other I-8024 functions.

Syntax:

```
[ C ]
void I8024_Initial(int slot)
```

Parameter:

slot :	[Input] Specify the LinCon-8000 system slot (Range: 1 to 7)

Return Value:

None

Example:

```
int slot=1;
I8024_Initial(slot);
// The I-8024 card is plugged into slot 1 of LinCon-8000 and initializes the I-8024 module.
```

Remark:

This function can be applied on module: I-8024.

■ I8024_VoltageOut

Description:

This function is used to send the voltage float value to the I-8024 module with the specified channel and slot in the LinCon-8000 system.

Syntax:

```
[ C ]
void I8024_VoltageOut(int slot, int ch, float data)
```

Parameter:

slot :	[Input] Specified the LinCon-8000 system slot (Range: 1 to 7)
ch :	[Input] Output channel (Range: 0 to 3)
data :	[Input] Output data with engineering unit (Voltage Output: -10~ +10)

Return Value:

None

Example:

```
int slot=1, ch=0;
float data=3.0f;
I8024_VoltageOut(slot, ch, data);
//The I-8024 module output the 3.0V voltage from the channel 0.
```

Remark:

This function can be applied on module: I-8024.

■ I8024_CurrentOut

Description:

This function is used to initialize the I-8024module in the specified slot for current output. Users must call this function before trying to use the other I-8024 functions for current output.

Syntax:

```
[ C ]
void I8024_CurrentOut(int slot, int ch, float cdata)
```

Parameter:

slot :	[Input] Specify the LinCon-8000 system slot (Range: 1 to 7)
ch :	[Input] Output channel (Range: 0 to 3)
cdata :	[Input] Output data with engineering unit (Current Output: 0~20 mA)

Return Value:

None

Example:

```
int slot=1, ch=0;
float data=10.0f;
I8024_CurrentOut(slot, ch, data);
// Output the 10.0mA current from the channel 0 of I-8024 module.
```

Remark:

This function can be applied on module: I-8024.

■ I8024_VoltageHexOut

Description:

This function is used to send the voltage value in the Hex format to the specified channel in the I-8024 module, which is plugged into the slot in the LinCon-8000 system.

Syntax:

```
[ C ]
void I8024_VoltageHexOut(int slot, int ch, int hdata)
```

Parameter:

slot :	[Input] Specify the LinCon-8000 system slot (Range: 1 to 7)
ch :	[Input] Output channel (Range: 0 to 3)
hdata :	[Input] Output data with hexadecimal (data range: 0h ~ 3FFFh→ Voltage Output: -10. ~ +10. V)

Return Value:

None

Example:

```
int slot=1, ch=0; data=0x3000;
I8024_VoltageHexOut(slot, ch, data);
// The I-8024 module output the 5.0V voltage from the channel 0.
```

Remark:

This function can be applied on module: I-8024.

■ I8024_CurrentHexOut

Description:

This function is used to send the current value in the Hex format to the specified channel in the analog output module I-8024, which is plugged into the slot in the LinCon-8000 system.

Syntax:

```
[ C ]
void I8024_CurrentHexOut(int slot, int ch, int hdata)
```

Parameter:

slot :	[Input] Specify the LinCon-8000 system slot (Range: 1 to 7)
ch :	[Input] Output channel (Range: 0 to 3)
hdata :	[Input] Output data with hexadecimal (data range: 0h ~ 3FFFh → Current Output: 0. ~ +20.mA)

Return Value:

None

Example:

```
int slot=1, ch=0; data=0x2000;
I8024_CurrentHexOut(slot, ch, data);
// Output the 10.0mA current from the channel 0 of I-8024 module.
```

Remark:

This function can be applied on module: I-8024.

6.6.2 For I-7000/I-8000/I-87000 modules via serial port

6.6.2.1 I-7000 series modules

■ AnalogOut

Description:

This function is used to obtain analog value from analog output module of I-7000 series modules.

Syntax:

```
[ C ]
WORD AnalogOut(WORD wBuf[], float fBuf[],char szSend[], char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7016/21/22/24
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Input] The analog output channel number
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Input] Analog output value
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7016;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
// wBuf[5] = 0;                    // I-7016 no used
wBuf[6] = 1;
fBuf[0] = 3.5                       // Excitation Voltage output +3.5V
AnalogOut (wBuf, fBuf, szSend, szReceive);    ”
Close_Com(COM3);
```

Remark:

■ AnalogOutReadBack

Description:

This function is used to obtain read back the analog value of analog output modules of I-7000 series modules. There are two types of read back functions, as described in the following :

1. Last value is read back by $AA6 command
2. Analog output of current path is read back by $AA8 command

Syntax:

```
[ C ]
WORD AnalogOutReadBack(WORD wBuf[], float fBuf[],char szSend[],
                       char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7016/21/22/24
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Input] 0 : command $AA6 read back 1 : command $AA8 read back Note 1) When the module is I-7016: Don’t care. 2) When the module is I-7021/22, analog output of current path read back ($AA8) 3) When the module is I-7024, the updating value in a specific Slew rate ($AA8) (For more information, please refer to I-7021/22/24 manual)
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
wBuf[7] :	[Input] The analog output channel No. (0~3) of module I-7024 No used for single analog output module
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Output] Analog output read back value
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
Float Volt;
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7021;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[5] = 0;                        // $AA6 command
wBuf[6] = 1;
wBuf[7] = 1;
AnalogOutReadBack (wBuf, fBuf, szSend, szReceive);
Volt = fBuf[0];                     // Receive: “!01+2.57” excitation voltage , Volt=2.57
Close_Com(COM3);
```

Remark:

■ AnalogOutHex

Description:

This function is used to obtain analog value of analog output modules through Hex format.

Syntax:

```
[ C ]
WORD AnalogOutHex(WORD wBuf[], float fBuf[],char szSend[], char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7021/21P/22
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Input] The analog output channel number (No used for single analog output module)
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
wBuf[7] :	[Input] Analog output value in Hexadecimal data format
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
```

```
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7022;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[5] = 1;                    // channel 1
wBuf[6] = 1;
wBuf[7] = 0x250
AnalogOutHex (wBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ AnalogOutFsr

Description:

This function is used to obtain analog value of analog output modules through % of span data format. This function only can be used after analog output modules is set as "FSR" output mode.

Syntax:

[C]

```
WORD AnalogOutFsr(WORD wBuf[], float fBuf[],char szSend[], char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7021/21P/22
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Input] The analog output channel number (No used for single analog output module)

wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf :	Float Input/Output argument table.
FBuf[0] :	[Input] Analog output value in % of Span data format.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7022;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[5] = 1;                    // channel 1
wBuf[6] = 1;
fBuf[0] = 50
AnalogOutFsr (wBuf, fBuf, szSend, szReceive);   ”
Close_Com(COM3);
```

Remark:

■ AnalogOutReadBackHex

Description:

This function is used to obtain read back the analog value of analog output modules in Hex format for I-7000 series modules. There are two types of read back functions, as described in the following :

1. Last value is read back by $AA6 command
2. Analog output of current path is read back by $AA8 command

Syntax:

```
[ C ]
WORD AnalogOutReadBackHex(WORD wBuf[], float fBuf[],char szSend[],
                          char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7021/21P/22
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Input] 0 : command $AA6 read back 1 : command $AA8 read back
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
wBuf[7] :	[Input] The analog output channel No. No used for single analog output module
wBuf[9] :	[Output] Analog output value in Hexadecimal data format.
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

WORD Volt;

```
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7021;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[5] = 0;                    // command $AA6
wBuf[6] = 1;
wBuf[7] = 0;
AnalogOutReadBackHex (wBuf, fBuf, szSend, szReceive);
Volt = wBuf[9];
Close_Com(COM3);
```

Remark:

■ AnalogOutReadBackFsr

Description:

This function is used to obtain read back the analog value of analog output modules throuth % of span data format for I-7000 series modules. There are two types of read back functions, as described in the following :

1. Last value is read back by $AA6 command
2. Analog output of current path is read back by $AA8 command

Syntax:

```
                              [ C ]
WORD AnalogOutReadBackFsr(WORD wBuf[], float fBuf[],char szSend[],
                              char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7021/21P/22
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 msecond
wBuf[5] :	[Input] 0 : command $AA6 read back 1 : command $AA8 read back
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
wBuf[7] :	[Input] The analog output channel No. No used for single analog output module
fBuf :	Float input/output argument table.
fBuf[0] :	[Output] Analog output value in % Span data format.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float Volt;
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7021;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[5] = 0;                    // command $AA6
wBuf[6] = 1;
wBuf[7] = 0;
AnalogOutReadBackFsr (wBuf, fBuf, szSend, szReceive);
Volt = fBuf[0];
Close_Com(COM3);
```

Remark:

6.6.2.2 I-8000 series modules

■ AnalogOut_8K

Description:

This function is used to obtain analog value of analog output module for I-8000 series modules.

Syntax:

```
                        [ C ]
WORD AnalogOut_8K(DWORD dwBuf[], float fBuf[],char szSend[],
                  char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7] :	[Input] Slot number
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Input] Analog output value
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
fBuf[0] = 2.55
AnalogOut_8K (dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ AnalogOutReadBack_8K

Description:

This function is used to read back the analog value of analog output module for I-8000 series modules.

Syntax:

```
[ C ]
WORD AnalogOutReadBack_8K(DWORD dwBuf[], float fBuf[],char szSend[],
                          char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond

dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Input] Analog output value
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float Valot;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
AnalogOutReadBack_8K (dwBuf, fBuf, szSend, szReceive);
Volt = fBuf[0];
Close_Com(COM3);
```

Remark:

■ ReadConfigurationStatus_8K

Description:

This function is used to read configuration status of analog output module for I-8000 series modules.

Syntax:

```
[ C ]
WORD ReadConfigurationStatus_8K(DWORD dwBuf[], float fBuf[],char szSend[],
                                char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7] :	[Input] Slot number
dwBuf[8] :	[Output] Output range: 0x30, 0x31,0x32
dwBuf[9] :	[Output] Slew rate
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD Status;
DWORD Rate;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
ReadConfigurationStatus_8K (dwBuf, fBuf, szSend, szReceive);
Status = dwBuf[8];
Rate = dwBuf[9];
Close_Com(COM3);
```

Remark:

■ SetStartUpValue_8K

Description:

This function is used to setting start-up value of analog output module for I-8000 series modules.

Syntax:

```
[ C ]
WORD SetStartUpValue_8K(DWORD dwBuf[], float fBuf[],char szSend[],
                         char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF

dwBuf[2] :	[Input] Module ID, 0x8024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7] :	[Input] Slot number
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
SetStartUpValue_8K (dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ SetStartUpValue_8K

Description:

This function is used to read start-up value of analog output module for I-8000 series modules.

Syntax:

```
[ C ]
WORD ReadStartUpValue_8K(DWORD dwBuf[], float fBuf[],char szSend[],
                         char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7] :	[Input] Slot number
fBuf :	Float input/output argument table.
fBuf[0] :	[Output] Start-Up value.
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float StartUp;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
ReadStartUpValue_8K (dwBuf, fBuf, szSend, szReceive);
StartUp = fBuf[0];
Close_Com(COM3);
```

Remark:

■ AnalogOutReadBack_8K

Description:

This function is used to read back the analog value of analog output module for I-8000 series modules.

Syntax:

```
                              [ C ]
WORD AnalogOutReadBack_8K(DWORD dwBuf[], float fBuf[],char szSend[],
                              char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Input] Analog output value
szSend :	[Input] Command string to be sent to I-8000 series modules.
szReceive :	[Output] Result string receiving from I-8000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float Volt;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x8024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
AnalogOutReadBack_8K(dwBuf, fBuf, szSend, szReceive);
Volt = fBuf[0];
Close_Com(COM3);
```

Remark:

6.6.2.3 I-87000 series modules

■ AnalogOut_87K

Description:

This function is used to output input value form I-87000 series analog input modules.

Syntax:

```
                              [ C ]
WORD AnalogOut_87K(DWORD dwBuf[], float fBuf[],char szSend[],
                               char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87021/22/24/26
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] Channel number for multi-channel
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Output] The analog output value
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules .

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
```

```
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
fBuf[0] = 2.55;           //+2.55V
AnalogOut_87K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ AnalogOutReadBack_87K

Description:

This function is used to read back the analog value of analog output module for I-87000 series modules. There are two types of read back functions, as described in the following:

1. Last value is read back by $AA6 command
2. Analog output of current path is read back by $AA8 command

Syntax:

```
[ C ]
WORD AnalogOutReadBack_87K(DWORD dwBuf[], float fBuf[],char szSend[],
char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8022/24/26
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Outut] Analog output read back value
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float Volt;
float fBuf[12];
char szSend[80];
char szReceive[80];
```

```
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
AnalogOutReadBack_87K (dwBuf, fBuf, szSend, szReceive);
Volt = fBuf[0];
Close_Com(COM3);
```

Remark:

■ AnalogOutHex_87K

Description:

This function is used to output the analog value of analog output I-87000 modules through Hex format.

Syntax:

```
[ C ]
WORD AnalogOutHex_87K(DWORD dwBuf[], float fBuf[],char szSend[],
                      char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87022/26 (0x87024 has no Hex type)

dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] The analog output channel number
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7] :	[Input] Analog output value in Hexadecimal data format
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87022;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;                    // channel 1
dwBuf[6] = 1;
dwBuf[7] = 0x250
AnalogOutHex_87K (wBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ AnalogOutFsr_87K

Description:

This function is used to output the analog value of analog output through % of span data format for I-87000 series modules. This function only can be used after analog output module is set as “FSR” output mode.

Syntax:

```
[ C ]
WORD AnalogOutFsr_87K(DWORD dwBuf[], float fBuf[],char szSend[],
                      char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87022/26 (0x87024 has no Fsr type)
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] The analog output channel number
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf :	Not used.
fBuf[0] :	[Input] Analog output value in % of Span data format.
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87022;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;                    // channel 1
dwBuf[6] = 1;
dwBuf[7] = 0x250
AnalogOutFsr_87K (wBuf, fBuf, szSend, szReceive);    ”
Close_Com(COM3);
```

Remark:

■ AnalogOutReadBackHex_87K

Description:

This function is used to read back the analog value of analog output module in Hex format for I-87000 series modules. There are two types of read back functions, as described in the following:

1. Last value is read back by $AA6 command
2. Analog output of current path is read back by $AA8 command

Syntax:

```
[ C ]
WORD AnalogOutReadBackHex_87K(DWORD dwBuf[], float fBuf[],char szSend[],
char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87022/26 (0x87024 has no Hex type)
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] 0: command $AA6 read back 1: command $AA8 read back
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7] :	[Input] The analog output channel No.
dwBuf[9] :	[Outut] Analog output value read back in Hexadecimal data format.
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
DWORD Volt;
float fBuf[12];
```

```
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87022;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
AnalogOutReadBackHex_87K (dwBuf, fBuf, szSend, szReceive);
Volt = dwBuf[9];
Close_Com(COM3);
```

Remark:

■ AnalogOutReadBackFsr_87K

Description:

This function is used to read back the analog value of analog output module through % of span data format for I-87000 series modules. There are two types of read back functions, as described in the following:

1. Last value is read back by $AA6 command
2. Analog output of current path is read back by $AA8 command

Syntax:

```
[ C ]
WORD AnalogOutReadBackFsr_87K(DWORD dwBuf[], float fBuf[],char szSend[],
char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87022/26 (0x87024 has no Fsr type)
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] 0: command $AA6 read back 1: command $AA8 read back
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7] :	[Input] The analog output channel No.
fBuf :	Not used.
fBuf[0] :	[Outut] Analog output value read back in % of Span data format.
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float Volt;
float fBuf[12];
```

```
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87022;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
AnalogOutReadBackFsr_87K (dwBuf, fBuf, szSend, szReceive);
Volt = fBuf[0];
Close_Com(COM3);
```

Remark:

■ ReadConfigurationStatus_87K

Description:

This function is used to read configuration status of analog output module for I-87000 series modules.

Syntax:

```
[ C ]
WORD ReadConfigurationStatus_87K(DWORD dwBuf[], float fBuf[],char szSend[],
char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7] :	[Input] Slot number
dwBuf[8] :	[Output] Output range: 0x30, 0x31,0x32
dwBuf[9] :	[Output] Slew rate
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD Status;
DWORD Rate;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
ReadConfigurationStatus_87K (dwBuf, fBuf, szSend, szReceive);
Status = dwBuf[8];
Rate = dwBuf[9];
Close_Com(COM3);
```

Remark:

SetStartUpValue_87K

Description:

This function is used to setting start-up value of analog output module for I-87000 series modules.

Syntax:

```
[ C ]
WORD SetStartUpValue_87K(DWORD dwBuf[], float fBuf[],char szSend[],
                         char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7] :	[Input] Slot number
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
```

```
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
SetStartUpValue_87K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ ReadStartUpValue_87K

Description:

This function is used to setting start-up value of analog output module for I-87000 series modules.

Syntax:

```
[ C ]
WORD SetStartUpValue_87K(DWORD dwBuf[], float fBuf[],char szSend[],
                         char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 msecond
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive

1 → Save to szSend & szReceive

dwBuf[7] :	[Input] Slot number
fBuf :	Float input/output argument table.
fBuf[0] :	Start-Up value.
szSend :	[Input] Command string to be sent to I-87000 series modules.
szReceive :	[Output] Result string receiving from I-87000 series modules.

Return Value:

0 is for Success

Not 0 is for Failure

Example:

```
Float StartUp;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 0;
dwBuf[6] = 1;
dwBuf[7] = 1;
ReadStartUpValue_87K(dwBuf, fBuf, szSend, szReceive);
StartUp = fBuf[0];
Close_Com(COM3);
```

Remark:

6.7 3-axis Encoder Functions

■ I8090_REGISTRATION

Description:

In order to distinguish more than one I-8090 card in the LinCon-8000 platform, the I-8090 modules should be registered before using it. If there are no I-8090 modules in the LinCon-8000 at the given address, this function will return 0 which means a failure.

Syntax:

```
[ C ]
unsigned char I8090_REGISTRATION(unsigned char slot, unsigned int address)
```

Parameter:

slot :	[Input] Specify the LinCon-8000 slot (Range: 1 to 7)
address :	This parameter is not used in the LinCon-8000.

Return Value:

1: Success registration

0: Failure registration

Example:

```
unsigned char slot=1;
unsigned int address=0x0;
I8090_REGISTRATION(slot, address);   // I-8090 is plugged in slot 1 of LinCon.
```

Remark:

This function can be applied on module: I-8090.

■ I8090_INIT_CARD

Description:

This function is applied to reset the I-8090 counter values of three axes in a specific slot of LinCon and set the modes of three counters.

Syntax:

```
[ C ]
void I8090_INIT_CARD(unsigned char cardNo, unsigned char x_mode,
                     unsigned char y_mode, unsigned char z_mode)
```

Parameter:

cardNo :	[Input] Specify the LinCon-8000 slot (Range: 1 to 7)
x_mode :	[Input] The X axis counter mode. Refer to the Remarks.
y_mode :	[Input] The Y axis counter mode. Refer to the Remarks.
z_mode :	[Input] The Z axis counter mode. Refer to the Remarks.

Return Value:

None

Example:

```
unsigned char slot=1;
unsigned int address=0x0;
I8090_REGISTRATION(slot, address);
I8090_INIT_CARD(slot, ENC_QUADRANT, ENC_QUADRANT,
                ENC_QUADRANT);
//The X, Y, Z axis encoder mode are ENC_QUADRANT mode.
```

Remark:

There are three modes for each axis of I-8090 :

(1) ENC_QUADRANT

(2) ENC_CW_CCW

(3) ENC_PULSE_DIR

■ I8090_GET_ENCODER

Description:

This function is used to obtain the counter value of the selected axis on the specified encoder card. This counter value is defined in the short (16-bit) format.

Syntax:

```
[ C ]
unsigned int I8090_GET_ENCODER(unsigned char cardNo, unsigned char axis)
```

Parameter:

cardNo :	[Input] Specify the LinCon-8000 slot (Range: 1 to 7)
axis :	[Input] The selected axis. 1: X_axis; 2: Y_axis; 3: Z_axis

Return Value:

A 16 bits unsigned integer value.

Example:

```
unsigned char slot=1;
unsigned int address=0x0;
unsigned int data;
I8090_REGISTRATION(slot, address);
I8090_INIT_CARD(slot, ENC_QUADRANT, ENC_QUADRANT,
                       ENC_QUADRANT);
data= I8090_GET_ENCODER(slot, X_axis);
//The data value is the X-axis encoder value.
```

Remark:

This function can be applied on module: I-8090.

■ I8090_RESET_ENCODER

Description:

This function is used to reset the counter value to be zero for the selected axis on the specified encoder card.

Syntax:

```
[ C ]
void I8090_RESET_ENCODER(unsigned char cardNo, unsigned char axis)
```

Parameter:

cardNo :	[Input] Specify the LinCon-8000 slot (Range: 1 to 7)
axis :	[Input] The selected axis. 1: X_axis; 2: Y_axis; 3: Z_axis

Return Value:

None

Example:

```
unsigned char slot=1;
unsigned int address=0x0;
I8090_REGISTRATION(slot, address);
I8090_INIT_CARD(slot, ENC_QUADRANT, ENC_QUADRANT,
                        ENC_QUADRANT);
I8090_RESET_ENCODER(slot, X_axis);     //Set X-axis counter value to be zero.
```

Remark:

This function can be applied on module: I-8090.

■ I8090_GET_ENCODER32

Description:

This function is used to obtain the counter value of the selected axis on the specified encoder card. The counter value is defined in the long (32-bit) format. Users must call I8090_ENCODER32_ISR() function before using this function.

Syntax:

```
                              [ C ]
long I8090_GET_ENCODER32(unsigned char cardNo,unsigned char axis)
```

Parameter:

cardNo :	[Input] Specify the LinCon-8000 slot (Range: 1 to 7).
axis :	[Input] The selected axis. 1: X_axis; 2: Y_axis; 3: Z_axis

Return Value:

A 32 bits integer value.

Example:

```
unsigned char slot=1;
unsigned int address=0x0;
long data;
I8090_REGISTRATION(slot, address);
I8090_INIT_CARD(slot, ENC_QUADRANT, ENC_QUADRANT,
                    ENC_QUADRANT);
I8090_ENCODER32_ISR(slot);
data=I8090_GET_ENCODER32(slot, X_axis);
// The data value is the X-axis encoder value.
```

Remark:

This function can be applied on module: I-8090.

■ I8090_RESET_ENCODER32

Description:

This function is applied to reset the counter variable of the function I8090_Get_Encoder32.

Syntax:

```
[ C ]
void I8090_RESET_ENCODER32(unsigned char cardNo, unsigned char axis)
```

Parameter:

cardNo :	[Input] Specify the LinCon-8000 slot (Range: 1 to 7)
axis :	[Input] The selected axis. 1: X_axis; 2: Y_axis; 3: Z_axis

Return Value:

None

Example:

```
unsigned char slot=1;
unsigned int address=0x0;
I8090_REGISTRATION(slot, address);
I8090_INIT_CARD(slot, ENC_QUADRANT, ENC_QUADRANT,
                ENC_QUADRANT);
I8090_RESET_ENCODER(slot, X_axis);     // X-axis encoder value set zero.
```

Remark:

This function can be applied on module: I-8090.

■ I8090_GET_INDEX

Description:

This function is used to get the value of the “INDEX” register on the specified card.

Syntax:

```
[ C ]
unsigned char I8090_GET_INDEX(unsigned char cardNo)
```

Parameter:

cardNo : [Input] Specify the LinCon-8000 slot (Range: 1 to 7).

Return Value:

Register	Add.	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INDEX	0x08	R						ZI	YI	XI

Example:

```
unsigned char slot, data;
data=I8090_GET_INDEX(slot);        //Returned value: data=0x07
```

Remark:

This function can be applied on module: i8090. The index input C+/C- can be read out from this register. These bits are highly active.

XI : Indicate the index of X-axis.

YI : Indicate the index of Y-axis.

ZI : Indicate the index of Z-axis.

■ I8090_ENCODER32_ISR

Description:

This function is used to calculate the pulse value between present and last time with a ”long” type format. Therefore, I8090_ENCODER32_ISR() function should be executed periodically (2~10ms) using the timer interrupt or manual method.

Syntax:

```
[ C ]
void I8090_ENCODER32_ISR(unsigned char cardNo)
```

Parameter:

cardNo :	[Input] Specify the LinCon-8000 slot (Range: 1 to 7).

Return Value:

None

Example:

```
unsigned char slot;
long data;
i8090_ENCODER32_ISR(slot);        // should be called in 2~20ms.
data=I8090_GET_ENCODER32(slot, X_axis);
```

Remark:

This function can be applied on module: I-8090.

6.8 2-axis Stepper/Servo Functions

■ I8091_REGISTRATION

Description:

This function is used to assign a card number “cardNo” to the I-8091 card in the specified slot. In order to distinguish more than one of the I-8091 cards in the LinCon-8000 platform, the I-8091 cards should be registered before using them. If there are no I-8091 modules at the given address, this command will return 0 which is a failure value.

Syntax:

```
[ C ]
unsigned char I8091_REGISTRATION(unsigned char cardNo, int slot)
```

Parameter:

cardNO :	[Input] The board number (0~19)
slot :	[Input] The specific slot which I-8091 card is pluged in (1~7)

Return Value:

1: card exist
0: card not exist

Example:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
// The I-8091 card is plugged into slot 1 of LinCon-8000.
```

Remark:

This function can be applied on module: I-8091.

■ i8091_RESET_SYSTEM

Description:

This function is used to reset and terminate the running command in the 8091 module. Users can apply this command in software emergencies as a stop function. It can also clear all the card settings. After calling this function, users need to configure all the parameters in the I-8091 card.

Syntax:

```
                              [ C ]
void i8091_RESET_SYSTEM(unsigned char cardNo)
```

Parameter:

cardNO :	[Input] 0~19, The selected card number.

Return Value:

None

Example:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_RESET_SYSTEM(CARD1);
// The I-8091 card plugged in slot 1 of LinCon-8000
```

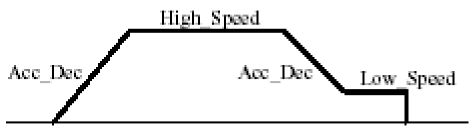
Remark:

This function can be applied on module: I-8091.

■ i8091_SET_VAR

Description:

This function is used to set the DDA cycle, plus accelerating/decelerating speeds, low-speed and the high-speed values in the specified I-8091 card.



Syntax:

[C]

```
void i8091_SET_VAR(unsigned char cardNo, unsigned char DDA_cycle,
          unsigned char Acc_Dec, unsigned int Low_Speed, unsigned int High_Speed)
```

Parameter:

cardNO :	[Input] 0~19, The selected card number.
DDA_cycle :	[Input] 1<=DDA_cycle<=254.
Acc_Dec :	[Input] 1<=Acc_Dec<=200.
Low_Speed :	[Input] 1<=Low_Speed<=200.
High_Speed :	[Input] Low_Speed<=High_Speed<=2047.

Return Value:

None

Example:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_SET_VAR(CARD1, 5, 2, 10, 150);
```

Remark:

This function can be applied on module: I-8091.

■ i8091_SET_DEFDIR

Description:

This function is used to define the rotating directions of the X and Y axes on the controlling motors. Sometimes, the output direction of the X-axis or Y-axis is in an undesired direction because of the wire connection to the motor or gear train mechanism. In order to unify the output direction, the CW/FW directions of the X/Y axis are defined as an outside going motion through the motor control, and similarly the CCW/BW directions are defined as the inward motion through the motor control.

Syntax:

```
[C]
void i8091_SET_DEFDIR(unsigned char cardNo, unsigned char defdirX,
                      unsigned char defdirY)
```

Parameter:

cardNO :	[Input] The board number (0~19)
defdirX :	[Input] X axis direction definition (0:NORMAL_DIR, 1:REVERSE_DIR)
defdirY :	[Input] Y axis direction definition (0:NORMAL_DIR, 1:REVERSE_DIR)

Return Value:

None

Example:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_SET_DEFDIR(CARD1, NORMAL_DIR, NORMAL_DIR);
```

Remark:

This function can be applied on module: I-8091.

■ i8091_SET_MODE

Description:

This function is used to set the motor control modes of the X and Y axes in the specified I-8091 card.

Syntax:

```
[ C ]
void i8091_SET_MODE(unsigned char cardNo, unsigned char modeX,
                    unsigned char modeY)
```

Parameter:

cardNO :	[Input] The board number (0~19)
modeX :	[Input] X axis output mode (0:CW/CCW mode, 1:Pulse/Direction mode)
modeY :	[Input] Y axis output mode (0:CW/CCW mode, 1:Pulse/Direction mode)

Return Value:

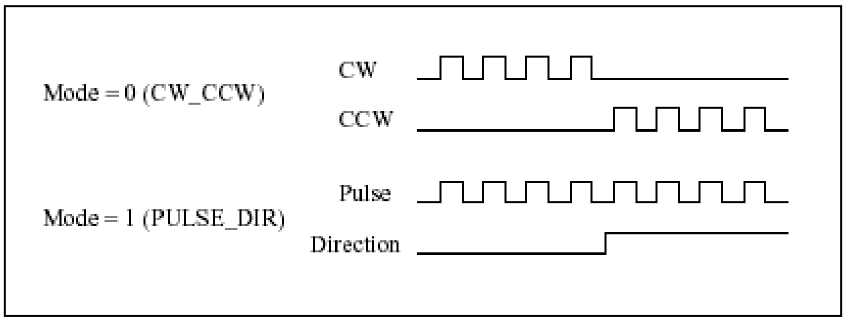
None

Example:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_SET_MODE(CARD1, CW_CCW, CW_CCW);
```

Remark:

This function can be applied on module: I-8091.



■ i8091_SET_SERVO_ON

Description:

This function is used to turn the servo function on/off to get the motor driver ready or to stop motor control.

Syntax:

```
[ C ]
void i8091_SET_SERVO_ON(unsigned char cardNo, unsigned char sonX,
unsigned char sonY)
```

Parameter:

cardNO :	[Input] The board number (0~19)
modeX :	[Input] X-axis servo/hold on switch (1:ON , 0:OFF)
modeY :	[Input] X-axis servo/hold on switch (1:ON , 0:OFF)

Return Value:

None

Example:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_SET_SERVO_ON(CARD1, ON, ON);
```

Remark:

This function can be applied on module: I-8091.

■ i8091_SET_NC

Description:

This function is used to set all of the following limit switches to N.C.(normal close) or N.O.(normal open). If users set the “sw” parameter as N.O, then those limit switches are active low. If users set the value as N.C, those limit switches are then “active high”. The auto-protection system will automatically change the judgments, whatever it is, to N.O. or N.C.

Limit switches: ORG1, LS11, LS14, ORG2, LS21, LS24, EMG.

Syntax:

[C]

```
void i8091_SET_NC(unsigned char cardNo, unsigned char sw)
```

Parameter:

cardNO :	[Input] The board number (0~19)
sw :	[Input] 0(NO) normal open (default), 1(YES) normal close.

Return Value:

None

Example:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_SET_NC(CARD1, NO, NO);
```

Remark:

This function can be applied on module: I-8091.

■ i8091_STOP_X

Description:

This function is used to stop the X-axis from running immediately.

Syntax:

```
[ C ]
void i8091_STOP_X(unsigned char cardNo)
```

Parameter:

cardNO :	[Input] The board number (0~19)

Return Value:

None

Example:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_STOP_X(CARD1);
// The X-axis is stopped.
```

Remark:

This function can be applied on module: i8091.

This command would stop the X axis immediately.

■ i8091_STOP_Y

Description:

This function is used to stop the Y-axis from running immediately.

