Software Guide

ICP DAS LP-8x4x SDK

Implement industry control with Linux Technique

(Version 1.8)

Warranty

All products manufactured by ICP DAS Inc. are warranted against defective materials for a period of one year from the date of delivery to the original purchaser.

Warning

ICP DAS Inc. assume no liability for any damage consequent to the use of this product. ICP DAS Inc. reserves the right to change this manual at any time without notice. The information furnished by ICP DAS Inc. is believed to be accurate and reliable. However, no responsibility is assumed by ICP DAS Co., Ltd. for its use, nor for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademarks

Names are used for identification purposes only and maybe registered trademarks of their respective companies.

License

The user can use, modify and backup this software **on a single machine.** The user may not reproduce, transfer or distribute this software, or any copy, in whole or in part.

Contents

1. Introduction

Nowadays, Linux has been adopted widely by many users because of the properties of stability, open source, and free of charge. In the mean while, owing to the great supports from more and more companies and the mature development, Linux is now becoming one of the most popular OS on the market. Furthermore the hardware requirements of Linux OS in embedded system is not high, just only 386 CPU or better and 8 MB RAM. Therefore, besides Win CE of Microsoft, Linux has been already become 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 LP-8x4x embbeded controller has been published to be a new generation product from ICP DAS and the Embedded-Linux OS has been adopted into the LP-8x4x. The LP-8x4x's main purpose is to allow the numerous enthusiastic linux users to control their own embedded systems easily within the Linux Environment.

LP-8x4x is the second generation PAC of ICP DAS. It equips with a powerful CPU module running a Linux kernel 2.6 operating system, various interfaces (VGA, USB, Ethernet, RS-232/485) and slots for high performance parallel I/O modules (high profile I-8K series) and serial-type I/O modules (high profile I-87K I/O modules).

Compared with the first generation LinCon-8000, it not only improves the CPU performance and upgrades OS (from Linux kernel 2.4 to Linux kernel 2.6), but also adds many reliability features, such as dual LAN, redundant power input, dual battery backup SRAM, etc. Those make LP-8x4x to be the most powerful control system.

ICP DAS provides the library file — libi8k.a which includes all the functions from the I-7000/8000/87000 series modules which are used in the LP-8x4x Embedded Controller. The libi8k.a is desiged specially for the I-7000/8000/87000 series modules on the Linux platform for use in the LP-8x4x. Users can easily develop applications in the LP-8x4x 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 LP-8x4x embedded controller are depicted in figure 1-1, which includes a **VGA**, **USB(Card Reader, Camera …)**, **Mouse**, **Keyboard**, **microSD/microSDHC card**, **Series ports(RS-232, RS-485), Ethernet(Hub…) and many I/O slots** in the picture. Presently, HTTP、FTP、Telnet、SSH、SFTP Servers are built in and users can transfer files or use remote control with the LP-8x4x more conveniently. In network communication, **wireless**, **Bluetooth** transfer and **Modem**, **GPRS, ADSL, Firewall** are also supported. Fig. 1-2 illustrates hardware architecture of the LP-8x4x.

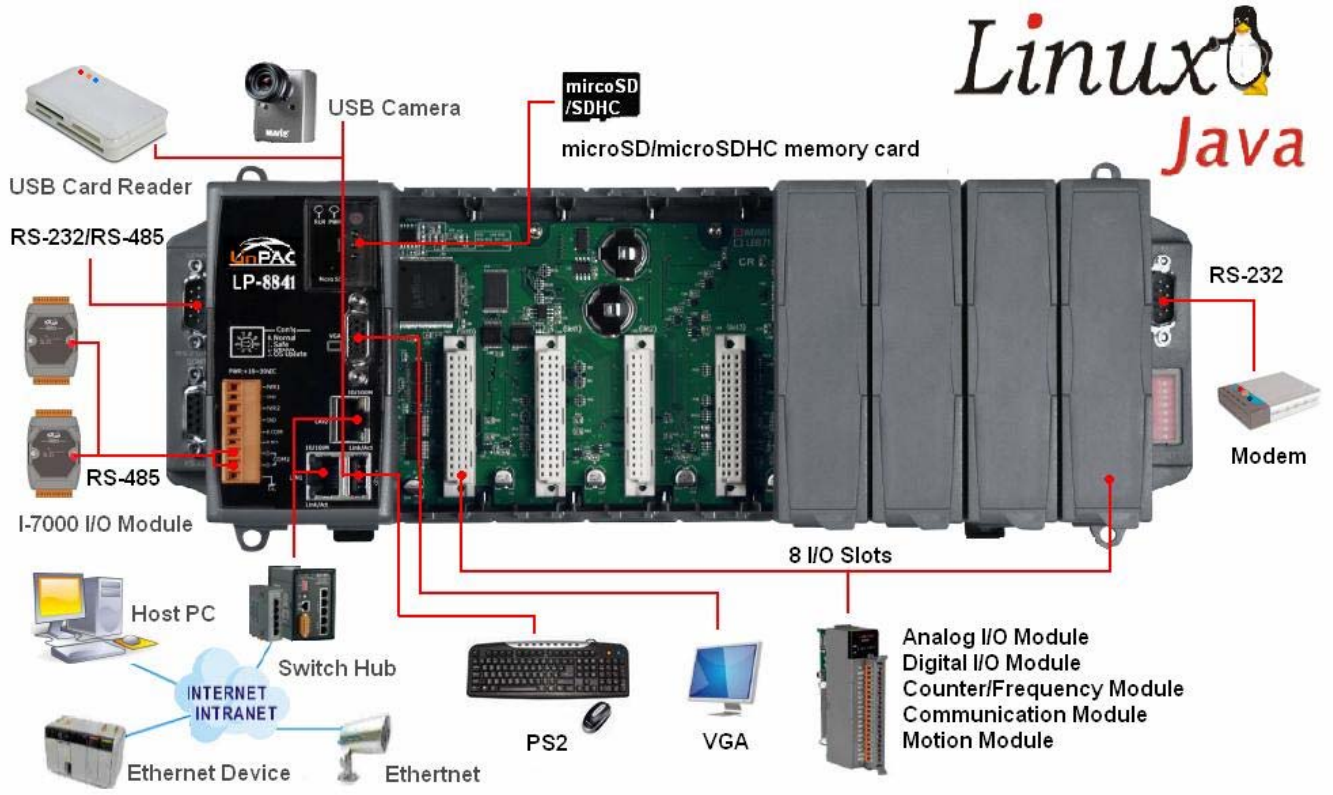


Fig. 1-1

LP-8x41 Series

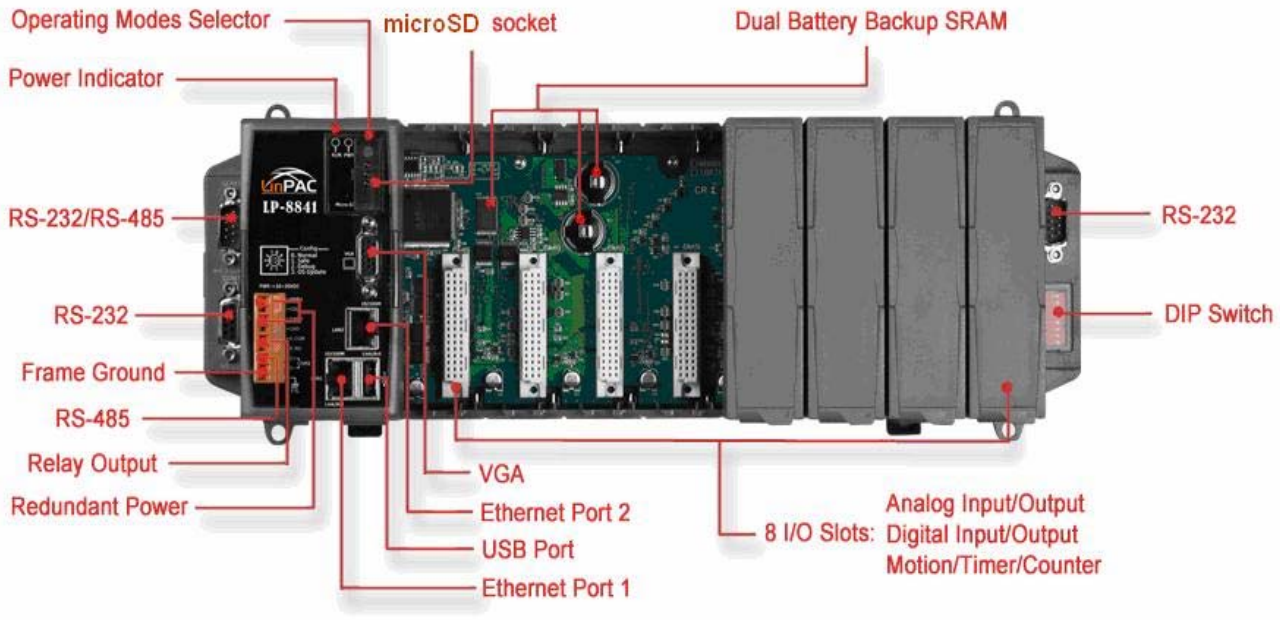


Fig. 1-2

2. Installation of LP-8x4x SDK

"LP-8x4x SDK" consists of the following major items.

- LinPAC SDK library files
- LinPAC SDK include files
- Demo files
- GNU ToolChain

From ftp://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x4x/sdk/, users can download the latest version of LP-8x4x SDK (hereinafter referred to as LP-8000 or LP-8K). Then follows below steps to install the development toolkit provided by ICP DAS for the application development of the LP-8x4x embedded controller platform easily.

2.1 Quick Installation of LP-8x4x SDK

(1) Quick Installation Guide for Windows

1. Please insert the installation CD into your CD-ROM driver.
2. Run the "lp8x4x_sdk_for_windows.exe" file under the folder \napdos\lp-8x4x\SDK\. Then click on the "Next" button, refer to Fig. 2-1.
3. Choose the option of "I accept the agreement" and click the "next" button, refer to Fig. 2-2 below.

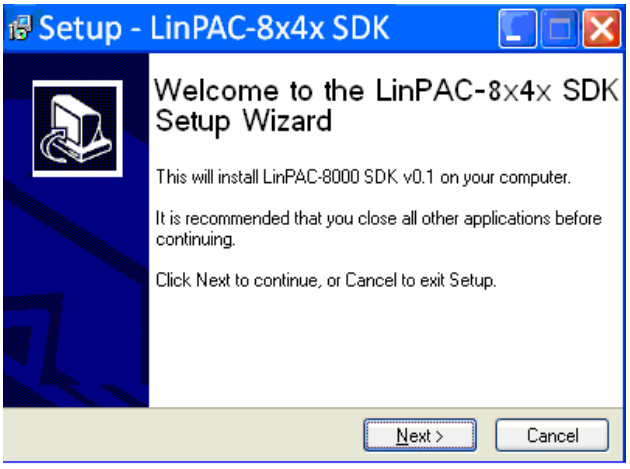


Fig. 2 -1

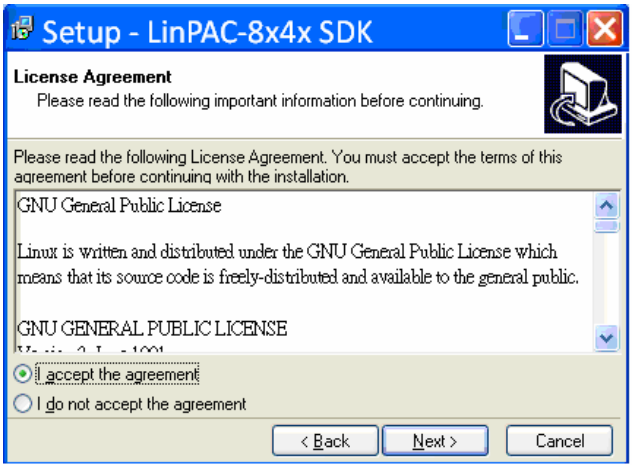


Fig. 2-2

4. To starting install the LP-8x4x SDK, refer to Fig 2-3.
5. After successfully installing the software, please click on the "Finish" button to finish the development toolkit installation, refer to Fig. 2-4.

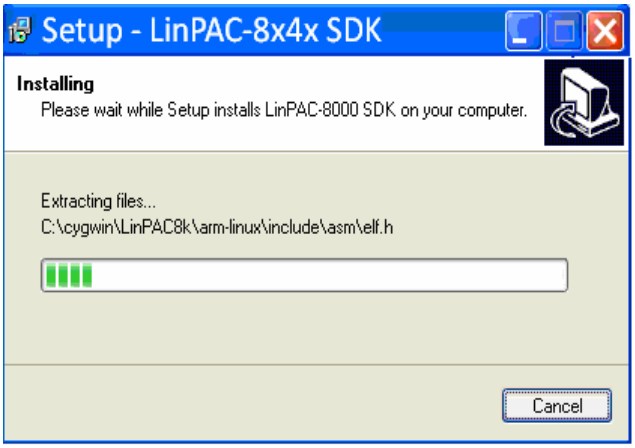


Fig. 2-3

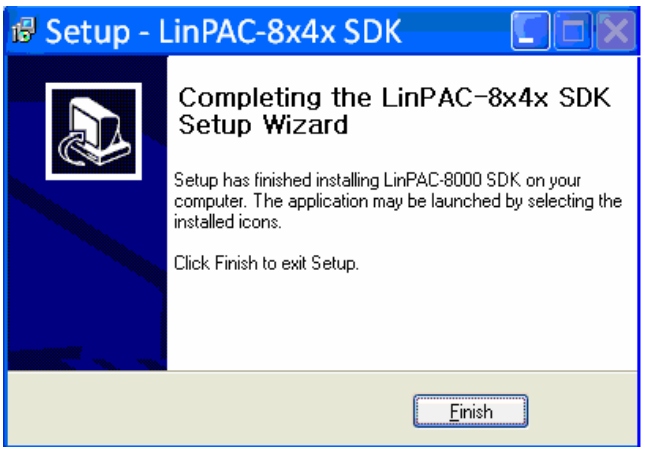


Fig. 2-4

6. Open the "**C:\cygwin\LinCon8k**" folder and see the content. Refer to Fig 2-5.

Fig. 2-5

7. Start using the "LP-8x4x Build Environment" by double clicking the shortcut for the "**LP-8x4x Build Environment**" on the desktop or by clicking through " Start ">" Programs ">" ICPDAS ">" LP-8x4x SDK ">" LP-8x4x Build Environment " icon. Then a special DOSBOX will be displayed in which we can compile applications for the LP-8x4x. refer to Fig. 2-6.

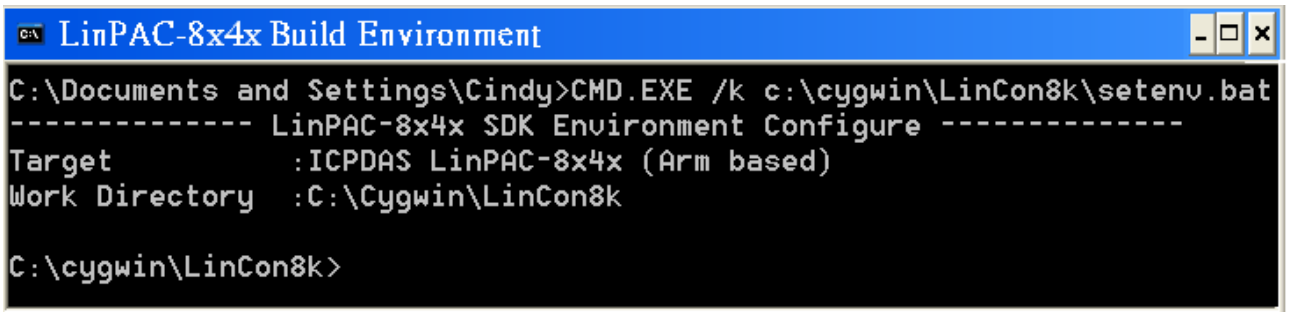


Fig. 2-6

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**"

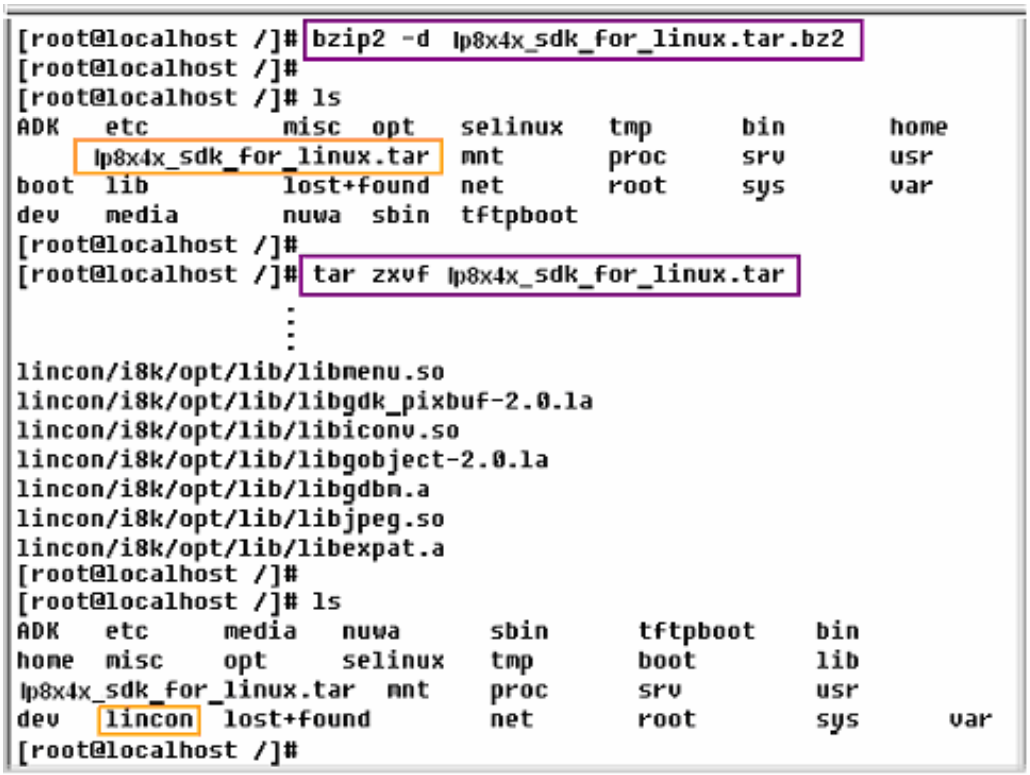
The demo path is "**C:\cygwin\LinCon8k\examples**".