Syntax:

```
[ C ]
void i8091_STOP_Y(unsigned char cardNo)
```

Parameter:

cardNO :	[Input] The board number (0~19)

Return Value:

None

Example:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_STOP_Y(CARD1);
// The Y-axis is stopped.
```

Remark:

This function can be applied on module: i8091.

This command would stop the Y axis immediately.

■ i8091_STOP_ALL

Description:

This function is used to stop both the X and Y axis immediately. It will clear all commands that are pending in the FIFO.

Syntax:

```
                         [ C ]
void i8091_STOP_ALL(unsigned char cardNo)
```

Parameter:

cardNO :	[Input] The board number (0~19)

Return Value:

None

Example:

```
#define CARD1   1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_STOP_ALL(CARD1);
//The X-axis and Y-axis are stopped.
```

Remark:

This function can be applied on module: i8091.

■ i8091_EMG_STOP

Description:

This function is the same as the i8091_STOP_ALL function, but can only be used in the interrupt routine. It can clear all the commands that are pending in the FIFO.

Syntax:

```
[ C ]
void i8091_EMG_STOP(unsigned char cardNo)
```

Parameter:

cardNO :	[Input] The board number (0~19)

Return Value:

None

Example:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_EMG_STOP(CARD1);
//The X-axis and Y-axis are stopped.
```

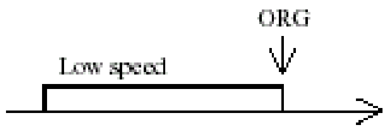
Remark:

This function can be applied on module: i8091.

■ i8091_LSP_ORG

Description:

This function is used to stop specific (X/Y) axis in low-speed movement when the ORG1/ORG2 limit switch is in contact.



Syntax:

```
[ C ]
void i8091_LSP_ORG(unsigned char cardNo, unsigned char DIR,
                   unsigned char AXIS)
```

Parameter:

cardNO :	[Input] The board number (0~19)
DIR :	[Input] The moving direction. (0:CW, 1:CCW)
AXIS :	[Input] The selected axis. (1: X_axis, 2: Y_axis)

Return Value:

None

Example:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_LSP_ORG(CARD1, CCW, X_axis);
i8091_LSP_ORG(CARD1, CCW, Y_axis);
```

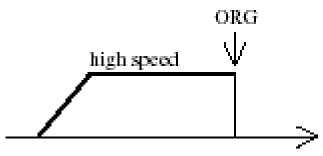
Remark:

This function can be applied on module: i8091.

■ i8091_HSP_ORG

Description:

This function drives the specified (X/Y) axis to search for their home position (ORG1/ORG2) in the high-speed mode motion and stops the motion when the ORG1/ORG2 limit switch is pushed.



Syntax:

```
[ C ]
void i8091_HSP_ORG(unsigned char cardNo, unsigned char DIR,
                   unsigned char AXIS)
```

```
[Visual Basic, C#]
void Wcon.ModuleName.HSP_ORG(byte cardNo, byte DIR, byte AXIS)
```

Parameter:

cardNO :	[Input] The board number (0~19)
DIR :	[Input] The moving direction. (0:CW, 1:CCW)
AXIS :	[Input] The selected axis. (1: X_axis, 2: Y_axis)

Return Value:

None

Example:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_HSP_ORG(CARD1, CCW, X_axis);
i8091_HSP_ORG(CARD1, CCW, Y_axis);
```

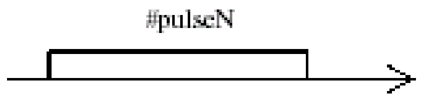
Remark:

This function can be applied on module: i8091.

■ i8091_LSP_PULSE_MOVE

Description:

This function drives the specified axis into motion toward the desired position from the given pulse number in low-speed mode.



Syntax:

```
[ C ]
void i8091_LSP_PULSE_MOVE(unsigned char cardNo, unsigned char AXIS,
long pulseN)
```

Parameter:

cardNO :	[Input] The board number (0~19)
AXIS :	[Input] The selected axis (1: X_axis, 2: Y_axis)
pulseN :	[Input] The moving number of pulse.

Return Value:

None

Example:

```
i8091_LSP_PULSE_MOVE(CARD1, X_axis, 20000);
i8091_LSP_PULSE_MOVE(CARD1, X_axis, -2000);
i8091_LSP_PULSE_MOVE(CARD1, Y_axis, 20000);
i8091_LSP_PULSE_MOVE(CARD1, Y_axis, -2000);
// when pulseN>0, move toward CW/FW direction
// when pulseN<0, move toward CCW/BW direction
```

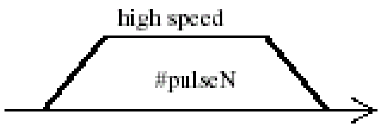
Remark:

This function can be applied on module: i8091.

■ i8091_HSP_PULSE_MOVE

Description:

This function drives the specified axis into motion toward the desired position from the given pulse number in high-speed mode.



Syntax:

[C++]

```
void i8091_HSP_PULSE_MOVE(unsigned char cardNo, unsigned char AXIS,
long pulseN)
```

Parameter:

cardNO :	[Input] The board number (0~19)
AXIS :	[Input] The selected axis (1: X_axis, 2: Y_axis)
pulseN :	[Input] The moving number of pulse.

Return Value:

None

Example:

```
i8091_HSP_PULSE_MOVE(CARD1, X_axis, 20000);
i8091_HSP_PULSE_MOVE(CARD1, X_axis, -2000);
i8091_HSP_PULSE_MOVE(CARD1, Y_axis, 20000);
i8091_HSP_PULSE_MOVE(CARD1, Y_axis, -2000);
// when pulseN>0, move toward CW/FW direction
// when pulseN<0, move toward CCW/BW direction
```

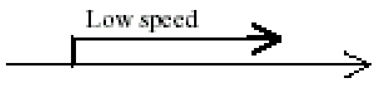
Remark:

This function can be applied on module: i8091.

■ i8091_LSP_MOVE

Description:

This function drives the specified axis into motion toward the selected direction in low-speed mode. It can be stopped by the i8091_STOP_X function, or the i8091_STOP_Y function, or the i8091_STOP_ALL function.



Syntax:

```
[ C ]
void i8091_LSP_MOVE(unsigned char cardNo, unsigned char DIR,
                    unsigned char AXIS)
```

Parameter:

cardNO :	[Input] The board number (0~19)
DIR :	[Input] The moving direction. (0:CW, 1:CCW)
AXIS :	[Input] The selected axis (1: X_axis, 2: Y_axis)

Return Value:

None

Example:

```
i8091_LSP_MOVE(CARD1, CW, X_axis);
i8091_LSP_MOVE(CARD1, CW, Y_axis);
```

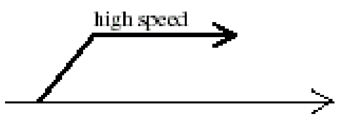
Remark:

This function can be applied on module: i8091.

■ i8091_HSP_MOVE

Description:

This function drives the specified axis into motion toward the selected direction in high-speed mode. It can be stopped by the i8091_STOP_X, or i8091_STOP_Y, or i8091_STOP_ALL functions.



Syntax:

```
[ C ]
void i8091_HSP_MOVE(unsigned char cardNo, unsigned char DIR,
                    unsigned char AXIS)
```

Parameter:

cardNO :	[Input] The board number (0~19)
DIR :	[Input] The moving direction. (0:CW, 1:CCW)
AXIS :	[Input] The selected axis (1: X_axis, 2: Y_axis)

Return Value:

None

Example:

```
i8091_HSP_MOVE(CARD1, CW, X_axis);
i8091_HSP_MOVE(CARD1, CW, Y_axis);
```

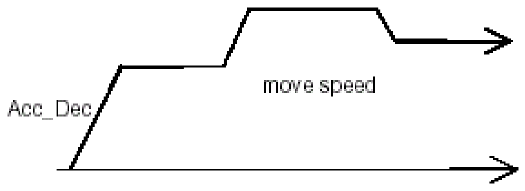
Remark:

This function can be applied on module: i8091.

■ i8091_CSP_MOVE

Description:

This function is used to accelerate/decelerate the motor on the selected axis into motion up/down to the desired speed. If commands are continuously applied to the I-8091, then this function can dynamically change the motor speed. The rotating motor can be stopped by using the following commands: i8091_STOP_X(), i8091_STOP_Y(), i8091_STOP_ALL(), or i8091_SLOW_STOP().



Syntax:

[C^{++}]

```
void i8091_CSP_MOVE(unsigned char cardNo, unsigned char dir,
                    unsigned char axis, unsigned int move_speed)
```

Parameter:

cardNO :	[Input] The board number (0~19)
dir :	[Input] The moving direction. (0:CW, 1:CCW)
axis :	[Input] The selected axis. (1: X_axis, 2: Y_axis)
move_speed :	[Input] 0 < move_speed <= 2040

Return Value:

None

Example:

```
i8091_CSP_MOVE(CARD1, CW, X_axis, 10);    //Delay(10000);
i8091_CSP_MOVE(CARD1, CW, X_axis, 20);
```

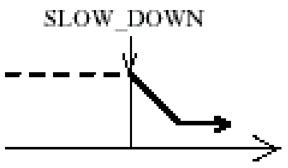
Remark:

This function can be applied on module: i8091.

■ i8091_SLOW_DOWN

Description:

This function is used to decelerate the motor motion to a specific low speed in order to be able to call either the i8091_STOP_X(), or i8091_STOP_Y(), or i8091_STOP_ALL functions so that the motor's motion can be stopped.



Syntax:

[C]

```
void i8091_SLOW_DOWN(unsigned char cardNo, unsigned char AXIS)
```

Parameter:

cardNO :	[Input] The board number (0~19)
AXIS :	[Input] The selected axis (1: X_axis, 2: Y_axis)

Return Value:

None

Example:

```
i8091_HSP_MOVE(CARD1, CW, X_axis);
i8091_SLOW_DOWN(CARD1, X_axis);
i8091_STOP_X(CARD1);
```

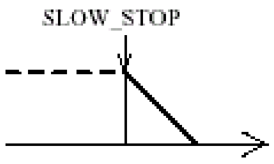
Remark:

This function can be applied on module: i8091.

■ i8091_SLOW_STOP

Description:

This function is used to decelerate the speed of a specified axis and then stop it (as shown in following figure).



Syntax:

```
[ C ]
void i8091_SLOW_STOP(unsigned char cardNo, unsigned char AXIS)
```

Parameter:

cardNO :	[Input] The board number (0~19)
AXIS :	[Input] The selected axis (1: X_axis, 2: Y_axis)

Return Value:

None

Example:

```
i8091_HSP_MOVE(CARD1, CW, Y_axis);
i8091_SLOW_STOP(CARD1, Y_axis);
```

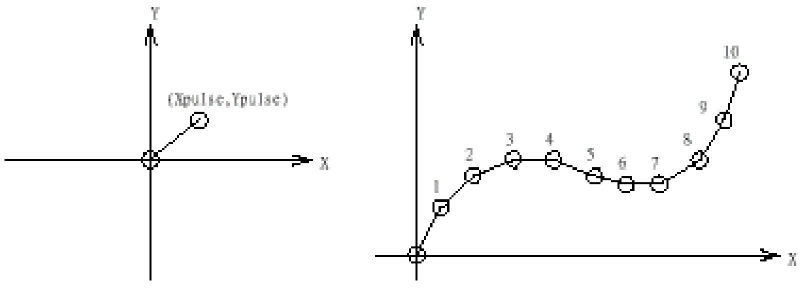
Remark:

This function can be applied on module: i8091.

■ i8091_INTP_PULSE

Description:

This function is used to move a short distance (interpolation short line) in the X-Y plane. This command provides a method for users to generate an arbitrary curve in a X-Y plane.



Syntax:

[C]

```
void i8091_INTP_PULSE(unsigned char cardNo, int Xpulse, int Ypulse)
```

Parameter:

cardNO :	[Input] The board number (0~19)
Xpulse :	[Input] －2047<= # Xpulse <=2047
Ypulse :	[Input] －2047<= # Ypulse <=2047

Return Value:

None

Example:

```
i8091_INTP_PULSE(CARD1, 20, 20);
i8091_INTP_PULSE(CARD1, 20, 13);
i8091_INTP_PULSE(CARD1, 20, 7);
i8091_INTP_PULSE(CARD1, 20, 0);
i8091_INTP_PULSE(CARD1,15, -5);
```

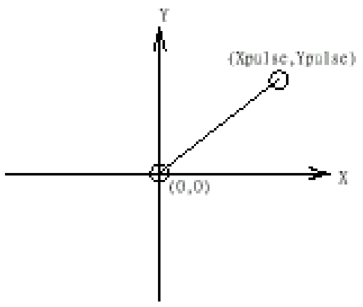
Remark:

This function can be applied on module: i8091.

■ i8091_INTP_LINE

Description:

This command will move a long distance (interpolation line) in the X-Y plane. The CPU on the I-8091 card will generate a trapezoidal speed profile of the X-axis and Y-axis, and then execute interpolation by way of a DDA chip.



Syntax:

```
[ C ]
void i8091_INTP_LINE(unsigned char cardNo, int Xpulse, int Ypulse)
```

Parameter:

cardNO :	[Input] The board number (0~19)
Xpulse :	[Input] －524287<= # Xpulse <=524287
Ypulse :	[Input] －524287<= # Ypulse <=524287

Return Value:

None

Example:

```
i8091_INTP_LINE(CARD1, 2000, -3000);
i8091_INTP_LINE(CARD1, -500, 200);
```

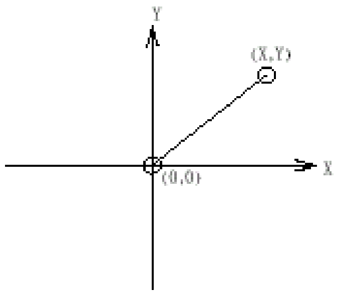
Remark:

This function can be applied on module: i8091.

■ i8091_INTP_LINE02

Description:

This function is used to move a long interpolation line in the X-Y plane. The host will automatically generate a trapezoidal speed profile of the X-axis and Y-axis via the state-machine-type calculation method. The i8091_INTP_LINE02 function only sets parameters into the driver. Users can directly utilize the do { } while (i8091_INTP_STOP( ) !=READY) method to apply the computing entity.



Syntax:

```
[ C ]
void i8091_INTP_LINE02(unsigned char cardNo, long x, long y,
                       unsigned int speed, unsigned char acc_mode)
```

Parameter:

cardNO :	[Input] The board number (0~19)
x :	[Input] The end point of the line relates to the present position
y :	[Input] The end point of the line relates to the present position
speed :	[Input] 0~2040
acc_mode :	[Input] 0: enables the acceleration and deceleration profiles 1: disables the acceleration and deceleration profiles

Return Value:

None

Example:

```
i8091_INTP_LINE02(CARD1, 1000, 1000, 100, 0);
do { } while(i8091_INTP_STOP()!=READY) ; //call state machine
```

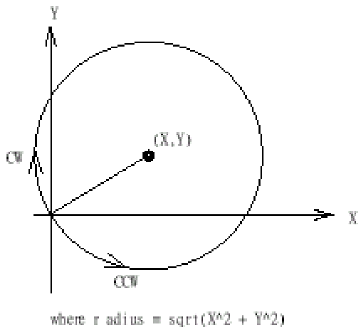
Remark:

This function can be applied on module: I-8091.

■ i8091_INTP_CIRCLE02

Description:

This function is used to generate an interpolation circle in the X-Y plane. The host will automatically generate a trapezoidal speed profile of the X-axis and Y-axis via the state-machine-type calculation method. The i8091_INTP_CIRCLE02 function only sets parameters into the driver. Users can directly utilize the do { } while (i8091_INTP_STOP( ) !=READY) method to execute the computing entity.



Syntax:

```
[ C ]
void i8091_INTP_CIRCLE02(unsigned char cardNo, long x, long y,
            unsigned chat dir, unsigned int speed, unsigned char acc_mode)
```

Parameter:

cardNO :	[Input] The board number (0~19)
x :	[Input] The center point of the circle relates to the present position
y :	[Input] The center point of the circle relates to the present position
dir :	[Input] The moving direction. (0:CW, 1:CCW)
speed :	[Input] 0~2040
acc_mode :	[Input] 0: enable acceleration and deceleration profile 1: disable acceleration and deceleration profile

Return Value:

None

Example:

```
i8091_INTP_CIRCLE02(CARD1, 2000, 2000, 100, 0);
do { } while(i8091_INTP_STOP()!=READY) ; //call state machine
```

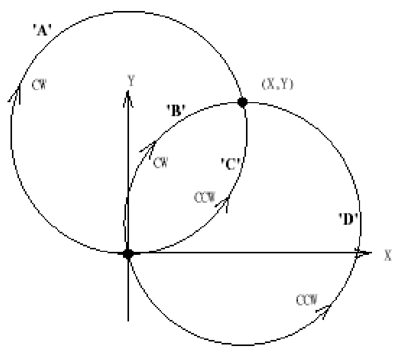
Remark:

This function can be applied on module: I-8091.

■ i8091_INTP_ARC02

Description:

This command generates an interpolation arc in the X-Y plane. The host will automatically generate a trapezoidal speed profile of the X-axis and Y-axis via the state-machine-type calculation method. The i8091_INTP_ARC02() only sets parameters into the driver. Users can directly call the do { } while (i8091_INTP_STOP( ) !=READY) method to execute the computing entity.



Syntax:

```
[ C ]
void i8091_INTP_ARC02(unsigned char cardNo, long x, long y, long R,
          unsigned char dir, unsigned int speed, unsigned char acc_mode)
```

Parameter:

cardNO :	[Input] The board number (0~19)
x :	[Input] The center point of the circle relates to the present position
y :	[Input] The center point of the circle relates to the present position
R :	[Input] The radius of arc
dir :	[Input] The moving direction (0:CW, 1:CCW)

R	dir	path of curve
R>0	CW	‘B’
R>0	CCW	‘C’
R<0	CW	‘A’
R<0	CCW	‘D’

speed :	[Input] 0~2040
acc_mode :	[Input] 0: enables the acceleration and deceleration profiles 1: disables the acceleration and deceleration profiles

Return Value:

None

Example:

```
i8091_INTP_ ARC02(CARD1, 2000, -2000, 2000, CW, 100, 0);
do { } while(i8091_INTP_STOP()!=READY) ; //call state machine
```

Remark:

This function can be applied on module: I-8091.

Restriction:

$$-2^{32}+1 \le \# x \le 2^{32}-1$$

$$-2^{32}+1 \le \# y \le 2^{32}-1$$

$$-2^{32}+1 \le \# R \le 2^{32}-1$$

$$R \ge \frac{\sqrt{x^2+y^2}}{2}$$

■ i8091_INTP_STOP

Description:

This function must be applied when stopping the motor motion control for the above described 3 state-machine-type interpolation functions, to be precise the i8091_INTP_LINE02, i8091_INTP_CIRCLE02 and i8091_INTP_ARC02 functions. The state-machine-type interpolation functions only set parameters into the driver. The computing entity is in the i8091_INTP_STOP function. This function computes the interpolation profile. It will return READY(0) for when the interpolation command is completed, and return BUSY(1) for when it is not yet completed.

Syntax:

```
[ C ]
unsigned char i8091_INTP_STOP(void)
```

Parameter:

None

Return Value:

0: READY

1: BUSY

Example:

```
i8091_INTP_CIRCLE02(CARD1,2000,2000,100,0);
do { } while(i8091_INTP_STOP()!=READY) ; //call state machine
```

Remark:

This function can be applied on module: i8091

■ i8091_LIMIT_X

Description:

This function is used to request the condition of the X-axis limit switches.

Syntax:

```
[ C ]
unsigned char i8091_LIMIT_X(unsigned char cardNo)
```

Parameter:

cardNO : [Input] The board number (0~19)

Return Value:

a unsigned char value

MSB 7	6	5	4	3	2	1	0
/EMG	/FFFF	/FFEF	/LS14	xx	xx	/LS11	/ORG1

/ORG1 : original point switch of X-axis, low active.

/LS11, /LS14 : limit switches of X-axis, low active, which must be configured as Fig.(5). (Refer to I-8091 User's Manual)

/EMG : emergency switch, low active.

/FFEF : active low, FIFO is empty

/FFFF : active low, FIFO is full

Example:

```
unsigned char limit1;
limit1 = i8091_LIMIT_X(CARD1);
```

Remark:

This function can be applied on module: I-8091.

■ i8091_LIMIT_Y

Description:

This function is used to request the condition of the Y-axis limit switches.

Syntax:

```
[ C ]
unsigned char i8091_LIMIT_Y(unsigned char cardNo)
```

Parameter:

cardNO : [Input] The board number (0~19)

Return Value:

a unsigned char value

MSB 7	6	5	4	3	2	1	0
Ystop	Xstop	xx	/LS24	xx	xx	/LS21	/ORG2

/ORG2: original point switch of Y-axis, low active.

/LS21, /LS24: limit switches of Y-axis, low active, which must be configured as Fig.(6). (Refer to I-8091 User's Manual)

Xstop: 1:indicate X-axis is stop.

Ystop: 1:indicate Y-axis is stop.

Example:

```
unsigned char limit2;
limit2 = i8091_LIMIT_Y(CARD1);
```

Remark:

This function can be applied on module: I-8091.

■ i8091_WAIT_X

Description:

This function is used to make the X-axis wait before going to the STOP state.

Syntax:

```
[ C ]
void i8091_WAIT_X(unsigned char cardNo)
```

Parameter:

cardNO :	[Input] The board number (0~19)

Return Value:

None

Example:

Remark:

This function can be applied on module: I8091.

■ i8091_WAIT_Y

Description:

This function is used to make the Y-axis wait before going to the STOP state.

Syntax:

```
[ C ]
void i8091_WAIT_Y(unsigned char cardNo)
```

Parameter:

cardNO :	[Input] The board number (0~19)

Return Value:

None

Example:

Remark:

This function can be applied on module: I-8091.

■ i8091_IS_X_STOP

Description:

This function is used to check whether the X-axis is in the stop state or not.

Syntax:

```
[ C ]
unsigned char i8091_IS_X_STOP(unsigned char cardNo)
```

Parameter:

cardNO :	[Input] The board number (0~19)

Return Value:

0 (NO) :	not yet stop
1 (YES) :	stop

Example:

```
unsigned char data;
data= i8091_IS_X_STOP(CARD1);
```

Remark:

This function can be applied on module: I-8091.

■ i8091_IS_Y_STOP

Description:

This function is used to check whether the Y-axis is in the stop state or not.

Syntax:

```
[ C ]
unsigned char i8091_IS_Y_STOP(unsigned char cardNo)
```

Parameter:

cardNO :	[Input] The board number (0~19)

Return Value:

0 (NO) :	not yet stop
1 (YES) :	stop

Example:

```
unsigned char data;
data= i8091_IS_Y_STOP(CARD1);
```

Remark:

This function can be applied on module: I-8091.

7. Demo of LinCon-8000 Modules With C Language

In this section, we will focus on examples for the description and application of the control functions on the I-7000/I-8000/I-87k series modules for use in the LinCon-8000. After you install the LinCon-8000 SDK, all these demo programs as below are in the path of **“c:/cygwin/lincon8k/examples”.**

7.1 I-7k Modules DIO Control Demo

This demo – **i7kdio.c** will illustrate how to control DI/DO with the I-7050 module (8 DO channels and 7 DI channels). The address and baudrate of the I-7050 module in the RS-485 network are 02 and 9600 separately.

The result of this demo allows the DO channels 0 ~ 7 output and DI channel 2 input. The source code of this demo program is as follows:

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80], ans;
WORD wBuf[12];
float fBuf[12];

/* ------------------------------------------------------------------ */
int main()
{
      int   wRetVal;

      // Check Open_Com3
      wRetVal = Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed!\n");
            return (-1);
      }
      //   *****   7050 DO && DI Parameter *******
```

```
    wBuf[0] = 3;                    // COM Port
    wBuf[1] = 0x02;                 // Address
    wBuf[2] = 0x7050;               // ID
    wBuf[3] = 0;                    // CheckSum disable
    wBuf[4] = 100;                  // TimeOut , 100 msecond
    wBuf[5] = 0x0ff;                // 8 DO Channels On
    wBuf[6] = 0;                    // string debug

    // 7050 DO Output
        wRetVal = DigitalOut(wBuf, fBuf, szSend, szReceive);
        if (wRetVal)
            printf("DigitalOut_7050 Error !, Error Code=%d\n", wRetVal);
        printf("The DO of 7050 : %u \n", wBuf[5]);

    // 7050 DI Input
        DigitalIn(wBuf, fBuf, szSend, szReceive);
        printf("The DI of 7050 : %u \n", wBuf[5]);
    Close_Com(COM3);
  return 0;
}
```

Follow the below steps to achieve the desired results：

STEP 1：**(Write i7kdio.c)**

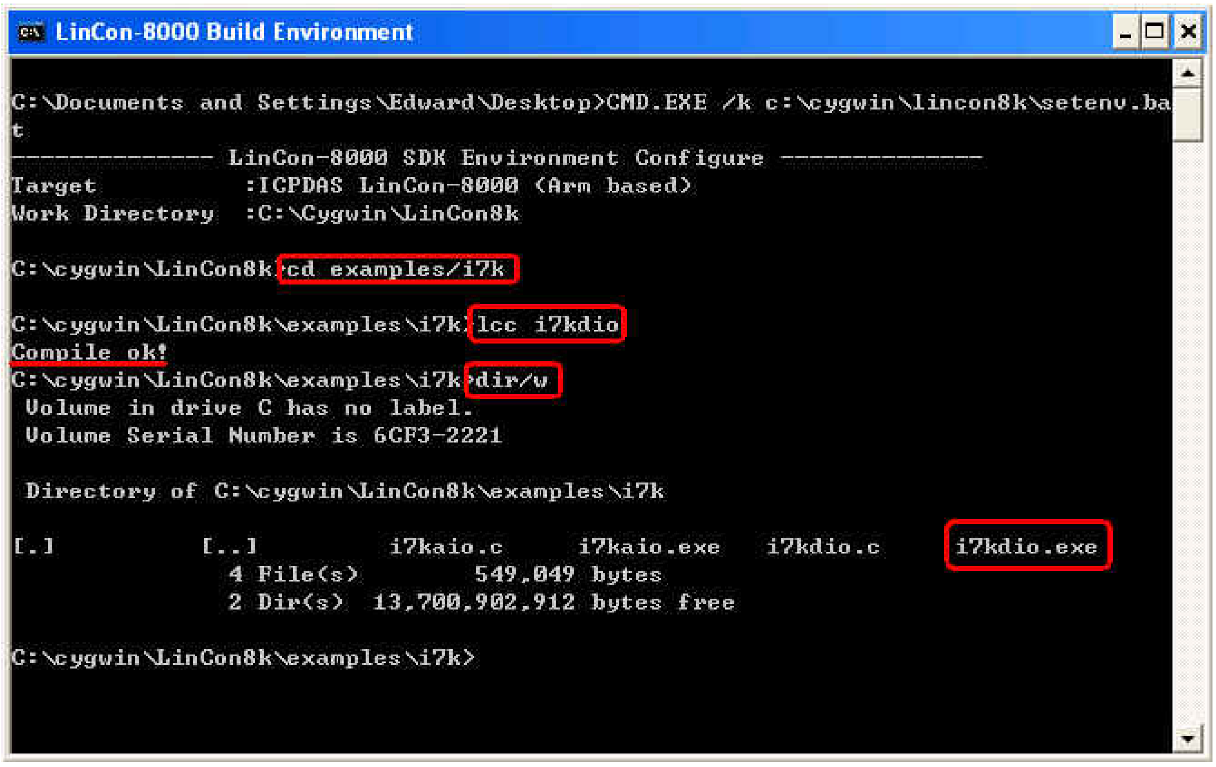
Copy the above source code and save it with the name - i7kdio.c or get the file from C:\cygwin\LinCon8k\examples\i7k.

STEP 2：**(Compile i7kdio.c to i7kdio.exe)**

Here we will introduce two methods to accomplish step 2.

< Method One > Using Batch File (lcc.bat)

Execute Start>Programs>ICPDAS>LinCon-8000 SDK> LinCon-8000 Build Environment to open **LinCon-8000 SDK** and change the path to C:\cygwin\LinCon8k\examples\i7k. Then type **lcc i7kdio** to compile i7kdio.c to i7kdio.exe. (refer to Fig. 7-1)

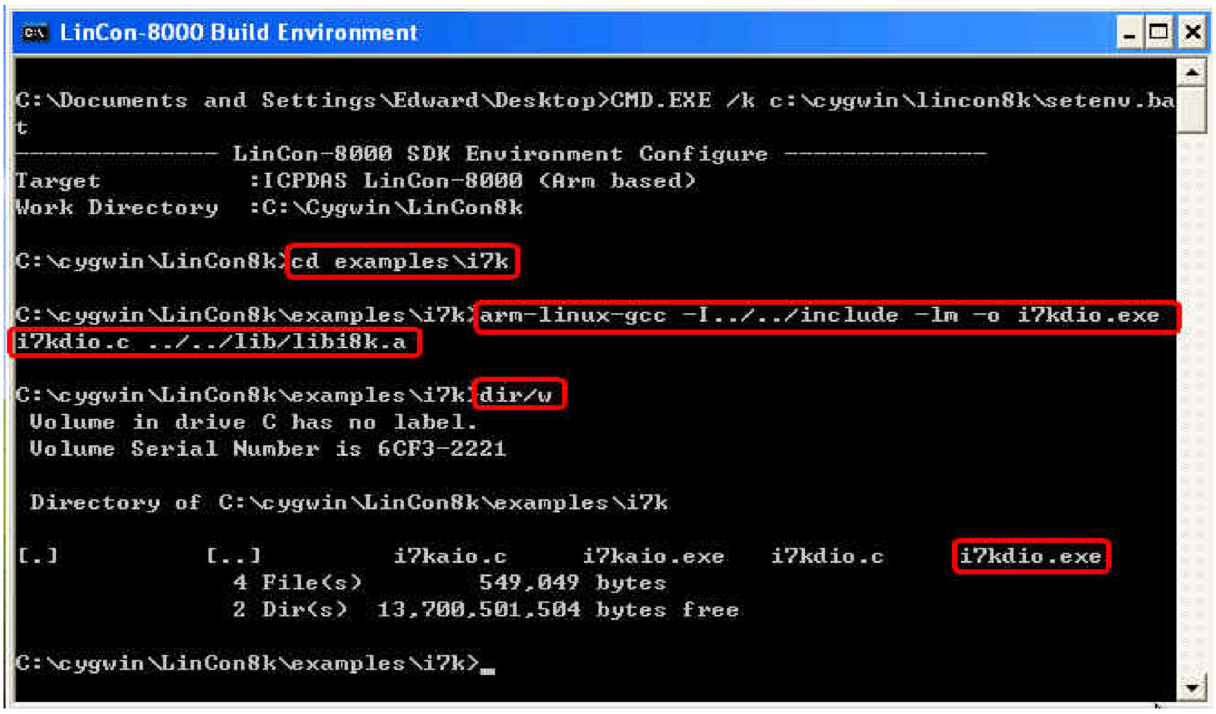


```
LinCon-8000 Build Environment

C:\Documents and Settings\Edward\Desktop>CMD.EXE /k c:\cygwin\lincon8k\setenv.ba
t
--------------- LinCon-8000 SDK Environment Configure ---------------
Target          :ICPDAS LinCon-8000 (Arm based)
Work Directory  :C:\Cygwin\LinCon8k

C:\cygwin\LinCon8k>cd examples/i7k

C:\cygwin\LinCon8k\examples\i7k>lcc i7kdio
Compile ok!
C:\cygwin\LinCon8k\examples\i7k>dir/w
 Volume in drive C has no label.
 Volume Serial Number is 6CF3-2221

 Directory of C:\cygwin\LinCon8k\examples\i7k

[.]            [..]           i7kaio.c       i7kaio.exe     i7kdio.c       i7kdio.exe
               4 File(s)        549,049 bytes
               2 Dir(s)  13,700,902,912 bytes free

C:\cygwin\LinCon8k\examples\i7k>
```

Fig. 7-1

< Method Two > Using Compile Instruction

If you choose this method, change the path to <u>C:\cygwin\LinCon8k\examples\i7k</u> and then type <u>arm-linux-gcc -I**../**../include –lm –o i7kdio.exe i7kdio.c **../**../lib/libi8k.a</u> to compile i7kdio.c to i7kdio.exe. (refer to Fig. 7-2)



```
LinCon-8000 Build Environment

C:\Documents and Settings\Edward\Desktop>CMD.EXE /k c:\cygwin\lincon8k\setenv.ba
t
--------------- LinCon-8000 SDK Environment Configure ---------------
Target          :ICPDAS LinCon-8000 (Arm based)
Work Directory  :C:\Cygwin\LinCon8k

C:\cygwin\LinCon8k>cd examples\i7k

C:\cygwin\LinCon8k\examples\i7k>arm-linux-gcc -I../../include -lm -o i7kdio.exe
i7kdio.c ../../lib/libi8k.a

C:\cygwin\LinCon8k\examples\i7k>dir/w
 Volume in drive C has no label.
 Volume Serial Number is 6CF3-2221

 Directory of C:\cygwin\LinCon8k\examples\i7k

[.]            [..]           i7kaio.c       i7kaio.exe     i7kdio.c       i7kdio.exe
               4 File(s)        549,049 bytes
               2 Dir(s)  13,700,501,504 bytes free

C:\cygwin\LinCon8k\examples\i7k>_
```

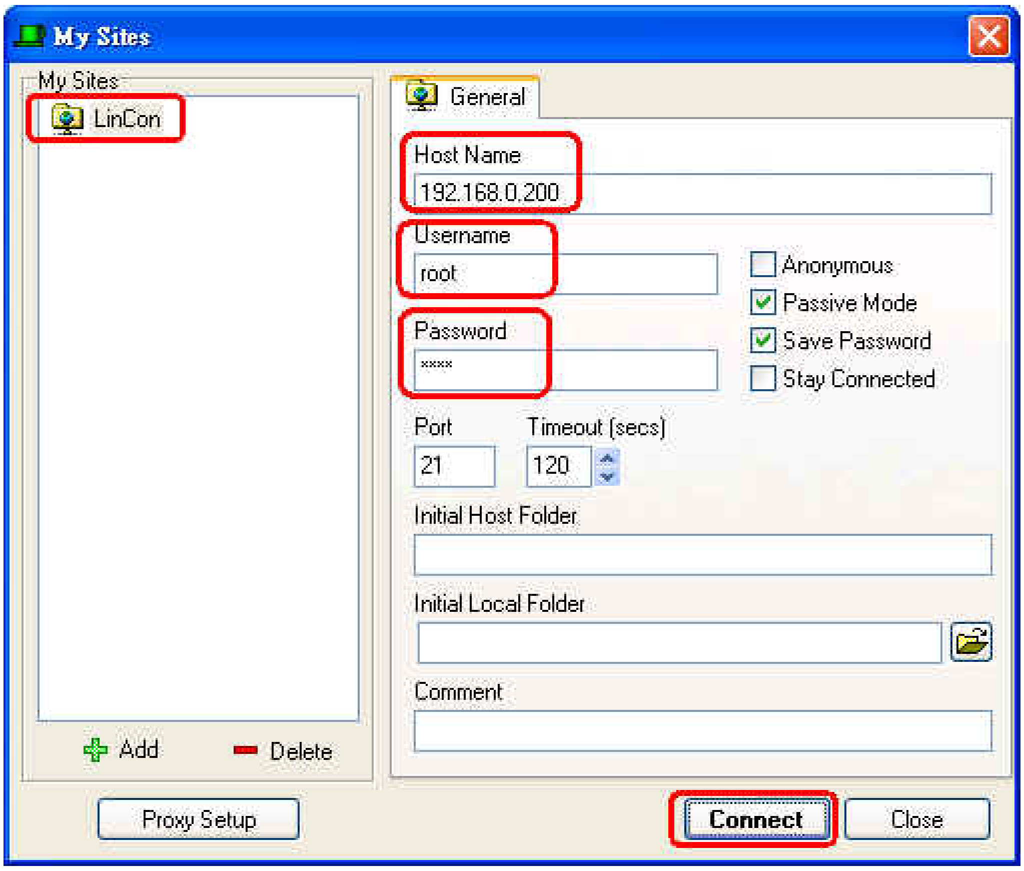
Fig. 7-2

STEP 3：**(Transfer i7kdio.exe to the LinCon-8000)**

Here we introduce two methods for achieving this purpose.

< Method One > Using FTP Software

(1) Open a FTP Software and add a ftp site of the LinCon-8000. The **User_Name** and **Password** default value is **" root "**. Then click the **"Connect"** button to connect to the ftp server of the LinCon-8000. (refer to Fig.7-3).


My Sites

LinCon

General

Host Name: 192.168.0.200

Username: root

Password: ****

Anonymous [ ]

Passive Mode [✓]

Save Password [✓]

Stay Connected [ ]

Port: 21

Timeout (secs): 120

Initial Host Folder

Initial Local Folder

Comment

Fig.7-3

(2) Upload the file – **i7kdio.exe** to the LinCon-8000. (refer to Fig.7-4).

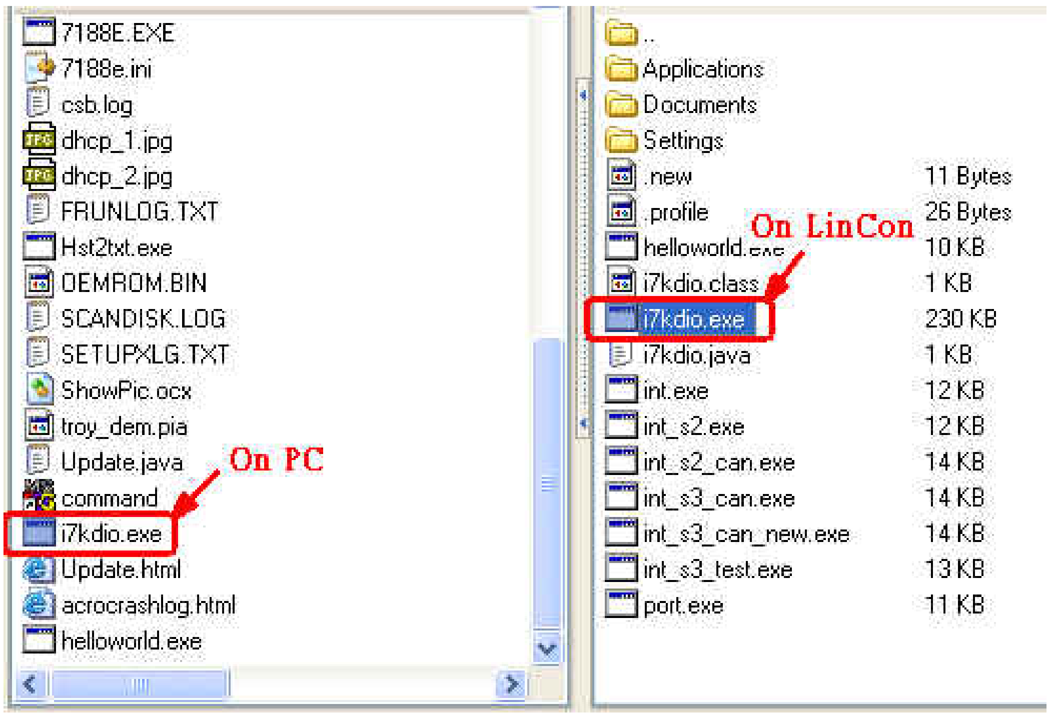
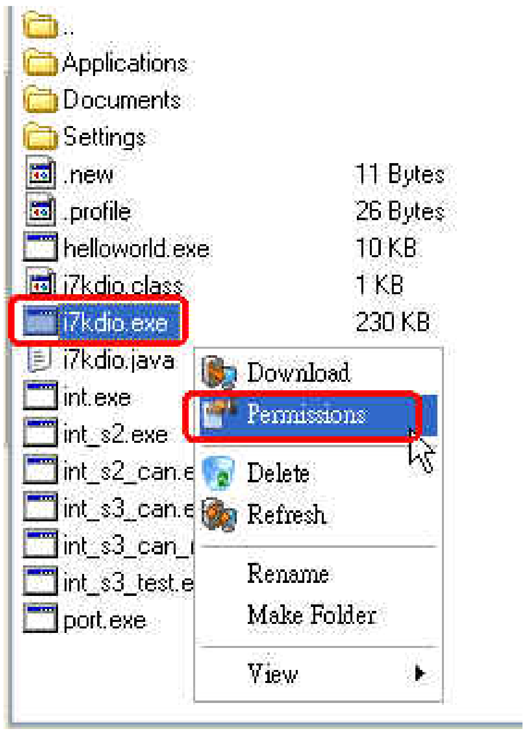
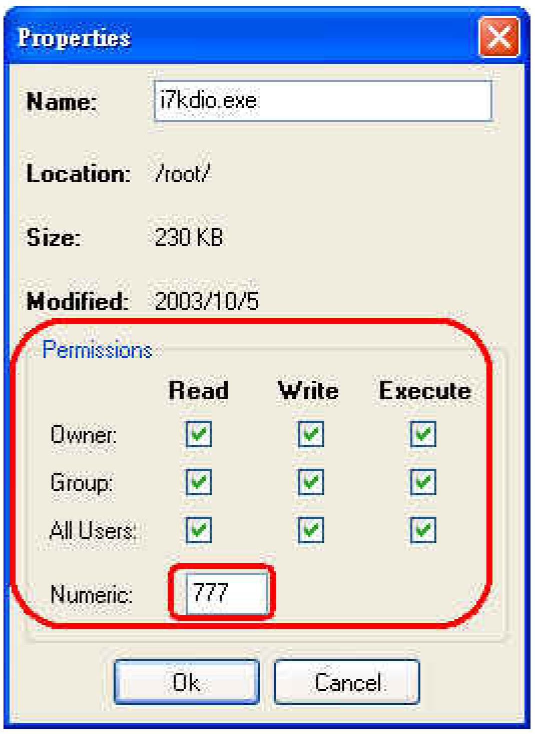


Fig.7-4

(3) Choose i7kdio.exe in the LinCon-8000 and Click the right mouse button to choose the “ **Permission** ”option. Then type 777 into the Numeric blank textbox. (refer to Fig.7-5 and refer to Fig.7-6).





Properties

Name: i7kdio.exe

Location: /root/

Size: 230 KB

Modified: 2003/10/5

Permissions

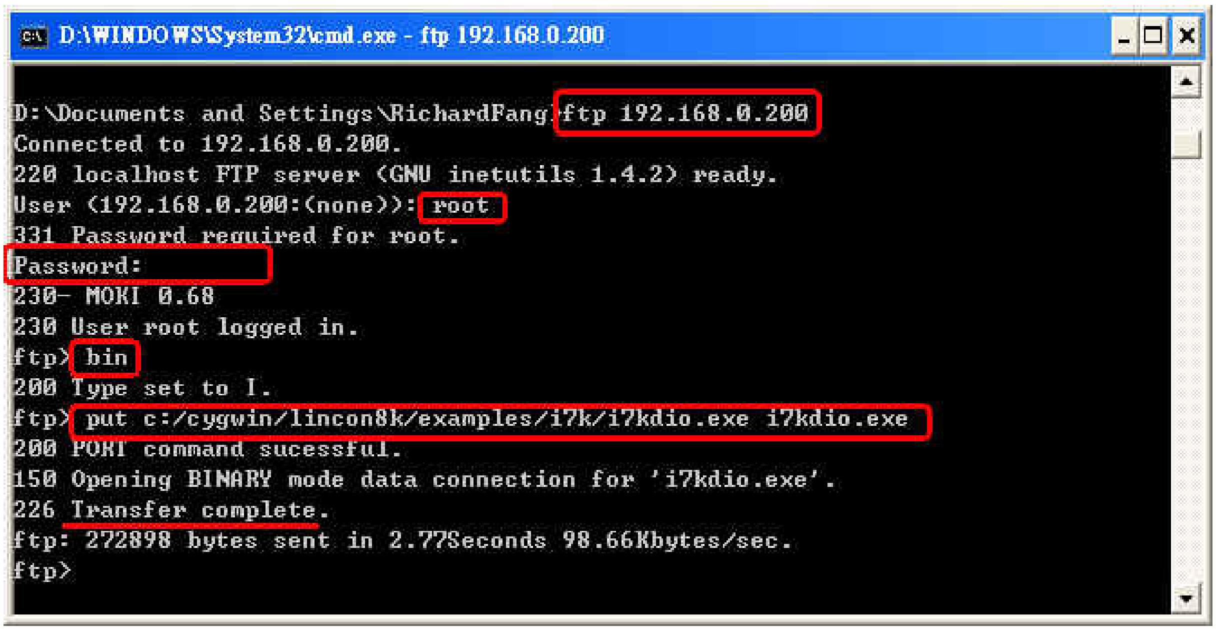
	Read	Write	Execute
Owner:	✓	✓	✓
Group:	✓	✓	✓
All Users:	✓	✓	✓

Numeric: 777

Fig.7-5 Fig.7-6

< Method Two > Using DOS Command Prompt

Open DOS Command Prompt and type ftp IP Address of LinCon-8000 in order to connect to the ftp server of the LinCon-8000. Then input **User Name** and **Password** (**root** is the default value) to login to the LinCon-8000. Type **bin** to make the file transference in "binary" mode. Then type put c:/cygwin/lincon8k/examples/i7k/i7kdio.exe i7kdio.exe to transfer the i7kdio.exe to the LinCon-8000. After the "Transfer complete" message appears, the process of transference would have been completed.(refer to Fig. 7-7)

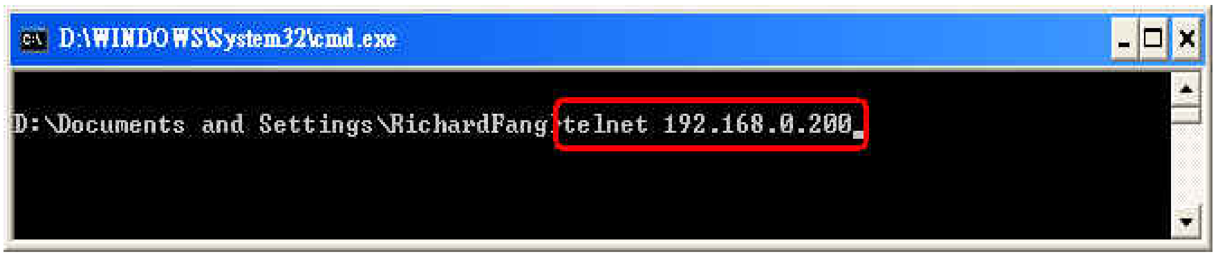


```
D:\Documents and Settings\RichardFang>ftp 192.168.0.200
Connected to 192.168.0.200.
220 localhost FTP server (GNU inetutils 1.4.2) ready.
User (192.168.0.200:(none)): root
331 Password required for root.
Password:
230- MOXI 0.68
230 User root logged in.
ftp> bin
200 Type set to I.
ftp> put c:/cygwin/lincon8k/examples/i7k/i7kdio.exe i7kdio.exe
200 PORT command sucessful.
150 Opening BINARY mode data connection for 'i7kdio.exe'.
226 Transfer complete.
ftp: 272898 bytes sent in 2.77Seconds 98.66Kbytes/sec.
ftp>
```

Fig. 7-7

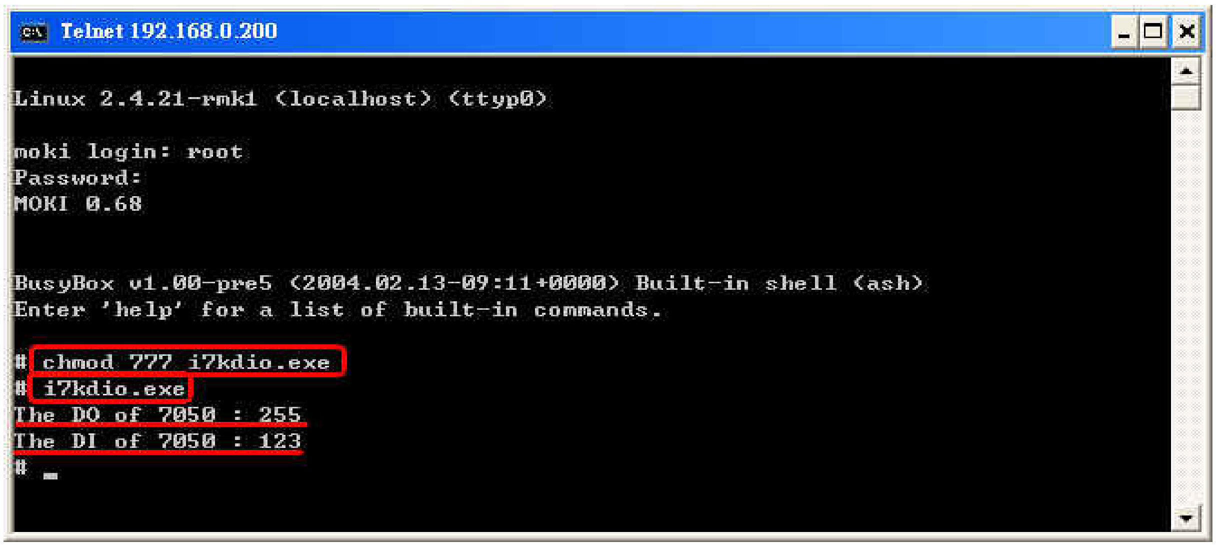
STEP 4 : **(Telnet to the LinCon-8000 to execute i7kdio.exe)**

Type telnet IP Address of LinCon-8000 into the remote control the LinCon-8000 and input your **User Name** and **Password** (**root** is the default value) to login to the LinCon-8000. And then type chmod 777 i7kdio.exe to make i7kdio.exe executable. Type i7kdio.exe to execute i7kdio.exe. (refer to Fig. 7-8 and Fig. 7-9)



```
D:\Documents and Settings\RichardFang>telnet 192.168.0.200_
```

Fig. 7-8



Telnet 192.168.0.200

```
Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.02.13-09:11+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# chmod 777 i7kdio.exe
# i7kdio.exe
The DO of 7050 : 255
The DI of 7050 : 123
# _
```

Fig. 7-9

" **The DO of I-7050：255** (=2^**8**-1)" means DO channel 0 ~ 7 will output and " **The DI of I-7050：123** (=127-2^**2**)" means there is input in DI channel 2.

7.2 I-7k Modules AIO Control Demo

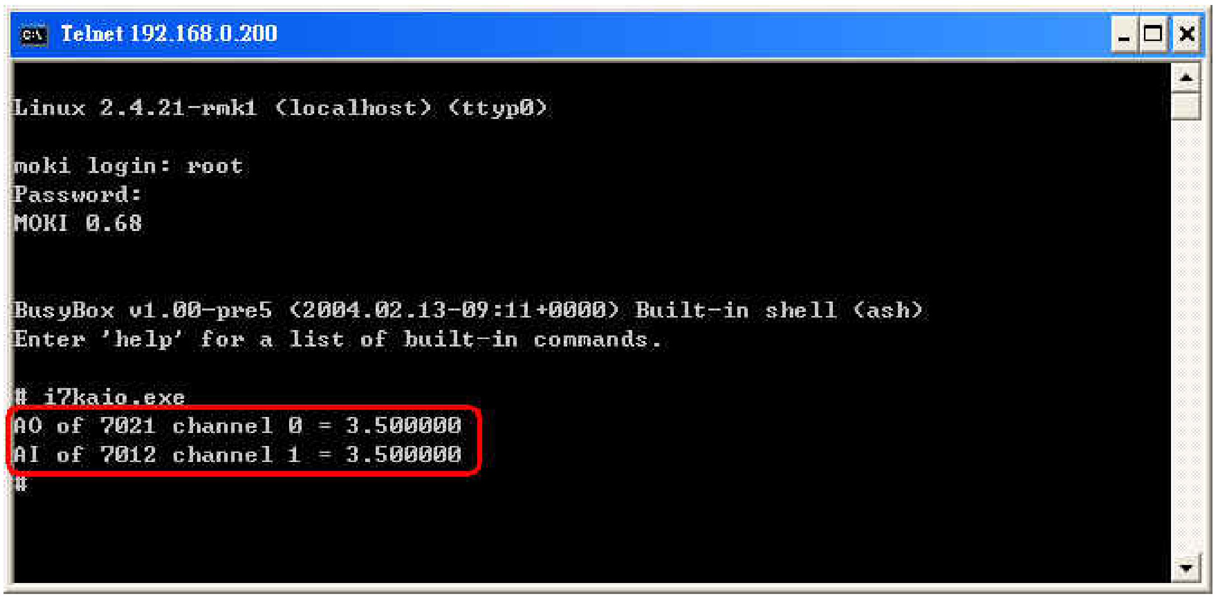
This demo – **i7kaio.c** will illustrate how to control the AI/AO with the I-7017 (8 AI channels) and I-7021 modules (1 AO channel). The address for the I-7021 and I-7017 modules are in the RS-485 network where 05 and 03 are separate and the baudrate is 9600.

The result of this demo allows the I-7021 module's AO channel to output 3.5V and the I-7017 's AI channel 2 to input. The source code of this demo program is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
WORD wBuf[12];
float fBuf[12];

/* ----------------------------------------------------------------- */
int main()
{
      int i,j, wRetVal;
      DWORD temp;
```

```
        wRetVal = Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
        if (wRetVal > 0) {
                printf("open port failed!\n");
                return (-1);
        }

        //--- Analog output ----    ****      7021 -- AO    ****
        i = 0;
        wBuf[0] = 3;                        // COM Port
        wBuf[1] = 0x05;                     // Address
        wBuf[2] = 0x7021;           // ID
        wBuf[3] = 0;                        // CheckSum disable
        wBuf[4] = 100;                      // TimeOut , 100 msecond
        //wBuf[5] = i;                      // Not used if module ID is 7016/7021
                                            // Channel No.(0 to 1) if module ID is 7022
                                            // Channel No.(0 to 3) if module ID is 7024
        wBuf[6] = 0;                        // string debug
        fBuf[0] = 3.5;                       // Analog Value

        wRetVal = AnalogOut(wBuf, fBuf, szSend, szReceive);
        if (wRetVal)
                printf("AO of 7021 Error !, Error Code=%d\n", wRetVal);
        else
                printf("AO of 7021 channel %d = %f \n",i,fBuf[0]);

        //--- Analog Input ----    ****      7017 -- AI    ****
        j = 1;
        wBuf[0] = 3;                        // COM Port
        wBuf[1] = 0x03;                     // Address
        wBuf[2] = 0x7017;           // ID
        wBuf[3] = 0;                        // CheckSum disable
        wBuf[4] = 100;                      // TimeOut , 100 msecond
        wBuf[5] = j;                        // Channel of AI
        wBuf[6] = 0;                        // string debug

        wRetVal = AnalogIn(wBuf, fBuf, szSend, szReceive);
        if (wRetVal)
                printf("AI of 7017 Error !, Error Code=%d\n", wRetVal);
        else
                printf("AI of 7017 channel %d = %f \n",j,fBuf[0]);

        Close_Com(COM3);

        return 0;
}
```

All the steps from programming to execution are the same as those in the section 7.1. The result of execution refers to Fig. 7-10.



```
Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.02.13-09:11+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# i7kaio.exe
AO of 7021 channel 0 = 3.500000
AI of 7012 channel 1 = 3.500000
#
```

Fig. 7-10

7.3 I-87k Modules DIO Control Demo

When using I-87k modules for I/O control of the LinCon-8000, the program will be a little different, according to the location of I-87k modules. There are three conditions for the location of the I-87k modules：

(1) When I-87k DIO modules are **in the LinCon-8000 slots**, the two functions “ Open_Slot ” and “ ChangeToSlot ”, must be added before using other functions for the I-87k modules and the function of “Close_Slot() “ also needs to be added to the end of the program. Please refer to demo in section 7.3.1.

(2) When I-87K DIO modules are **in the I-87k I/O expansion unit slots**, then please refer to the demo in section 7.3.2.

(3) When the I-87k DIO modules are **in the I-8000 controller slots**, then the I-87k modules will be regarded as I-8k modules and so please refer to I/O control of I-8k modules in section 7.5.2

7.3.1 I-87k Modules in slots of LinCon-8000

This demo – **i87kdio.c** will illustrate how to control the DI/DO with the I-87054 module (8 DO channels and 8 DI channels). The I-87054 module is in slot 3 of the LinCon-8000. The address and baudrate in the LinCon-8000 are constant and they are 00 and 115200 respectively. The result of this demo lets DO channel 0 ~ 7 of I-87054 output and DI channel 1 of I-87054 input. The source code of this demo program is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
DWORD dwBuf[12];
float fBuf[12];

/* --------------------------------------------------------------- */
int main()
{
     int i, wRetVal;
     DWORD temp;

     //Check Open_Slot
     wRetVal = Open_Slot(0);
     if (wRetVal > 0) {
           printf("open Slot failed!\n");
           return (-1);
     }

     //Check Open_Com1
     wRetVal = Open_Com(COM1, 115200, Data8Bit, NonParity, OneStopBit);
     if (wRetVal > 0) {
           printf("open port failed!\n");
           return (-1);
     }

     //Choose Slot3
      ChangeToSlot(3);
     //--- digital output ----   **(DigitalOut_87k()**)
     dwBuf[0] = 1;                        // COM Port
     dwBuf[1] = 00;                       // Address
     dwBuf[2] = 0x87054;                  // ID
     dwBuf[3] = 0;                        // CheckSum disable
     dwBuf[4] = 100;                      // TimeOut , 100 msecond
     dwBuf[5] = 0xff;                     // digital output
     dwBuf[6] = 0;                        // string debug
    wRetVal = DigitalOut_87k(dwBuf, fBuf, szSend, szReceive); // DO Output
   printf("DO Value= %u", dwBuf[5]);


   //--- digital Input ----     **(DigitalIn_87k()**)
     dwBuf[0] = 1;                      // COM Port
     dwBuf[1] = 00;                     // Address
     dwBuf[2] = 0x87054;                // ID
     dwBuf[3] = 0;                      // CheckSum disable
     dwBuf[4] = 100;                    // TimeOut , 100 msecond

     dwBuf[6] = 0;                      // string debug
     getch();
     DigitalIn_87k(dwBuf, fBuf, szSend, szReceive);     // DI Input
    printf("DI= %u",dwBuf[5]);

  //--- digital output ----     ** Close DO **
     dwBuf[0] = 1;                      // COM Port
     dwBuf[1] = 00;                     // Address
     dwBuf[2] = 0x87054;                // ID
     dwBuf[3] = 0;                      // CheckSum disable
     dwBuf[4] = 100;                    // TimeOut , 100 msecond
     dwBuf[5] = 0x00;                   // digital output
     dwBuf[6] = 0;                      // string debug
     getch();                          // push any key to continue
    wRetVal = DigitalOut_87k(dwBuf, fBuf, szSend, szReceive);

     Close_Com(COM1);
     Close_SlotAll();
     return 0;
}
```

7.3.2 I-87k Modules in slots of I-87k I/O expansion unit

If the I-87k modules are in the slots of the I-87k I/O expansion unit, the above program needs to be modified in three parts：

(1) The functions of **Open_Slot()** , **ChangeToSlot(), Close_SlotAll()** will be deleted.

(2) The **address** and **baudrate** of I-87k modules in the network of RS-485 need to be set by DCON Utility

(3) **Open com1**(internal serial port of LinCon-8000) will be modified to **open com3** (RS-485 port of LinCon-8000)

The address and baudrate of the I-87054 in the RS-485 network are set to be 06 and 9600 separately by the DCON Utility. The source code of this demo program – **i87kdio_87k.c** is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
DWORD dwBuf[12];
float fBuf[12];

/* ---------------------------------------------------------------- */
int main()
{
      int i, wRetVal;
      DWORD temp;
      //Check Open_Com3
      wRetVal = Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed!\n");
            return (-1);
      }
      //--- digital output ----    **(DigitalOut_87k()**)
      dwBuf[0] = 3;                 // COM Port
      dwBuf[1] = 06;                // Address
      dwBuf[2] = 0x87054;           // ID
      dwBuf[3] = 0;                 // CheckSum disable
      dwBuf[4] = 100;               // TimeOut , 100 msecond
      dwBuf[5] = 0xff;              // digital output
      dwBuf[6] = 0;                 // string debug

     wRetVal = DigitalOut_87k(dwBuf, fBuf, szSend, szReceive);   // DO Output
     printf("DO Value= %u", dwBuf[5]);

     //--- digital Input ----     **(DigitalIn_87k()**)
      dwBuf[0] = 3;                 // COM Port
      dwBuf[1] = 06;                // Address
      dwBuf[2] = 0x87054;           // ID
      dwBuf[3] = 0;                 // CheckSum disable
      dwBuf[4] = 100;               // TimeOut , 100 msecond
      dwBuf[6] = 0;                 // string debug
      getch();

      DigitalIn_87k(dwBuf, fBuf, szSend, szReceive);     // DI Input
      printf("DI= %u",dwBuf[5]);
      //--- digital output ----     ** Close DO **
      dwBuf[0] = 3;                 // COM Port
      dwBuf[1] = 06;                // Address
      dwBuf[2] = 0x87054;           // ID
      dwBuf[3] = 0;                 // CheckSum disable
      dwBuf[4] = 100;               // TimeOut , 100 msecond
      dwBuf[5] = 0x00;              // digital output
      dwBuf[6] = 0;                 // string debug
      getch();                     // push any key to continue
```

```
   wRetVal = DigitalOut_87k(dwBuf, fBuf, szSend, szReceive);

   Close_Com(COM3);
return 0;
}
```

All the steps from programming to execution are the same as those in the section 7.1. The result of execution refers to Fig. 7-11.

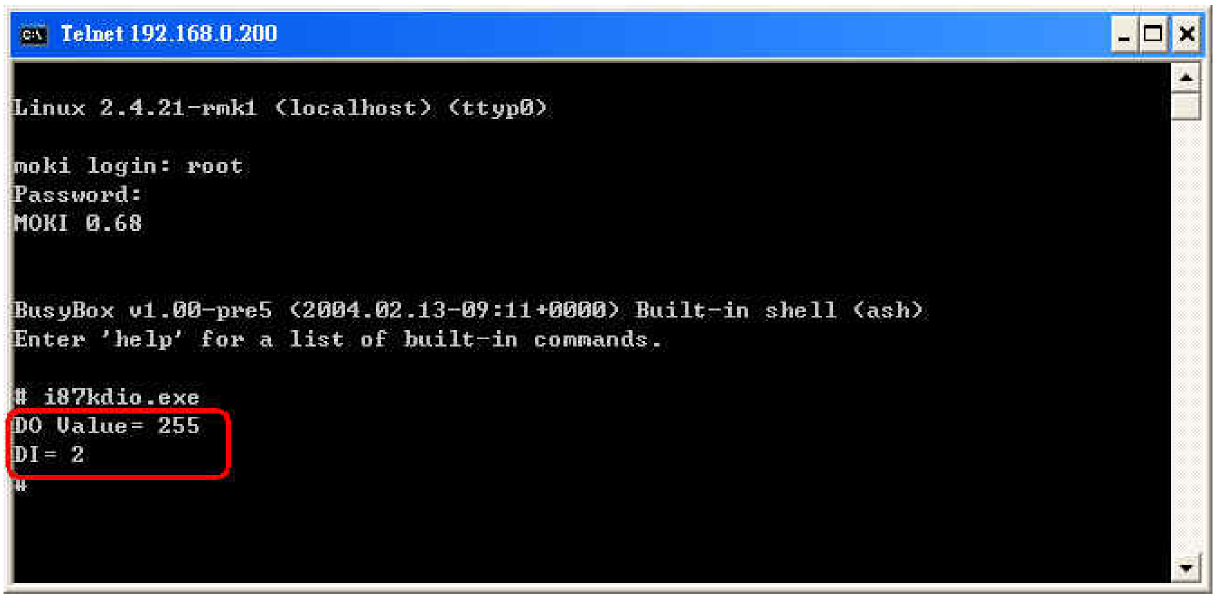

```
Telnet 192.168.0.200

Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.02.13-09:11+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# i87kdio.exe
DO Value= 255
DI= 2
#
```

Fig. 7-11

7.3.3 I-87k Modules in slots of I-8000 Controller

If the I-87k DIO modules are in the I-8000 controller slots, I-87k modules will be regarded as I-8k modules and so please refer to DI/DO control of I-8k modules in the section 7.5.

7.4 I-87k Modules AIO Control Demo

When using I-87k modules for I/O control of the LinCon-8000, according to the location of the I-87k modules, the program will be a little different. There are three conditions for the location of the I-87k modules：

(1) When the I-87k AIO modules are **in the LinCon-8000 slots**, the two functions “ Open_Slot ” and “ ChangeToSlot ” must be added before using the other functions of the I-87k modules and the function “ Close_Slot() “ also needs to be added to the end of the program. Please refer to the demo in section 7.4.1.