(2) Quick Installation Guide for Linux

1. Before you install LinPAC-8x4x SDK, you must complete several tasks as the root user by ‘sudo’ or ‘su’ command.
2. Download the “lp8x4x_sdk_for_linux.tar.bz2” file under the folder \napdos\lp-8x4x\SDK\ or visit our website to download the latest LP-8x4x SDK.

   $ bzip2 -d lp8x4x_sdk_for_linux.tar.bz2

   $ tar zxvf lp8x4x_sdk_for_linux.tar



4. To run the shell startup script and set the environment variables, enter the following command:

   $. /lincon/linpac.sh

2.2 The LP-8x4x SDK Introduction

In this section, we will discuss some techniques that are adopted in the LP-8x4x. Through our detailed explanations, users can learn how to use the LP-8x4x easily. LP-8x4x 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 LP-8x4x applications quickly. Therefore after you have written your applications, you can compile them through the LP-8x4x SDK into executable files that can be run in your LP-8x4x 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 LinPAC-8x4x SDK

- **For Windows system : (**Extract the .exe file into to the **C: driver.)**
  linpacsdk_for_windows.exe as below:

ftp://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x4x/sdk/linpacsdk_for_windows.exe

- **For Linux system : (**Extract the .bz2 file into to the **root (/) directory.)**
  linpacsdk_for_linux.tar.bz2 as below:

ftp://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x4x/sdk/linpacsdk_for_linux.tar.bz2

Note: We recommend user to change user ID to become root by 'sudo' or 'su' command.

3.The Architecture of LIBI8K.A in the LP-8x4x

The **libi8k.a** is a library file that is designed for I7000/8000/87000 applications running in the LP-8x4x 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：

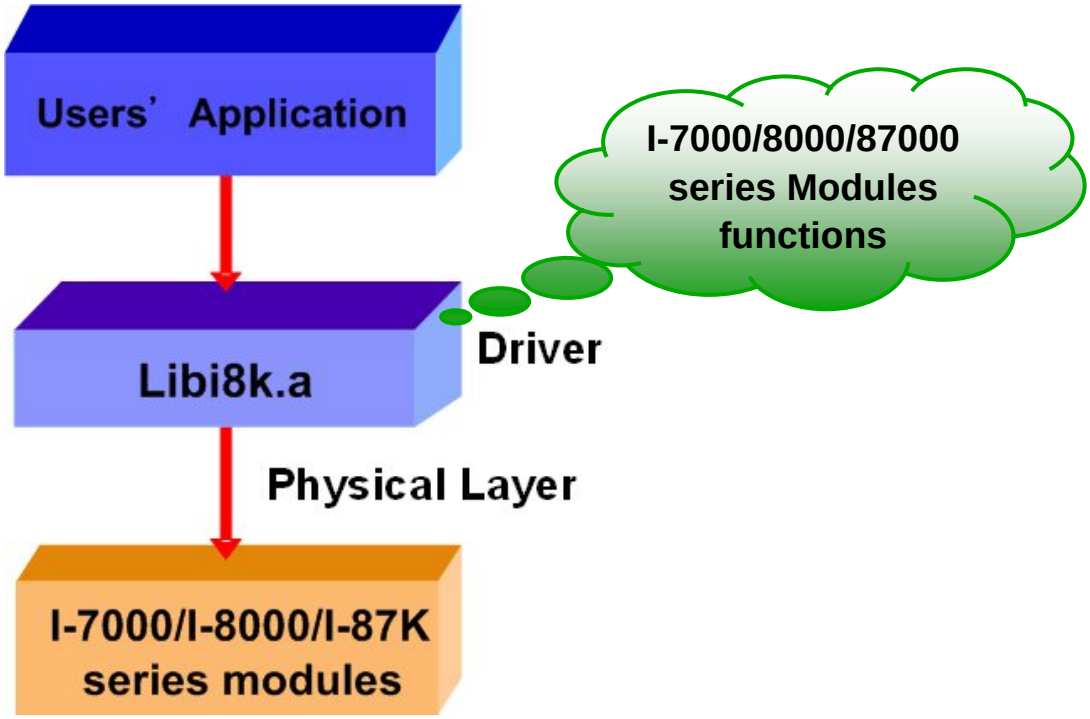


Fig. 3-1

Functions for LP-8x4x 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 LP-8x4x. 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. LP-8x4x System Settings

In this section, we will introduce how to setup the LP-8x4x configuration. Let users can use the LP-8x4x more easily.

4.1 Settings for the LP-8x4x Network

The LP-8x4x network setting includes two ways. One is **DHCP** and the other is "**Assigned IP**". DHCP is the default setting after the LP-8x4x 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 LP-8x4x 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)

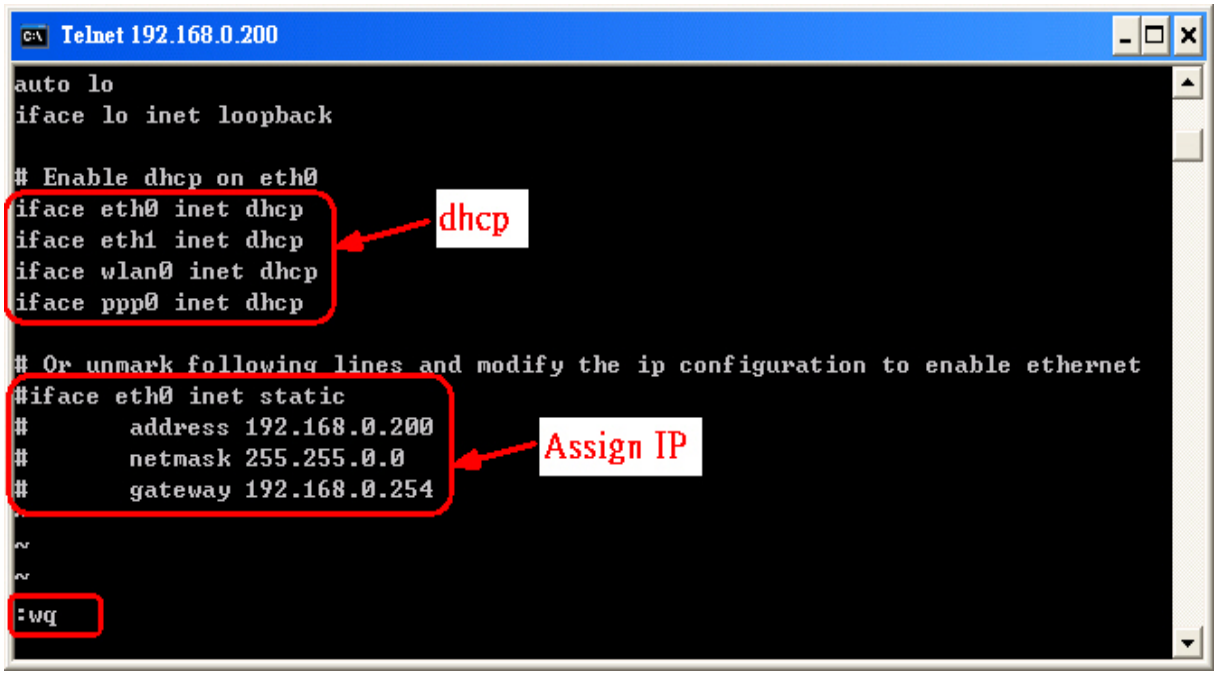


Fig 4-1

(2) Using “Assigned IP” :

Boot up LP-8x4x 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)

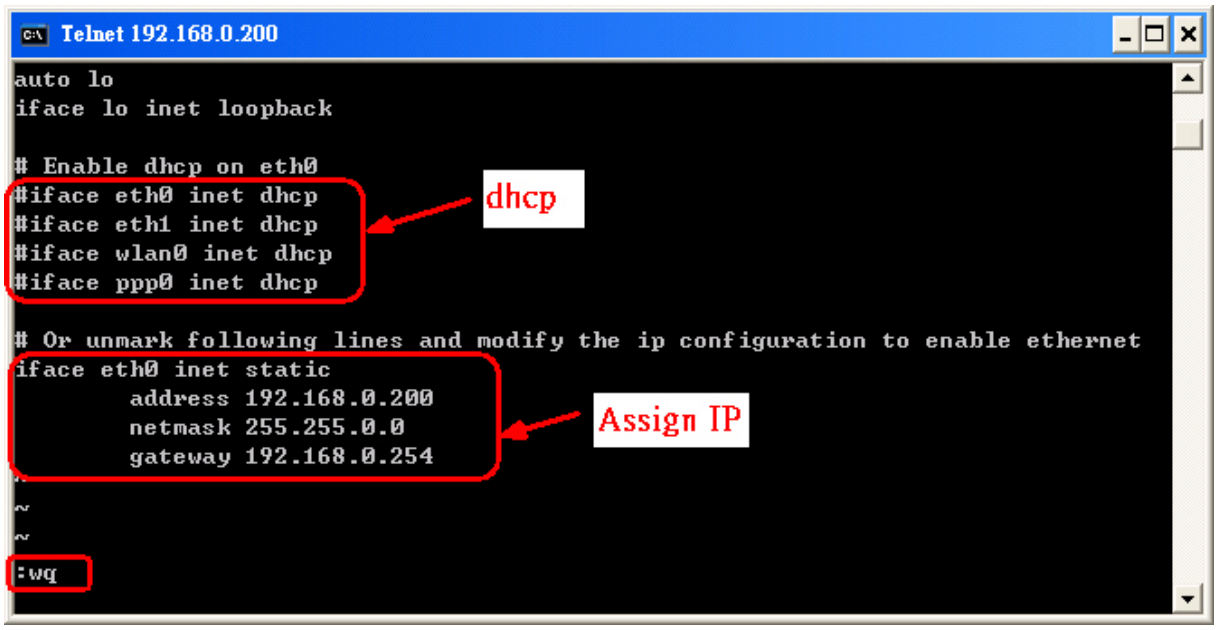


Fig 4-2

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

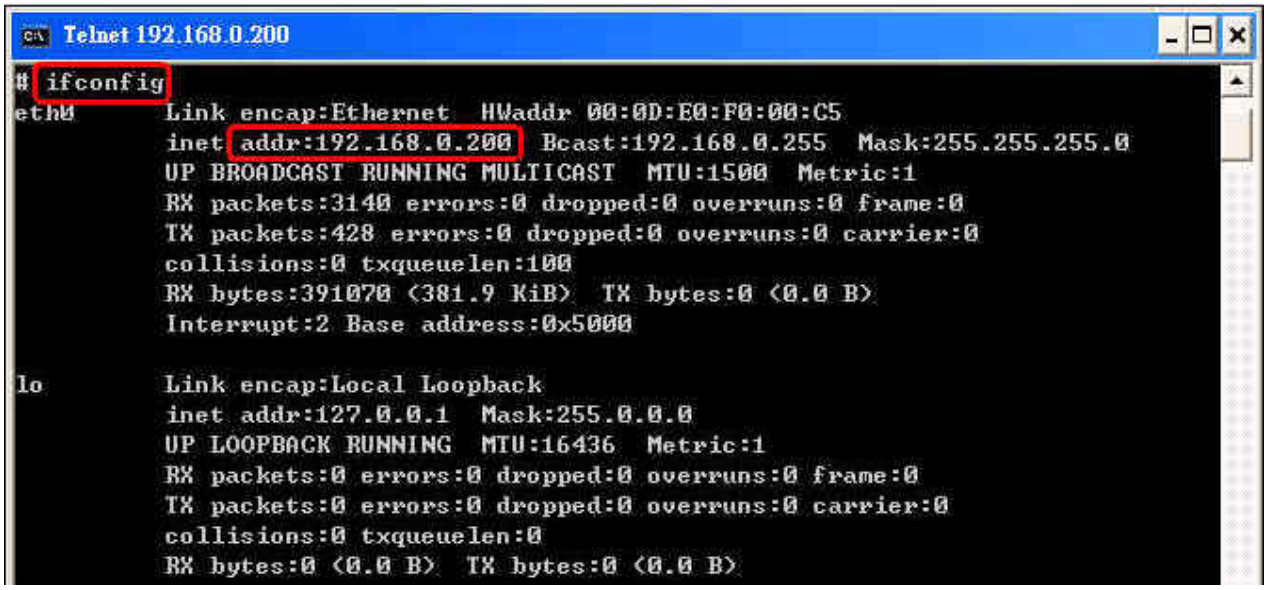


Fig 4-3

4.1.2 Setting of DNS

Boot up LP-8x4x 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 LP-8x4x to make the setting work. (Refer to the Fig 4-4)

Fig 4-4

4.2 microSD Card Usage

Users can access the files of microSD card in the **/mnt/hda** directory (Refer to Fig 4-5).

```
# mount
rootfs on / type rootfs (rw)
/dev/root on / type jffs2 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
tmpfs on /var type tmpfs (rw)
/dev/mmcblk0p1 on /mnt/hda type vfat (rw,fmask=0022,dmask=0022,codepage=cp437,iocharset=iso8859-1)
/dev/ramU on /mnt/ramfs type minix (rw)
#
```

Fig 4-5

When using the microSD card, pay attention to the following notes:

1. Umount the microSD card before unplugging it.
2. Please do not power off or reboot the LP-8x4x while data is being written to or read from the microSD card.
3. The microSD memory must be formatted in the VFAT/EXT2/EXT3 file system.

4.2.1 Mount microSD Card

If want to use the microSD card, you can insert the microSD card into the socket in the LP-8x4x (Refer to Fig. 1-3). It will be auto-mounted in the LP-8x4x at boot time, and you can access the files of SD card in the **/mnt/hda** directory.

If not, type in “**/etc/init.d/sd start** ”, user can mount microSD card by manual.

4.2.2 Umount microSD Card

Before you want to pull out the microSD card from the LP-8x4x, please type the following steps:

(1) **/etc/init.d/startx stop**

(2) **/etc/init.d/apachect1 stop**

(3) **umount /mnt/hda**

Then you can unplug the microSD card safely to prevent the damage to microSD card.

4.2.3 Scan and repair microSD Card

The microSD card at boot will be named " **/dev/mmcblk0p1** ". User could be umount microSD card first before scan or repair microSD card.

- **blockdev** : call block device ioctls from the command line
  ex. blockdev --report /dev/mmcblk0p1 (print a report for device)
  blockdev -v --getra --getbz /dev/mmcblk0p1 (get readhead and blocksize)
- **fsck.minix** : perform a consistency check for the Linux MINIX filesystem
  ex. fsck.minix -r /dev/mmcblk0p1 (performs interactive repairs)
  fsck.minix -s /dev/mmcblk0p1 (outputs super-block information)
- **fsck.vfat** : check and repair MS-DOS file systems
  ex. fsck.vfat -a /dev/mmcblk0p1 (automatically repair the file system)
  fsck.vfat -l /dev/mmcblk0p1 (list path names of files being processed)
- **mkfs** : build a Linux file system on a device, usually a hard disk partition.
  ex. mkfs -t vfat /dev/mmcblk0p1 (specifies the type of file system to be built)
  mkfs -c vfat /dev/mmcblk0p1
  (check the device for bad blocks before building the file system)
- **mkfs.minix** : make a MINIX filesystem
  ex. mkfs.minix /dev/mmcblk0p1 (create a Linux MINIX file-system)
  mkfs.minix -c /dev/mmcblk0p1
  (check the device for bad blocks before creating the file system)
- **mkfs.vfat** : make an MS-DOS filesystem
  ex. mkfs.vfat -A /dev/mmcblk0p1 (use Atari variation of the MS-DOS filesystem)
  mkfs.vfat -v /dev/mmcblk0p1 (verbose execution)

4.3 USB Storage Device Usage

Users need to mount the USB storage device to the LP-8x4x, before they can access the USB storage device. This is because it will not auto-mount the USB storage device in the LP-8x4x

4.3.1 Mount USB Storage 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 storage device to the usb directory and type " **ls /mnt/usb** " to see the content of USB storage device. (Refer to Fig 4-6)

```
# mkdir /mnt/usb
#
# cat /proc/diskstats | grep sda*
   8   0 sda 37 62 106 260 0 0 0 0 0 254 260
   8   1 sda1 98 98 0 0
#
# mount /dev/sda1 /mnt/usb
# mount
rootfs on / type rootfs (rw)
/dev/root on / type jffs2 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
tmpfs on /var type tmpfs (rw)
shmfs on /dev/shm type tmpfs (rw)
usbfs on /proc/bus/usb type usbfs (rw)
/dev/mmcblk0p1 on /mnt/hda type vfat
(rw,fmask=0022,dmask=0022,codepage=cp437,iocharset=iso8859-1)
/dev/ram0 on /mnt/ramfs type minix (rw)
/dev/sda1 on /mnt/usb type vfat
(rw,fmask=0022,dmask=0022,codepage=cp437,iocharset=iso8859-1)
#
# ls /mnt/usb
0429.doc    2009.avi
```

Fig 4-6

4.3.2 Umount USB Storage Device

Before users pull out the USB storage device from the LP-8x4x, users need to type the " **umount /mnt/usb** " command first. Then pull out the USB storage device to prevent any damage to usb storage device.

4.4 Adjust VGA Resolution

There are two modes -- **640x480**、**800x600** supported in the LinPAC 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 640x480. First, add **" # "** in the 800x600 column and then remove **" # "** in the 640x480 column. Type **" :wq "** to save the setting. (Refer to Fig 4-7)

❑ Open the file : /etc/init.d/fbman, user will see the following lines:

#/usr/sbin/fbset -n 640x480-60

/usr/sbin/fbset -n **800x600**-70

It means that the resolution setting is 800x600.