(2) When I-87K AIO modules are **in the I-87k I/O expansion unit slots**, please refer to the demo in section 7.4.2.

(3) When the I-87k AIO modules are **in the I-8000 controller slots**, the I-87k modules will be regarded as I-8k modules and so please refer to I/O control of I-8k modules in section 7.4.3

7.4.1 I-87k Modules in slots of LinCon-8000

This demo – **i87kaio.c** will illustrate how to control the AI/AO with the I-87022 module (2 AO channels) and the I-87017 module (8 AI channels).The I-87022 and I-87017 modules are plugged into slot 2 and slot 3 of the LinCon-8000 separately. The address and baudrate in the LinCon-8000 are constant and they are 00 and 115200 separately. The result of this demo lets AO channel 0 of I-87022 output 2.5V and AI channel 1 of I-87017 input. The source code of this demo program is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
DWORD wBuf[12];
DWORD   wBuf7[12];
float fBuf[12];

/* -------------------------------------------------------------- */
int main()
{
      int i,j, wRetVal;
      DWORD temp;

   //Check Open_Slot
   wRetVal = Open_Slot(0);
      if (wRetVal > 0) {
            printf("open Slot failed!\n");
            return (-1);
      }

      //Check Open_Com1
      wRetVal = Open_Com(COM1, 115200, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed!\n");
            return (-1);
      }
```

```
    ChangeToSlot(2);
    //--- Analog output ----   ****    87022 -- AO   ****
    i=0;
    wBuf[0] = 1;              // COM Port
    wBuf[1] = 0x00;           // Address
    wBuf[2] = 0x87022;        // ID
    wBuf[3] = 0;              // CheckSum disable
    wBuf[4] = 100;            // TimeOut , 100 msecond
    wBuf[5] = i;              // Channel Number of AO
    wBuf[6] = 0;              // string debug
    fBuf[0] = 2.5;            // AO Value

    wRetVal = AnalogOut_87k(wBuf, fBuf, szSend, szReceive);
    if (wRetVal)
        printf("AO of 87022 Error !, Error Code=%d\n", wRetVal);
    else
        printf("AO of 87022 channel %d = %f \n",i,fBuf[0]);

    ChangeToSlot(3);
    //--- Analog Input ----   ****    87017 -- AI   ****
    j=1;
    wBuf7[0] = 1;             // COM Port
    wBuf7[1] = 0x00;          // Address
    wBuf7[2] = 0x87017;       // ID
    wBuf7[3] = 0;             // CheckSum disable
    wBuf7[4] = 100;           // TimeOut , 100 msecond
    wBuf7[5] = j;             //Channel Number of AI
    wBuf7[6] = 0;             // string debug

    wRetVal = AnalogIn_87k(wBuf7, fBuf, szSend, szReceive);
    if (wRetVal)
        printf("AI of 87017 Error !, Error Code=%d\n", wRetVal);
    else
        printf("AI of 87017 channel %d = %f \n",j,fBuf[0]);

    Close_Com(COM1);
    Close_SlotAll();
    return 0;
}
```

7.4.2 I-87k Modules in slots of I-87k I/O expansion unit

If the I-87k modules are in slots of I-87k I/O expansion unit, the above program needs to be modified in three parts：

(1) The functions of **Open_Slot()** , **ChangeToSlot(), Close_SlotAll()** will be deleted.

(2) The **addrss** and **baudrate** of I-87k modules in the network of RS-485 need to be

set by DCON Utility

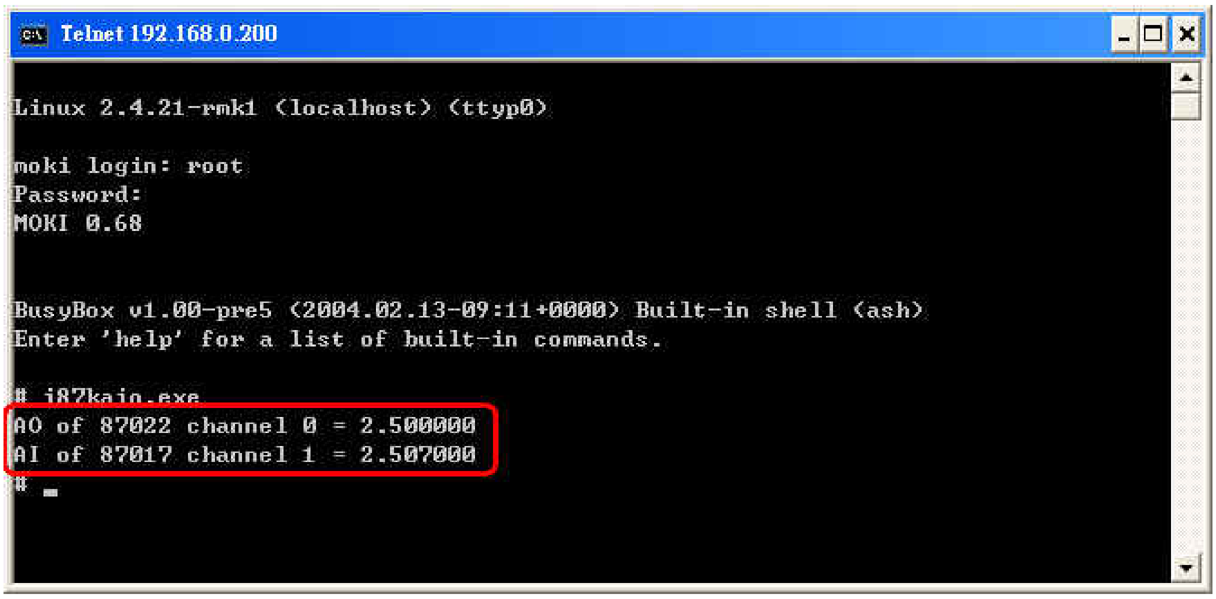
(3) **Open com1**(internal serial port of LinCon-8000) will be modified to **open com3** (RS-485 port of LinCon-8000)

The addresses I-87022 and I-87017 are in the RS-485 network and are set to be 01 and 02 separately and the baudrate is 9600 by DCON Utility. The source code of this demo program – **i87kaio_87k.c** is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
DWORD wBuf[12];
DWORD   wBuf7[12];
float fBuf[12];

/* --------------------------------------------------------------- */
int main()
{
      int i,j, wRetVal;
      DWORD temp;

      //Check Open_Com3
      wRetVal = Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed!\n");
            return (-1);
      }

      //--- Analog output ----   ****    87022 -- AO   ****
      i=0;
      wBuf[0] = 3;                // COM Port
      wBuf[1] = 0x01;             // Address
      wBuf[2] = 0x87022;          // ID
      wBuf[3] = 0;                // CheckSum disable
      wBuf[4] = 100;              // TimeOut , 100 msecond
      wBuf[5] = i;                // Channel Number of AO
      wBuf[6] = 0;                // string debug
      fBuf[0] = 2.5;              // AO Value

      wRetVal = AnalogOut_87k(wBuf, fBuf, szSend, szReceive);
      if (wRetVal)
                  printf("AO of 87022 Error !, Error Code=%d\n", wRetVal);
      else
                  printf("AO of 87022 channel %d = %f \n",i,fBuf[0]);
```

```
    //--- Analog Input ----    ****     87017 -- AI    ****
    j=1;
    wBuf7[0] = 3;                 // COM Port
    wBuf7[1] = 0x02;              // Address
    wBuf7[2] = 0x87017;           // ID
    wBuf7[3] = 0;                 // CheckSum disable
    wBuf7[4] = 100;               // TimeOut , 100 msecond
    wBuf7[5] = j;                 //Channel Number of AI
    wBuf7[6] = 0;                 // string debug

    wRetVal = AnalogIn_87k(wBuf7, fBuf, szSend, szReceive);
    if (wRetVal)
            printf("AI of 87017 Error !, Error Code=%d\n", wRetVal);
    else
            printf("AI of 87017 channel %d = %f \n",j,fBuf[0]);

    Close_Com(COM3);

    return 0;
}
```

All the steps from programming to execution are the same as those in the section 7.1. The result of execution refers to Fig. 7-12.



```
Telnet 192.168.0.200

Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.02.13-09:11+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# i87kaio.exe
AO of 87022 channel 0 = 2.500000
AI of 87017 channel 1 = 2.507000
# _
```

Fig. 7-12

7.4.3 I-87k Modules in slots of I-8000 Controller

If the I-87k AIO modules are in slots of I-8000 controller, I-87k modules will be regarded as I-8k modules and refer to AI/AO control of I-8k modules in the section 7.6.

7.5 I-8k Modules DIO Control Demo

I8000.c of Libi8k.a is the source file for i8k modules in slots of I-8000 controller. **Slot.c** of Libi8k.a is the source file for i8k modules in slots of LinCon-8000. Therefore the functions for i8k modules in slots of LinCon-8000 and in slots of I-8000 controller are different completely. There are two conditions for the location of the I-8k modules：

(1) When I-8K DIO modules are **in the LinCon-8000**, then please refer to the demo in section 7.5.1.

(2) When I-8K DIO modules are **in the I-8000 controller**, then please refer to the demo in section 7.5.2.

7.5.1 I-8k Modules in slots of LinCon-8000

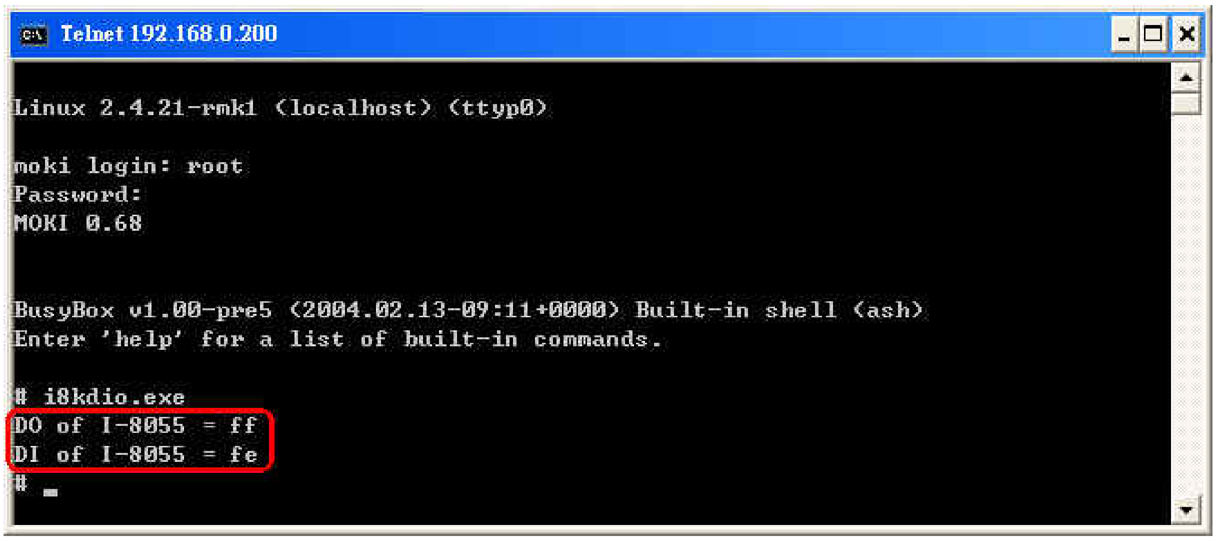
In this section, this demo program – **i8kdio.c** will introduce how to control the DI/DO with the I-8055 (8 DO channels and 8 DI channels) module and it is plugged into slot 3 of the LinCon-8000.

The address and baudrate in the LinCon-8000 are constant and they are 00 and 115200 separately. The result of this demo lets DO channel 0 ~7 of I-8055 output and DI channel 0 of I-8055 input. The source code of this demo program is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
DWORD dwBuf[12];
float fBuf[12];
/* --------------------------------------------------------------- */
int main()
{
      int i,j, wRetVal;
      WORD DOval,temp;

      wRetVal = Open_Slot(3);
      if (wRetVal > 0) {
            printf("open Slot failed!\n");
            return (-1);
      }
```

```
        //I-8055_DO
        DO_8(3,255);
        printf("DO of I-8055 = 0x%x \n", 255);

        //I-8055_DI
        printf("DI of I-8055 = %x",DI_8(3));

        Close_Slot(3);
        return 0;
}
```

All the steps from programming to execution are the same as those in the section 7.1. The result of execution refers to Fig. 7-13.



```
Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.02.13-09:11+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# i8kdio.exe
DO of I-8055 = ff
DI of I-8055 = fe
#
```

Fig. 7-13

7.5.2 I-8k Modules in slots of I-8000 Controller

In this section, this demo program – **i8kdio_8k.c** will illustrate how to control the DI/DO with the I-8055 (8 DO channels and 8 DI channels) module. Please follow the below steps to configure the hardware：

(1) Put the I-8055 module in slot 0 of I-8000 controller.

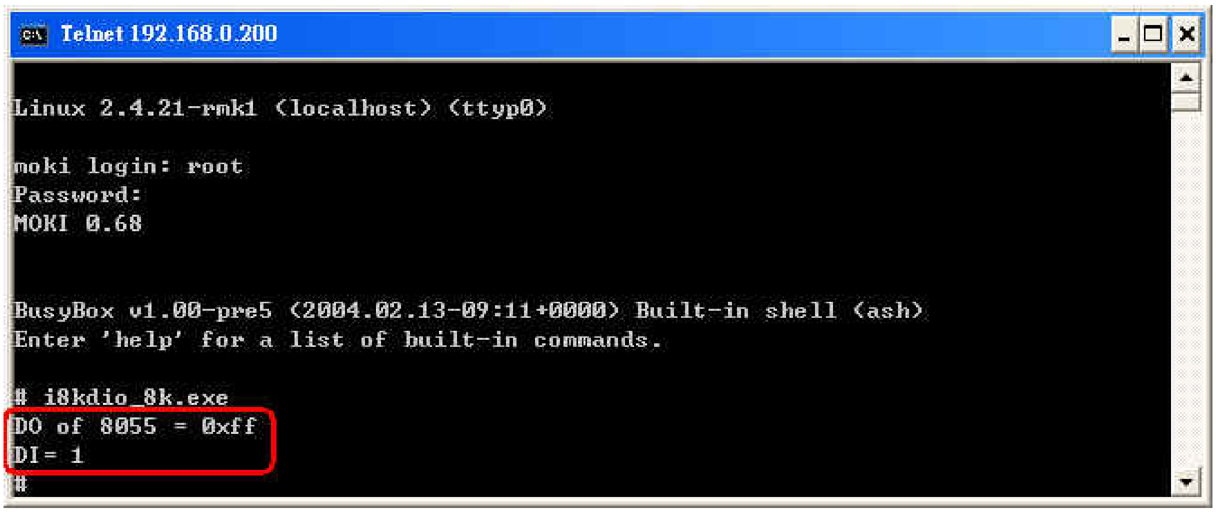
(2) Install 8k232.exe or R232_300.exe to flash memory of I-8000 controller as firmware.

(3) Connect the com2 of LinCon-8000 to the com1 of I-8000 controller with the RS-232 cable.

The address of I-8000 controller is 01 and the baudrate is 115200 that can be modified by DCON Utility. The result of this demo lets DO channel 0 ~7 of I-8055 output and DI channel 0 of I-8055 input. The source code of this demo program is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
DWORD dwBuf[12];
float fBuf[12];

/* --------------------------------------------------------------- */
int main()
{
      int i, wRetVal;
      DWORD temp;

      //Check Open_Com2
      wRetVal = Open_Com(COM2, 115200, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed!\n");
            return (-1);
      }

      //--- digital output ----   **(DigitalOut_8K()**)
      dwBuf[0] = 2;                    // COM Port
      dwBuf[1] = 01;                   // Address
      dwBuf[2] = 0x8055;               // ID
      dwBuf[3] = 0;                    // CheckSum disable
      dwBuf[4] = 100;                  // TimeOut , 100 msecond
      dwBuf[5] = 0xff;                 // digital output
      dwBuf[6] = 0;                    // string debug
      dwBuf[7] = 1;                   // slot number

      wRetVal = DigitalOut_8K(dwBuf, fBuf, szSend, szReceive);
      if (wRetVal)
                  printf("DO of 8055 Error !, Error Code=%d\n", wRetVal);
      else
                  printf("DO of 8055 = 0x%x" ,dwBuf[5]);

   //--- digital Input ----     **(DigitalIn_8K()**)
      dwBuf[0] = 2;                       // COM Port
      dwBuf[1] = 01;                      // Address
      dwBuf[2] = 0x8055;                  // ID
      dwBuf[3] = 0;                       // CheckSum disable
      dwBuf[4] = 100;                     // TimeOut , 100 msecond

      dwBuf[6] = 0;                       // string debug
      dwBuf[7] = 1;                      // slot number

      getch();
      DigitalIn_8K(dwBuf, fBuf, szSend, szReceive);
      printf("DI = %u",dwBuf[5]);
   //--- digital output ----      ** Close DO **
      dwBuf[0] = 2;                    // COM Port
      dwBuf[1] = 01;                   // Address
      dwBuf[2] = 0x8055;               // ID
      dwBuf[3] = 0;                    // CheckSum disable
      dwBuf[4] = 100;                  // TimeOut , 100 msecond
      dwBuf[5] = 0x00;                 // digital output
      dwBuf[6] = 0;                    // string debug
      dwBuf[7] = 1;                   // slot number
      getch();                        // push any key to continue
   wRetVal = DigitalOut_8K(dwBuf, fBuf, szSend, szReceive);

      Close_Com(COM2);
      return 0;
}
```

All the steps from programming to execution are the same as those in the section 7.1. The result of execution refers to Fig. 7-14.



```
Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.02.13-09:11+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# i8kdio_8k.exe
DO of 8055 = 0xff
DI= 1
#
```

Fig. 7-14

7.6 I-8k Modules AIO Control Demo

I8000.c of Libi8k.a is the source file for i8k modules in slots of I-8000 controller. **Slot.c** of Libi8k.a is the source file for i8k modules in slots of the LinCon-8000. Therefore the functions for the i8k modules in LinCon-8000 slots and in the I-8000 controller slots are completely different. There are two conditions for the location of the I-8k modules：

(1) When I-8K AIO modules are **in the LinCon-8000**, then please refer to the demo in section 7.6.1.

(2) When I-8K AIO modules are **in the I-8000 controller**, then please refer to the demo in section 7.6.2.

7.6.1 I-8k Modules in slots of LinCon-8000

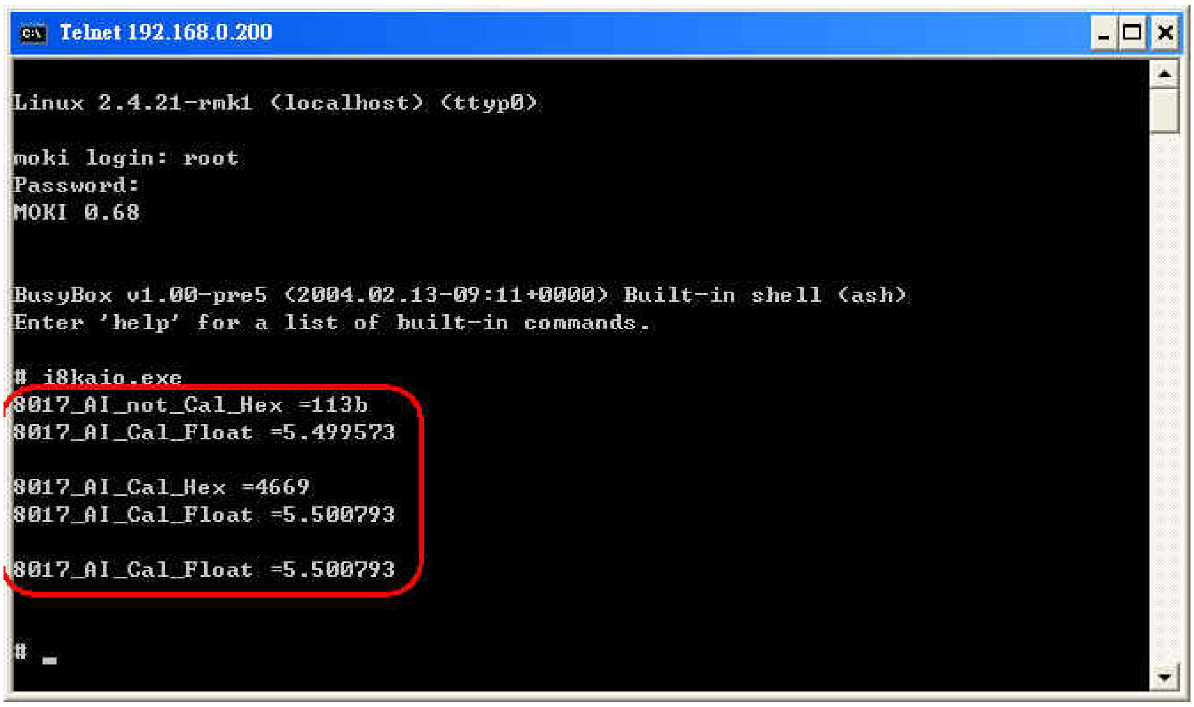
In this section, this demo program – **i8kaio.c** will illustrate how to control the AI/AO with the I-8024 (4 AO channels) and I-8017 (8 AI channels) module and they are in slot 1 and slot 2 of the LinCon-8000 separately.

The address and baudrate in the LinCon-8000 are constant and they are 00 and 115200 separately. The result of this demo lets AO voltage channel 0 of I-8024 output 5.5V and AI channel 2 of I-8017H input. The source code of this demo program is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
DWORD dwBuf[12];
float fBuf[12];

/* ----------------------------------------------------------------- */
int main()
{
    int i, wRetVal,j;
    float fAi;
    int hAi, chAi, Succ;
    int Arr_hAi[5];
    float Arr_fAi[5];

    //I-8024
    wRetVal = Open_Slot(1);
    if (wRetVal > 0) {
        printf("open Slot failed!\n");
        return (-1);
    }

    //I8024 Initial
    I8024_Initial(1);

    //I8024_AO Output
    I8024_VoltageOut(1,0,5.5);
    Close_Slot(1);

    /***************************************/
    //I-8017H
    wRetVal = Open_Slot(2);
    if (wRetVal > 0) {
        printf("open Slot failed!\n");
        return (-1);
    }
```

```
        //I8017H Initial
        I8017_Init(2);
        //I8017H _Channel Setup
        I8017_SetChannelGainMode(2,2,0,0);

        // First Method：Get AI Value
        hAi = I8017_GetCurAdChannel_Hex(2);     //Get Not-calibrated AI Hex Value
        printf("8017_AI_not_Cal_Hex =%x\n",hAi);
        fAi = HEX_TO_FLOAT_Cal(hAi,2,0);          //Not-calibrated AI Hex Value modify to
calibrated AI Float Value
        printf("8017_AI_Cal_Float =%f\n\n",fAi);

        // Second Method：Get AI Value
        hAi = I8017_GetCurAdChannel_Hex_Cal(2);    //Get Calibrated AI Hex Value
        printf("8017_AI_Cal_Hex =%x\n",hAi);
        fAi = CalHex_TO_FLOAT(hAi,0);                       //Calibrated AI Hex Value modify to
Calibrated AI Float Value
        printf("8017_AI_Cal_Float =%f\n\n",fAi);

        // Third Method：Get AI Value
        fAi = I8017_GetCurAdChannel_Float_Cal(2);    //Get Calibrated AI Float Value
        printf("8017_AI_Cal_Float =%f\n\n\n",fAi);

        Close_Slot(2);
        return 0;
}
```

All the steps from programming to execution are the same as those in the section 7.1. The result of execution refers to Fig. 7-15.



```
Telnet 192.168.0.200

Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.02.13-09:11+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# i8kaio.exe
8017_AI_not_Cal_Hex =113b
8017_AI_Cal_Float =5.499573

8017_AI_Cal_Hex =4669
8017_AI_Cal_Float =5.500793

8017_AI_Cal_Float =5.500793


# _
```

Fig. 7-15

7.6.2 I-8k Modules in slots of I-8000 Controller

In this section, this demo program – **i8kaio_8k.c** will introduce how to control the AI/AO with the I-8024 (4 AO channels) and I-8017 (8 AI channels) module and they are plugged into slot 0 and slot 1 of the I-8000 controller separately. Please follow the below steps to configure the hardware：

(1) Put the I-8024 and I-8017 modules in slot 0 and slot 1 of I-8000 controller.

(2) Install 8k232.exe or R232_300.exe to flash memory of I-8000 controller as firmware.

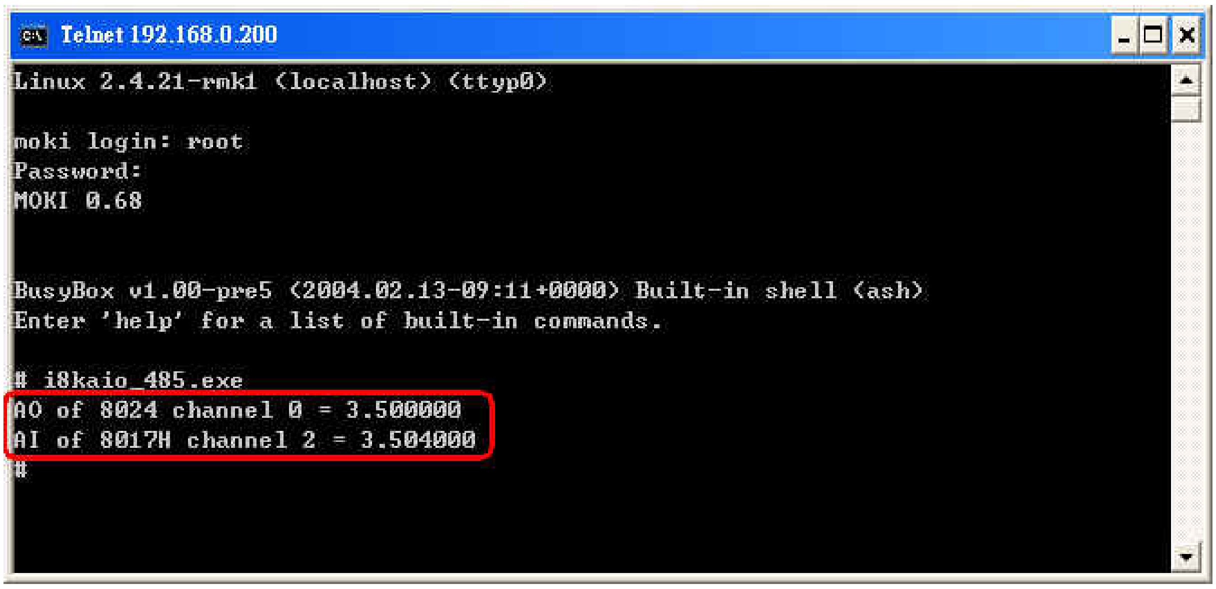
(3) Connect com2 of LinCon-8000 to com1 of I-8000 controller with RS-232 cable.

The address and baudrate of I-8000 controller are 01 and 115200 that can be modified by DCON Utility. The result of this demo lets AO voltage channel 0 of 8024 output 3.5V and AI channel 2 of 8017H input. The source code of this demo program is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
DWORD wBuf[12];
float fBuf[12];

/* --------------------------------------------------------------- */
int main()
{
      int i,j, wRetVal;
      DWORD temp;

      wRetVal = Open_Com(COM2, 115200, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed!\n");
            return (-1);
      }

      //--- Analog output ----   ****    8024 -- AO    ****
      i = 0;
      wBuf[0] = 2;                 // COM Port
      wBuf[1] = 0x01;              // Address
      wBuf[2] = 0x8024;            // ID
      wBuf[3] = 0;                 // CheckSum disable
      wBuf[4] = 100;               // TimeOut , 100 msecond
      wBuf[5] = i;                 // Channel No. of AO
```

```
        wBuf[6] = 0;                  // string debug
        wBuf[7] = 0;                  // Slot Number
        fBuf[0] = 3.5;

        wRetVal = AnalogOut_8K(wBuf, fBuf, szSend, szReceive);
        if (wRetVal)
              printf("AO of 8024 Error !, Error Code=%d\n", wRetVal);
        else
              printf("AO of 8024 channel %d = %f \n",i,fBuf[0]);

        //--- Analog Input ----   ****    8017H -- AI   ****
        j = 2;
        wBuf[0] = 2;                  // COM Port
        wBuf[1] = 0x01;               // Address
        wBuf[2] = 0x8017;             // ID
        wBuf[3] = 0;                  // CheckSum disable
        wBuf[4] = 100;                // TimeOut , 100 msecond
        wBuf[5] = j;                  // Channel of AI
        wBuf[6] = 0;                  // string debug
        wBuf[7] = 1;                  // Slot Number

        wRetVal = AnalogIn_8K(wBuf, fBuf, szSend, szReceive);
        if (wRetVal)
              printf("AI of 8017H Error !, Error Code=%d\n", wRetVal);
        else
              printf("AI of 8017H channel %d = %f \n",j,fBuf[0]);

        Close_Com(COM2);
        return 0;
}
```

All the steps from programming to execution are the same as those in the section 7.1. The result of execution refers to Fig. 7-16


Telnet 192.168.0.200

```
Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.02.13-09:11+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# i8kaio_485.exe
AO of 8024 channel 0 = 3.500000
AI of 8017H channel 2 = 3.504000
#
```

Fig. 7-16

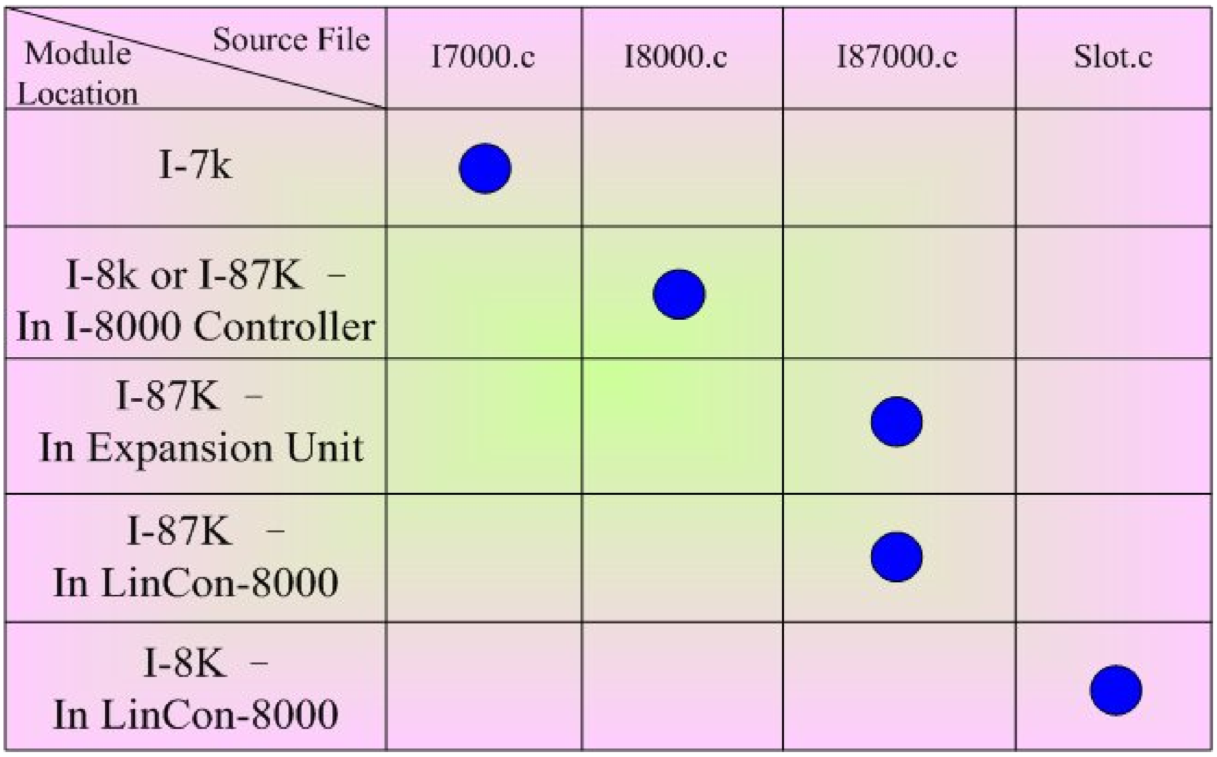
7.7 Conclusion of Module Control Demo

Fig. 7-17 is the table of communication functions for the I-7000/I-8000/I-87000 modules in different locations. When using the ICPDAS modules in the LinCon-8000, this table will be helpful to let users understand which functions of communication should be used.

Module Location / Communication Functions	I-8k – In LinCon-8000	I-87k – In LinCon-8000	I-87k – In Expansion Unit	I-8k or I-87k – In I-8000 Controller	I-7k
Open_Slot( )	✓	✓			
Open_Com( )		✓	✓	✓	✓
Close_Slot( )	✓	✓			
Close_Com( )		✓	✓	✓	✓
ChangeToSlot( )		✓			
sio_open( ) (i-811x and i-814x only)	✓				
sio_close( ) (i-811x and i-814x only)	✓				

Fig. 7-17

Fig. 7-18 is the table of source files for the I-7000/I-8000/I-87000 modules in different locations. When using ICPDAS modules in the LinCon-8000, this table will be helpful to let users understand which source files of the libi8k.a should be called.



Module Location \ Source File	I7000.c	I8000.c	I87000.c	Slot.c
I-7k	●			
I-8k or I-87K – In I-8000 Controller		●		
I-87K – In Expansion Unit			●	
I-87K – In LinCon-8000			●	
I-8K – In LinCon-8000				●

Fig. 7-18

7.8 Timer Function Demo

If users want to use "**Timer**" function in the LinCon-8000, please refer to the demo – timer.c and time2.c in the LinCon SDK -- (C:\cygwin\LinCon8k\examples\common). **timer.c** is for the execution period between 0.5~10 ms (Real-Time) and **timer2.c** is for the execution period more than 10 ms (General). So users need to use different timer function according to their requirement.

8. Introduction of LinCon-8000 Serial Ports

This section describes the function of the three serial ports (RS-232/RS-485 interface) in the LinCon-8000 embedded controller (see Fig 8-1). The information in this section is organized as follows:

- COM1 Port
- COM2 Port
- COM3 Port

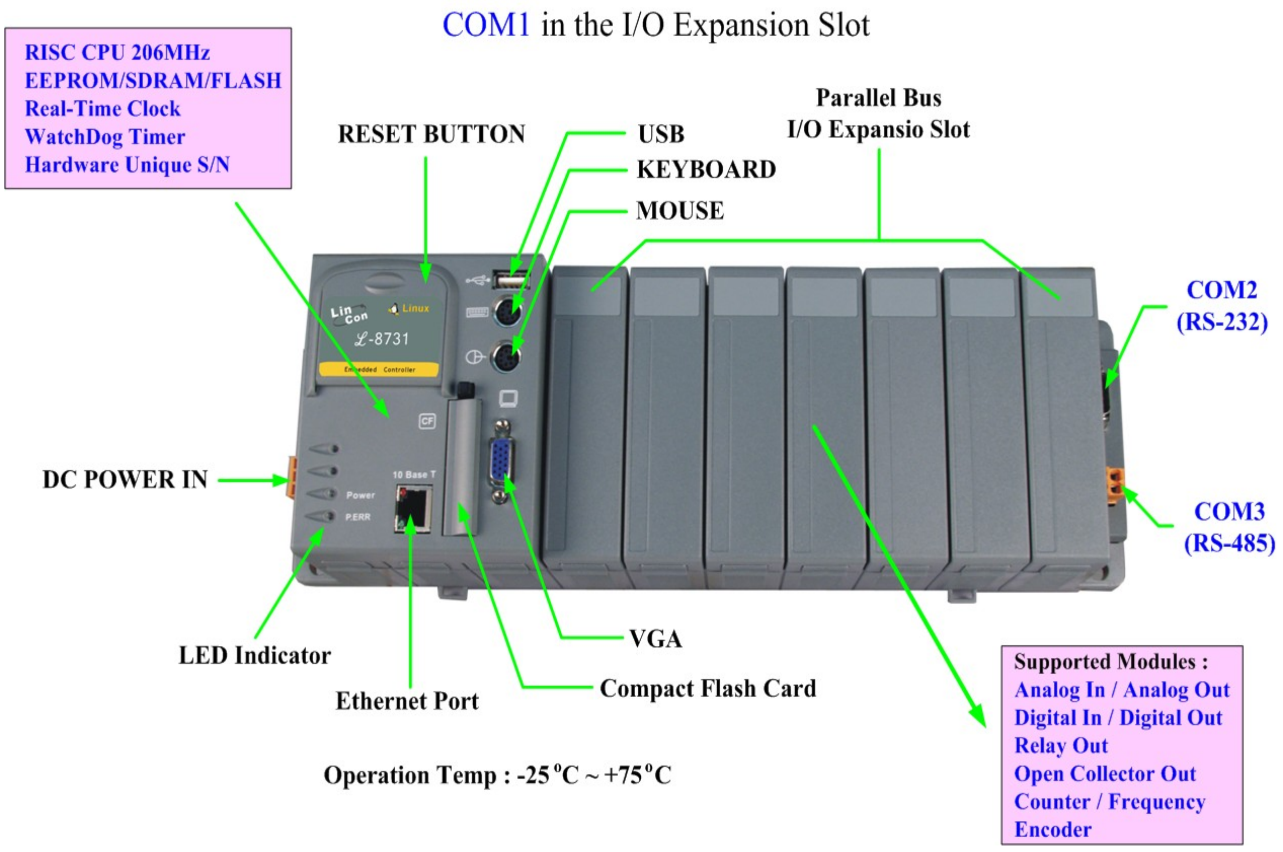


Fig. 8-1

8.1 Introduction of COM1 Port of LinCon-8000

COM1 is an internal I/O expansion port of the LinCon-8000. This port is used to connect the I-87k series module plugged into the LinCon-8000 embedded controller. Users must use the serial command to control the I-87k series module. For controlling the I-87k module, you must input the Com-port parameters and call the Open_Com function to open the com1 port based on the appropriate settings. Finally, the user can call the ChangeToSlot(slot) function in order to change the control slot. This is like the serial address, and you can send out the control commands to the I/O module which is plugged into this slot. Therefore the module’s serial address for its slot is 0. A detailed example is provided below:

For Example:

```
int slot=1;
unsigned char port=1;
DWORD baudrate=115200;    // for all modules in com1 of LinCon-8000
char data=8;
char parity=0;
char stopbit=1;
Open_Slot(slot);
Open_Com(port, baudrate, data, parity, stopbit);
ChangeSlotToI-87k(slot);
// send command...
Close_Com(port);
Close_Slot(slot);
```

8.2 Introduction of COM2 Port of LinCon-8000

This COM2 port is located on the right-upper corner on the LinCon-8000. It is a standard RS-232 serial port, and it provides TxD, RxD, RTS, CTS, GND, non-isolated and a maximum speed of 115.2K bps. It can also connect to the I-7520 module in order to provide a general RS-485 communication. The COM2 port can also connect to a wireless modem so that it can be controlled from a remote device. The application example and code is demonstrated below:

For Example:

```
unsigned char port=2;
DWORD baudrate=9600;
char data=8;
char parity=0;
char stopbit=1;
Open_Com(port, baudrate, data, parity, stopbit);
// send command...
```

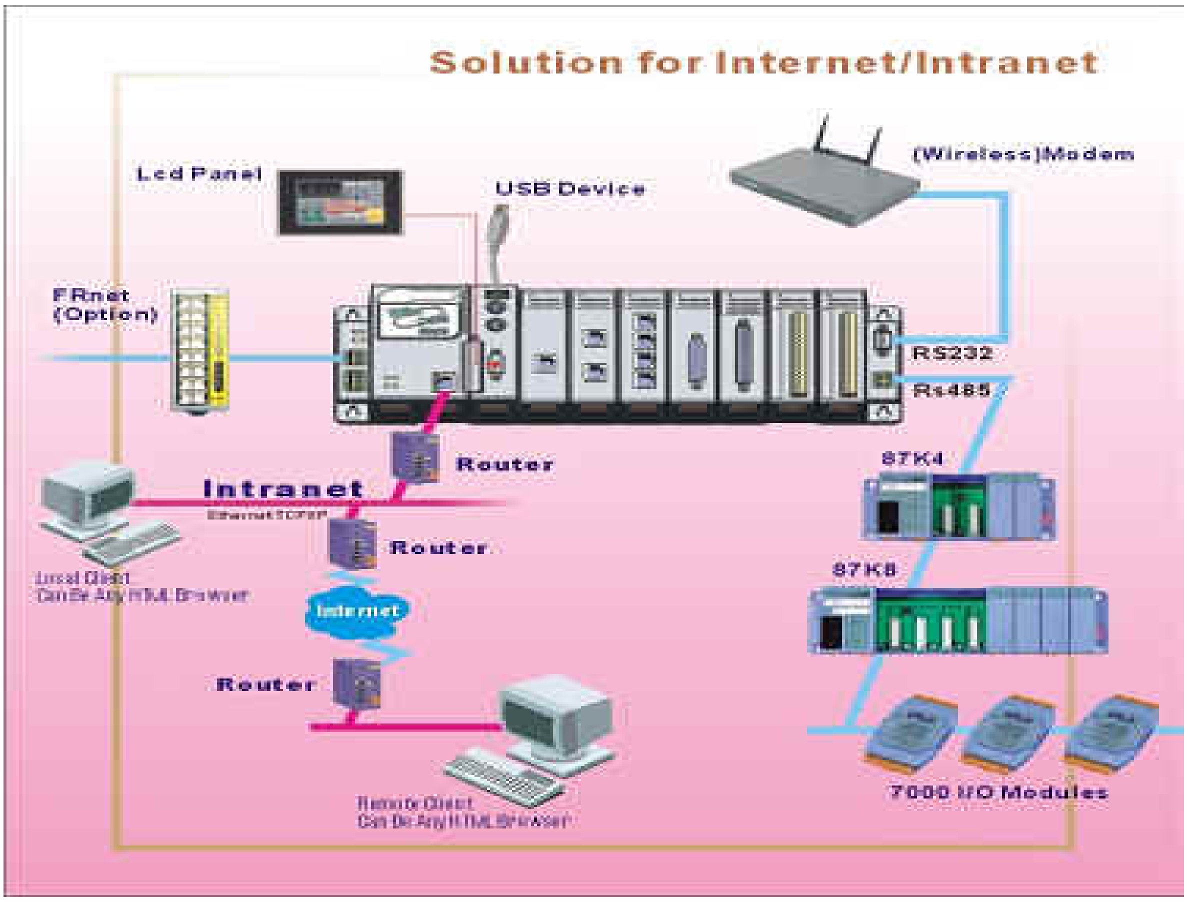


Fig. 8-2

8.3 Introduction of COM3 Port of LinCon-8000

This COM3 port provides RS-485 serial communication (DATA+ and DATA-) and is located on bottom-right corner in the LinCon-8000. You can connect to the RS-485 device with modules like the <u>I-7000, I-87k and I-8000 serial modules</u>(DCON Module) via this port. That is, you can control the ICP DAS I-7000/I-87k/I-8000 series modules directly from this port with any converter. ICP DAS will provide very easy to use functions with libi8k.a and then you can easily handle the I-7000/I-87k/I-8000 series modules. Below is an example of the program code demo.

For Example:

```
unsigned char port=3;
DWORD baudrate=9600;
char data=8;
char parity=0;
char stopbit=1;
Open_Com(port, baudrate, data, char parity, stopbit);
// send command …
```

9. LinCon-8000 Library Reference in C Language

In this chaper, all the functions of **libi8k** will be listed to allow users to able to look them up quickly.

9.1 List Of System Information Functions

```
int Open_Slot(int slot)
void Close_Slot(int slot)
int Open_Slot(void)
void Close_SlotAll(void)
void ChangeToSlot(char slot)
WORD Open_Com(char port, DWORD baudrate, char cData, char cParity, char cStop)
BOOL Close_Com(char port)
WORD Send_Receive_Cmd (char port, char szCmd[ ], char szResult[ ], WORD wTimeOut,
                       WORD wChksum, WORD *wT)
WORD Send_Cmd (char port, char szCmd[ ], WORD wTimeOut, WORD wChksum)
WORD Receive_Cmd (char port, char szResult[ ], WORD wTimeOut, WORD wChksum)
WORD Send_Binary(char port, char szCmd[ ], int iLen)
WORD Receive_Binary(char cPort, char szResult[], WORD wTimeOut, WORD wLen,
                    WORD *wT)
int sio_open(int slot)
int sio_close(int slot)
int GetModuleType(char slot)
int GetNameOfModule(char slot)
void Read_SN(unsigned char serial_num[ ])
```

9.2 List Of Digital Input/Output Functions

9.2.1 For I-8000 modules only

```
void DO_8(int slot, unsigned char data)
void DO_16(int slot, unsigned int data)
void DO_32(int slot, unsigned int data)
unsigned char DI_8(int slot)
unsigned int DI_16(int slot)
```

unsigned long DI_32(int slot)

void DIO_DO_8(int slot, unsigned char data)

void DIO_DO_16(int slot, unsigned int data)

unsigned char DIO_DI_8(int slot)

unsigned char DIO_DI_16(int slot)

unsigned short UDIO_WriteConfig_16

unsigned short UDIO_ReadConfig_16

void UDIO_DO16(int slot, unsigned short config)

unsigned short UDIO_DI16(int slot)

9.2.2 For I-7000/8000/87000 modules via serial port

9.2.2.1 For I-7000 modules

WORD DigitalOut(WORD wBuf[], float fBuf[], char szSend[], char szReceive[])

WORD DigitalBitOut(WORD wBuf[ ], float fBuf[ ], char szSend[ ], char szReceive[ ])

WORD DigitalOutReadBack(WORD wBuf[ ], float fBuf[ ],char szSend[ ], char szReceive[ ])

WORD DigitalOut_7016(WORD wBuf[], float fBuf[], char szSend[],char szReceive[])

WORD DigitalIn(WORD wBuf[], float fBuf[], char szSend[], char szReceive[])

WORD DigitalInLatch(WORD wBuf[], float fBuf[], char szSend[], char szReceive[])

WORD ClearDigitalInLatch(WORD wBuf[], float fBuf[],char szSend[],char szReceive[])

WORD DigitalInCounterRead(WORD wBuf[], float fBuf[], char szSend[],char szReceive[])

WORD ClearDigitalInCounter(WORD wBuf[], float fBuf[],char szSend[],char szReceive[])

WORD ReadEventCounter(WORD wBuf[], float fBuf[],char szSend[],char szReceive[])

WORD ClearEventCounter(WORD wBuf[], float fBuf[], char szSend[],char szReceive[])

9.2.2.2 For I-8000 modules

WORD DigitalOut_8K(DWORD dwBuf[], float fBuf[],char szSend[],char szReceive[])

WORD DigitalBitOut_8K(DWORD dwBuf[], float fBuf[],char szSend[],char szReceive[])

WORD DigitalIn_8K(DWORD dwBuf[], float fBuf[], char szSend[], char szReceive[])

WORD DigitalInCounterRead_8K(DWORD dwBuf[], float fBuf[], char szSend[],
char szReceive[])

WORD ClearDigitalInCounter_8K(DWORD dwBuf[], float fBuf[],char szSend[],
char szReceive[])

WORD DigitalInLatch_8K(DWORD dwBuf[], float fBuf[], char szSend[],char szReceive[])

WORD ClearDigitalInLatch_8K(DWORD dwBuf[], float fBuf[], char szSend[],
char szReceive[])

9.2.2.3 For I-87000 modules

WORD DigitalOut_87K(DWORD dwBuf[], float fBuf[], char szSend[],char szReceive[])

WORD DigitalOutReadBack_87K(DWORD dwBuf[], float fBuf[], char szSend[],
char szReceive[])

WORD DigitalBitOut_87K(DWORD dwBuf[], float fBuf[], char szSend[],char szReceive[])
WORD DigitalIn_87K(DWORD dwBuf[], float fBuf[],char szSend[], char szReceive[])
WORD DigitalInLatch_87K(DWORD dwBuf[], float fBuf[], char szSend[],char szReceive[])
WORD ClearDigitalInLatch_87K(DWORD dwBuf[], float fBuf[], char szSend[],
                                        char szReceive[])
WORD DigitalInCounterRead_87K(DWORD dwBuf[], float fBuf[], char szSend[],
                                        char szReceive[])
WORD ClearDigitalInCounter_87K(DWORD dwBuf[], float fBuf[], char szSend[],
                                        char szReceive[])

9.3 List Of Watch Dog Timer Functions

void EnableWDT(unsigned int msecond)
void DisableWDT(void)
unsigned int WatchDogSWEven(void)
void ClearWDTSWEven(unsigned int rcsr)

9.4 List Of EEPROM Read/Write Functions

void Enable_EEP(void)
void Disable_EEP(void)
unsigned char Read_EEP(int block, int offset)
void Write_EEP(int block, int offset, unsigned char data)

9.5 List Of Analog Input Functions

9.5.1 For I-8000 modules via parallel port

int I8017_Init(int slot)
void I8017_SetLed(int slot, unsigned int led)
void I8017_SetChannelGainMode (int slot, int ch, int gain, int mode)
int I8017_GetCurAdChannel_Hex (int slot)
int I8017_AD_POLLING(int slot, int ch, int gain, unsigned int datacount, int *DataPtr)
float HEX_TO_FLOAT_Cal(int HexValue, int slot, int gain)
void ARRAY_HEX_TO_FLOAT_Cal(int *HexValue, float *FloatValue, int slot,
                                        int gain,int len)
int I8017_Hex_Cal(int data)

int I8017_Hex_Cal_Slot_Gain(int slot, int gain, int data)

float CalHex_TO_FLOAT(int HexValue,int gain)

void ARRAY_CalHex_TO_FLOAT(int *HexValue, float *FloatValue, int gain, int len)

int I8017_GetCurAdChannel_Hex_Cal(int slot)

int I8017_AD_POLLING_Cal(int slot, int ch, int gain, unsigned int datacount, int *DataPtr)

int I8017_GetCurAdChannel_Float_Cal(int slot)

9.5.2 For I-7000/8000/87000 modules via serial port

9.5.2.1 For I-7000 modules

WORD AnalogIn(wBuf, fBuf, szSend, szReceive)

WORD AnalogInHex(wBuf, fBuf, szSend, szReceive)

WORD AnalogInFsr (wBuf, fBuf, szSend, szReceive)

WORD AnalogInAll (wBuf, fBuf, szSend, szReceive)

WORD ThermocoupleOpen_7011(wBuf, fBuf, szSend, szReceive)

WORD SetLedDisplay (wBuf, fBuf, szSend, szReceive)

WORD GetLedDisplay (wBuf, fBuf, szSend, szReceive)

9.5.2.2 For I-8000 modules

WORD AnalogIn_8K(dwBuf, fBuf, szSend, szReceive)

WORD AnalogInHex_8K(dwBuf, fBuf, szSend, szReceive)

WORD AnalogInFsr_8K(dwBuf, fBuf, szSend, szReceive)

WORD AnalogInAll_8K(dwBuf, fBuf, szSend, szReceive)

9.5.2.3 For I-87000 modules

WORD AnalogIn_87K(dwBuf, fBuf, szSend, szReceive)

WORD AnalogInHex_87K(dwBuf, fBuf, szSend, szReceive)

WORD AnalogInHex_87K(dwBuf, fBuf, szSend, szReceive)

WORD AnalogInAll_87K(dwBuf, fBuf, szSend, szReceive)

9.6 List Of Analog Output Functions

9.6.1 For I-8000 modules via parallel port

void I8024_Initial(int slot)

void I8024_VoltageOut(int slot, int ch, float data)

void I8024_CurrentOut(int slot, int ch, float cdata)

void I8024_VoltageHexOut(int slot, int ch, int hdata) void I8024_CurrentHexOut(int slot, int ch, int hdata)

9.6.2 For I-7000/8000/87000 modules via serial port

9.6.2.1 For I-7000 modules

WORD AnalogOut(wBuf, fBuf, szSend, szReceive);

WORD AnalogOutReadBack(wBuf, fBuf, szSend, szReceive)

WORD AnalogOutHex(wBuf, fBuf, szSend, szReceive)

WORD AnalogOutFsr(wBuf, fBuf, szSend, szReceive)

WORD AnalogOutReadBackHex(wBuf, fBuf, szSend, szReceive)

WORD AnalogOutReadBackFsr(wBuf, fBuf, szSend, szReceive)

9.6.2.2 For I-8000 modules

WORD AnalogOut_8K(dwBuf, fBuf, szSend, szReceive)

WORD AnalogOutReadBack_8K(dwBuf, fBuf, szSend, szReceive)

WORD ReadConfigurationStatus_8K(dwBuf, fBuf, szSend, szReceive)

WORD SetStartUpValue_8K(dwBuf, fBuf, szSend, szReceive)

WORD ReadStartUpValue_8K(dwBuf, fBuf, szSend, szReceive)

9.6.2.3 For I-87000 modules

WORD AnalogOut_87K(dwBuf, fBuf, szSend, szReceive)

WORD AnalogOutReadBack_87K(dwBuf, fBuf, szSend, szReceive)

WORD AnalogOutHex_87K(wBuf, fBuf, szSend, szReceive)

WORD AnalogOutFsr_87K(wBuf, fBuf, szSend, szReceive)

WORD AnalogOutReadBackHex_87K(dwBuf, fBuf, szSend, szReceive)

WORD AnalogOutReadBackFsr_87K(dwBuf, fBuf, szSend, szReceive)

WORD ReadConfigurationStatus_87K(dwBuf, fBuf, szSend, szReceive)

WORD SetStartUpValue_87K(dwBuf, fBuf, szSend, szReceive)

WORD ReadStartUpValue_87K(dwBuf, fBuf, szSend, szReceive)

9.7 List Of 3-axis Encoder Functions

```
unsigned char i8090_REGISTRATION(unsigned char slot, unsigned int address)
void i8090_INIT_CARD(unsigned char cardNo, unsigned char x_mode,
                     unsigned char y_mode, unsigned char z_mode)
unsigned int i8090_GET_ENCODER(unsigned char cardNo, unsigned char axis)
```

```
void i8090_RESET_ENCODER(unsigned char cardNo, unsigned char axis)
long i8090_GET_ENCODER32(unsigned char cardNo,unsigned char axis)
void i8090_RESET_ENCODER32(unsigned char cardNo, unsigned char axis)
unsigned char i8090_GET_INDEX(unsigned char cardNo)
void i8090_ENCODER32_ISR(unsigned char cardNo)
```

9.8 List Of 2-axis Stepper/Servo Functions

```
unsigned char i8091_REGISTRATION(unsigned char cardNo, int slot)
void i8091_RESET_SYSTEM(unsigned char cardNo)
void i8091_SET_VAR(unsigned char cardNo, unsigned char DDA_cycle,
                   unsigned char Acc_Dec, unsigned int Low_Speed,
                   unsigned int High_Speed)
void i8091_SET_DEFDIR(unsigned char cardNo, unsigned char defdirX,
                      unsigned char defdirY)
void i8091_SET_MODE(unsigned char cardNo, unsigned char modeX,
                    unsigned char modeY)
void i8091_SET_SERVO_ON(unsigned char cardNo, unsigned char sonX,
                        unsigned char sonY)
void i8091_SET_NC(unsigned char cardNo, unsigned char sw)
void i8091_STOP_X(unsigned char cardNo)
void i8091_STOP_Y(unsigned char cardNo)
void i8091_STOP_ALL(unsigned char cardNo)
void i8091_EMG_STOP(unsigned char cardNo)
void i8091_LSP_ORG(unsigned char cardNo, unsigned char DIR, unsigned char AXIS)
void i8091_HSP_ORG(unsigned char cardNo, unsigned char DIR, unsigned char AXIS)
void i8091_LSP_PULSE_MOVE(unsigned char cardNo, unsigned char AXIS, long pulseN)
void i8091_HSP_PULSE_MOVE(unsigned char cardNo, unsigned char AXIS, long pulseN)
void i8091_LSP_MOVE(unsigned char cardNo, unsigned char DIR, unsigned char AXIS)
void i8091_HSP_MOVE(unsigned char cardNo, unsigned char DIR, unsigned char AXIS)
void i8091_CSP_MOVE(unsigned char cardNo, unsigned char dir, unsigned char axis,
                    unsigned int move_speed)
void i8091_SLOW_DOWN(unsigned char cardNo, unsigned char AXIS)
void i8091_SLOW_STOP(unsigned char cardNo, unsigned char AXIS)
void i8091_INTP_PULSE(unsigned char cardNo, int Xpulse, int Ypulse)
void i8091_INTP_LINE(unsigned char cardNo, int Xpulse, int Ypulse)
void i8091_INTP_LINE02(unsigned char cardNo, long x, long y, unsigned int speed,
                                        unsigned char acc_mode)
void i8091_INTP_CIRCLE02(unsigned char cardNo, long x, long y, unsigned chat dir,
                                            unsigned int speed, unsigned char acc_mode)
void i8091_INTP_ARC02(unsigned char cardNo, long x, long y, long R, unsigned char dir,
                                        unsigned int speed, unsigned char acc_mode)
unsigned char i8091_INTP_STOP(void)
unsigned char i8091_LIMIT_X(unsigned char cardNo)
unsigned char i8091_LIMIT_Y(unsigned char cardNo)
void i8091_WAIT_X(unsigned char cardNo)
void i8091_WAIT_Y(unsigned char cardNo)
unsigned char i8091_IS_X_STOP(unsigned char cardNo)
unsigned char i8091_IS_Y_STOP(unsigned char cardNo)
```

10. Demo of LinCon-8000 Modules With Java Language

Java is an independent operating system platform for software development. It consists of a programming language, utility programs and a run time environment (JVM). A Java program can be developed on one computer that must be installed with the JDK and run on any other computer with the correct run time environment. This language is freely available to the public to use on anything from personal Web sites to corporate enterprise systems to NASA spacecraft. Java was written from the ground up to be a clean, safe, secure, and object-orientated programming language. Therefore **JDK for linux** is also installed in the LinCon-8000, so that users can compile and run Java programs in it. In the meanwhile, we provide the **Java I/O Driver** – **JIOD** to allow users to be able to control I/O modules of ICPDAS with Java language.

10.1 JIOD and JAVA

Java I/O Driver (JIOD) is the Java platform technology of choice for extending and enhancing JVM to make many industry control applications possible. JIOD includes I/O packages for I-7k,I-8k and I-87k remote I/O modules and PCI bus series add-on cards. JIOD provides developers with a simple and easy mechanism for extending the functionality of JVM and accessing ICPDAS products.

The **JIOD** contains three packages namely：**com.icpdas.ixpio, com.icpdas.ixpci** and **com.icpdas.comm** which will support variant ICPDAS's various products. The three pakcages are all included in **icpdas.jar** and their functionalities are listed below：

Packages Summary	
com.icpdas.comm	For i7k,i87,i87k series remote I/O modules
com.icpdas.ixpci	For PCI series add-on cards
com.icpdas.ixpio	For PIO series add-on cards

These packages in the JIOD are easy to understand. they provide powerful, easy-to-use packages for developing your data acquisition application. The program can use these packages within applications, applets and servlets with ease. To speed-up your developing process, some demonstration source programs are provided. The relationship between the JIOD and the user's application is depicted in Fig 10-1.

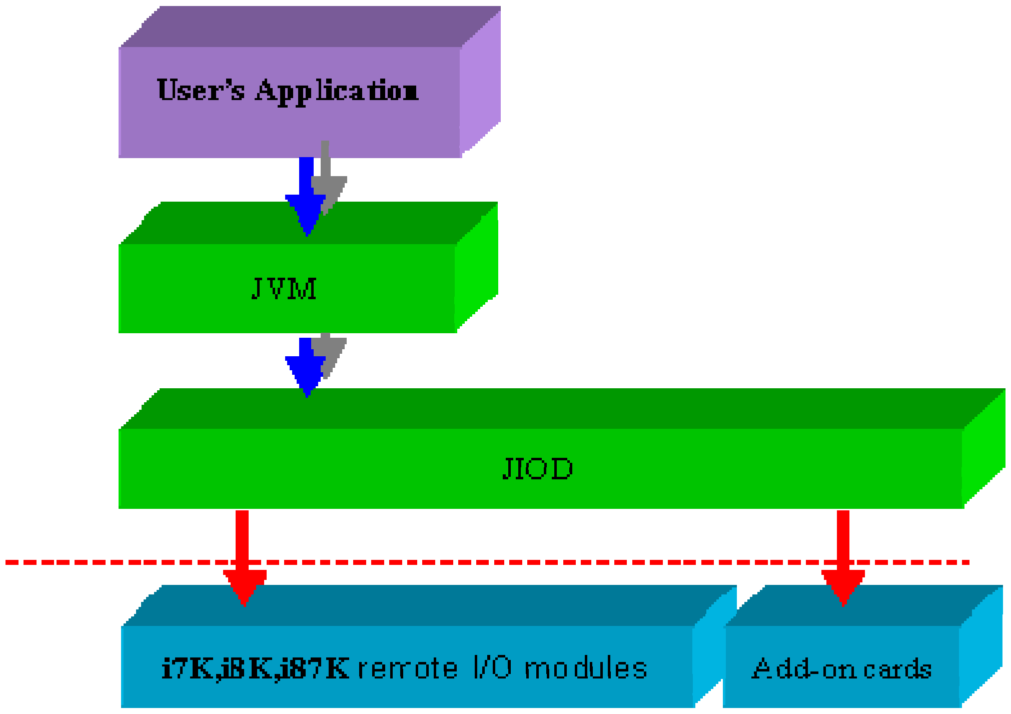


Fig 10-1

10.2 How to Use JIOD

If you want to use Java language to write programs to control ICPDAS products, you must first install the JIOD driver. In order to download the JIOD driver or get more information about it, you are welcome to visit our web site at either - http://w3.icpdas.com/moki/ or http://www.icpdas.com/download/index.htm

Using JIOD is very similar to that for C language users because **icpdas.jar** for Java is just like **libi8k.a** for C. The key points for this are provided below：

1. After you install the JIOD driver, you will see the file **- icpdas.jar**. Then please add the icpdas.jar path to the environment variable - **CLASSPATH**. For more detailed steps relating to this, you can refer to the documents on the web site - **http://w3.icpdas.com/moki/.**

2. Import JIOD into your source program.

10.3 Introduction of com.icpdas.comm Package

Here we just introduce the package - **com.icpdas.comm** of JIOD**.** The com.icpdas.comm package can be used to write platform-independent industry applications. We provides the necessary classes to control ICPDAS remote modules – i7k, i87k, i8k. The command set for these modules will be compatible to ADAM, Nudam, and 6B of Analog Devices in the future.

(ICPDAS remote I/O modules)

10.3.1 Class of com.icpdas.comm

```
java.lang.Object
  |
  +--com.icpdas.comm.Comm
```

The Comm Class includes both the low level serial communication method and the high level remote I/O modules control method. A serial port can be opened for reading and writing data. Once the application is done with the port, it must call the close method before ending the program.

Class Summary	
Comm	Defines methods for communication to serial devices
IoBuf	Remote modules control matrix.

10.3.2 Class Comm

Method Summary	
int	**open**(int portno, int baudrate, int databit, int isparity, int stopbit) Initialize the COM port.
void	**close**(int portno) Free all the resources used by open. This method must be **called before** the program exit.
int	**setSendCmd**(IoBuf ioArg) Create a thread to send a command to module.
int	**getReceiveCmd**(IoBuf ioArg) Create a thread to receive the response-result from module.
int	**getSendReceiveCmd**(IoBuf ioArg) Create a thread to send a command and receive the response-result from module.
int	**getAnalogIn**(IoBuf ioArg) Read the analog input value from module.
int	**getAnalogInAll**(IoBuf ioArg) Read all channels of analog input values from module.
int	**setAnalogOut**(IoBuf ioArg) Send the analog output command to module.
int	**getAnalogOutReadBack**(IoBuf ioArg) Read back the current D/A output value of module.
int	**getDigitalIn**(IoBuf ioArg) Read the digital input value from module.
int	**setDigitalOut**(IoBuf ioArg) To set the digital output value for module.
int	**getDigitalOutReadBack**(IoBuf ioArg) Read back the digital output value of module.
int	**setDigitalBitOut**(IoBuf ioArg) Set the digital value of digital output channel No.
int	**getDigitalInLatch**(IoBuf ioArg) Obtain the latch value of the high or low latch mode of Digital Input module.

int	**setClearDigitalInLatch**(IoBuf ioArg) Clear the latch status of digital input module when latch function has been enabled.
int	**getDigitalInCounter**(IoBuf ioArg) Obtain the counter event value of the channel number of Digital Input module.
int	**setClearDigitalInCounter**(IoBuf ioArg) Clear the counter value of the digital input channel No.
int	**setAlarmMode**(IoBuf ioArg) To set module enter *momentary alarm mode* or *latch alarm mode*.
int	**setAlarmConnect**(IoBuf ioArg) Set the link between DO and AI module.
int	**setClearLatchAlarm**(IoBuf ioArg) To clear the latch alarm for module.
int	**setAlarmLimitValue**(IoBuf ioArg) To set a high or low alarm limit value for module.
int	**getAlarmLimitValue**(IoBuf ioArg) To get the high or low alarm limit value for module.
int	**getAlarmStatus**(IoBuf ioArg) Reading the alarm status for a module.
int	**getAlarmMode**(IoBuf ioArg) Reading the alarm mode for a module.
int	**getConfigStatus**(IoBuf ioArg) Obtain the configuration status of the modules.
int	**setStartUpValue**(IoBuf ioArg) Configure the initial analog output of analog output module when its power is on.
int	**getStartUpValue**(IoBuf ioArg) Obtain the initial output setting value of analog output module when the power is on.