❑ If user want to change the setting to be **640*480**, please see the following setting result :

/usr/sbin/fbset -n 640x480-60

#/usr/sbin/fbset -n 800x600-70

```
start)
    echo -n "Setting framebuffer ..."
    #/usr/bin/clear
    /usr/sbin/fbset -n 640x480-60
    #/usr/sbin/fbset -n 800x600-70
    EXITCODE=0
    ;;
stop)
    echo -n "Restore framebuffer ..."
    echo "done."
    EXITCODE=0
    ;;
restart)
    $0 stop
    $0 start
    EXITCODE=$?
    ;;
*)
    usage
    ;;
esac
:wq
```

Fig 4-7

(3) Type **" reboot "** to reboot LP-8x4x, and you will find the new setting. (Refer to Fig 4-8)

```
# fbset
mode "640x480-60"
        # D: 26.000 MHz, H: 31.401 kHz, V: 59.926 Hz
        geometry 640 480 640 480 16
        timings 38461 78 46 22 10 64 12
        accel false
        rgba 5/11,6/5,5/0,0/0
endmode
#
```

Fig 4-8

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 LP-8x4x.

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) To create a script. 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 4-9)

Note: Set up of environment variable **PATH** and **LD_LIBRARY_PATH** in your script if necessary, user could pay a visit to /etc/init.d/webcam. (Refer to the Fig 4-10)

(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 4-11)

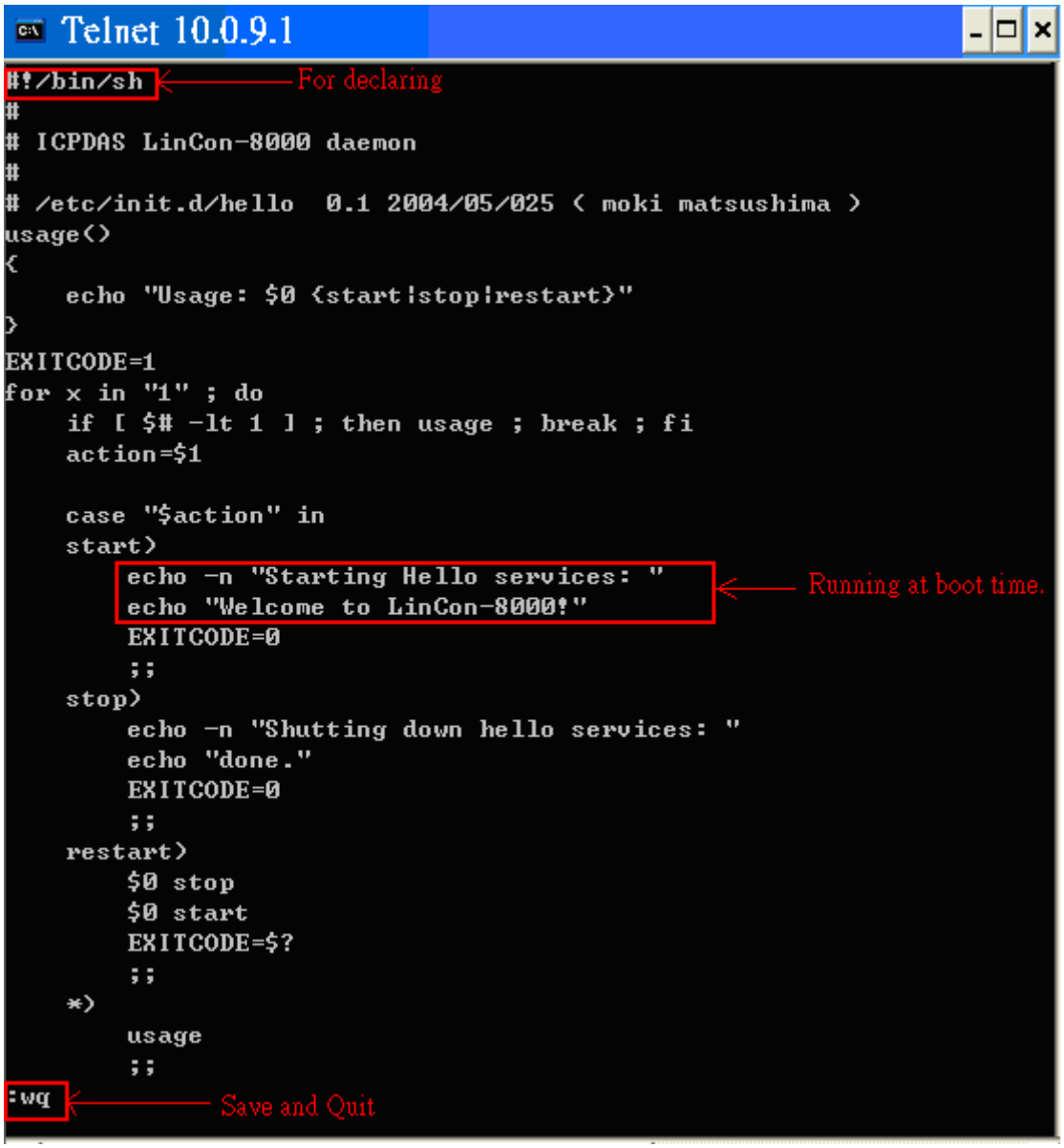


Fig. 4-9

```
# cat /etc/init.d/webcam
#!/bin/sh
#
#  Sample start-up script
#
export LOGNAME=root
export HOME=/$LOGNAME
export LD_LIBRARY_PATH=/opt/X11R6/lib:/opt/lib:/usr/local/lib:/lib:/usr/lib:/opt/kaffe/jre/lib/arm
:/opt/php/lib:/opt/mysql/lib:/opt/apache2/lib:/etc/user/lib:/mnt/hda/opt/lib:/opt/local/lib
export PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/opt/X11R6/bin:/opt/bin:/
opt/kaffe/bin:/opt/php/bin:/opt/mysql/bin:/opt/apache2/bin:/etc/user/bin:/etc/user/sbin:/opt/local
/bin:/opt/local/sbin:/mnt/hda/opt/bin:/opt/sbin:.
export CLASSPATH=/opt/kaffe/lib/kjc.jar:/opt/kaffe/lib/icpdas.jar:/opt/kaffe/lib/swingall.jar:.
export JAVA_HOME=/opt/kaffe
```

Fig. 4-10

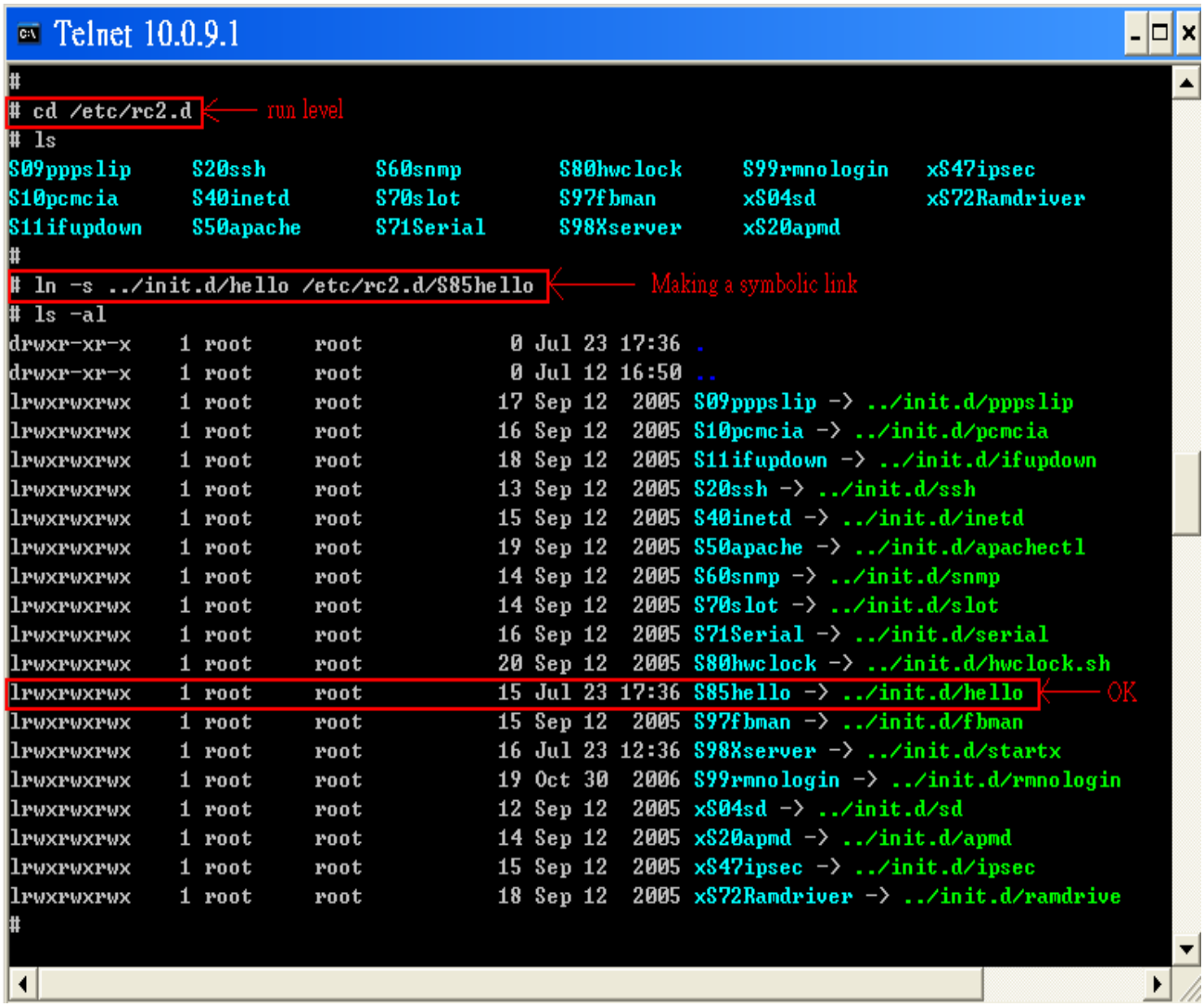


Fig. 4-11

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.

4.6 Automatic login

Log the specified user onto the console (normally /dev/tty1) when the system is first booted without prompting for a username or password using **mingetty** command.

The steps are as follows :

(1) Login as root and edit **/etc/inittab**

(2) Modify the entry for the first terminal— **tty1**

Below user can see the modified part of LP-8x4x /etc/inittab file (Refer to the Fig 4-12), and it will autologins into the root account after reboot the LP-8x4x.

```
# /sbin/getty invocations for the runlevels.
#
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
#
# Format:
#  <id>:<runlevels>:<action>:<process>
#
1:2345:respawn:/sbin/mingetty -noclear --autologin root tty1
#1:2345:respawn:/sbin/getty 38400 tty1
2:2345:respawn:/sbin/getty 38400 tty2
3:2345:respawn:/sbin/getty 38400 tty3
4:2345:respawn:/sbin/getty 38400 tty4
5:2345:respawn:/sbin/getty 38400 tty5
6:2345:respawn:/sbin/getty 38400 tty6
```

Fig. 4-12

5. Instructions for the LP-8x4x

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)

rmdir –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 －＞ 111(read, write, execute) 101(read, write, execute) 100(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 LinPAC to PC (Ex：get /mnt/hda/test.exe c:/test.exe)

put：upload file from PC to LinPAC (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 LP-8x4x)

5.1.15 date：print or set system date and time

5.1.16 hwclock：query and set the hardware clock (RTC)

Parameter：

(1) -r: read the hardware clock and print the time on standard output.

(2) -w: set the hardware clock to the current system time.

5.1.17 netstat：show the state of network

Parameter [-a]：list all states (Example：netstat -a)

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

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

ping IPAddress (Example：ping 192.168.0.1)

5.1.20 clear：clear the screen

5.1.21 passwd：change the password

5.1.22 reboot：reboot the LinPAC (or ‘shutdown –r now’)

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：

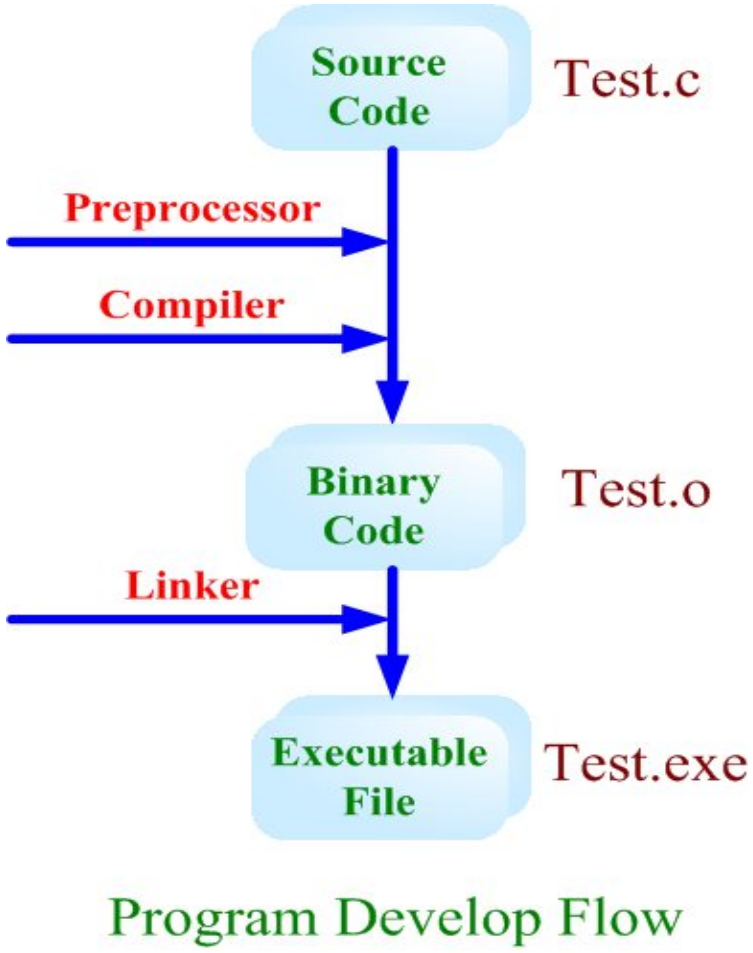


Fig. 5-1

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 LP-8x4x library

(1) Purpose：*. c　to　*. exe

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

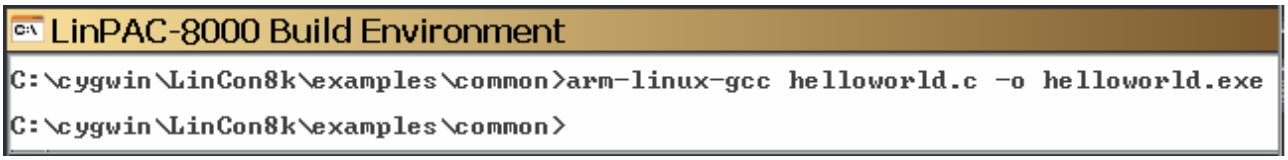
Parameter：

-o target：assign the name of output file

source.c：source code of C

Example：arm-linux-gcc　–o　helloworld.exe　helloworld.c

Output File：helloworld.exe



(2) Purpose：*. c　...　*.c　to　*. exe

Command：arm-linux-gcc　–c　source.c

Command：arm-linux-gcc　–o　target　object.o

Parameter：

-o target：assign the name of output file

source.c：source code of C

object.o：object file

Example：arm-linux-gcc　–c　main.c　helloworld.c　hi.c

arm-linux-gcc　–o　main.exe　main.o　helloworld.o　hi.o

Output File：main.exe

5.2.2 Compile with linking the LP-8x4x library (libi8k.a)

(1) Purpose：*. c　to　*. o

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

Parameter：

-IincludeDir：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 –IincludeDIR -lm –o target source.o library

Parameter：

-IincludeDir：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 –IincludeDIR -lm –o target source.c library

Parameter：

-IincludeDir：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 LP-8x4x by using FTP. Finally executes this file via the Telnet Server on the LP-8x4x. These steps can be accomplished in one pc without another monitor for the LP-8x4x. In this example, no ICP DAS modules are used. If you want to use the modules of ICP DAS 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 LP-8x4x 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)

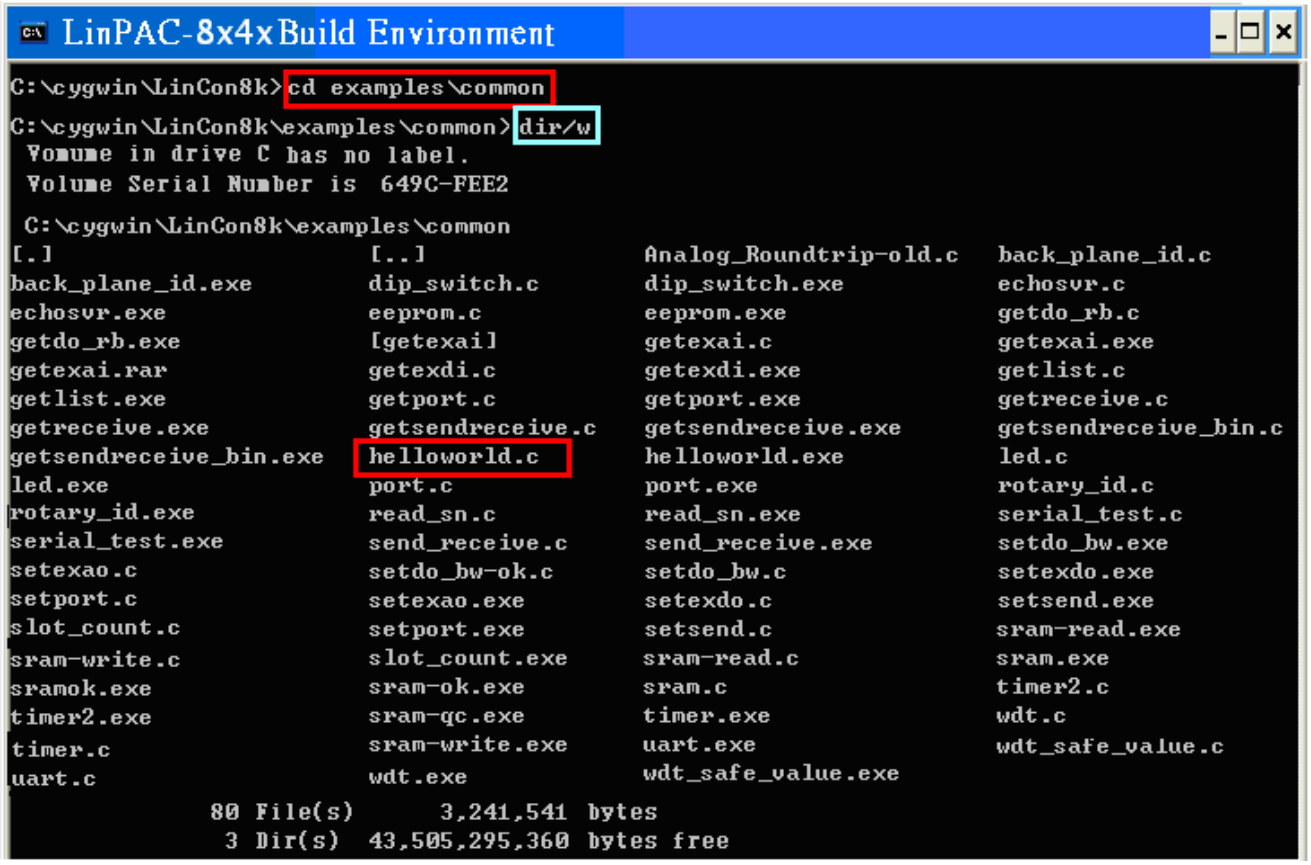


Fig. 5-2

(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)