Field Summary

DATABITS_5, DATABITS_6, DATABITS_7, DATABITS_8, PARITY_EVEN, PARITY_NONE, PARITY_ODD, STOPBITS_1, STOPBITS_1_5, STOPBITS_2

10.3.3 Class IoBuf

java.lang.Object
   |
   +--com.icpdas.comm.IoBuf

The IoBuf class provides a variable that the high level remote I/O modules control method use.

Field Summary
dwBuf, **fBuf**, **szSend**, **szReceive**

Field Detail

dwBuf

public int dwBuf[]

Double word length matrix for module control.

fBuf

public float fBuf[]

Floating matrix for module control.

szSend

public java.lang.String szSend

Send a Command string to the module.

szReceive

public java.lang.String szSend

Receives a string from the module.

10.4 I-7k Modules DIO Control Demo

This demo – **i7kdio.java** will illustrate how to control the DI/DO with the I-7065 module (5 DO channels and 4 DI channels). The address and baudrate of I-7065 module in the RS-485 network are 01 and 9600 separately.

The result of this demo lets DO channel 0 ~ 5 output and DI channel 2 input. The source code of this demo program is as follows：

```
import java.io.*;                           //For System.in.read()
import com.icpdas.comm.*;                   //ICPDAS communication packages

public class i7kdio
{
    public static void main(String[] args) throws java.io.IOException
    {
        int rev;
        int fd;
        byte a[] = new byte[100];
        Comm comm1 = new Comm();            //ICPDAS communication object
        IoBuf i7kBuf = new IoBuf();         //control matrix

        rev = comm1.open(3,9600,comm1.DATABITS_8,comm1.PARITY_NONE,
                         comm1.STOPBITS_1); //open serial port

        if (rev!=0) System.out.println("Open port error code : "+rev);
        else{
            i7kBuf.dwBuf[0] = 3;            //Serial Port no
            i7kBuf.dwBuf[1] = 1;            //Address
            i7kBuf.dwBuf[2] = 0x7065 ;      //0x7060; //module name
            i7kBuf.dwBuf[3] = 0;            //check sum disable
            i7kBuf.dwBuf[4] = 100;          //Timeout 100ms
            i7kBuf.dwBuf[5] = 31;           //DO Value
            i7kBuf.dwBuf[6] = 1;            //Enable String Debug
            rev = comm1.setDigitalOut(i7kBuf);    //DO 7065 module
            System.out.println("DO Send = "+ i7kBuf.szSend);
            if (rev!=0) System.out.println("Digital Out Error Code : "+ rev+"\n");

            rev = comm1.getDigitalIn(i7kBuf); //Digital Input to i7065 module
            System.out.println("DI Send = "+ i7kBuf.szSend+"\n");
            if (rev!=0) System.out.println("Digital In Error Code : "+ rev);
            else System.out.println("DI ACK : "+ i7kBuf.dwBuf[5]);
        }
        comm1.close(3);
    }
}
```

Follow the below steps to achieve the desiredresult：

STEP 1：**(Write i7kdio.java)**

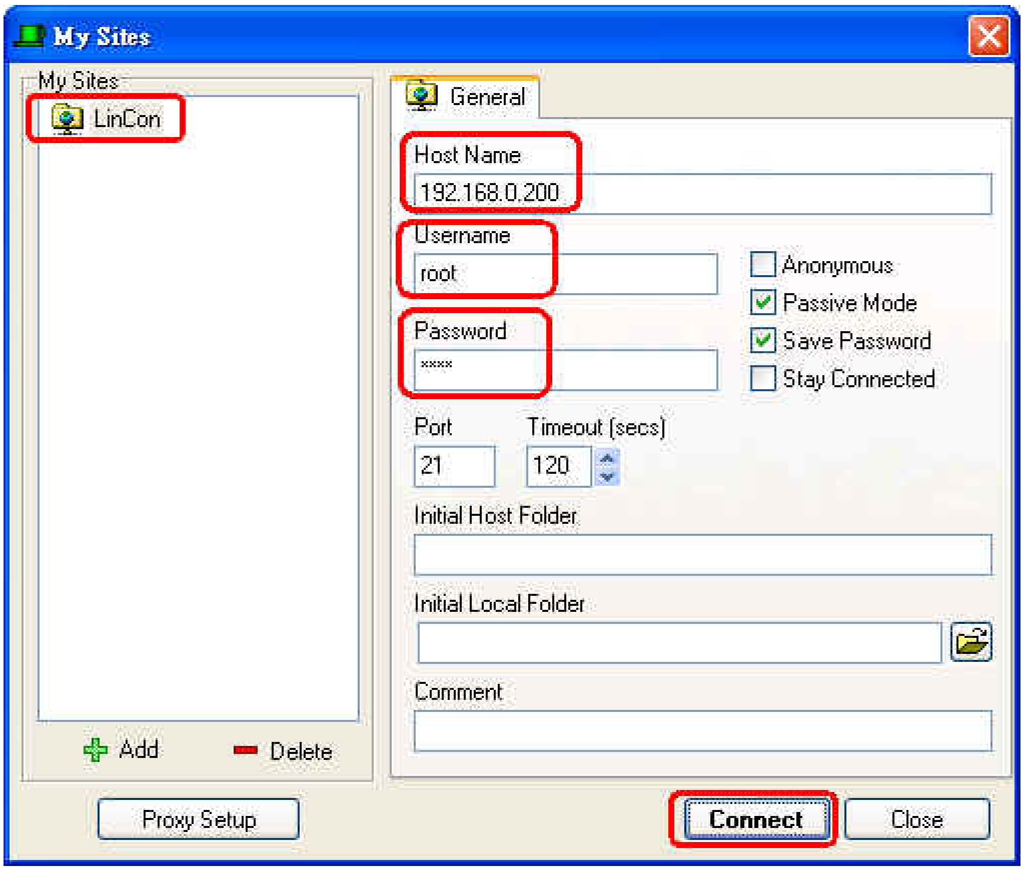
Copy the above source code and save it with the name **- i7kdio.java**. Remember! the file name – i7kdio.java must be the same as the class name. Or get the file from C:\cygwin\LinCon8k\examples\Java\i7k.

STEP 2：**(Transfer i7kdio.java to LinCon-8000)**

Here we will introduce two methods to accomplish step 2.

< Method One > Using FTP Software

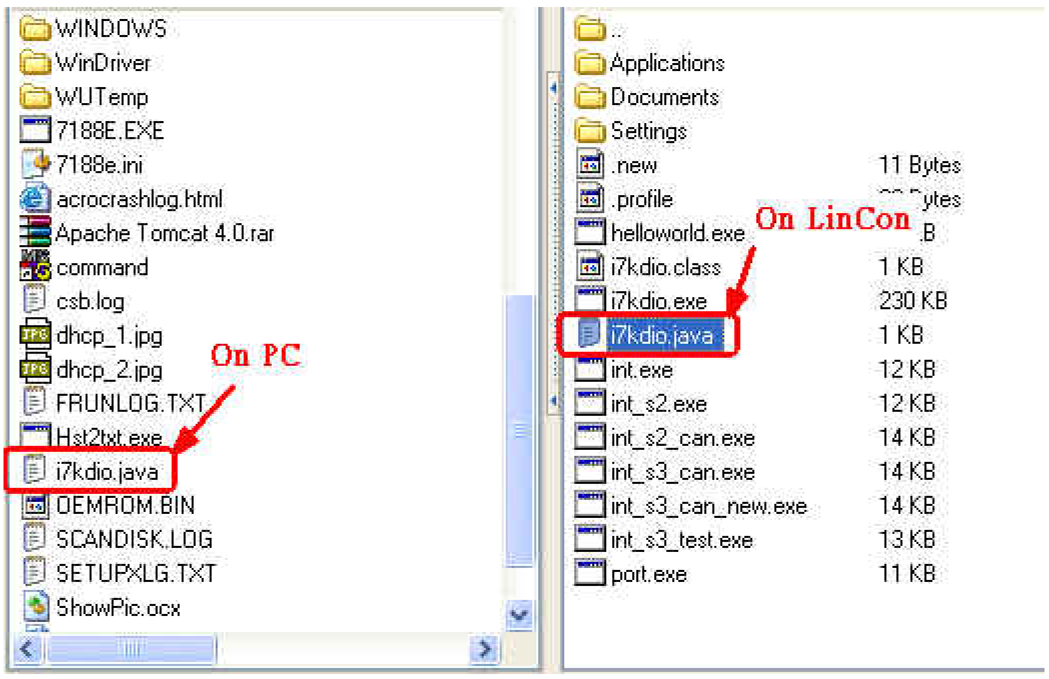
(1) Open a FTP Software and add a ftp site of the LinCon-8000. The **User_Name** and **Password** default value is **" root "**. (refer to Fig.10-2).



Host Name: 192.168.0.200
Username: root
Password: ****
[ ] Anonymous
[✓] Passive Mode
[✓] Save Password
[ ] Stay Connected
Port: 21
Timeout (secs): 120
Initial Host Folder
Initial Local Folder
Comment

Fig.10-2

(2) Click the **"Connect"** button to connect to the ftp server on the LinCon-8000. Then upload the file – **i7kdio.java** to the LinCon-8000. (refer to Fig.10-3).



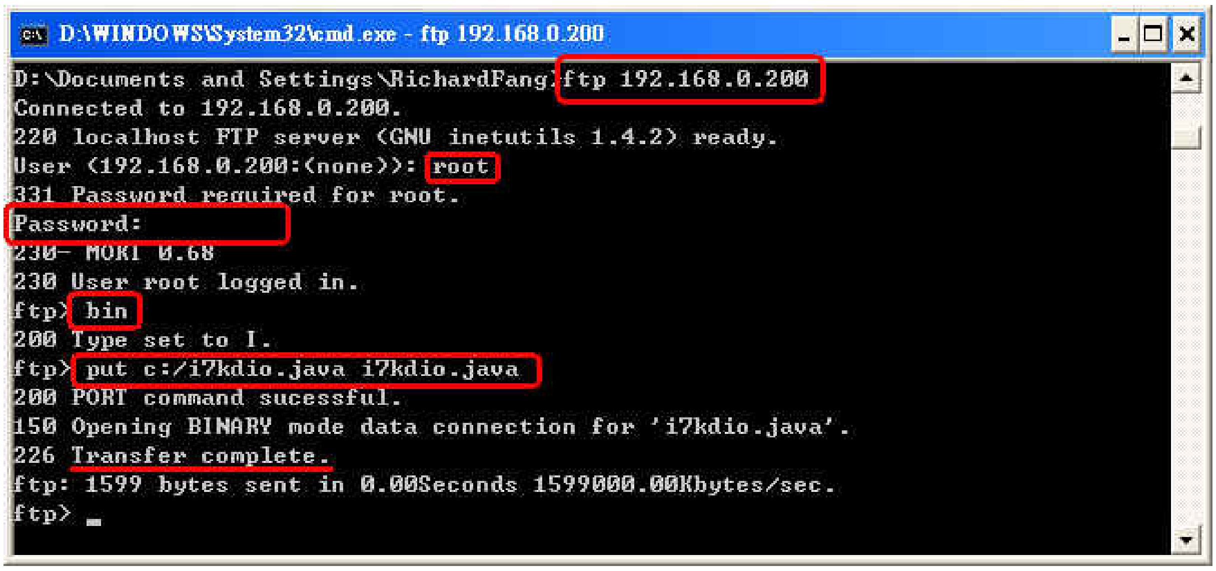
```
WINDOWS
WinDriver
WUTemp
7188E.EXE
7188e.ini
acrocrashlog.html
Apache Tomcat 4.0.rar
command
csb.log
dhcp_1.jpg
dhcp_2.jpg
FRUNLOG.TXT
Hst2txt.exe
i7kdio.java
OEMROM.BIN
SCANDISK.LOG
SETUPXLG.TXT
ShowPic.ocx
```

```
..
Applications
Documents
Settings
.new                 11 Bytes
.profile                  ytes
helloworld.exe             B
i7kdio.class         1 KB
i7kdio.exe           230 KB
i7kdio.java          1 KB
int.exe              12 KB
int_s2.exe           12 KB
int_s2_can.exe       14 KB
int_s3_can.exe       14 KB
int_s3_can_new.exe   14 KB
int_s3_test.exe      13 KB
port.exe             11 KB
```

Fig.10-3

< Method Two > Using DOS Command Prompt

Open a DOS Command Prompt and type in the ftp IP Address of LinCon-8000 to connect to the ftp server of your LinCon-8000. Then input your **User Name** and **Password** (**root** is the default value) to login to the LinCon-8000. Type in **bin** to make transfer files by using the binary mode. Then type **put c:/i7kdio.java i7kdio.java** to transfer the i7kdio.exe file to the LinCon-8000. After the message of Transfer complete has appeared, the transfer will be complete.(refer to Fig. 10-4)

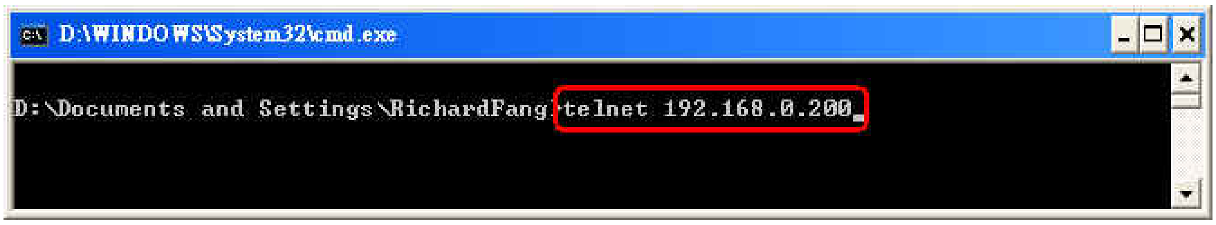


```
D:\WINDOWS\System32\cmd.exe - ftp 192.168.0.200
D:\Documents and Settings\RichardFang>ftp 192.168.0.200
Connected to 192.168.0.200.
220 localhost FTP server (GNU inetutils 1.4.2) ready.
User (192.168.0.200:(none)): root
331 Password required for root.
Password:
230- MOKI 0.68
230 User root logged in.
ftp> bin
200 Type set to I.
ftp> put c:/i7kdio.java i7kdio.java
200 PORT command sucessful.
150 Opening BINARY mode data connection for 'i7kdio.java'.
226 Transfer complete.
ftp: 1599 bytes sent in 0.00Seconds 1599000.00Kbytes/sec.
ftp> _
```

Fig. 10-4

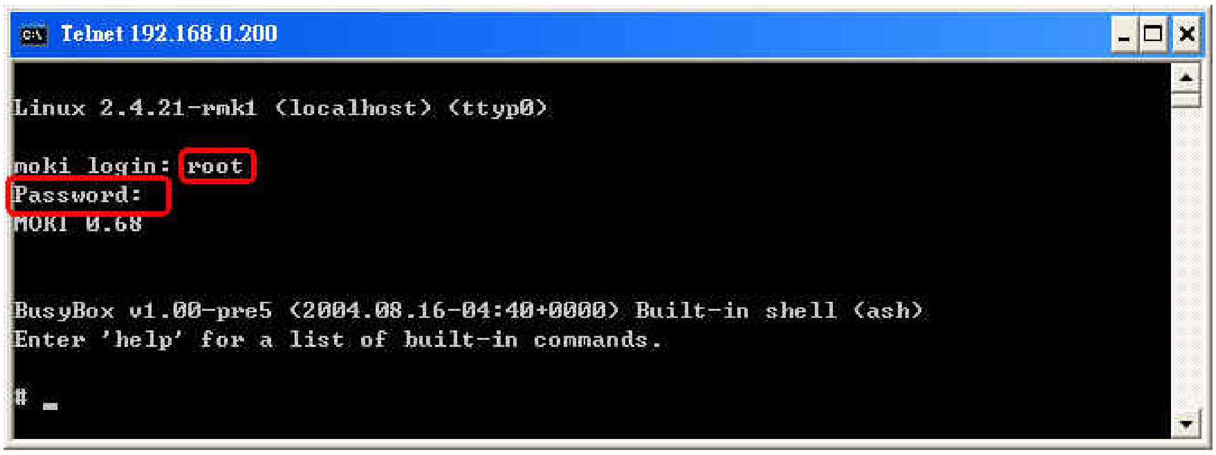
STEP 3：**(Telnet to LinCon-8000)**

Type the <u>telnet IP Address of LinCon-8000</u> to remote control LinCon-8000 and input your **User Name** and **Password** (**root** is the default value) to login to the LinCon-8000. (refer to Fig. 10-5 and Fig. 10-6)



```
D:\Documents and Settings\RichardFang>telnet 192.168.0.200
```

Fig. 10-5

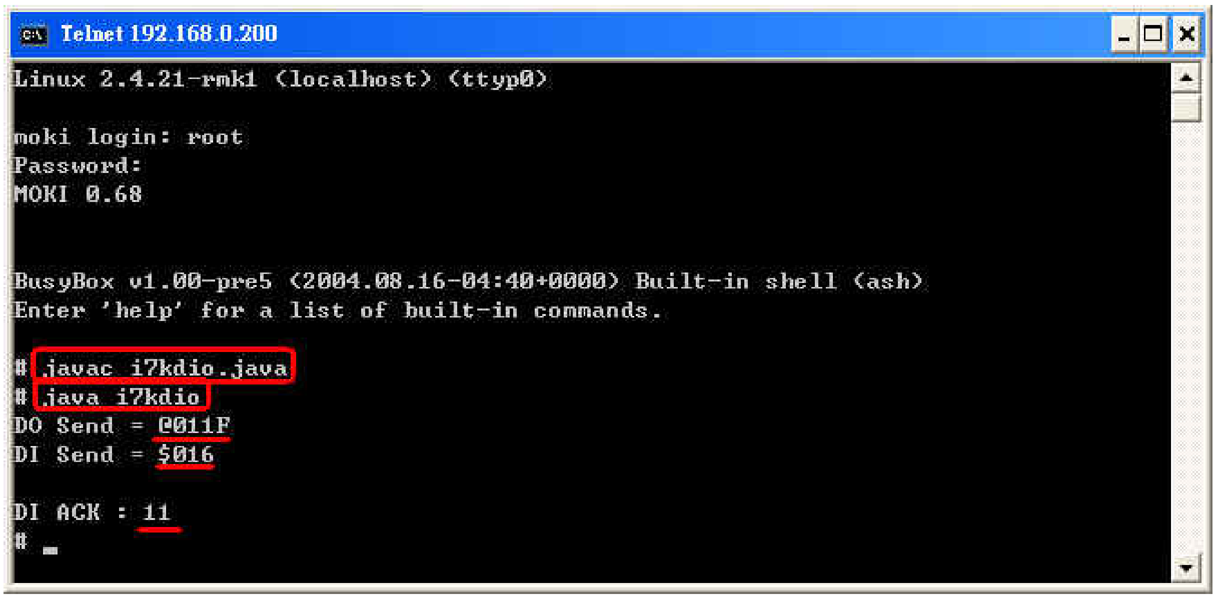


```
Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.08.16-04:40+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# _
```

Fig. 10-6

STEP 4：**(Compile i7kdio.java to i7kdio.class and execute i7kdio.class)**

Type in **<u>javac i7kdio.java</u>** to compile i7kdio.java to i7kdio.class. The process of compiling in the LinCon-8000 is slower than that in the PC. Then type **<u>java i7kdio</u>** to execute the i7kdio.class. (refer to Fig. 10-7)



```
Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.08.16-04:40+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# javac i7kdio.java
# java i7kdio
DO Send = @011F
DI Send = $016

DI ACK : 11
# _
```

Fig. 10-7

“ @**01**1**F** “ is an ICPDAS command type where **01** means address and **1F** means the DO channel 0 ~ 5 will output. “ **$016** “ is used to read back all DI Channels status. “ **DI ACK : 11** (15-2^**2**)“ means there is an input in DI channel 2.

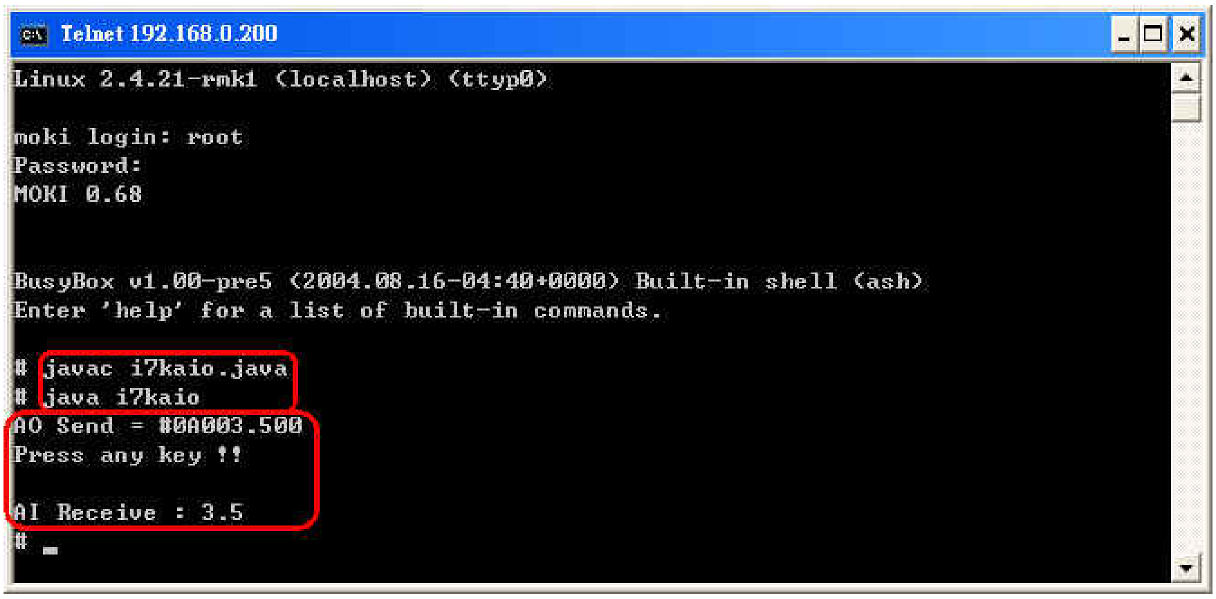
10.5 I-7k Modules AIO Control Demo

This demo – **i7kaio.java** will illustrate how to control the AI/AO with the I-7012 (1 AI channels) and I-7022 module (2 AO channel). The address of I-7012 and I-7022 module in the RS-485 network are 11 and 10 separately and the baudrate is 9600.

The result of this demo is to allow the AO channel 0 of I-7022 to output at 3.5V and AI channel of I-7012 input from the AO channel 0 of I-7022. The source code of this demo program is provided below :

```
import java.io.*;
import com.icpdas.comm.*;

public class i7kaio
{
    public static void main(String[] args) throws java.io.IOException
    {
        int rev;
        int fd;
        float ao=3.5f;
        byte a[] = new byte[100];
        Comm comm1 = new Comm();
        IoBuf i7kBuf = new IoBuf();
        rev = comm1.open(3,9600,comm1.DATABITS_8,comm1.PARITY_NONE,
                                comm1.STOPBITS_1);

        if (rev!=0) System.out.println("Open port error code : "+rev);
        else{
            i7kBuf.dwBuf[0] = 3;          //Serial Port
            i7kBuf.dwBuf[1] = 10;          //Address
            i7kBuf.dwBuf[2] = 0x7022;   //0x7022; //module name
            i7kBuf.dwBuf[3] = 0;          //check sum disable
            i7kBuf.dwBuf[4] = 100;        //Timeout 100ms
            i7kBuf.dwBuf[6] = 1;          //Enable String Debug

            //I-7022 AO
            i7kBuf.fBuf[0] = ao;         //Output AO Value
            rev = comm1.setAnalogOut(i7kBuf);     //AO Output
            if (rev!=0) System.out.println("Analog Out Error Code : "+ rev);

            System.out.println("AO Send = "+ i7kBuf.szSend);
            System.out.println("Press any key !!");
            System.in.read(a);

            //I-7012 AI
            i7kBuf.dwBuf[1] = 11;          //Address
            i7kBuf.dwBuf[2] = 0x7012;   //0x7012; //module name

            rev = comm1.getAnalogIn(i7kBuf);
            if (rev!=0) System.out.println("AI Error Code : "+ rev);

            //System.out.println("szSend = "+ i7kBuf.szSend);
            System.out.println("AI Receive : "+i7kBuf.fBuf[0]);
        }
        comm1.close(3);
    }
}
```

All the steps from programming to execution are the same as those in the section 10.4. The result of execution refers to Fig. 10-8.



```
Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.08.16-04:40+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# javac i7kaio.java
# java i7kaio
AO Send = #0A003.500
Press any key !!

AI Receive : 3.5
# _
```

Fig. 10-8

10.6 I-87k Modules DIO Control Demo

When using the I-87k modules for I/O control in the LinCon-8000, the program will be a little different depending on the location of I-87k modules. There are three conditions for the location of I-87k modules：

(1) When I-87k DIO modules are **in the LinCon-8000 slots**, please refer to the demo in the section 10.6.1.

(2) When I-87K DIO modules are **in the I-87k I/O expansion unit slots**, please refer to the demo in the section 10.6.2.

(3) When I-87k DIO modules **in the I-8000 controller slots**, I-87k modules will be regarded as I-8k modules so please refer to the I/O control of I-8k modules in the section 10.6.2

10.6.1 I-87k Modules in slots of LinCon-8000

I-87057 and I-87051 are plugged in the slot 01 and 02 in the LinCon-8000 separately. The source code of this demo program –**i87kdio.java** is as follows：

```
import java.io.*;                //For System.in.read()
import com.icpdas.comm.*;        //ICPDAS communication packages
import com.icpdas.slot.*;        //Slot I/O packages
```

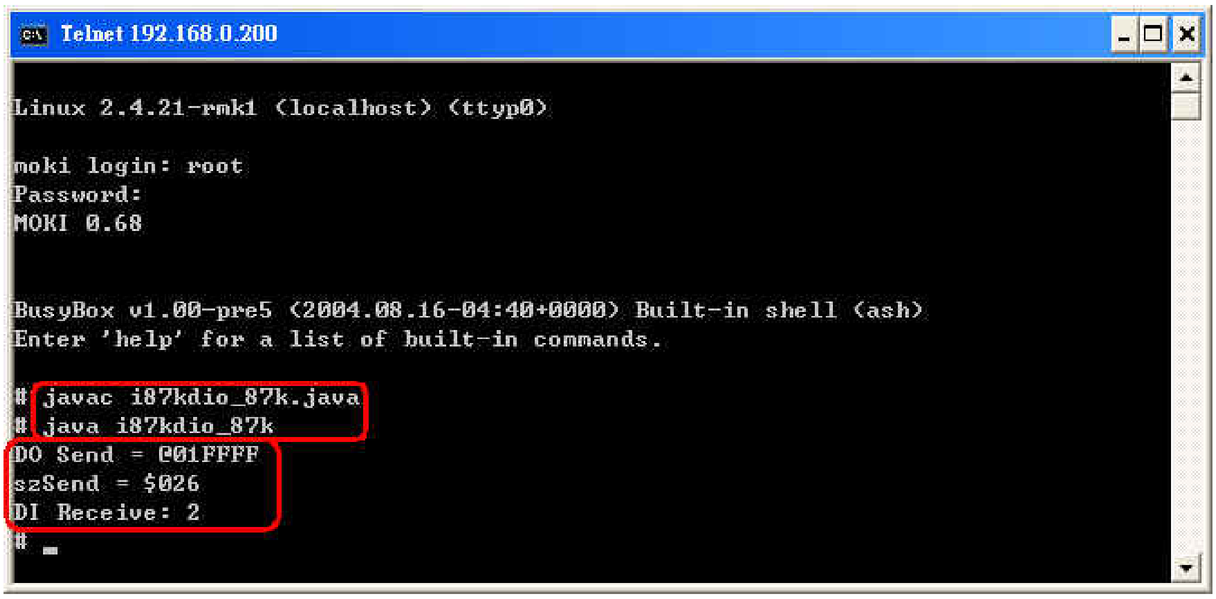
```
public class i87kdio
{
    public static void main(String[] args) throws java.io.IOException
    {
        int rev;
        int fd;
        int i=1;
        byte a[] = new byte[100];
        Comm comm1 = new Comm();                //ICPDAS communication object
        Slot slot1 = new Slot();
        IoBuf i87kBuf = new IoBuf();            //control matrix
        slot1.open(0);                          //open slot 0 fot i87k slot select

        rev = comm1.open(1,115200,comm1.DATABITS_8,comm1.PARITY_NONE,
                            comm1.STOPBITS_1);//open serial port
        if (rev!=0) System.out.println("Open port error code : "+rev);
        else{
            i87kBuf.dwBuf[0] = 1;                           //Serial Port
            i87kBuf.dwBuf[1] = 0;                           //Module Address
            i87kBuf.dwBuf[3] = 0;                           //check sum disable
            i87kBuf.dwBuf[4] = 100;                         //Timeout 100ms
            i87kBuf.dwBuf[6] = 1;                           //Enable String Debug

            while(a[0]!=113) {
                i87kBuf.dwBuf[2] = 0x87057 ;                //module name
                i87kBuf.dwBuf[5] =i;                        //Digital Output Value
                slot1.setChangeToSlot(1);                   //change to slot 1
                rev = comm1.setDigitalOut(i87kBuf);
                System.out.println("szSend = " + i87kBuf.szSend +
                                   " szReceive = "+i87kBuf.szReceive);

                if (rev!=0) System.out.println("Digital Out Error Code : "+ rev);

                i87kBuf.dwBuf[2] = 0x87051 ;                //module name
                slot1.setChangeToSlot(2);                   //change to slot 2
                rev = comm1.getDigitalIn(i87kBuf);
                System.out.println("szSend = " + i87kBuf.szSend +
                                   " szReceive = "+i87kBuf.szReceive);

                if (rev!=0) System.out.println("Digital In Error Code : "+ rev);
                else System.out.println("Digital In : "+ i87kBuf.dwBuf[5]);

                System.in.read(a);
                i=(i>255)?1:(i<<1);
            }
        }
        slot1.close(0);
        comm1.close(1);
        System.out.println("End of program");
    }
}
```

10.6.2 I-87k Modules in slots of I-87k I/O expansion unit

The address of I-87057 and I-87051 in the RS-485 network are set to be 01 and 02 separately and the baudrate is 9600 via the DCON Utility. The source code of this demo program – **i87kdio_87k.java** is as follows：

```
import java.io.*;                          //For System.in.read()
import com.icpdas.comm.*;                  //ICPDAS communication packages

public class i87kdio_87k
{
    public static void main(String[] args) throws java.io.IOException
    {
        int rev;
        int fd;
        int i=65535;                       //(int)(Math.pow(2,16)-1)
        byte a[] = new byte[100];
        Comm comm1 = new Comm();           //ICPDAS communication object
        IoBuf i87kBuf = new IoBuf();       //control matrix
        rev = comm1.open(3,9600,comm1.DATABITS_8,comm1.PARITY_NONE,
                         comm1.STOPBITS_1);//open serial port

        if (rev!=0) System.out.println("Open port error code : "+rev);
        else{
            i87kBuf.dwBuf[0] = 3;                  //Serial Port
            i87kBuf.dwBuf[3] = 0;                  //check sum disable
            i87kBuf.dwBuf[4] = 100;                //Timeout 100ms
            i87kBuf.dwBuf[6] = 1;                  //Enable String Debug

            //I-87057 DO
            i87kBuf.dwBuf[1] = 1;              //Module Address
            i87kBuf.dwBuf[2] = 0x87057 ;       //module name
            i87kBuf.dwBuf[5] =i;               //Digital Output Value
            rev = comm1.setDigitalOut(i87kBuf);

            System.out.println("DO Send = "+ i87kBuf.szSend);
            if (rev!=0) System.out.println("Digital Out Error Code : "+ rev);


            //I-87051 DI
            i87kBuf.dwBuf[1] = 2;                  //Module Address
            i87kBuf.dwBuf[2] = 0x87051 ;           //0x87051 //module name
            rev = comm1.getDigitalIn(i87kBuf);
            System.out.println("szSend = "+ i87kBuf.szSend);
            if (rev!=0) System.out.println("Digital In Error Code : "+ rev);
            else System.out.println("DI Receive: "+ i87kBuf.dwBuf[5]);
        }
        comm1.close(3);
    }
}
```

All the steps from writing to execution are the same as those in the section 10.4 The result of execution refers to Fig. 10-9.



```
Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.08.16-04:40+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# javac i87kdio_87k.java
# java i87kdio_87k
DO Send = @01FFFF
szSend = $026
DI Receive: 2
# _
```

Fig. 10-9

10.6.3 I-87k Modules in slots of I-8000 Controller

If the I-87k DIO modules are in the I-8000 controller slots, the I-87k modules will be regarded as I-8k modules. In this case, users can refer to DI/DO control demo of I-8k modules in the section 10.8.

10.7 I-87k Modules AIO Control Demo

When using I-87k modules for I/O control of the LinCon-8000, the program will be a little different depending on the location of I-87k modules. There are three conditions for the location of I-87k modules：

(1) When I-87k AIO modules are **in the LinCon-8000 slots**, , please refer to the demo in the section 10.7.1.

(2) When I-87K AIO modules are **in the I-87k I/O expansion unit slots**, please refer to the demo in the section 10.7.2.

(3) When I-87k AIO modules **in the I-8000 controller slots**, I-87k modules will be regarded as I-8k modules. In this case users can refer to I/O control demo of I-8k modules in the section 10.7.3

10.7.1 I-87k Modules in slots of LinCon-8000

I-87024 and I-87017 are plugged in the slot 01 and 02 in the LinCon-8000 separately. The source code of this demo program –**i87kaio.java** is as follows：

```
import java.io.*;                        //For System.in.read()
import com.icpdas.comm.*;                //ICPDAS communication packages
import com.icpdas.slot.*;                //Slot I/O packages
public class i87kaio
{
    public static void main(String[] args) throws java.io.IOException
    {
        int rev;
        int fd;
        float aoValue = 0.5F;
        byte a[] = new byte[100];
        Comm comm1 = new Comm();             //ICPDAS communication object
        Slot slot1 = new Slot();
        IoBuf i87kBuf = new IoBuf();         //control matrix
        slot1.open(0);                       //open slot 0 fot i87k slot select

        rev = comm1.open(1,115200,comm1.DATABITS_8,comm1.PARITY_NONE,
                         comm1.STOPBITS_1);//open serial port
        if (rev!=0) System.out.println("Open port error code : "+rev);
        else{
            i87kBuf.dwBuf[0] = 1;            //Serial Port
            i87kBuf.dwBuf[1] = 0;            //Address
            i87kBuf.dwBuf[3] = 0;            //check sum disable
            i87kBuf.dwBuf[4] = 100;          //Timeout 100ms
            i87kBuf.dwBuf[5] = 0;            //channel 0
            i87kBuf.dwBuf[6] = 1;            //Enable String Debug

            while(a[0]!=113) {
                i87kBuf.dwBuf[2] = 0x87024 ;           //module name
                i87kBuf.fBuf[0] = aoValue;             //AO output value
                slot1.setChangeToSlot(1);              //change to slot 1
                rev = comm1.setAnalogOut(i87kBuf);  //Set Analog Output Value

                if (rev!=0) System.out.println("Analog Out Error Code : "+ rev);

                System.out.println("szSend = " + i87kBuf.szSend +
                                   " szReceive = " +i87kBuf.szReceive);
                System.out.println("Press 'Enter' to Read AI");

                System.in.read(a);
                i87kBuf.dwBuf[2] = 0x87017 ;           //module name
                slot1.setChangeToSlot(2);              //change to slot 2
                rev = comm1.getAnalogIn(i87kBuf);      //Get Analog Output Value
```

```
                    if (rev!=0) System.out.println("Analog In Error Code : "+ rev);
                    System.out.println("szSend = "+ i87kBuf.szSend +
                                        " szReceive = "+i87kBuf.szReceive);
                    System.out.println("Analog In Value : "+i87kBuf.fBuf[0]);
                    System.in.read(a);
                    aoValue=(aoValue>8.0F)?0.5F:(aoValue + 1.0F);
                }
            }
            slot1.close(0);
            comm1.close(1);
            System.out.println("End of program");
        }
}
```

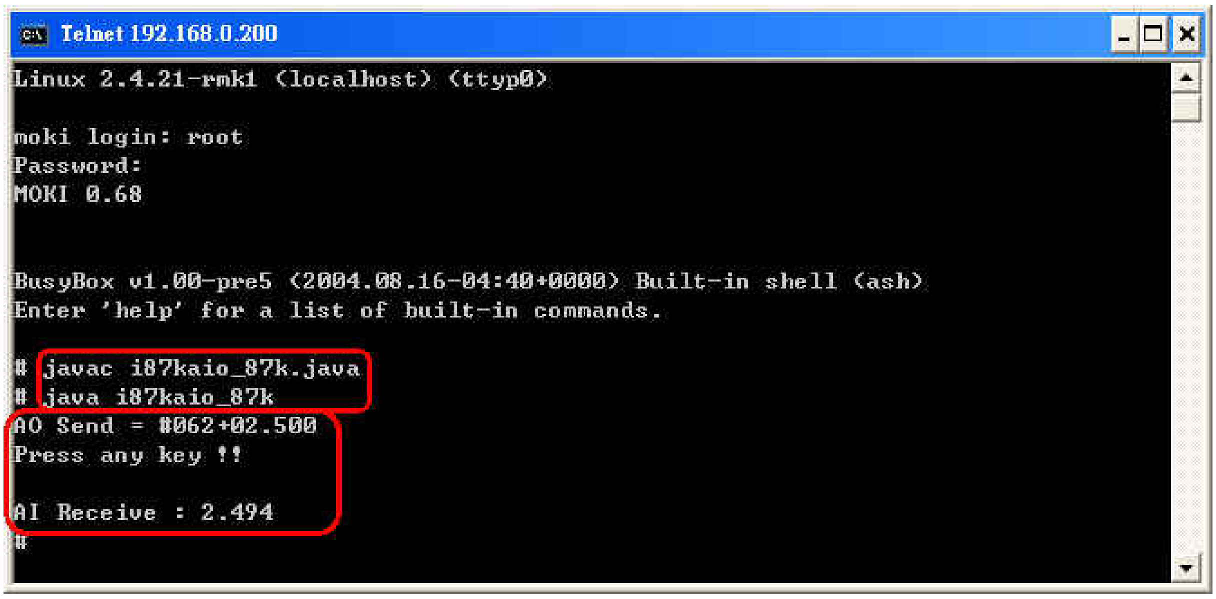
10.7.2 I-87k Modules in slots of I-87k I/O expansion unit

The address of I-87024 and I-87017 in the RS-485 network are set to be 06 and 07 separately and the baudrate is 9600 via the DCON Utility. The source code of this demo program – **i87kaio_87.java** is as follows：

```
import java.io.*;                                   //For System.in.read()
import com.icpdas.comm.*;                           //ICPDAS communication
packages

public class i87kaio_87k
{
    public static void main(String[] args) throws java.io.IOException
    {
        int rev;
        int fd;
        float ao=2.5f;
        byte a[] = new byte[100];
        Comm comm1 = new Comm();                    //ICPDAS communication object
        IoBuf i87kBuf = new IoBuf();                //control matrix

        rev = comm1.open(3,9600,comm1.DATABITS_8,comm1.PARITY_NONE,
                            comm1.STOPBITS_1);     //open serial port
        if (rev!=0) System.out.println("Open port error code : "+rev);
        else{
            i87kBuf.dwBuf[0] = 3;                   //Serial Port
            i87kBuf.dwBuf[3] = 0;                   //check sum disable
            i87kBuf.dwBuf[4] = 100;                 //Timeout 100ms
            i87kBuf.dwBuf[5] = 2;                   //channel 2
            i87kBuf.dwBuf[6] = 1;                   //Enable String Debug

            //I-87024 AO
            i87kBuf.dwBuf[1] = 6;                   //Address
            i87kBuf.dwBuf[2] = 0x87024 ;            //0x87024; //module name
            i87kBuf.fBuf[0] = ao;
```

```
                rev = comm1.setAnalogOut(i87kBuf);   //Set Analog Output Value
                if (rev!=0) System.out.println("Analog Out Error Code : "+ rev);

                System.out.println("AO Send = "+ i87kBuf.szSend);
                System.out.println("Press any key to read AI Data !!");
                System.in.read(a);

                //I-87017 AI
                i87kBuf.dwBuf[1] = 7;                //Address
                i87kBuf.dwBuf[2] = 0x87017 ;         //0x87017; //module name

                rev = comm1.getAnalogIn(i87kBuf);    //Get Analog Output Value
                if (rev!=0) System.out.println("Analog In Error Code : "+ rev);
                System.out.println("AI Receive : "+i87kBuf.fBuf[0]);
            }
            comm1.close(3);
        }
}
```

All the steps from writing to execution are the same as those in the section 10.4. The result of execution refers to Fig.10-10



```
Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.08.16-04:40+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# javac i87kaio_87k.java
# java i87kaio_87k
AO Send = #062+02.500
Press any key !!

AI Receive : 2.494
#
```

Fig.10-10

10.7.3 I-87k Modules in slots of I-8000 Controller

If the I-87k AI/AO modules are in the I-8000 controller slots, I-87k modules will be regarded as I-8k modules and please refer to AI/AO control demo of I-8k modules in the section 10.9.

10.8 I-8k Modules DIO Control Demo

When using I-8k modules for I/O control of the LinCon-8000, the program will be a little different depending on the location of I-8k modules. There are two conditions for the location of I-8k modules：

(1) When I-8k DIO modules are **in the LinCon-8000 slots.** Please refer to I/O control demo of I-8k DIO modules in the section 10.8.1

(2) When I-8k DIO modules **in I-8000 controller slots.** Please refer to I/O control demo of I-8k DIO modules in the section 10.8.2

10.8.1 I-8k Modules in slots of LinCon-8000

I-8055 is plugged in the slot 03 of LinCon-8000. The source code of this demo program – **i8kdio.java** is as follows：

```
import java.io.*;                                   //For System.in.read()
import com.icpdas.slot.*;                           //Slot I/O packages

public class i8kdio
{
     public static void main(String[] args) throws java.io.IOException
     {
          int rev;
          long diValue;
          int i=1;
          byte a[] = new byte[100];

          Slot slot1 = new Slot();
          rev = slot1.open(3);                             //open slot 1 for digital in/out
          if (rev!=0) System.out.println("Open port error code : "+rev);
          else{
               while(a[0]!=113) {
                    rev = slot1.setDigitalOut(3,8,i);
                    if (rev!=0) System.out.println("Digital Out Error Code : "+ rev);
                    diValue = slot1.getDigitalIn(3,8);
                    System.out.println("Digital Out :" + i + "    Digital In : "+ diValue);
                    System.in.read(a);
                    i=(i>255)?1:(i<<1);
               }
          }
          slot1.close(1);
          System.out.println("End of program");
     }
}
```

10.8.2 I-8k Modules in slots of I-8000 Controller

In this section, the demo program - **i8kdio_8k.java** will introduce how to control the DI/DO with the I-8055 (8 DO channels and 8 DI channels) module. Please follow the below steps to configure the hardware correctly：

(1) Put the I-8055 module into slot 0 on the I-8000 controller.

(2) Install the **R232_300.exe** to flash memory of I-8000 controller as firmware.

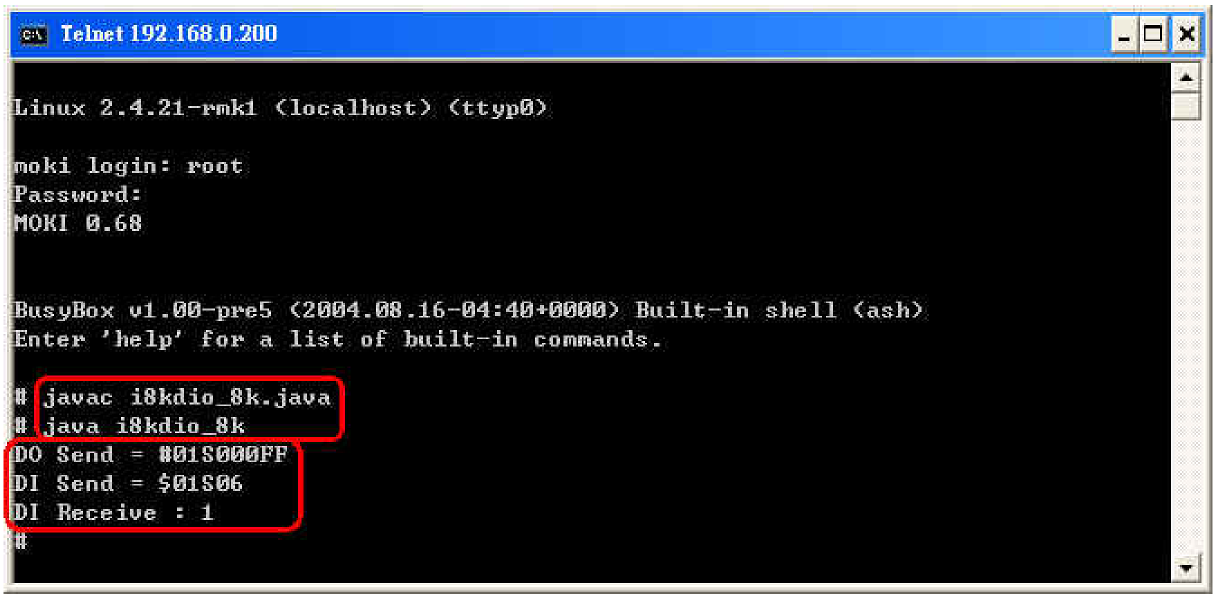
(3) Connect **com2** on the LinCon-8000 to the com1 on the I-8000 controller with the RS-232 cable.

The address of I-8000 controller is 01 and the baudrate is 115200 and they can be modified via the DCON Utility. The result of this demo allows DO channels 0 ~7 of I-8055 output and DI channel 0 of I-8055 input. The source code of this demo program is as follows：

```
import java.io.*;                               //For System.in.read()
import com.icpdas.comm.*;                       //ICPDAS communication packages

public class i8kdio_8k
{
    public static void main(String[] args) throws java.io.IOException
    {
        int rev;
        int fd;
        int i=255;
        byte a[] = new byte[100];
        Comm comm1 = new Comm();                //ICPDAS communication object
        IoBuf i87kBuf = new IoBuf();            //control matrix

        rev = comm1.open(2,115200,comm1.DATABITS_8,comm1.PARITY_NONE,
                         comm1.STOPBITS_1);//open serial port
        if (rev!=0) System.out.println("Open port error code : "+rev);
        else {
            i87kBuf.dwBuf[0] = 2;               //Serial Port
            i87kBuf.dwBuf[1] = 1;               //Address
            i87kBuf.dwBuf[3] = 0;               //check sum disable
            i87kBuf.dwBuf[4] = 100;             //Timeout 100ms
            i87kBuf.dwBuf[6] = 1;               //Enable String Debug

                //I-8055 DO
                i87kBuf.dwBuf[2] = 0x8055 ;     //0x8055; //module name
                i87kBuf.dwBuf[5] =i;            //Digital out
                i87kBuf.dwBuf[7] = 0;           //Slot number
                rev = comm1.setDigitalOut(i87kBuf);//Digital Output Value
                System.out.println("DO Send = "+ i87kBuf.szSend);
```

```
                    if (rev!=0) System.out.println("Digital Out Error Code : "+ rev);

                    //I-8055 DI
                    rev = comm1.getDigitalIn(i87kBuf);// Digital Input Value
                    System.out.println("DI Send = "+ i87kBuf.szSend);
                    if (rev!=0) System.out.println("Digital In Error Code : "+ rev);
                    else System.out.println("DI Receive : "+ i87kBuf.dwBuf[5]);
            }
            comm1.close(2);
        }
}
```

All the steps from writing to execution are the same as those in the section 10.4. The result of execution refers to Fig.10-12



```
Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.08.16-04:40+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# javac i8kdio_8k.java
# java i8kdio_8k
DO Send = #01S000FF
DI Send = $01S06
DI Receive : 1
#
```

Fig.10-12

10.9 I-8k Modules AIO Control Demo

When using I-8k modules for I/O control of the LinCon-8000, the program will be a little different depending on the location of I-8k modules. There are two conditions for the location of I-8k modules：

(1) When I-8k AIO modules are **in the LinCon-8000 slots.** Please refer to I/O control of I-8k AIO modules in the section 10.9.1

(2) When I-8k AIO modules **in the I-8000 controller slots.** Please refer to I/O control of I-8k AIO modules in the section 10.9.2

10.9.1 I-8k Modules in slots of LinCon-8000

I-8024 (4 AO channels) and I-8017 (8 AI channels) is plugged in the slot 1 and slot 2 of LinCon-8000. The source code of this demo program – **i8kaio.java** is as follows：

```
import java.io.*;                              //For System.in.read()
import com.icpdas.slot.*;                      //Slot I/O packages
public class i8kaio
{
    public static void main(String[] args) throws java.io.IOException
    {
        int rev;
        float aiValue;
        float i=1;
        byte a[] = new byte[100];
        Slot slot1 = new Slot();
        rev = slot1.open(1);                   //open slot 1 for analog output

        if (rev!=0) System.out.println("Open slot error code : "+rev);
        else{
            rev = slot1.init8024(1);
            rev = slot1.open(2);               //open slot 2 for analog input
            if (rev!=0) System.out.println("Open slot error code : "+rev);
            else
            {
                rev = slot1.init8017(2);
                rev = slot1.setChannelGainMode(2,0,0,0);

                while(a[0]!=113)
                {
                    rev = slot1.setVoltageOut(1,0,i);
                    if (rev!=0) System.out.println("Analog Out Error Code : "+ rev);
                    aiValue = slot1.getAnalogIn(2);
                    System.out.println("Analog Out :" + i + " Analog In : "+ aiValue);
                    System.in.read(a);
                    i=(i>8)?1:(i+1.5F);
                }
            }
        }

        slot1.close(1);
        slot1.close(2);
        System.out.println("End of program");
    }
}
```

10.9.2 I-8k Modules in slots of I-8000 Controller

In this section, the demo program - **i8kaio_8k.java** will illustrate how to control the AI/AO with the I-8024 (4 AO channels) module and I-8017 (8 AI channels) module when they are plugged into the slot 0 and slot 1 on the I-8000 controller separately. Please follow the below steps to configure the hardware correctly：

(1) Put the I-8024 and I-8017 modules into slot 0 and slot 1 of the I-8000 controller.

(2) Install the R232_300.exe to flash memory of the I-8000 controller as firmware.

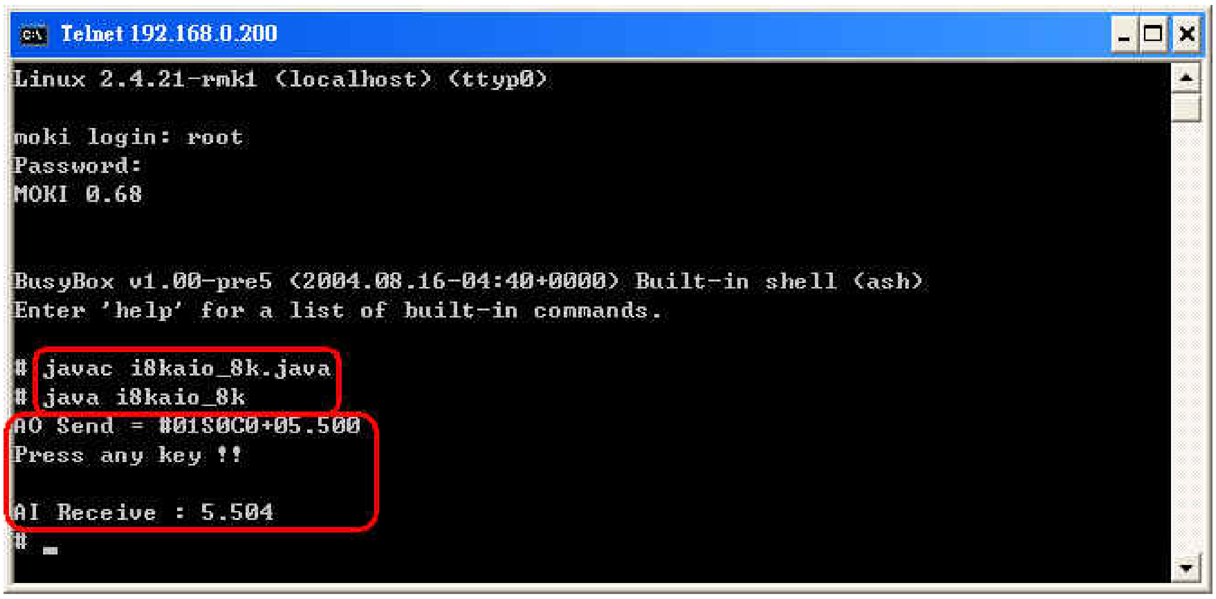
(3) Connect com2 on the LinCon-8000 to com1 on the I-8000 controller with RS-232 cable.

The address and baudrate of I-8000 controller are 01 and 115200 and they can be modified via the DCON Utility. The result of this demo allows the channel 0 of I-8024 output at 3.5V and the AI channel 0 of I-8017H input from AO channel 0 of I-8024. The source code of this demo program is as follows：

```
import java.io.*;                          //For System.in.read()
import com.icpdas.comm.*;                  //ICPDAS communication packages

public class i8kaio_8k
{
      public static void main(String[] args) throws java.io.IOException
      {
            int rev;
            int fd;
            float ao=5.5f;
            byte a[] = new byte[100];
            Comm comm1 = new Comm();              //ICPDAS communication object
            IoBuf i8kBuf = new IoBuf();           //control matrix
            rev = comm1.open(2,115200,comm1.DATABITS_8,comm1.PARITY_NONE,
                                   comm1.STOPBITS_1);//open serial port
            if (rev!=0) System.out.println("Open port error code : "+rev);
            else {
                  i8kBuf.dwBuf[0] = 2;              //Serial Port
                  i8kBuf.dwBuf[1] = 1;              //Address
                  i8kBuf.dwBuf[3] = 0;              //check sum disable
                  i8kBuf.dwBuf[4] = 100;            //Timeout 100ms
                  i8kBuf.dwBuf[6] = 1;              //Enable String Debug

                     //I-8024 AO
                     i8kBuf.dwBuf[2] = 0x8024 ;     //0x8024; //module name
                        i8kBuf.dwBuf[7] = 0;         //Slot number
                        i8kBuf.fBuf[0] = ao;
                        rev = comm1.setAnalogOut(i8kBuf);         //Set AO Value
```

```
                    if (rev!=0) System.out.println("Analog Out Error Code : "+ rev);
                    System.out.println("AO Send = "+ i8kBuf.szSend);
                    System.out.println("Press any key !!");
                    System.in.read(a);

                    //I-8017 AI
                    i8kBuf.dwBuf[2] = 0x8017 ;
                    i8kBuf.dwBuf[7] = 1;
                    rev = comm1.getAnalogIn(i8kBuf);            //Get AI Value
                    if (rev!=0) System.out.println("Analog In Error Code : "+ rev);
                    System.out.println("AI Receive : "+i8kBuf.fBuf[0]);

            }
            comm1.close(2);
        }
}
```

All the steps from writing to execution are the same as those in the section 10.4. The result of execution refers to Fig.10-14



```
Linux 2.4.21-rmk1 (localhost) (ttyp0)

moki login: root
Password:
MOKI 0.68


BusyBox v1.00-pre5 (2004.08.16-04:40+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# javac i8kaio_8k.java
# java i8kaio_8k
AO Send = #01S0C0+05.500
Press any key !!

AI Receive : 5.504
# _
```

Fig.10-14

11. Additional Support

In this chapter, ICPDAS provides extra module supported and instructions to enhance LinCon-8000 functionality and affinity.

11.1 N-Port Module (I-8114, I-8112, I-8142, I-8144) Support

The **I-8114/I-8144** and **I-8112/I-8142** modules provide **four** and **two serial ports** respectively. Users can insert them into the LinCon-8000 slots. In this way, users can use more serial ports in the LinCon-8000 and the expanded maximum number of serial port in the LinCon-8000 will be twenty-nine. The LinCon-8000 is a multi-tasking uint, therefore users can control all the serial ports simultaneously. **The serial port number** of I-8114 and I-8112 modules are presented in the fig.11-1 and fig.11-2 and it is **fixed** according to their slot position in the LinCon-8000.

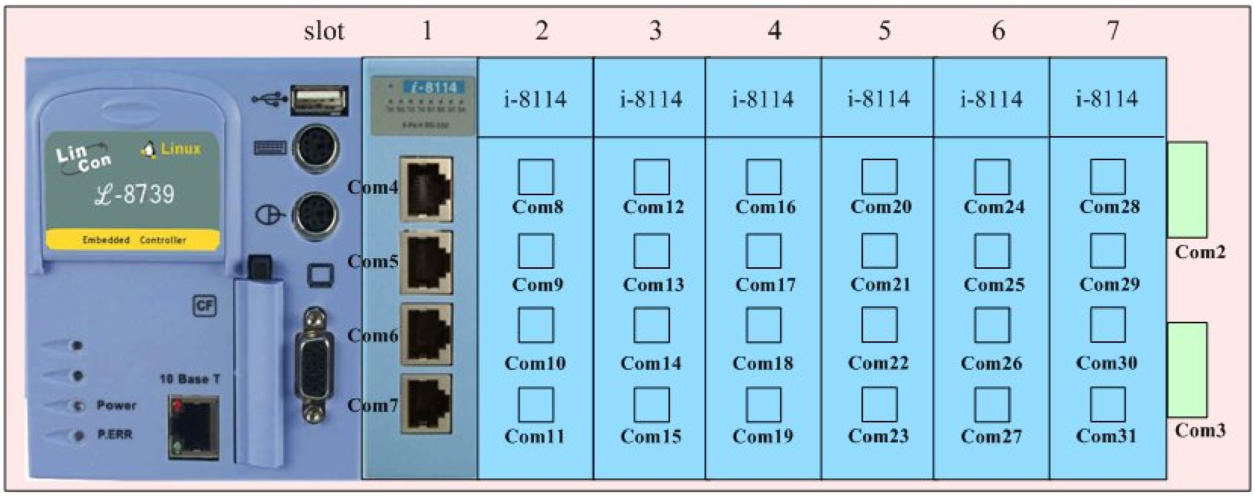


Fig.11-1

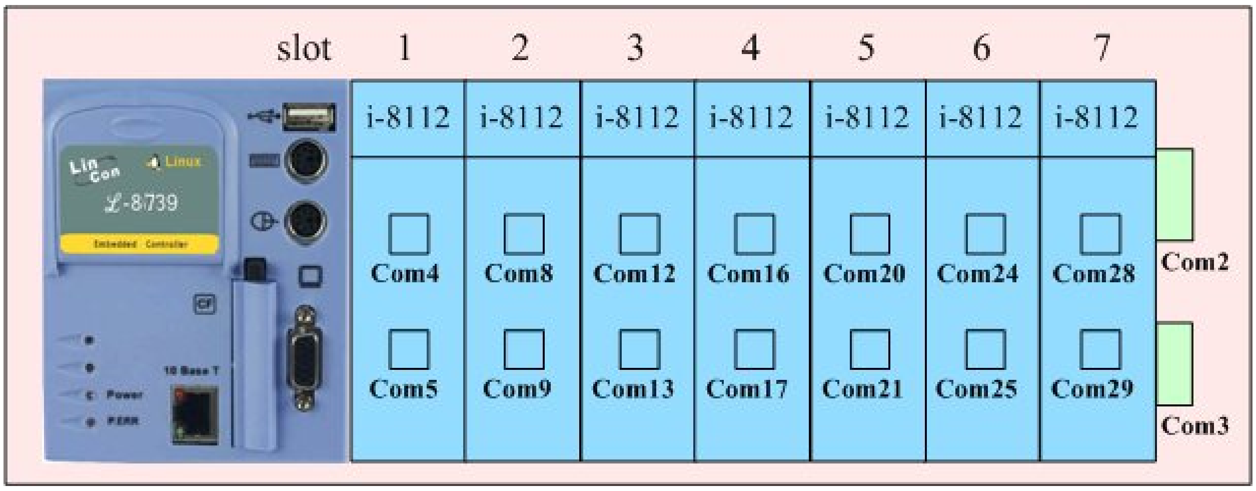


Fig.11-2

Fig.11-3 is the serial port number corresponding to the **device name** in the LinCon-8000.

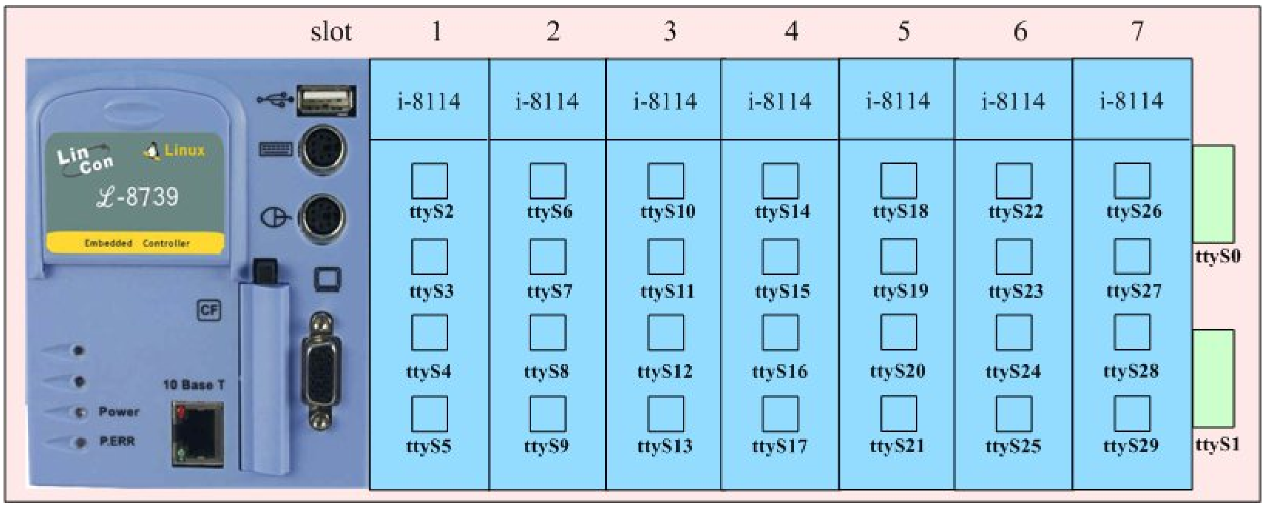


Fig.11-3

The demo - **i7kdio_8114.c** will illustrate how to use the I-8114 module in the LinCon-8000. In this demo, we will control the I-7044 (8 DO channels and 4 DI channels) module through the second serial port on the I-8114 plugged into the slot 2 on the LinCon-8000. The address and baudrate of the I-7044 module in the RS-485 network are 02 and 115200 separately. Fig.11-4 is the control diagram.