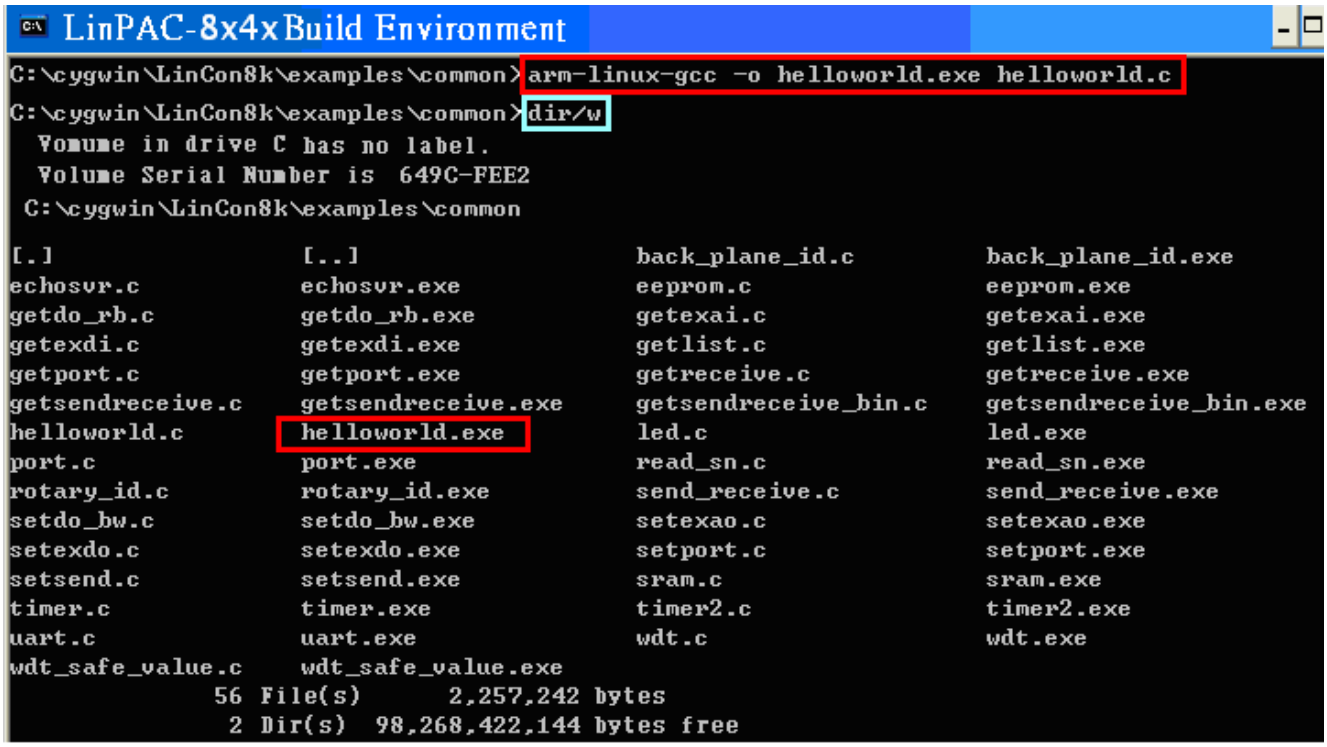


Fig. 5-3

STEP 2 ：(Transfer helloworld.exe to the LP-8x4x)

There are two methods for transferring files to the LP-8x4x：

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

(1) Open a “DOS Command Prompt” and type in the ftp IPAddress of the LP-8x4x (Example：**ftp 192.168.0.200**) to connect to the FTP Server on the LP-8x4x. Then type the **User_Name** and **Password (“ root ” is the default value.)** to accomplish the connection from the PC to the LP-8x4x.

(2) Before transferring your files to the LP-8x4x, type in the “**bin**” command to make the file transfer to the LP-8x4x in **binary mode**. (refer to Fig.5-4)

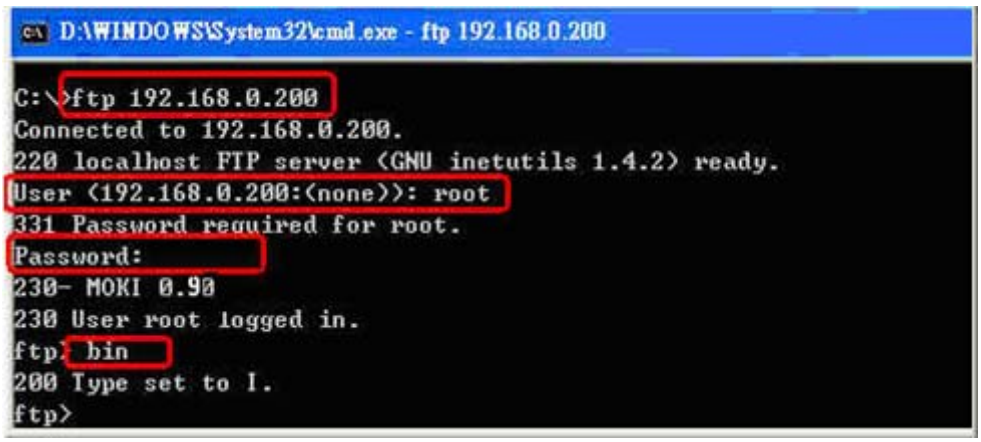


Fig.5-4

(3) Type in " **put C:/cygwin/LinCon8k/examples/common/helloworld.exe helloworld.exe** " to transfer helloworld.exe to the LP-8x4x. If it shows the message of " **Transfer complete** ", then the whole transferring process has been accomplished. If you need to disconnect from the LP-8x4x, type in the " **bye** " command to return to the PC console. (refer to Fig.5-5).

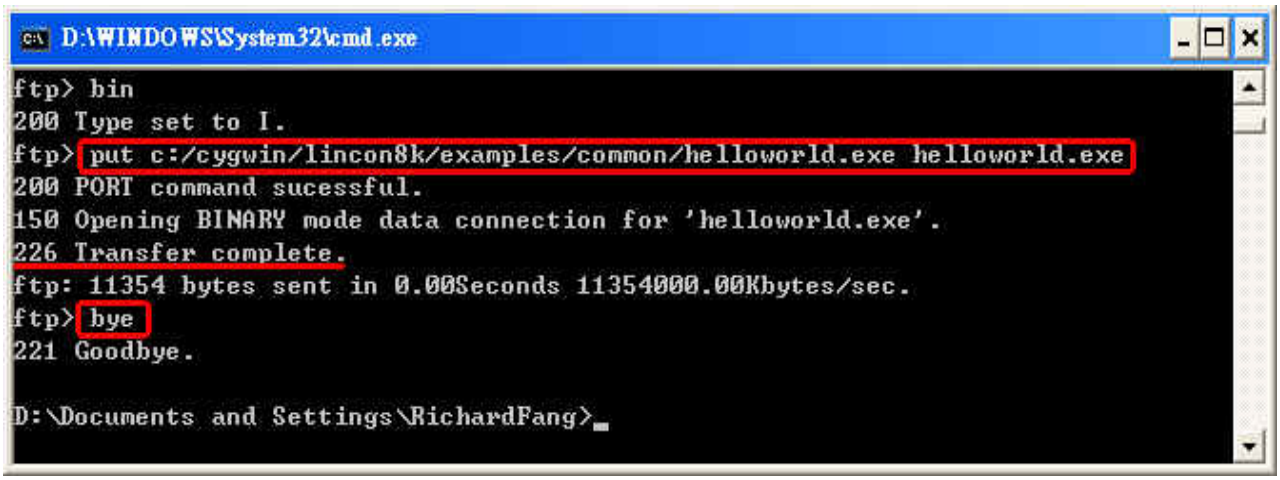


Fig.5-5

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

(1) Open the FTP Software and add a ftp site to the LP-8x4x. The **User_Name** and **Password** default value is **" root "**. Then click the **"Connect"** button to connect to the ftp server of the LP-8x4x. (refer to Fig.5-6).

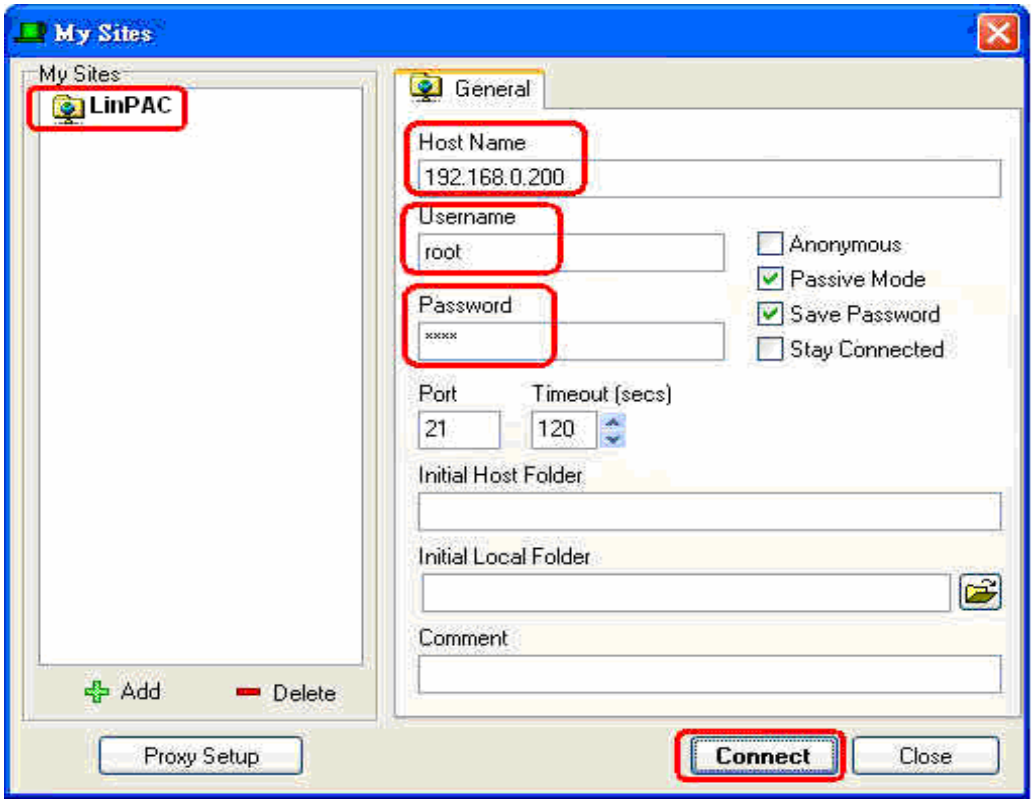


Fig.5-6

(2) Upload the file - **Helloworld.exe** to the LP-8x4x. (refer to Fig.5-7).

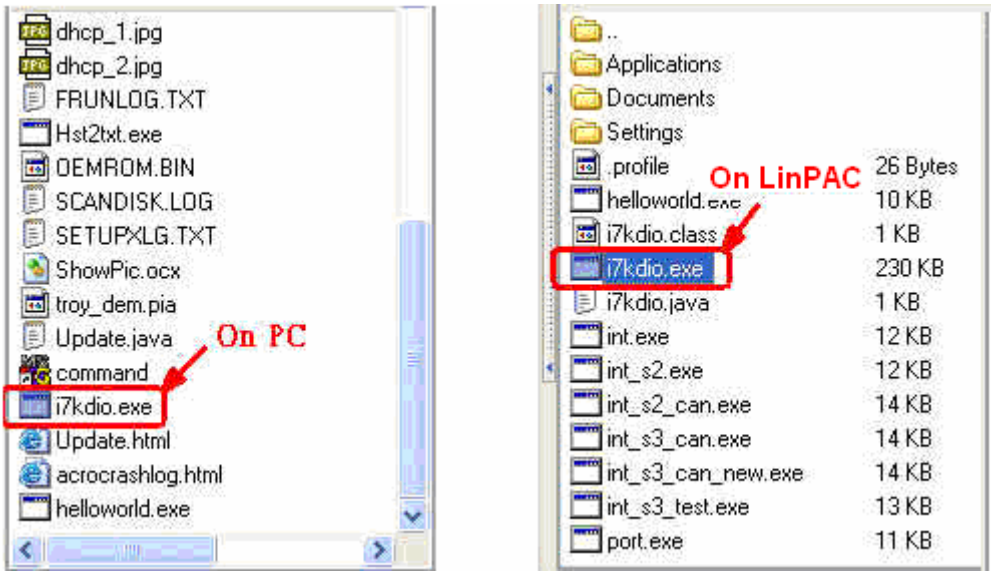


Fig.5-7

(3) Choose helloworld.exe in the LP-8x4x 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).

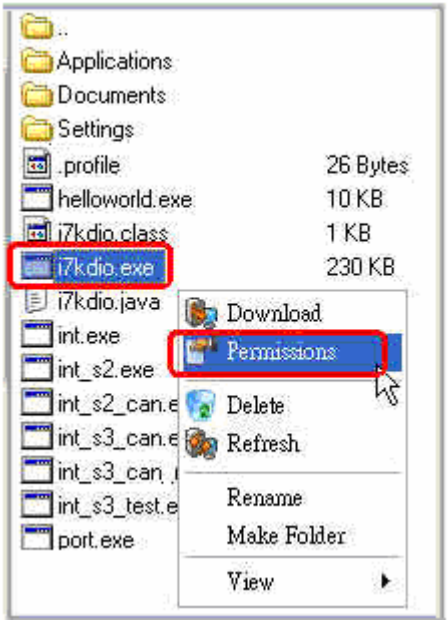


Fig.5-8

Fig.5-9

STEP 3 ：(Telnet to the LP-8x4x and execute program)

(1) Open a " DOS Command Prompt " and then type in the telnet IPAddress of the LP-8x4x (Example：**telnet 192.168.0.200**) to connect to the telnet server of the LP-8x4x. Then type the **User_Name** and **Password (" root " is the default value.)**. If it shows the **"** # **"** prompt character, the process of connecting from your PC to the telnet server of the LP-8x4x is finished. (refer to Fig.5-10)

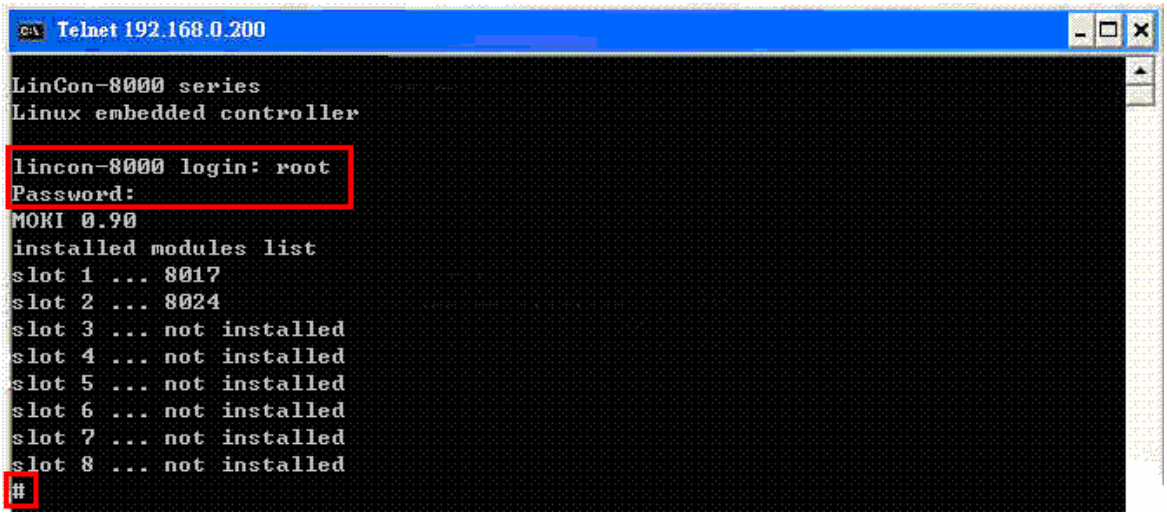


Fig.5-10

(2) Type in the "**ls -l**" command in order to list all the files in /root and to see the helloworld.exe file. Then type in the "**chmod 777 helloworld.exe**" command to change the authority of helloworld.exe and then type in the "**ls -l**" command again to see "helloworld.exe". This means that the file is executable. Type in "**./helloworld.exe**" to execute the file and it will show " Welcome to LP-8x4x ". Then all the steps from **compile**、 **transfer** to **telnet to execute program** will be completed. (refer to Fig.5-11)