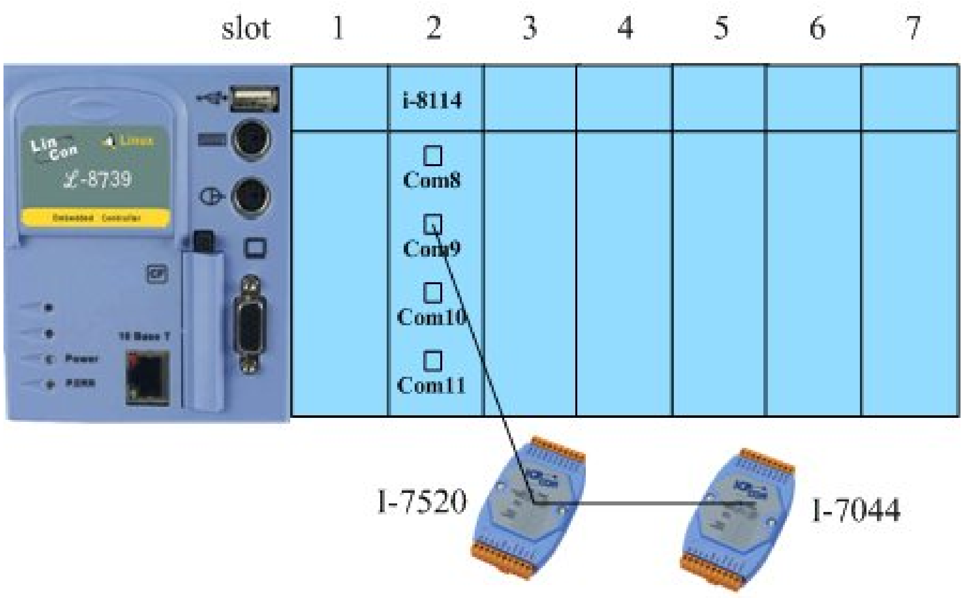
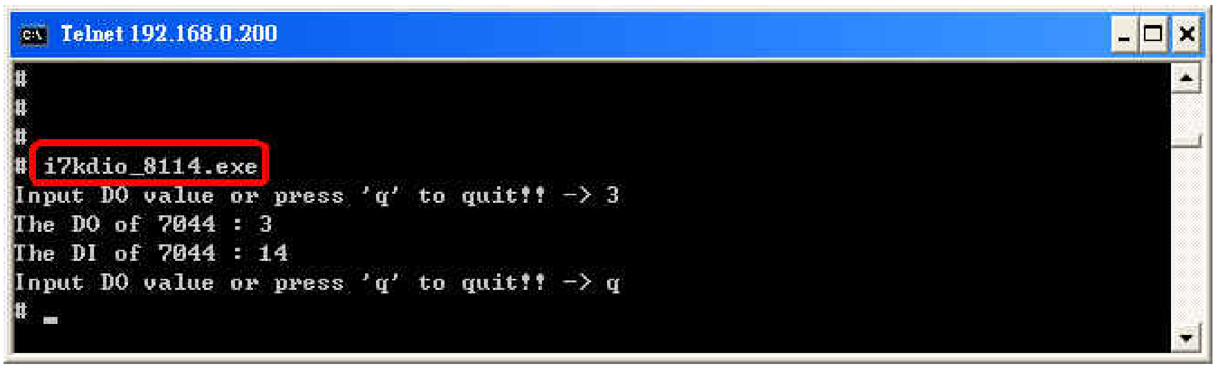


Fig.11-4

The result of this demo allows users to control DO channels' state and returns the state of the DI channels. The source code of this demo program is as follows:

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80], ans;
WORD wBuf[12];
float fBuf[12];

/* ------------------------------------------------------------------- */
int main()
{
    int   wRetVal,j=0;
    char i[10];

    // Check Open_Com9 in I-8114
    wRetVal = Open_Com(COM9, 115200, Data8Bit, NonParity, OneStopBit);
    if (wRetVal > 0) {
        printf("open port failed!\n");
        return (-1);
    }

    //   *****   7044 DO & DI Parameter *******
    wBuf[0] = 9;                    // COM Port
    wBuf[1] = 0x02;                 // Address
    wBuf[2] = 0x7044;               // ID
    wBuf[3] = 0;                    // CheckSum disable
    wBuf[4] = 100;                  // TimeOut , 100 msecond
    wBuf[6] = 0;                    // string debug
    // 7044 DO
    while(j!=113) {
        printf("Input DO value or press 'q' to quit!! -> ");
        scanf("%s",i);

        if (i[0]=='q') {
            wBuf[5] = 0;            // All DO Channels Off
            wRetVal = DigitalOut(wBuf, fBuf, szSend, szReceive);
            break;
        }

        j=atoi(i);
        if (j>=0 & j<=255)
            wBuf[5] = j;       // DO Channels On
        else if (j>255)
            wBuf[5] = 255;

        wRetVal = DigitalOut(wBuf, fBuf, szSend, szReceive);
        if (wRetVal)
            printf("DigitalOut_7044 Error !, Error Code=%d\n", wRetVal);

        printf("The DO of 7044 : %u \n", wBuf[5]);

        // 7044 DI
        DigitalIn(wBuf, fBuf, szSend, szReceive);
        printf("The DI of 7044 : %u \n", wBuf[5]);
    }
    Close_Com(COM9);

    return 0;
}
```

All the steps from programming to execution are the same as those in the section 7.1. The result of execution refers to Fig. 11.5.



```
#
#
#
# i7kdio_8114.exe
Input DO value or press 'q' to quit!! -> 3
The DO of 7044 : 3
The DI of 7044 : 14
Input DO value or press 'q' to quit!! -> q
# _
```

Fig. 11.5

11.2 Crash Free Support

If it is unfortunate, the LinCon-8000 is crashed and can't reboot. Please follow the steps to make the LinCon recover the normal state :

(1) Reboot the LinCon-8000.

(2) When the " **boot up screen** " shows up, connect the pin 1 and pin 5 of the second row of any slot with a wire immediately. (Refer to the Fig. 11-6 and the left side is down edge of LinCon Slot)

(3) The LinCon-8000 will boot up with its correct " **/etc** " directory built in and user's " **/etc** " directory will be placed to the path – " **/tmp/etc** ". When the LinCon-8000 boot up successfully, users should copy the correct files in the /etc to the path - /tmp/etc or correct the files in the /tmp/etc.

(4) After the correction, remove the connection wire and reboot LinCon-8000 again.

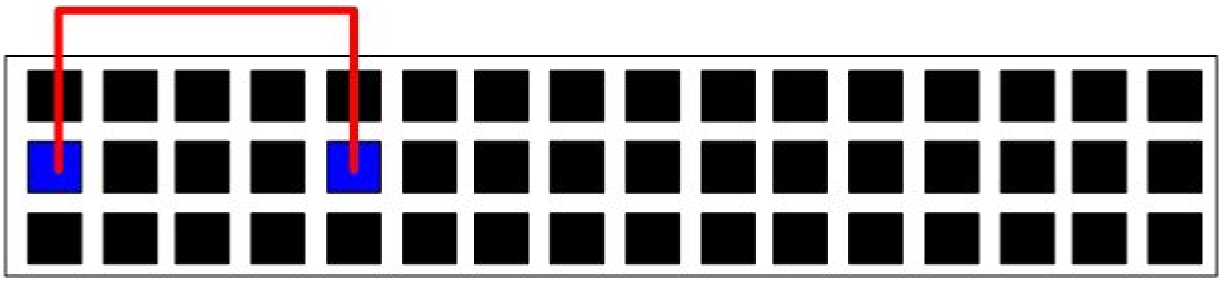


Fig. 11-6

11.3 GUI Funtion Support

Now " **X-window** " is supported in the LinCon-8000 and when the LinCon-8000 boot up, the GUI like " **Windows screen** " will show up. The most important thing is that users can write GUI programs and run them in the LinCon-8000. The GUI Library in the LinCon-8000 is provided with **GTK+ v1.2 & v2.0** Library. Therefore users can design their own " **SCADA** " screen by the GTK+ Library in the LinCon-8000. In the meanwhile, we provide some GUI demo programs to control I/O modules of ICPDAS and assist users to develop own GUI programs quickly. These demo programs are placed in the path – **C:\cygwin\LinCon8k\examples\gui** after users install the LinCon-8000 SDK. (Refer to the Fig. 11-7)

Except GTK+ GUI Function, **" Java GUI "** is also supported in the LinCon-8000. So if users are familiar with Java, users can also use Java to develop own GUI programs. But just Awe and Swing v1.1 elements below are supported in the LinCon-8000. To execute Java GUI program – Stylepad.jar in the LinCon-8000, users just type in **" java -jar Stylepad.jar -cp .:Stylepad.jar ".** Then it will take some time to run up the Java GUI program.

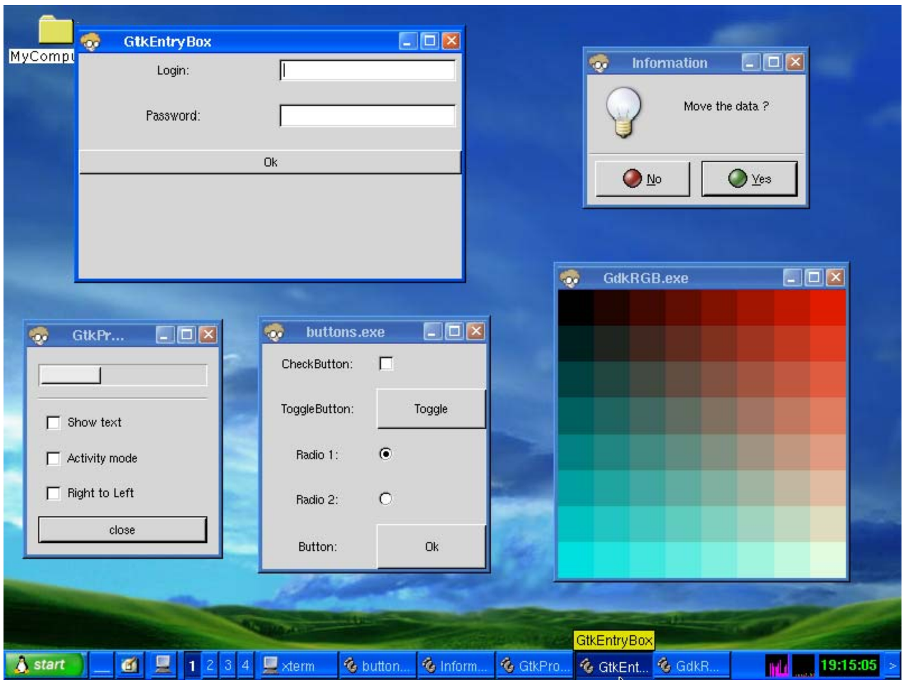


Fig. 11-7

11.3.1 How to boot LinCon-8000 without loading X-window

LinCon-8000 can boot without loading X-window by the steps as follows :

(1) Type " **cd /etc/rc2.d** " to into default run level.

(2) Type " **ls -al** " to see the S98Xserver link into ../init.d/startx.

(3) Type " **mv S98Xserver xS98Xserver** " to rename the S98Xserver for turn off X-window. Then exit and reboot LinCon-8000.

11.3.2 Enabling X-window load at boot time

If you type the " **ls -al /etc/rc2.d** " that can fine the link about ../init.d/startx, and then type the " **mv xS98Xserver S98Xserver** " to rename the xS98Xserver for turn on X-window or else if you can't fine any link about ../init.d/startx, and please follow the below steps :

(1) Type “ **cd /etc/rc2.d** “ to into default run level.

(2) Type ” **ln -s ../init.d/startx /etc/rc2.d/S98Xserver** “ to make a symbolic link into the script file of X-window for turn on X-window. Then exit and reboot LinCon-8000.

11.4 ScreenShot Support

There is a screenshot program — **“ fbshot ”** built in to let users to catch the LinCon-8000 screen conveniently. Users just type in **“ fbshot -d /dev/fb0 /mnt/hda/catch1.png ”** and the screen will be catched and saved to the file — /mnt/hda/catch1.png. If users want to take a look the picture, just type in **“ vi /mnt/hda/catch1.png”.** (Note : vi is placed in the path : /mnt/hda/opt/bin so users need to plug CF Card in the LinCon first.) If users want to know the detailed parameters of fbshot, just type in **“ fbshot --help ”**.

11.5 WebCAM Support

WebCAM is also supported in the LinCon-8000 and Logitech brand works successfully now. Other brands will need to do a test. Please follow the steps to make the Webcam work smoothly :

(1) Connect the webcam to the LinCon-8000 with **“ USB Interface “**.

(2) Reboot the LinCon-8000.

(3) Open a **“ Command Prompt “**. Type in **“ insmod pwcx.o “** to load the gqcam program decompressor and then type in **“ gqcam “** to see the webcam screen. If users want to know the detailed parameters of gqcam, just type in **“ gqcam --help ”**.

If users want to catch the picture through webcam, users can use gqcam program to do that. Please follow the steps as below :

(1) Click **“ File/Save Image… “**

(2) At **“ Gqcam: Save Image “** screen, input the path and file name in the **“ File Field “** and then click “ OK “ button.

11.6 Screen Resolution Setting

There are three modes to adjust the screen resolution of LinCon and they are **640*480**, **800*600**, **1024*768**. Users can edit the file : **/etc/init.d/fbman** to modify the setting and follow the below steps :

(1) When users open the file : **/etc/init.d/fbman**, users can see the following lines :

```
#/usr/sbin/fbset -n -a -g 640 480 640 480 32
/usr/sbin/fbset -n -a -g 800 600 800 600 32
#/usr/sbin/fbset -n -a -g 1024 768 1024 768 16
```

It means that the resolution setting is 800*600.

(2) If users want to change the setting to be **1024*768**, just remove the "#" mark in line 3 and add the "#" mark in line 2. Please see the following setting result :

```
#/usr/sbin/fbset -n -a -g 640 480 640 480 32
#/usr/sbin/fbset -n -a -g 800 600 800 600 32
/usr/sbin/fbset -n -a -g 1024 768 1024 768 16
```

After rebooting the LinCon, the setting will work.

11.7 Network Support

There are many network functions already built in the LinCon-8000. Here are the network functions supported in the LinCon-8000 :

(1) Support UPnP :

UPnP is **" Universal Plug and Play "** and allows automatic discovery and control of services available on the network from other devices without user intervention. Devices that act as servers can advertise their services to clients. Client systems, known as control points, can search for specific services on the network. When they find the devices with the desiredservices, the control points can retrieve detailed descriptions of the devices and services and interact from that point on.

(2) Support VPN

VPN is **“ Virtual Private Network “** and describes a network that includes secure remote access for client computers. It can be explained best by looking at its parts. **“ Virtual “** describes the fact that the network doesn't need to be physically connected directly. The **“ Private “** confirms that the data is encrypted and can only be viewed by a defined group. The last word, **“ Network “**, means that the users configured for VPN can be connected and share files or information. So it's extremely difficult for anyone to snoop on confidential information through VPN. (Refer to the Fig. 11-8)

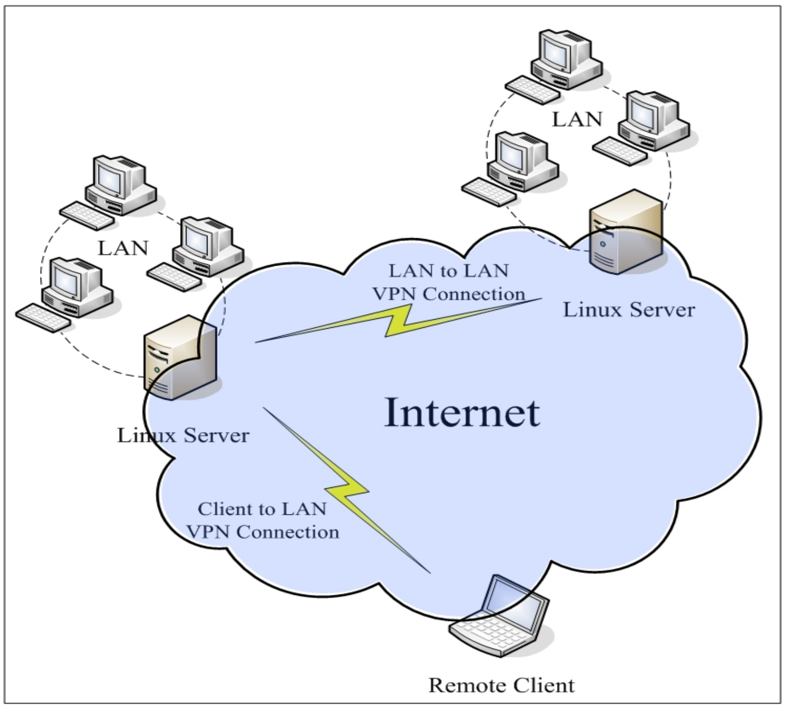


Fig. 11-8

(3) Support QoS

QoS is **“ Quality of Service “**. It means when the kernel has several packets to send out over a network device, it has to decide which ones to send first, which ones to delay, and which ones to drop. With Linux QoS subsystem, it is possible to make very flexible traffic control. Let users be able to control flow rate of assigned port to improve the network quality.

(4) Support Wireless LAN

“ Wireless communication ” is a networking technology allowing the connection of computers without any wires and cables, mostly using **radio** technology (and sometime **infrared**). It's called LAN because the range targeted is small (generally within an office, a building, a store, a small campus, a house...). This technology is slowly growing and Linux is able to take advantage of some of the wireless networks available.

If users plug wireless card in the LinCon, users need to modify /etc/network/interfaces.

(5) Support Dual LAN

Dual LAN means that users can combine wireless and cable network together through LinCon-8000. Therefore the communication between Cable LAN and Wireless LAN. If one of these LANs can connect to internet, then all the PC can connect to internet. (Refer to Fig. 11-9)

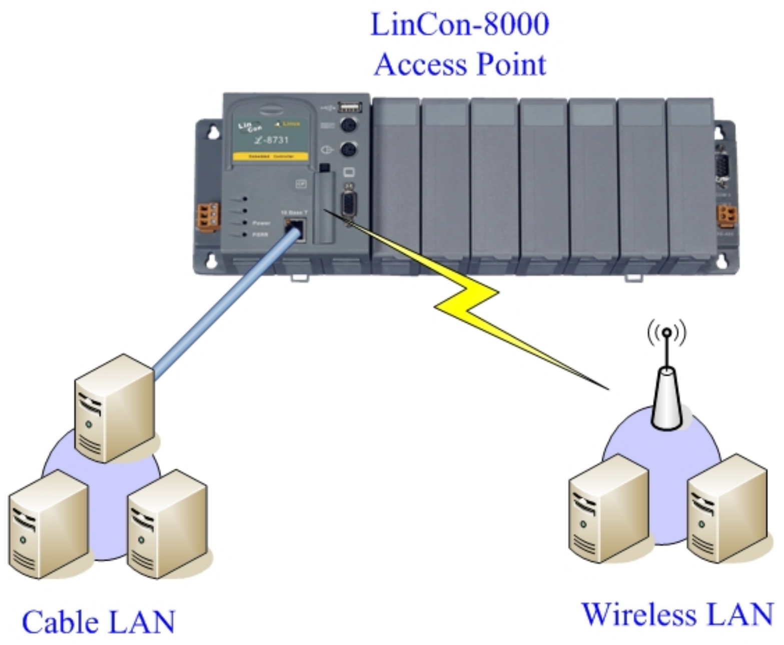


Fig. 11-9

(6) Support BlueTooth

The Bluetooth wireless technology is a worldwide specification for a small-form factor, low-cost radio solution that provides links between mobile computers, mobile phones, other portable handheld devices, and connectivity to the Internet. Now **“ BlueZ ”** is built in the LinCon-8000 and provides support for the core Bluetooth layers and protocols. It is flexible, efficient and uses a modular implementation.

(7) Support Modem / GPRS / ADSL

LinCon-8000 can be connected to the Internet with “ Modem “, “ GPRS “ or “ ADSL “ mode. The setup method is described separately as follows :

[Modem]

[GPRS]

The default GPRS baudrate is **“ 115200 “** in the LinCon, so if users finish the setting of gprs modem and connect the gprs modem to the COM 2 of LinCon-8000, just type in **“ pppd call wavecom ”** and then LinCon-8000 will be connected to the internet automatically. Remember that the network interface card of LinCon should stop first, just type in **“ ifdown eth0 ”** to stop it. If users type in “ ifconfig “ will see the **“ ppp0 ”** option.

[ADSL]

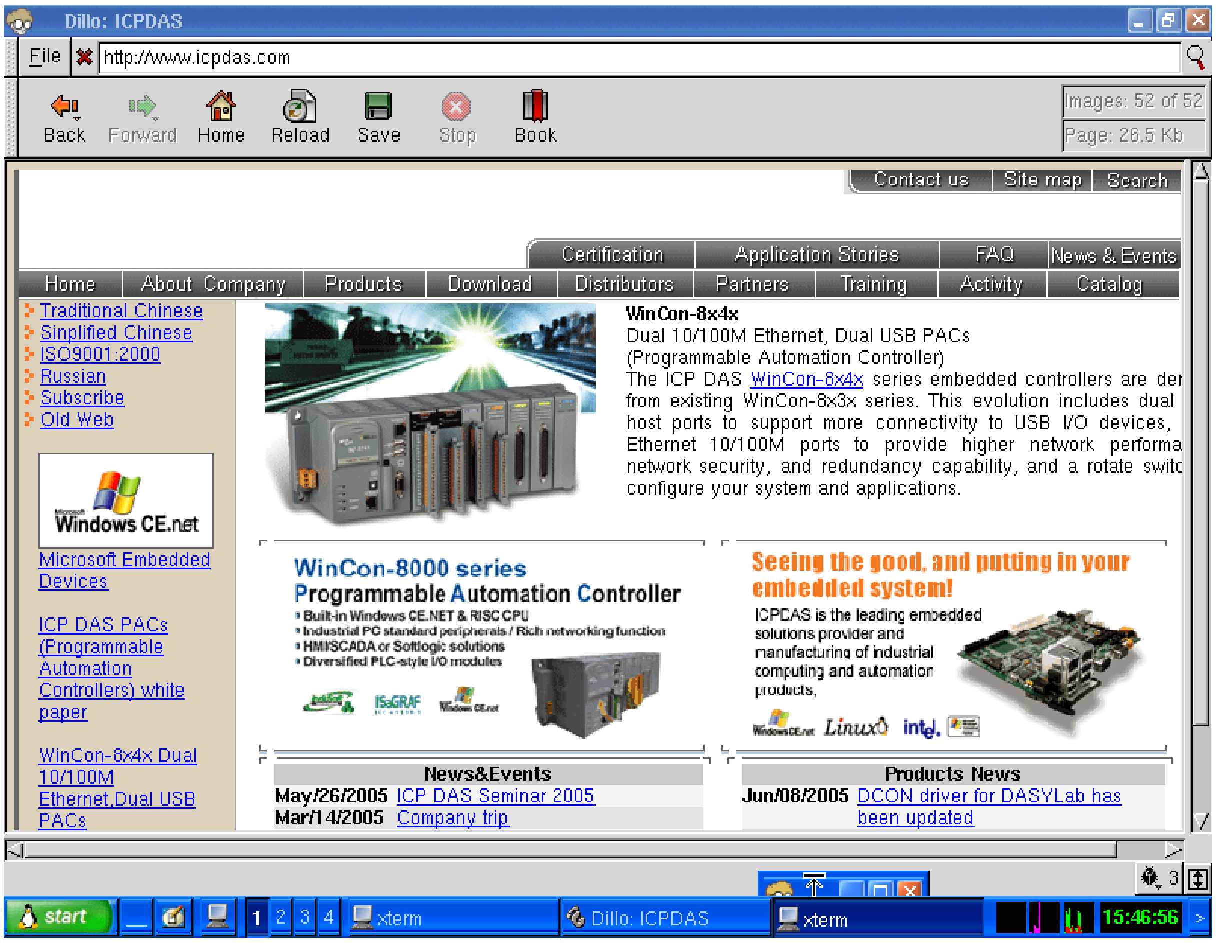
Users need to type in **“ adsl-setup ”** first to setup ADSL options. After that, users need to type in **“ adsl-connect ”** to make LinCon-8000 connect to the internet. If users want to stop adsl connection, just type in **“ adsl-stop ”**.

(8) Support Firewall (iptables function)

A firewall can controls outside access to a local network, locking out intruders to ensure your systems and data safe on the inside, even against an intentional attack from outside network.

(9) Provide Web Browser

Users can see the Web Page by using the Web Browser built in the LinCon-8000. Just type in **“ dillo ”** to open the web browser and input the web site address. (Refer to Fig 11-10) (Note : dillo is placed in the path : /mnt/hda/opt/bin so users need to plug CF Card in the LinCon first.)



WinCon-8x4x
Dual 10/100M Ethernet, Dual USB PACs
(Programmable Automation Controller)
The ICP DAS WinCon-8x4x series embedded controllers are der
from existing WinCon-8x3x series. This evolution includes dual
host ports to support more connectivity to USB I/O devices,
Ethernet 10/100M ports to provide higher network performa
network security, and redundancy capability, and a rotate switc
configure your system and applications.

WinCon-8000 series
Programmable Automation Controller
• Built-in Windows CE.NET & RISC CPU
• Industrial PC standard peripherals / Rich networking function
• HMI/SCADA or Softlogic solutions
• Diversified PLC-style I/O modules

Seeing the good, and putting in your embedded system!
ICPDAS is the leading embedded solutions provider and manufacturing of industrial computing and automation products.

News&Events
May/26/2005 ICP DAS Seminar 2005
Mar/14/2005 Company trip

Products News
Jun/08/2005 DCON driver for DASYLab has been updated

Fig 11-10

(10) Provide Apache Server

The Web Server — “ **Apache Server** “ has been built in the LinCon-8000 and it will be started automatically when boot up. These files are placed in the path — **/opt/apache2**. Users can type like “ **http://192.168.0.200** “ to connect to the web server in the LinCon-8000. If it returns a successful web page, it means that the web server in the LinCon-8000 has been started. The index web page of Apache Server is in the path : “ **/opt/apache2/htdocs/** “.

These files placed in the CF Card are full functions of Apache Server. So if users want to use other function of Apache Server that are not supported in the LinCon-8000, users just copy them to the path : **/opt/apache2** and reboot.

11.8 Other Optional Function

These optional functions are listed below all supported in the LinCon-8000. Users can choose which function to be used in the LinCon-8000 and just copy the corresponding file directory to the " opt " directory of CF Card. Then reboot LinCon-8000 and the function users choose will work automatically.

(1) Support MySQL

MySQL is a small database server and it is " Relational DataBase Management System (RDBMS) ". By using MySQL, users can add or delete data easily and it is open source and supports many platforms, like UNIX、Linux or Windows operating system. If users want to use MySQL in the LinCon-8000, remember to copy the " mysql " directory to the " opt " directory of CF Card and reboot LinCon-8000.

(2) Support PHP

PHP is a kind of " open source script language " and used to design active web page. When PHP combined with MySQL are cross-platform. It means that users can develop in Windows and serve on a Linux platform. (Refer to Fig 11-11)

PHP has been built in the LinCon-8000 Kernel so users just boot up LinCon-8000 and can use PHP directly in the LinCon-8000.

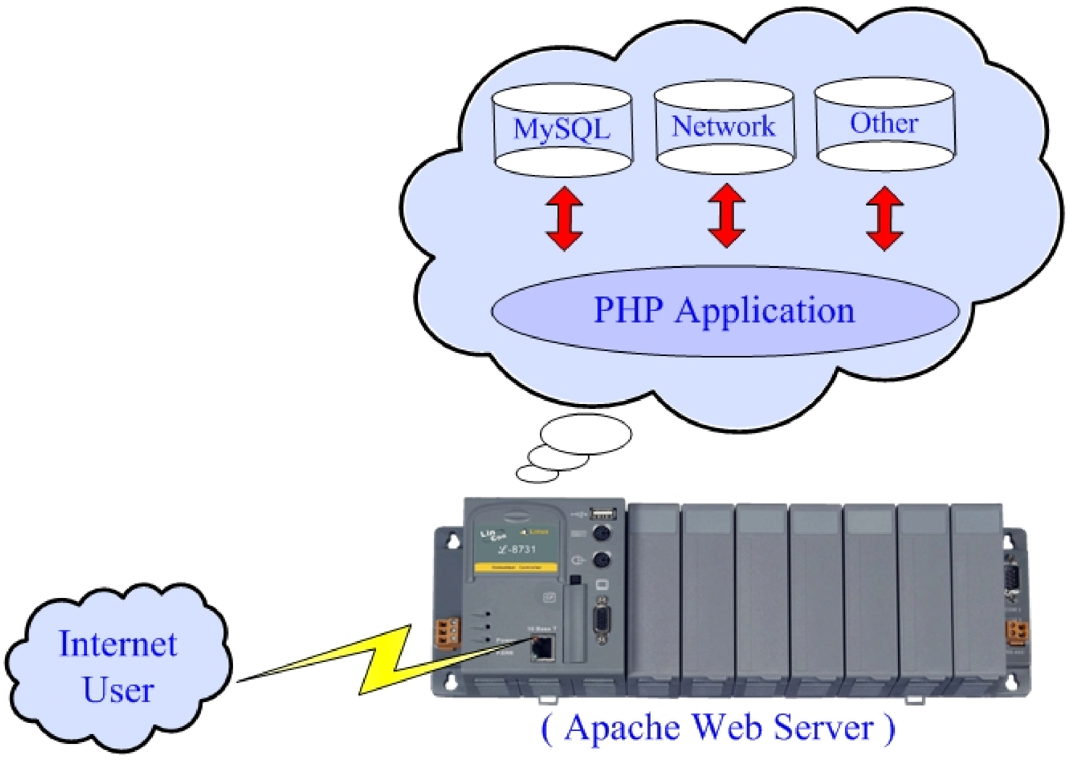


Fig 11-11

(3) Support Perl

Perl（Practical Extraction and Report Language）is also a “ open source script language “ and has been built in the LinCon-8000 Kernel so users just boot up LinCon-8000 and can use Perl directly in the LinCon-8000.

Appendix A. Service Information

This appendix will show how to contact ICPDAS when you have problems in the LinCon-8000 or other products.

Internet Service :

The internet service provided by ICPDAS will be satisfied and it includes Technical Support, Driver Update, OS_Image, LinCon_SDK and User's Manual Download etc. Users can refer to the following web site to get more information :

1. ICPDAS Web Site :

http://www.icpdas.com/

2. LinCon Introduction :

http://www.icpdas.com/products/PAC/lincon-8000/introduction.htm

3. Software Download :

http://www.icpdas.com/download/index.htm

4. Java Supported Document :

http://w3.icpdas.com/moki/

5. E-mail for Technical Support :

service@icpdas.com

Manual Revision :

Manual Edition	Revision Date	Revision Details
v 2.0	2004.12	1. Modifiy the LinCon_SDK installation path 2. Add demo description in chapter 7
v 3.0	2005.01	1. Add CF Card & USB Device usage method 2. Add I-talk Utility description 3. Add Java Demo Programs 4. Add i-8114 & i-8112 module demo
v 4.0	2005.06	1. Add Crash Free Support 2. Add GUI Function Support 3. Add VGA Resolution Modification Support 4. Add i-Talk utility support 7k/8k/87k modules 5. Add WebCAM support 6. Add ScreenShot support 7. Add Network support 8. Add MySQL support 9. Add PHP support 10. Add Perl support 11. Add LinCon Slot Java Demo
V4.1	2005.12	1.Add WDT Function 2.Modify I-8090 Function 3.Modify I-8091 Function 4.Add Timer function Demo
V4.2	2006.07	1.Add DO/DI ReadBack Function 2.Add DO Bit Write Function
V4.3	2007.07	1.Add UDIO_WriteConfig_16 Function 2. Add UDIO_ReadConfig_16 Function 3. Add UDIO_DO16 Function 4. Add UDIO_DI16 Function 5. Add auto run program support 6. Add turn on/off X-window support 7. Add SDK for Linux system support 8. Modify the LinCon SDK download path
V4.4	2007.08	1. Add sio_open Function 2. Add sio_close Function
V4.5	2008.12	1. Add DIO function for I-7K/8K/87K modules via serial port. 2. Add AIO function for I-7K/8K/87K modules via serial port.
V4.6	2009. 02	1. Add Read_SN Function 2. Add Send_Binary Function 3. Add Receive_Binary Function