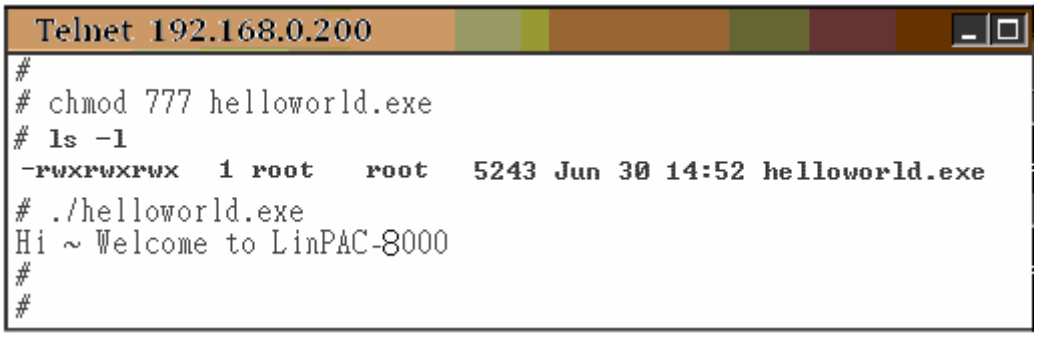


Fig.5-11

5.4 i-Talk Utility

The **i-Talk utility** provides **fifteen instructions** that make it convenient for users to access the modules and hardware in the LP-8x4x 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 **LinPAC-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 **LinPAC** COM port
getreceive	Receive String from **LinPAC** COM port
getsendreceive	Send/Receive String from **LinPAC** COM port
read_sn	Get Hardware Serial Number of **LinPAC-8000**

Fig. 5-12

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

Instruction	Demo
getlist	getlist list all the modules name in the
setdo	setdo slot data => setdo 3 3 set the I-8055W channel 1 and 2 on
setao	setao slot channel data => setdo 1 0 2.2 set the I-8024W channel 0 output 2.2V
getdi	getdi slot type => setdo 3 8 get the 8 bit DI value from I-8055W
getai	getai slot channel gain mode => getai 2 0 0 0 get the AI value I-8017HW
setexdo	setexdo slot 1 data => For slot setexdo slot comport data baudrate address => For Com Port
setexao	setexao slot 1 data channel => For slot setexao slot comport data channel baudrate address => For Com Port
getexdi	getexdi slot 1 => For slot getexdi slot comport baudrate address => For Com Port
getexai	getexai slot 1 channel => For slot getexai slot comport channel baudrate address => For Com Port
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

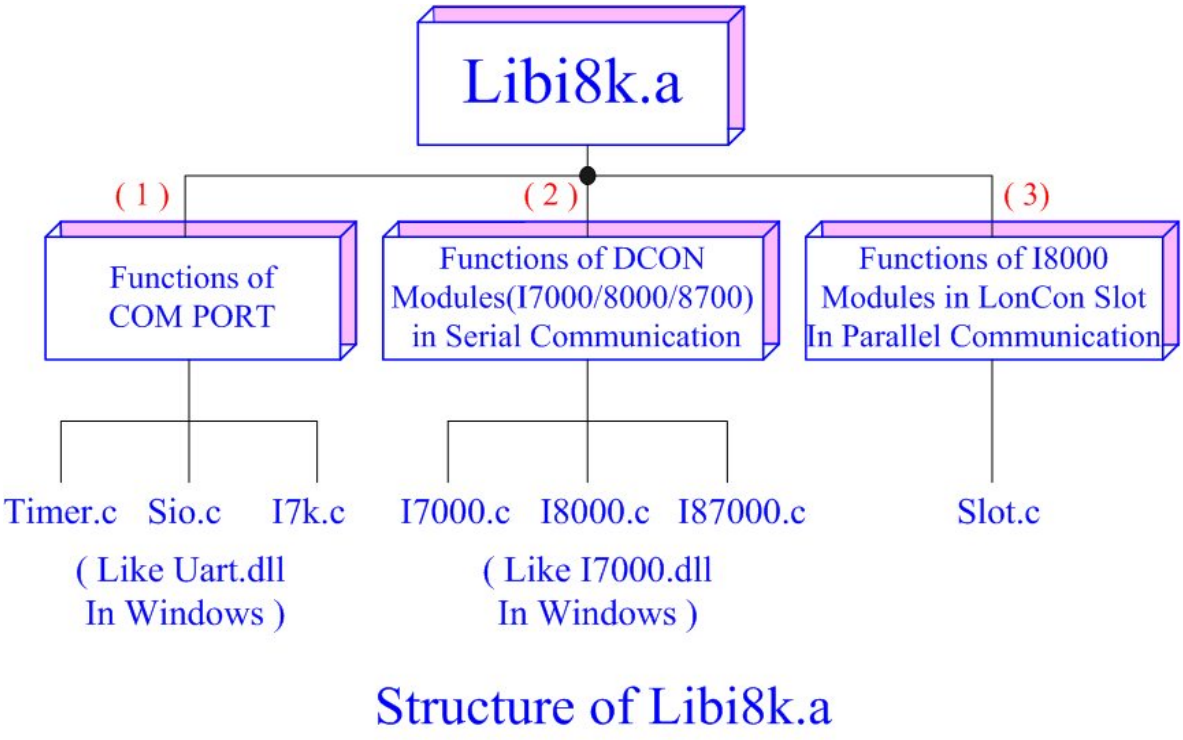


Fig. 6-1

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 LP-8x4x slots. They are different from functions (1) and (2) because the communication of I-8000 modules in the LP-8x4x slots are parallel and not serial. Therefore ICP DAS rewrote I8000.c to Slot.c especially for I-8000 modules in the LP-8x4x 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 ICP DAS I/O remote modules like i7k, i8k and i87k **through COM2 or COM3 or COM4 of the LP-8x4x**, 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 LP-8x4x, 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 specified slot in the LP-8x4x. The I-8k or I-87k modules in the LP-8x4x 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 LP-8x4x will be open and initiated.
```

Remark:

■ Close_Slot

Description:

If you have used the function of Open_Slot() to open the specified slot in the LP-8x4x, you need to use the Close_Slot() function to close the specified slot in the LP-8x4x. The I-8k or I-87k modules in the LP-8x4x 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 LP-8x4x will be closed.
```

Remark:

■ Open_SlotAll

Description:

This function is used to open and initiate **all slots** in the LP-8x4x. 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 LP-8x4x will be open and initiated.
```

Remark:

■ Close_SlotAll

Description:

If you have used the function Open_SlotAll() to open all the slots in the LP-8x4x, you can use the Close_SlotAll() function to close all the slots in the LP-8x4x. 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 LP-8x4x 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 LP-8x4x 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=2;
ChangeToSlot (slot);
// The first slot is specified as COM1 port in LP-8x4x.
```

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);
```

■ 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, 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);
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. Note that this function is usually applied to communicate with the other device, but not for ICP DAS DCON (I-7000/8000/87K) series modules.

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. Note that this function is usually applied to communicate with the other device, but not for ICP DAS DCON (I-7000/8000/87K) series modules.

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 10 character
Send _Binary(m_port, m_szSend, m_length);
// receive 10 character
Receive_Binary( m_port, m_szResult, m_timeout, m_wlength, &m_wt);
Close_Com (m_port);
```

Remark:

■ sio_open

Description:

This function is used to open and initiate a specified serial port in the LP-8x4x. The n-port modules in the LP-8x4x 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(const char *port, speed_t baudrate, tcflag_t data, tcflag_t parity,
                tcflag_t stop)
```

Parameter:

port :	[Input] device name: /dev/ttyS2, /dev/ttyS3…/dev/ttyS34
baudrate:	[Input] B1200/ B2400/ B4800/ B9600/ B19200/ B38400/ B57600/ B115200
date :	[Input] DATA_BITS_5/ DATA_BITS_6/ DATA_BITS_7/ DATA_BITS_8
parity :	[Input] NO_PARITY / ODD_PARITY / EVEN_PARITY
stop :	[Input] ONE_STOP_BIT / TWO_STOP_BITS

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-8144W 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 i8114W is plug in slot 1 and the first port will be open and initiated.
```

Remark:

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

■ sio_close

Description:

If you have used the function of sio_open() to open the specified serial port in the LP-8x4x, you need to use the sio_close() function to close the specified serial port in the LP-8x4x. 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/ttyS34

Return Value:

None

Example:

```
#define  COM_M2  "/dev/ttyS3"      // Defined the second port of I-8144iW 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 i8144iW in slot 1 will be closed.
```

Remark:

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

■ 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 LP-8x4x. This function performs a supporting task in the collection of information related to the system's hardware configurations.

Syntax:

[C]

```
int GetModuleType(int 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
_CAN	0XF0

Example:

```
int slot=1;
int moduleType;
Open_Slot(slot);
printf("GetModuleType= 0x%X \n", GetModuleType(slot));
Close_Slot(slot);
// The I-8057W card is plugged in slot 1 of LP-8x4x 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 LP-8x4x. This function supports the collection of system hardware configurations.

Syntax:

[C]

```
int GetNameOfModule(int 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-8017W will return 8017.

Example:

```
int slot=1;
int moduleID;
Open_Slot(slot);
moduleID=GetNameOfModule(slot);
Close_Slot(slot);
// The I-8017 card plugged in slot 1 of LP-8x4x
// Returned Value: moduleName=” 8017 ”
```

Remark:

■ Read_SN

Description:

This function is used to retrieves the hardware serial identification number on the LP-8x4x 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:

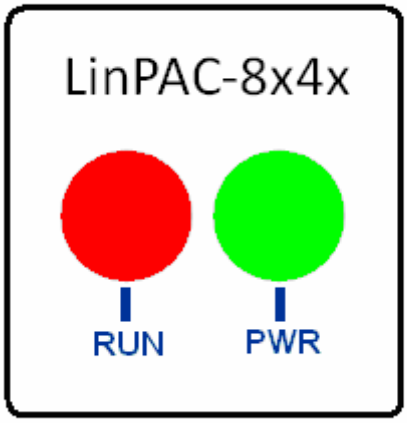
```
int slot ;
unsigned char serial_num[8];
Open_Slot(0);
Read_SN(serial_num);
printf("SN=%x%x%x%x%x%x%x%x\n",serial_num[0],serial_num[1], serial_ num[2]
      ,serial_num[3],serial_num[4],serial_num[5],serial_num[6],serial_num[7]);
```

Remark:

■ SetLED

Description:

This function is used to turn the LP-8x4x LED's on/off.



Address	**RUN**	**PWR**
Color	**Red**	**Green**
Programmable	Yes	No
Function	Start	Power

Syntax:

[C]

```
void SetLED(unsigned int led)
```

Parameter:

led : [Input] **1** : Turn on the LED

0 : Turn off the LED

Return Value:

None

Example:

```
unsigned int led;
led=1;
SetLED(led);
// The LED will turn on in LP-8x4x.
```

Remark:

■ GetBackPlaneID

Description:

This function is used to retrieve the back plane ID number in the LP-8x4x.

Syntax:

[C]

```
int GetBackPlaneID()
```

Parameter:

None

Return Value:

Backplane ID number.

Example:

```
int id;
id=GetBackPlaneID();
printf("GetBackPlanel =%d \n", id);
// Get the LP-8x4x backplane id . Returned Value: GetBackPlanel = 2
```

Remark:

■ GetSlotCount

Description:

This function is used to retrieve the number of slot in the LP-8x4x.

Syntax:

[C]

```
int GetSlotCount()
```

Parameter:

None

Return Value:

Number of slot.

Example:

```
int number;
number= GetSlotCount();
printf("GetSlotCount =%d \n", number);
// Get the LinPAC-8841 slot count.
// Returned Value: GetSlotCount = 8
```

Remark:

■ GetDIPswitch

Description:

This function is used to retrieve the DIP switch value in the LP-8x4x.

Syntax:

[C]

```
int GetDIPswitch()
```

Parameter:

None

Return Value:

DIP switch value.

Example:

```
int value;
value= GetDIPswitch();
printf("GetDIPswitch =%d \n", value);
// Get the LP-8x4x DIP switch value.
// Returned Value: GetDIPswitch = 128
```

Remark:

This function can be applied on PAC: LP-8441, LP-8841.

■ GetRotaryID

Description:

This function is used to retrieve the rotary ID number in the LP-8x4x.

Syntax:

[C]

```
int GetRotaryID(int type, & id)
```

Parameter:

type : [Input] number of slot.

id

Return Value:

0 is for Success

Not 0 is for Failure

SW	0	1	2	3	4	5	6	7	8	9
ID	79	78	77	76	75	74	73	72	71	70

Example:

```
int id, slot, type, wRetVal;
switch(type){
    case 1:
        slot = 0;   //lp-8x4x
        break;
    case 2:
        slot = 8;
        break;
    default:
        slot = 0;
        break;
}
wRetVal = Open_Slot(slot);
if (wRetVal > 0) {
        printf("open Slot%d failed!\n",slot);
        return (-1);
}
id= GetRotaryID(slot);
printf("GetRotaryID =%d \n",id);   // Get the LP-8x4x rotary id. If user turn the rotary switch to the 1 position, would get the returned value: GetRotaryID = 78
```

Remark:

■ GetSDKversion

Description:

This function is used to retrieve the version of LP-8x4x SDK.

Syntax:

[C]

```
float GetSDKversion(void)
```

Parameter:

None

Return Value:

Version number.

Example:

```
printf(" GetSDKversion = %4.2f \n ", GetSDKversion());
// Get the LP-8x4x SDK version number.
// Returned Value: GetSDKversion = 1.
```

Remark:

6.2 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 LinPAC will reset automatically.

Syntax:

[C]

```
void EnableWDT(unsigned int msecond)
void DisableWDT(void)
```

Parameter:

msecond: LinPAC 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 LinPAC 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 LinPAC)

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:

6.3 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 LP-8x4x 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 LP-8x4x 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:

6.4 Digital Input/Output Functions

6.4.1 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-8064W is plugged in slot 1 of LP-8x4x and can turn on channel 0 and 1.
```

Remark:

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

■ 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-8057W is plugged in slot 1 of LP-8x4x and can turn on channel 0 and 1.
```

Remark:

This function can be applied on modules: I-8037W, I-8056W, I-8057W and I-8046W.

■ 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-8041W is plugged in slot 1 of LP-8x4x and can turn on channel 0 and 1.
```

Remark:

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

■ 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-8058W is plugged in slot 1 of LP-8x4x and has inputs in channel 0 and 1.
// Returned value: data=0xfC
```

Remark:

There are two kind of Input type:

Input Type	On State	Off State	Modules
1	LED On, Readback as **1**	LED Off, Readback as **0**	I-8058W
2	LED On, Readback as **0**	LED Off, Readback as **1**	I-8048W, I-8052W

■ 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-8053W is plugged in slot 1 of LP-8x4x and has inputs in channel 0 and 1.
// Returned value: data=0xfffC
```

Remark:

There are two kind of Input type:

Input Type	On State	Off State	Modules
1	LED On, Readback as **1**	LED Off, Readback as **0**	I-8046W
2	LED On, Readback as **0**	LED Off, Readback as **1**	I-8051W, I-8053W, I-8053PW

■ 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-8040W plugged is in slot 1 of LP-8x4x and has inputs in
// channels 0 and 1.
// Returned value: data=0xfffffffC
```

Remark:

Input Type	On State	Off State	Modules
	LED On, Readback as **0**	LED Off, Readback as **1**	I-8040W

■ 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-8054W is plugged in slot 1 of LP-8x4x 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-8054W, I-8055W, and I-8063W.

■ 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-8042W is plugged in slot 1 of LP-8x4x 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-8042W and I-8050W.

■ 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-8054W is plugged in slot 1 of LP-8x4x and has inputs in channel 0 and 1.
// Returned value: data=0xfC
```

Remark:

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

■ 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-8042W is plugged in slot 1 of LP-8x4x and has inputs in channel 0 and 1.
// Returned value: data=0xfffC
```

Remark:

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

■ 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-8041W module is plugged in slot 1 of LP-8x4x and return all DO channel status.
```

Remark:

These functions can be applied on modules:

DO 8 channel：I-8060W, I-i8064W, I-8065W, I-8066W, I-8068W and I-8069W.

DO 16 channel：I-8037W, I-8056W, I-8057W and I-8046W.

DO 32 channel：I-8041W

■ 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-8041W module is plugged in slot 1 of LP-8x4x and just turn on channel 0 of
   I-8041W.
```

Remark:

These functions can be applied on modules:

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

DO 16 channel：I-i8037W, I-8056W and I-8057W

DO 32 channel：I-8041W

■ 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-8040W module is plugged in slot 1 of LP-8x4x and return channel 0
// status. ( 0：ON；1：OFF ).
```

Remark:

These functions can be applied on modules:

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

DI 16 channel：I-8051W and I-8053W

DI 32 channel：I-8040W

■ 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-8064W is plugged in slot 1 of LP-8x4x.
// WriteConfig: 0xffff (ch 0~ch15 is DO mode)
```

Remark:

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

■ 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-8050W is plugged in slot 1 of LP-8x4x.
// WriteConfig: 0x0000 (ch 0~ch15 is DI mode)
// Read the I/O Type is: 0x0000
```

Remark:

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

■ 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-8050W is plugged in slot 1 of LP-8x4x
// 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-8050W.

■ 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-8050W is plugged in slot 1 of LP-8x4x.
// WriteConfig: 0xff00 (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-8050W.

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

6.4.2.1 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 : [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] = 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:

6.4.2.2 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:

6.4.2.3 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:

6.5 Analog Input Functions

6.5.1 For I-8000 modules via parallel port

■ I8017_GetFirmwareVersion

Description:

This function is used to get the lattice version of I-8017HW at specific slot.

Syntax:

[C]

```
int I8017_GetFirmwareVersion(int slot)
```

Parameter:

slot : [Input] specified slot of the LP-8x4x system (Range: 1 to 8)

Return Value:

> 0 : Version no.

<=0 :error

Example:

```
int slot=1,ver;
ver=I8017_ GetFirmwareVersion (slot);
// The I-8017HW card is plugged in slot 1 of LP-8x4x and initializes the module.
```

Remark:

This function can be applied on module: I-8017HW.

■ I8017_Init

Description:

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

Syntax:

[C]

```
int I8017_Init(int slot)
```

Parameter:

slot : [Input] specified slot of the LP-8x4x system (Range: 1 to 8)

Return Value:

The version of library

Example:

```
int slot=1,ver;
ver=I8017_Init(slot);
// The I-8017HW is plugged in slot 1 of LP-8x4x and initializes the module.
```

Remark:

This function can be applied on module: I-8017HW.

■ I8017_SetLed

Description:

Turns the I-8017HW 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 LP-8x4x system (Range: 1 to 8)

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 LED light on channel 0 of the I-8017HW card which is plugged in slot 1 on the LP-8x4x.
```

Remark:

This function can be applied on module: I-8017HW.

■ I8017_GetSingleEndJumper

Description:

This function is used to get the mode of input channels, single-end or differential.

Syntax:

[C]

```
void I8017_GetSingleEndJumper(int slot)
```

Parameter:

slot : [Input] specified slot of the LP-8x4x system (Range: 1 to 8)

Return Value:

1: Single-End mode

0: Differential mode.

Example:

```
int slot=1;
printf("mode=%d", I8017_GetSingleEndJumper(slot));
```

Remark:

This function can be applied on module: I-8017HW.

■ I8017_SetChannelGainMode

Description:

This function is used to configure the range and mode of the analog input channel for the I-8017HW 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 LP-8x4x system (Range: 1 to 8)

ch : [Input] I-8017H : Range 0 to 7.

I-8017HS/I-8017HW : Single-end mode→ Range 0 to 15

Differentail mode→ Range 0 to 7

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:

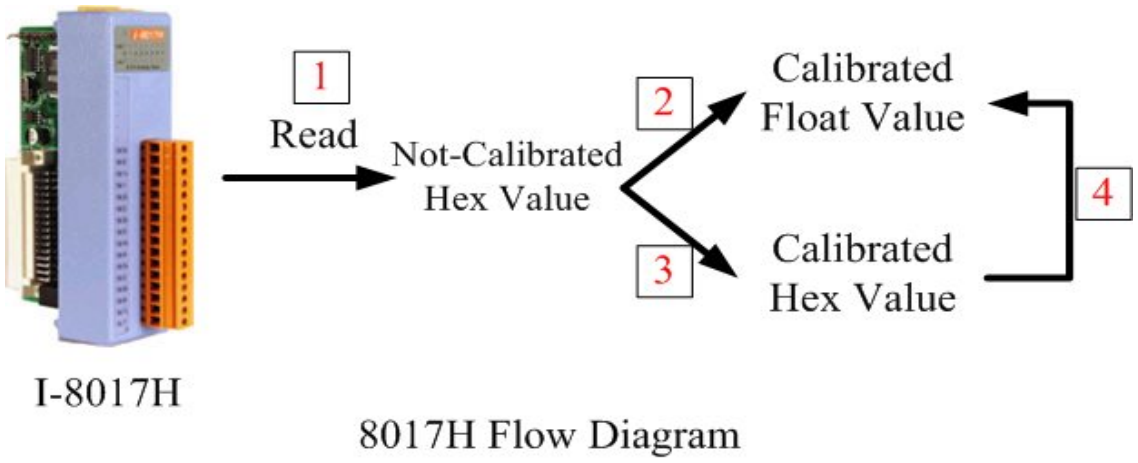
None

Example:

```
int slot=1,ch=0,gain=0;
I8017_SetChannelGainMode (slot, ch, gain,0);
// The I-8017HW card is plugged in slot 1 of LP-8x4x, 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-8017WH.



8017H Flow Diagram

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-8017HW modules. Please refer to Fig. 6-2

Syntax:

[C]

```
int I8017_GetCurAdChannel_Hex (int slot)
```

Parameter:

slot : [Input] specified slot of the LP-8x4x system (Range: 1 to 8)

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-8017HW is plugged into slot 1 of LP-8x4x 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 LP-8x4x system (Range: 1 to 8)

ch : [Input] I-8017H : Range 0 to 7.
I-8017HS/I-8017HW : Single-end mode→ Range 0 to 15
Differentail mode→ Range 0 to 7

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.
```

Remark:

You can use ARRAY_HEX_TO_FLOAT_Cal() or HEX_TO_FLOAT_Cal() to calibrate the data and convert to float value.

Function of [2]

■ I8017_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 I8017_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 LP-8x4x system (Range: 1 to 8)

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_I8017_GetCurAdChannel_Hex (slot);
fdata = I8017_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-8017HW .

■ I8017_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 I8017_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 LP-8x4x system (Range: 1 to 8)

gain : [Input] Input range:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

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);
I8017_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-8017HW.

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-8017HW.

■ 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). 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 LP-8x4x system (Range: 1 to 8)

gain : [Input] Input range:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

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-8017HW.

Function of [4]

■ I8017_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 I8017_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 = I8017_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-8017HW.

■ I8017_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 I8017_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:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

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 = I8017_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-8017HW.

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, I-8017H/I-8017HS/I-8017HW. This function is a combination of the "**I8017_GetCurAdChannel_Hex**" and the "**I8017_Hex_Cal**" function. Please refer to Fig. 6-2.

Syntax:

[C]

```
int I8017_GetCurAdChannel_Hex_Cal(int slot)
```

Parameter:

slot : [Input] specified slot of the LP-8x4x system (Range: 1 to 8)

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 LP-8x4x 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-8017HW.

■ 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 LP-8x4x system (Range: 1 to 8)

ch : [Input] I-8017HW : Range 0 to 7.
I-8017HS/I-8017HW : Single-end mode→ Range 0 to 15
Differentail mode→ Range 0 to 7

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-8017HW module.
```

Function of [1]+[2]

■ I8017_GetCurAdChannel_Float_Cal

Description:

Obtains the calibrated analog input value in the Float format directly from the analog input modules. This function is a combination of the "**I8017_GetCurAdChannel_Hex**" 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 LP-8x4x system (Range: 1 to 8)

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-8017HW is plugged into slot 1 of LP-8x4x 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-8017HW.

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" , szReceive=">+001.9"
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, 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/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, 0x8017
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:

6.5.2.3 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, 0x87013/15/16/17/18/19
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, 0x87013/15/16/17/18/19
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, 0x87013/15/16/17/18/19
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, 0x87013/15/16/17/18/19
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-8024W 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 LP-8x4x system slot (Range: 1 to 8)

Return Value:

None

Example:

```
int slot=1;
I8024_Initial(slot);
// The I-8024W is plugged into slot 1 of LP-8x4x and initializes the I-8024W module.
```

Remark:

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

■ I8024_VoltageOut

Description:

This function is used to send the voltage float value to the I-8024W module with the specified channel and slot in the LP-8x4x system.

Syntax:

[C]

```
void I8024_VoltageOut(int slot, int ch, float data)
```

Parameter:

slot : [Input] Specified the LP-8x4x system slot (Range: 1 to 8)

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-8024WW module output the 3.0V voltage from the channel 0.
```

Remark:

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

■ I8024_CurrentOut

Description:

This function is used to initialize the I-8024W module 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 LP-8x4x system slot (Range: 1 to 8)

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

Remark:

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

■ I8024_VoltageHexOut

Description:

This function is used to send the voltage value in the Hex format to the specified channel in the I-8024W module, which is plugged into the slot in the LP-8x4x system.

Syntax:

[C]

```
void I8024_VoltageHexOut(int slot, int ch, int hdata)
```

Parameter:

slot : [Input] Specify the LP-8x4x system slot (Range: 1 to 8)

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-8024W module output the 5.0V voltage from the channel 0.
```

Remark:

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

■ 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-8024W, which is plugged into the slot in the LP-8x4x system.

Syntax:

[C]

```
void I8024_CurrentHexOut(int slot, int ch, int hdata)
```

Parameter:

slot : [Input] Specify the LP-8x4x system slot (Range: 1 to 8)

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-8024W module.
```

Remark:

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

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, 0x87024
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, 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
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:

■ 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 Error Code Explanation

Error Code	Explanation
0	NoError
1	FunctionError
2	PortError
3	BaudrateError
4	DataError
5	StopError
6	ParityError
7	CheckSumError
8	ComPortNotOpen
9	SendThreadCreateError
10	SendCmdError
11	ReadComStatusError
12	StrCheck Error
13	CmdError
14	X
15	TimeOut
16	X
17	ModuleId Error
18	AdChannelError
19	UnderRang
20	ExceedRange
21	InvalidateCounterValue
22	InvalidateCounterValue
23	InvalidateGateMode
24	InvalidateChannelNo
25	ComPortInUse

6.8 3-axis Encoder Functions

■ I8090_REGISTRATION

Description:

In order to distinguish more than one I-8090W card in the LP-8x4x platform, the I-8090W modules should be registered before using it. If there are no I-8090W modules in the LP-8x4x 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 LP-8x4x slot (Range: 1 to 8)

ddress : This parameter is not used in the LP-8x4x.

Return Value:

1: Success registration

0: Failure registration

Example:

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

Remark:

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

■ I8090_INIT_CARD

Description:

This function is applied to reset the I-8090W counter values of three axes in a specific slot of LinPAC 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 LP-8x4x slot (Range: 1 to 8)

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-8090W :

(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 LP-8x4x slot (Range: 1 to 8)

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-8090W.

■ 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 LP-8x4x slot (Range: 1 to 8)

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-8090W.

■ 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 LP-8x4x slot (Range: 1 to 8).

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-8090W.

■ 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 LP-8x4x slot (Range: 1 to 8)

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-8090W.

■ 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 LP-8x4x slot (Range: 1 to 8).

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: I-8090W. 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 LP-8x4x slot (Range: 1 to 8).

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-8090W.

6.9 2-axis Stepper/Servo Functions

■ I8091_REGISTRATION

Description:

This function is used to assign a card number “cardNo” to the I-8091W card in the specified slot. In order to distinguish more than one of the I-8091 cards in the LP-8x4x platform, the I-8091W cards should be registered before using them. If there are no I-8091W 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 i8091 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-8091W card is plugged into slot 1 of LP-8x4x.
```

Remark:

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

■ i8091_RESET_SYSTEM

Description:

This function is used to reset and terminate the running command in the I-8091W 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-8091W 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-8091W card plugged in slot 1 of LP-8x4x
```

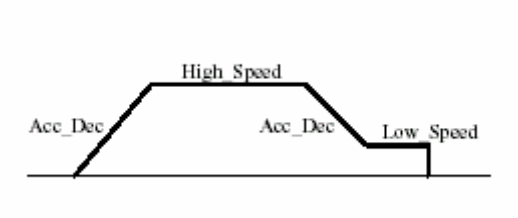
Remark:

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

■ 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-8091W.

■ 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-8091W.

■ i8091_SET_MODE

Description:

This function is used to set the motor control modes of the X and Y axes in the specified I-8091W.

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:

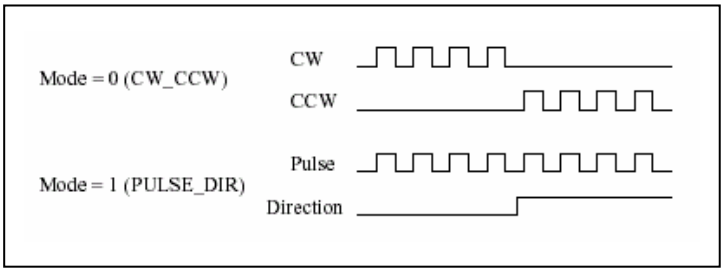
None

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-8091W.



■ 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-8091W.

■ i8091_SET_NC

Description:

This function is used to set all of the following limit switches to N.C.(normally closed) or N.O.(normally 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) normally open (default), 1(YES) normally closed.

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-8091W.

■ 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: I-8091W.

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: I-8091W.

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: I-8091W.

■ 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.
```

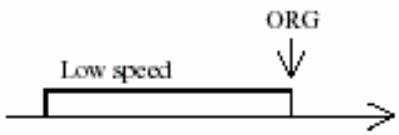
Remark:

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

■ 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);
```

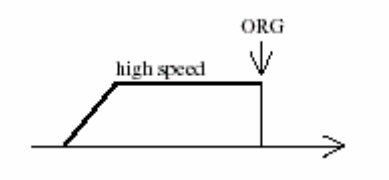
Remark:

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

■ 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)

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);
```

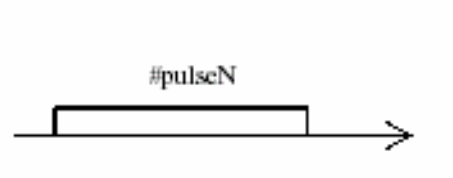
Remark:

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

■ 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
```

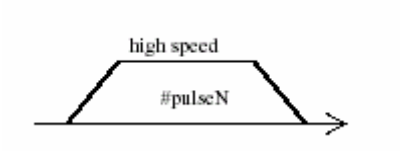
Remark:

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

■ 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
```

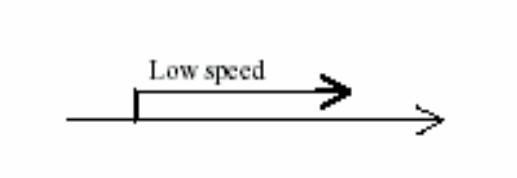
Remark:

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

■ 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);
```

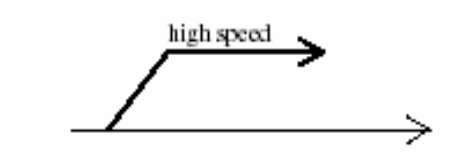
Remark:

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

■ 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);
```

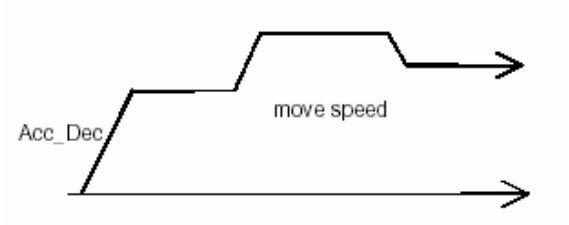
Remark:

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

■ 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);
```

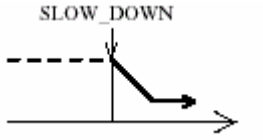
Remark:

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

■ 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);
```

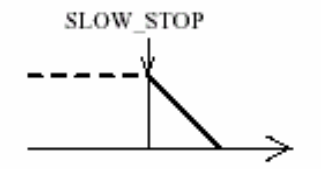
Remark:

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

■ 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);
```

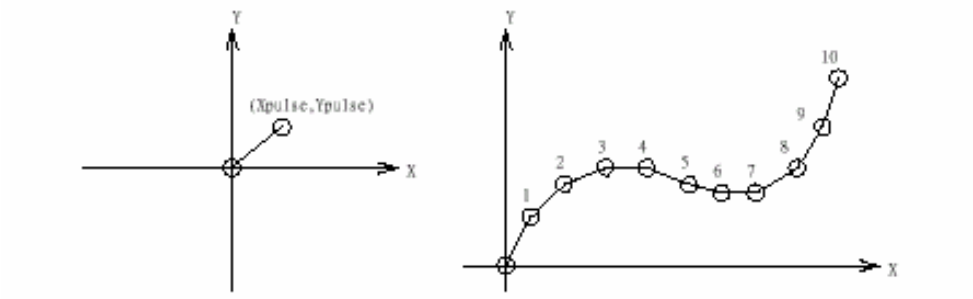
Remark:

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

■ 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);
```

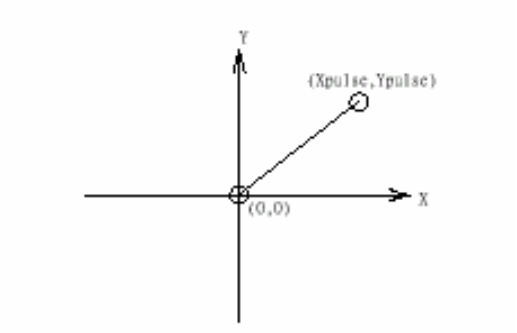
Remark:

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

■ i8091_INTP_LINE

Description:

This command will move a long distance (interpolation line) in the X-Y plane. The CPU on the I-8091W 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);
```

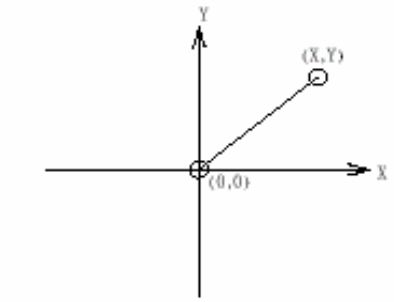
Remark:

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

■ 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
```

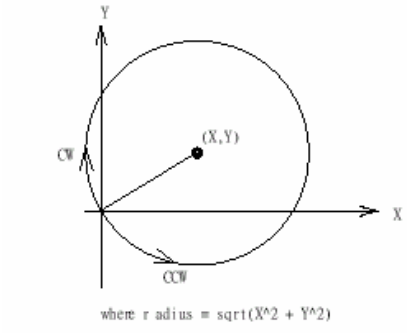
Remark:

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

■ 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
```

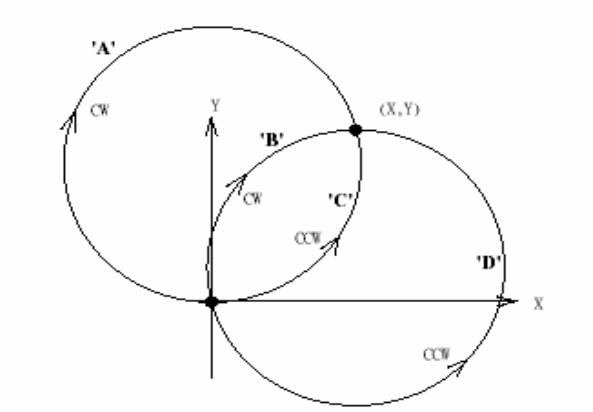
Remark:

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

■ 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-8091W.

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: I-8091W

■ 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-8091W.

■ 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-8091W.

■ 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: I8091W.

■ 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-8091W.

■ 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-8091W.

■ 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-8091W.

7. Demo of LP-8x4x 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 LP-8x4x. After you install the LP-8x4x 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] = 0x0f;               // 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>LP-8x4x SDK> LP-8x4x Build Environment to open **LP-8x4x 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)

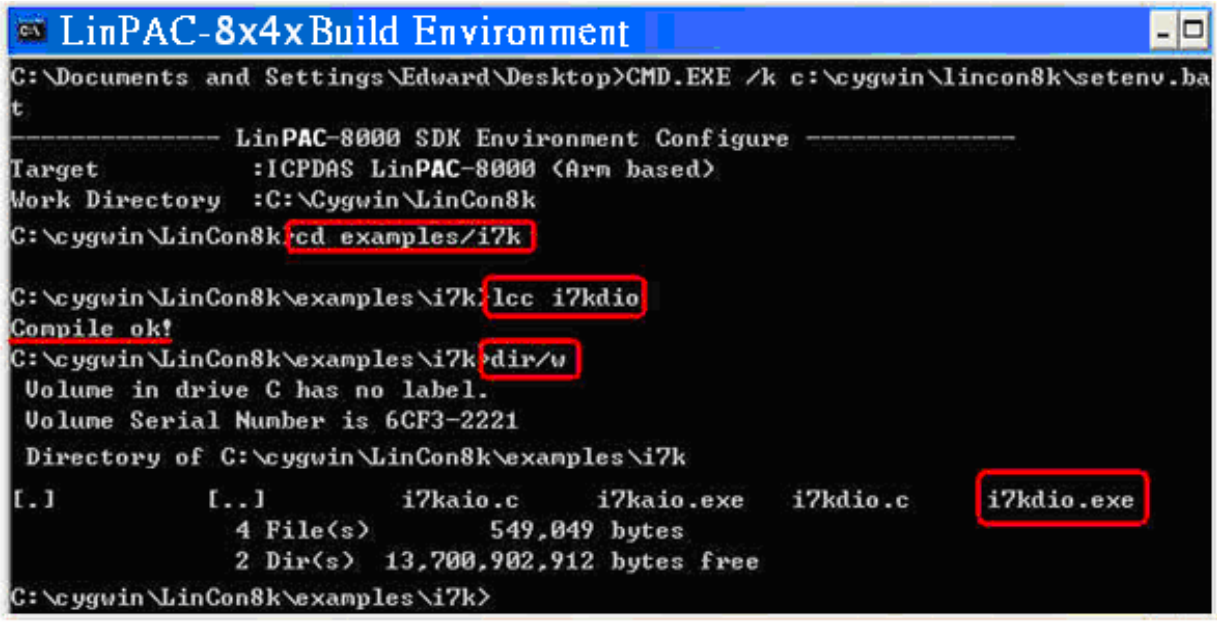


Fig. 7-1

< Method Two > Using Compile Instruction

If you choose this method, change the path to C:\cygwin\LinCon8k\examples\i7k and then type arm-linux-gcc -I**../../**include –lm –o i7kdio.exe i7kdio.c **../../**lib/libi8k.a to compile i7kdio.c to i7kdio.exe. (refer to Fig. 7-2)

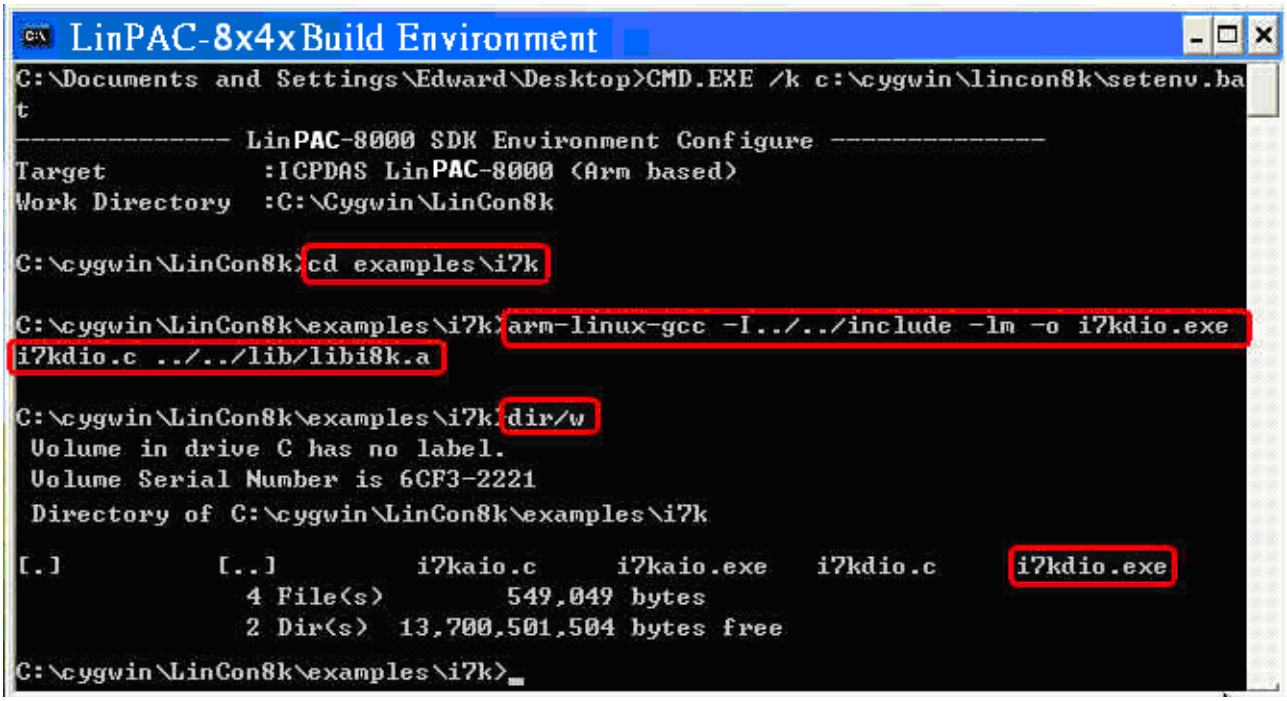


Fig. 7-2

STEP 3：**(Transfer i7kdio.exe to the LP-8x4x)**

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 LP-8x4x. The **User_Name** and **Password** default value is **" root "**. Then click the **"Connect"** button to connect to the ftp server of the LP-8x4x. (refer to Fig.7-3).

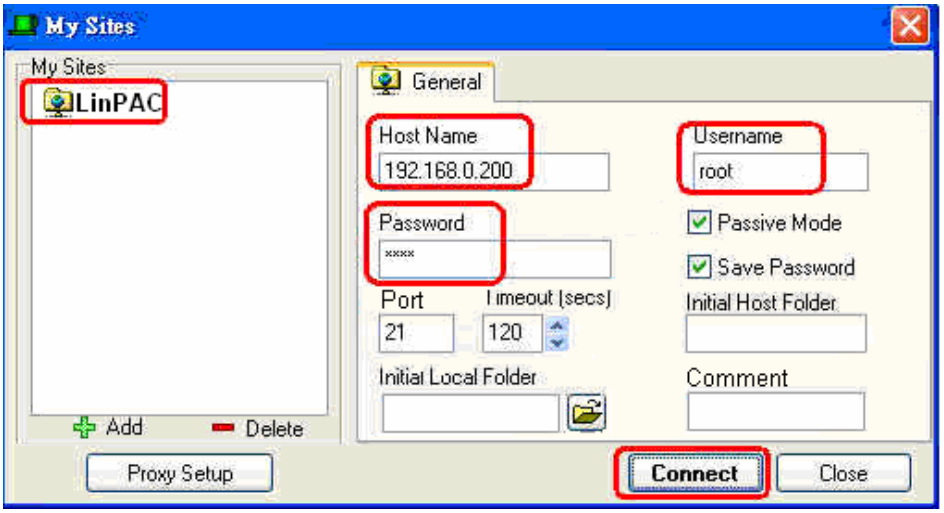


Fig.7-3

(2) Upload the file – **i7kdio.exe** to the LP-8x4x. (refer to Fig.7-4).

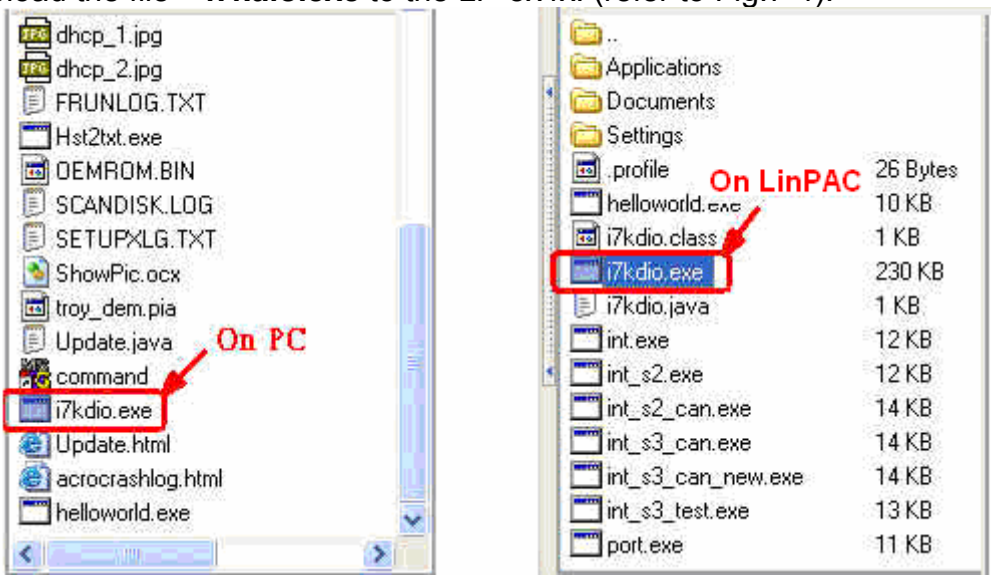


Fig.7-4

(3) Choose i7kdio.exe in the LP-8x4x 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).

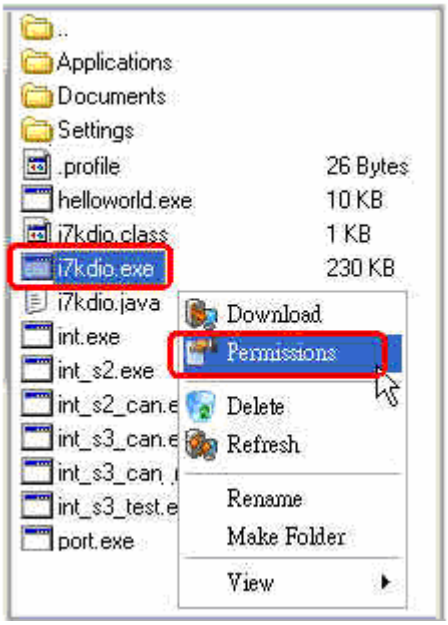


Fig.7-5

Fig.7-6

< Method Two > Using DOS Command Prompt

Open DOS Command Prompt and type ftp IP Address of LP-8x4x in order to connect to the ftp server of the LP-8x4x. Then input **User Name** and **Password** (**root** is the default value) to login to the LP-8x4x. 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 LP-8x4x. After the "Transfer complete" message appears, the process of

transference would have been completed.(refer to Fig. 7-7)

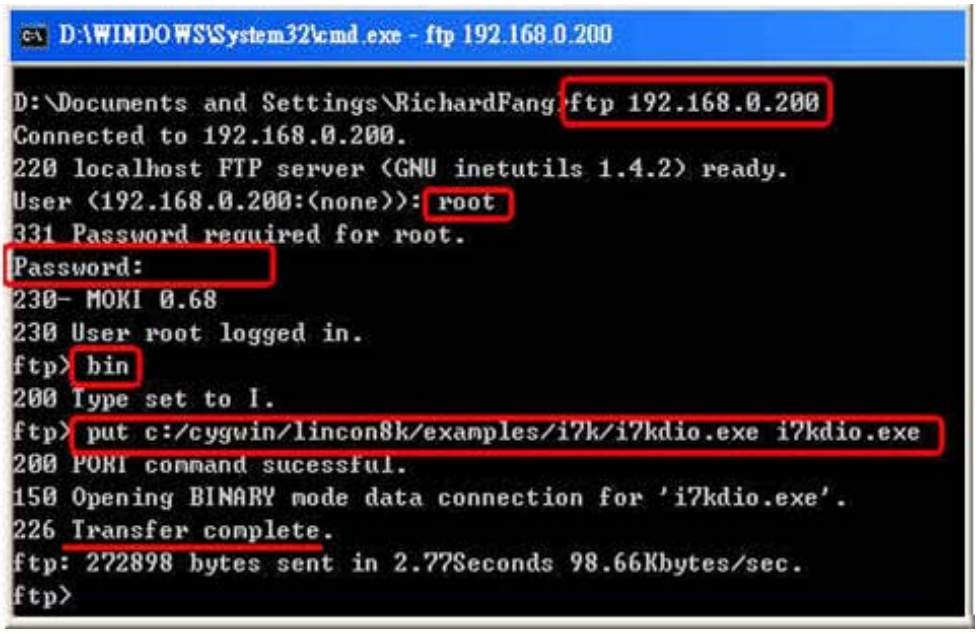


Fig. 7-7

STEP 4：**(Telnet to the LP-8x4x to execute i7kdio.exe)**

Type telnet IP Address of LP-8x4x into the remote control the LP-8x4x and input your **User Name** and **Password** (**root** is the default value) to login to the LP-8x4x. 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)

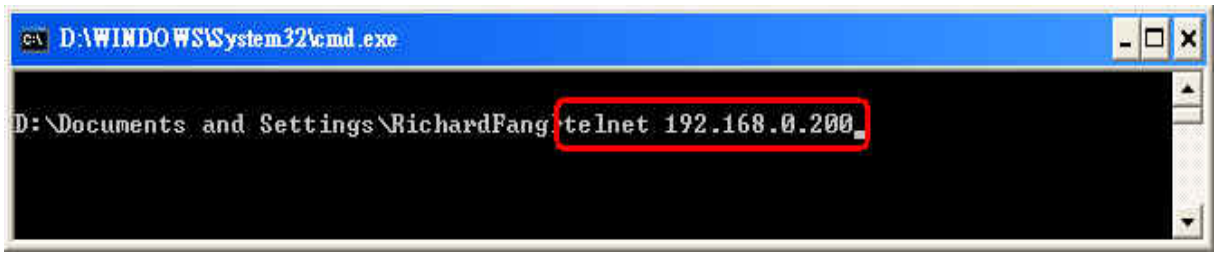


Fig. 7-8

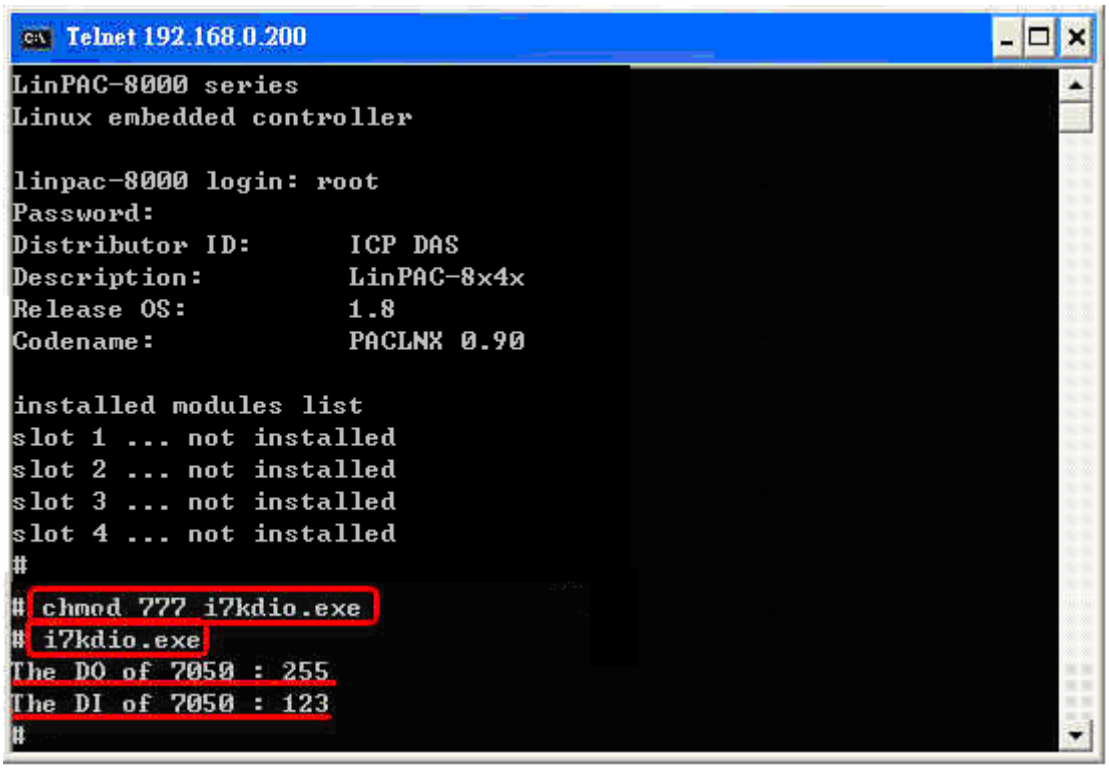


Fig. 7-9

" **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.

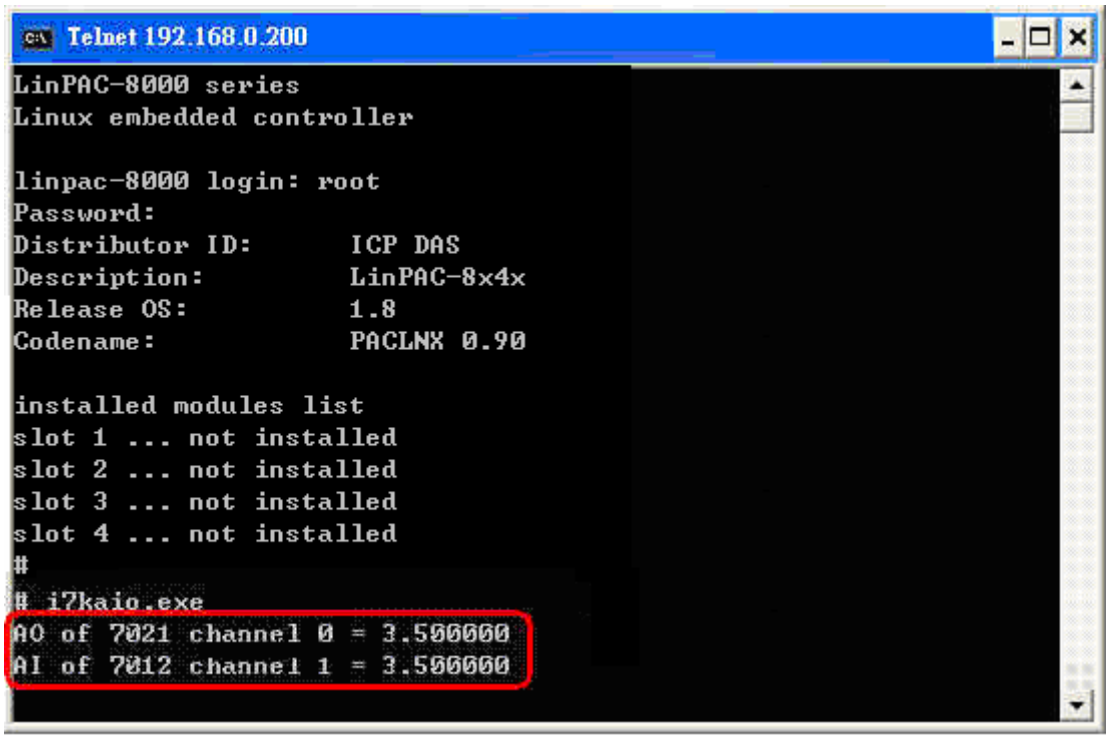


Fig. 7-10

7.3 I-87KW Modules DIO Control Demo

When using I-87KW modules for I/O control of the LP-8x4x, 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-87KW DIO modules are **in the LP-8x4x slots**, the two functions “ Open_Slot ” and “ ChangeToSlot ”, must be added before using other functions for the I-87KW 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-87KW 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-87KW DIO modules are **in the I-8000 controller slots**, then the I-87KW modules will be regarded as I-8KW modules and so please refer to I/O control of I-8KW modules in section 7.5.2

7.3.1 I-87KW Modules in slots of LP-8x4x

This demo – **i87kdio.c** will illustrate how to control the DI/DO with the I-87054W module (8 DO channels and 8 DI channels). The I-87054W module is in slot 3 of the LP-8x4x. The address and baudrate in the LP-8x4x are constant and they are 00 and 115200 respectively. The result of this demo lets DO channel 0 ~ 7 of I-87054W output and DI channel 1 of I-87054W 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-87KW Modules in slots of I-87KW I/O expansion unit

If the I-87KW modules are in the slots of the I-87KW 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-87KW modules in the network of RS-485 need to be set by DCON Utility(http://www.icpdas.com/products/dcon/introduction.htm).

(3) **Open com1**(internal serial port of LP-8x4x) will be modified to **open com3**(RS-485 port).

The address and baudrate of the I-87054W 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.

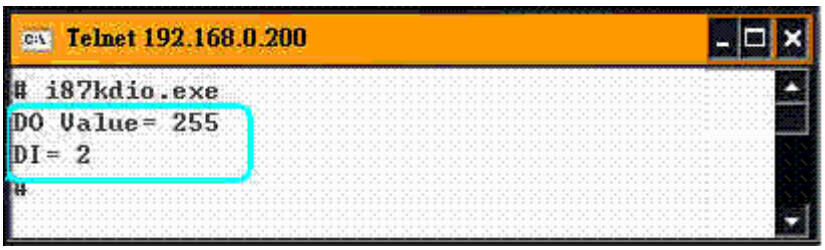


Fig. 7-11

7.3.3 I-87KW Modules in slots of I-8000 Controller

If the I-87KW DIO modules are in the I-8000 controller slots, I-87KW modules will be regarded as I-8KW modules and so please refer to DI/DO control of I-8KW modules in the section 7.5.

7.4 I-87KW Modules AIO Control Demo

When using I-87KW modules for I/O control of the LP-8x4x, according to the location of the I-87KW modules, the program will be a little different. There are three conditions for the location of the I-8KW modules：

(1) When the I-87KW AIO modules are **in the LP-8x4x slots**, the two functions " Open_Slot " and " ChangeToSlot " must be added before using the other functions of the I-87KW 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-87KW 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-87KW AIO modules are **in the I-8000 controller slots**, the I-87KW modules will be regarded as I-8KW modules and so please refer to I/O control of I-8KW modules in section 7.6.2.

7.4.1 I-87KW Modules in slots of LP-8x4x

This demo – **i87kaio.c** will illustrate how to control the AI/AO with the **I-87022W** module (2 AO channels) and the **I-87017W** module (8 AI channels). The I-87022W and I-87017W modules are plugged into slot 2 and slot 3 of the LP-8x4x separately. The address and baudrate in the LP-8x4x are constant and they are 00 and 115200 separately. The result of this demo lets AO channel 0 of I-87022W output 2.5V and AI channel 1 of I-87017W 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-87KW Modules in slots of I-87KW I/O expansion unit

If the I-87KW modules are in slots of I-87KW 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-87KW modules in the network of RS-485 need to be set by DCON Utility

(3) **Open com1** (internal serial port of LP-8x4x) will be modified to **open com3** (RS-485 port of LP-8x4x)

The addresses I-87022W and I-87017W 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.

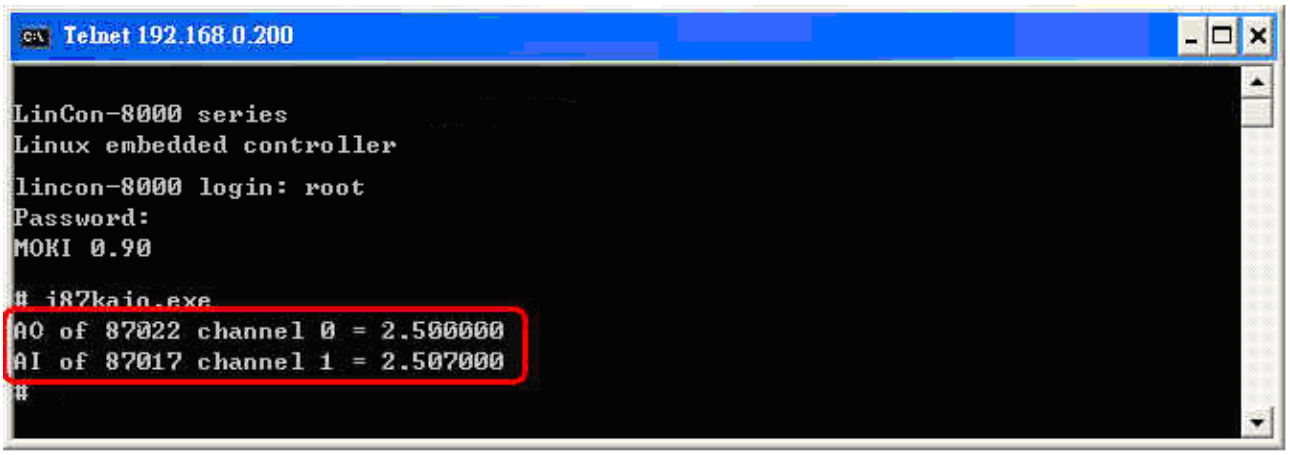


Fig. 7-12

7.4.3 I-87KW Modules in slots of I-8000 Controller

If the I-87KW AIO modules are in slots of I-8000 controller, I-87KW modules will be regarded as I-8KW modules and refer to AI/AO control of I-8KW modules in the section 7.6.

7.5 I-8KW Modules DIO Control Demo

I8000.c of Libi8k.a is the source file for I-8KW modules in slots of I-8000 controller. **Slot.c** of Libi8k.a is the source file for I-8KW modules in slots of LP-8x4x. Therefore the functions for I-8KW modules in slots of LP-8x4x and in slots of I-8000 controller are different completely. There are two conditions for the location of the I-8KW modules：

(1) When I-8KW DIO modules are **in the LP-8x4x**, then please refer to the demo in section 7.5.1.

(2) When I-8KW DIO modules are **in the I-8000 controller**, then please refer to the demo in section 7.5.2.

7.5.1 I-8KW Modules in slots of LP-8x4x

In this section, this demo program – **i8kdio.c** will introduce how to control the DI/DO with the I-8055W(8 DO channels and 8 DI channels) module and it is plugged into slot 3 of the LP-8x4x.

The address and baudrate in the LP-8x4x are constant and they are 00 and 115200 separately. The result of this demo lets DO channel 0 ~7 of I-8055W output and DI channel 0 of I-8055W 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-8055W_DO
    DO_8(3,255);
    printf("DO of I-8055 = 0x%x \n", 255);

    //I-8055W_DI
    printf("DI of I-8055 = %x",DI_8(3));

    Close_Slot(3);
    return 0;
}
```

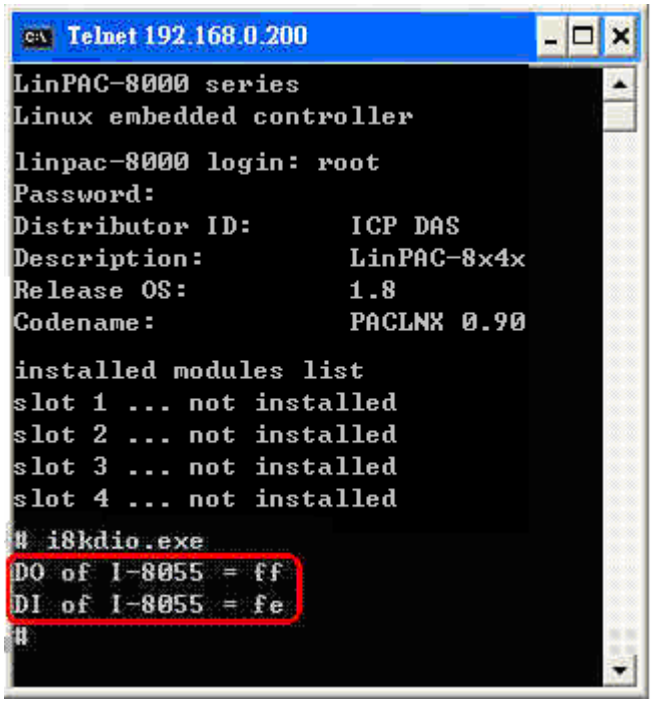


Fig. 7-13

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.

7.5.2 I-8KW 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-8055W(8 DO channels and 8 DI channels) module. Please follow the below steps to configure the hardware：

(1) Put the I-8055W 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 **com3** of LP-8x4x 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-8055W output and DI channel 0 of I-8055W 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_Com3
    wRetVal = Open_Com(COM3, 115200, Data8Bit, NonParity, OneStopBit);
    if (wRetVal > 0) {
        printf("open port failed!\n");
        return (-1);
    }

    //--- digital output ----   **(DigitalOut_8K()**)
    dwBuf[0] = 3;                   // 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] = 3;                   // 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] = 3;                   // 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(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. 7-14.

Fig. 7-14

7.6 I-8KW Modules AIO Control Demo

I8000.c of Libi8k.a is the source file for I-8KW modules in slots of I-8000 controller. **Slot.c** of Libi8k.a is the source file for I-8KW modules in slots of the LP-8x4x. Therefore the functions for the I-8KW modules in LP-8x4x slots and in the I-8000 controller slots are completely different. There are two conditions for the location of the I-8KW modules：

(1) When I-8KW AIO modules are **in the LP-8x4x**, then please refer to the demo in section 7.6.1.
(2) When I-8KW AIO modules are **in the I-8000 controller**, then please refer to the demo in section 7.6.2.

7.6.1 I-8KW Modules in slots of LP-8x4x

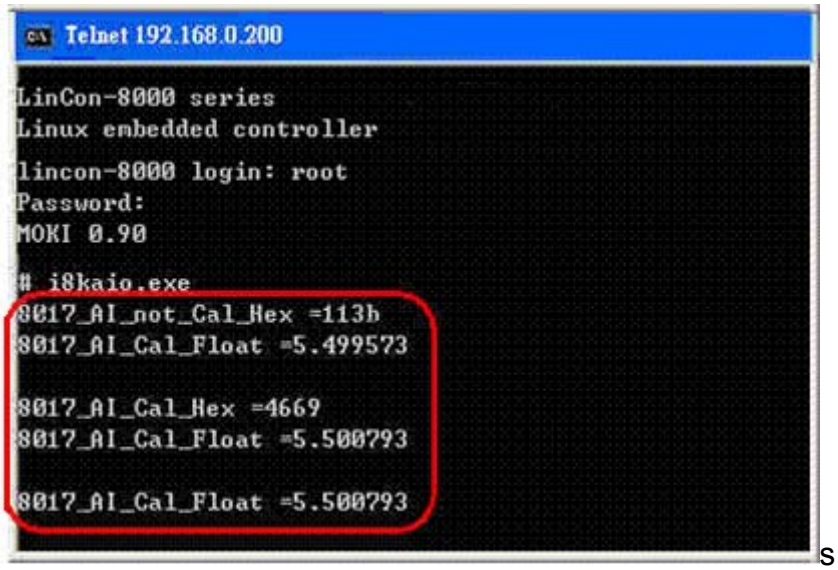
In this section, this demo program – **i8kaio.c** will illustrate how to control the AI/AO with the I-8024W (4 AO channels) and I-8017HW (8 AI channels) module and they are in slot 1 and slot 2 of the LP-8x4x separately.

The address and baudrate in the LP-8x4x are constant and they are 00 and 115200 separately. The result of this demo lets AO voltage channel 0 of I-8024W output 5.5V and AI channel 2 of I-8017HW 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-8024W
    wRetVal = Open_Slot(1);
    if (wRetVal > 0) {
        printf("open Slot failed!\n");
```

```
        return (-1);
    }

    //I8024W Initial
    I8024_Initial(1);
    //I8024_AO Output
    I8024_VoltageOut(1,0,5.5);
    Close_Slot(1);

    //I-8017HW
    wRetVal = Open_Slot(2);
    if (wRetVal > 0) {
        printf("open Slot failed!\n");
        return (-1);
    }
    //I8017HW Initial
    I8017_Init(2);
    //I8017HW _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.



s

Fig. 7-15

7.6.2 I-8KW 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-8024W (4 AO channels) and I-8017HW (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-8024W and I-8017HW 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 **com3** of LP-8x4x 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 I-8024W output 3.5V and AI channel 2 of I-8017HW 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(COM3, 115200, Data8Bit, NonParity, OneStopBit);

    if (wRetVal > 0) {
        printf("open port failed!\n");
        return (-1);
    }

    //--- Analog output ----    ****     8024 -- AO    ****
    i = 0;
    wBuf[0] = 3;                // 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] = 3;                // 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(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. 7-16

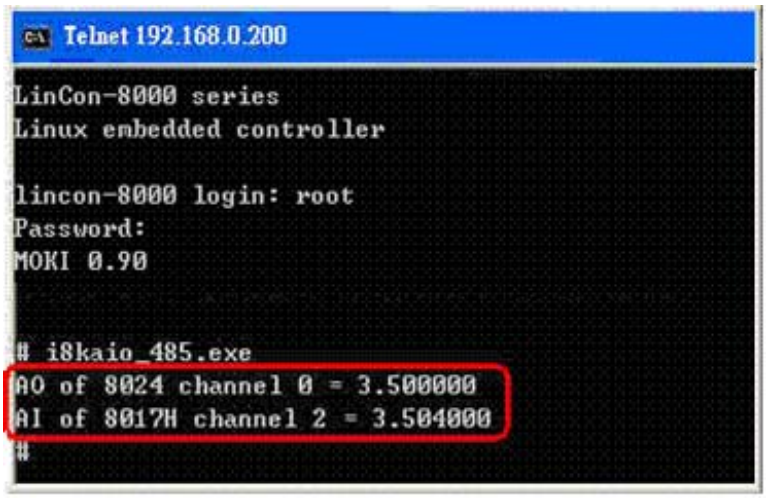


Fig. 7-16

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 ICP DAS modules in the LP-8x4x, this table will be helpful to let users understand which functions of communication should be used. **Attention please, the Open_slot()/Close_Slot() and sio_open()/sio_close() cannot for use in same slot**.

Module Location / Communication Functions	I-8KW – In LP-8x4x	I-87KW – In LP-8x4x	I-87KW – In Expansion Unit	I-8KW or I-87KW – 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 ICP DAS modules in the LP-8x4x, this table will be helpful to let users understand which source files of the libi8k.a should be called.

Source / Module / File / Loaction	I7000.c	I8000.c	I87000.c	slot.c
I-7K	✓			
I-8KW or I-87KW in I-8000 Controller		✓		
I-87KW in Expansion Unit			✓	
I-87KW in LinPAC			✓	
I-8KW in LinPAC				✓

Fig. 7-18

7.8 Timer Function Demo

If users want to use "**Timer**" function in the LinPAC-8x4x, please refer to the demo – timer.c and time2.c in the LinPAC 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).

8. Introduction of LinPAC-8x4x Serial Ports

This section describes the function of the three serial ports (RS-232/RS-485 interface) in the LinPAC-8x4x embedded controller (see Fig 8-1 for LP-8841 and Fig 8-2 for LP-8141). The information in this section is organized as follows:

- **COM1 Port** – Internal communication with the I-87KW modules in slots
- **COM2 Port** – RS-485 (D2+, D2-; self-tuner ASIC inside)
- **COM3 Port** – RS-232/RS-485
  (RXD, TXD, CTS, RTS and GND for RS-232, Data+ and Data- for RS-485)
- **COM36 Port** – RS-232 (RXD, TXD, CTS, RTS, DSR, DTR, CD, RI and GND)

COM port	Definitions in LP-8x4x SDK	Device name	Default baudrate
None	COM1	None	115200
1 (console)	None	None	115200
2 (RS-485)	COM2	ttyS0	9600
3 (RS-232/485)	COM3(LP-8441/8841 only)	ttyS1	9600
4 (RS-232)	COM36(LP-8441/8841 only)	ttyS34	9600

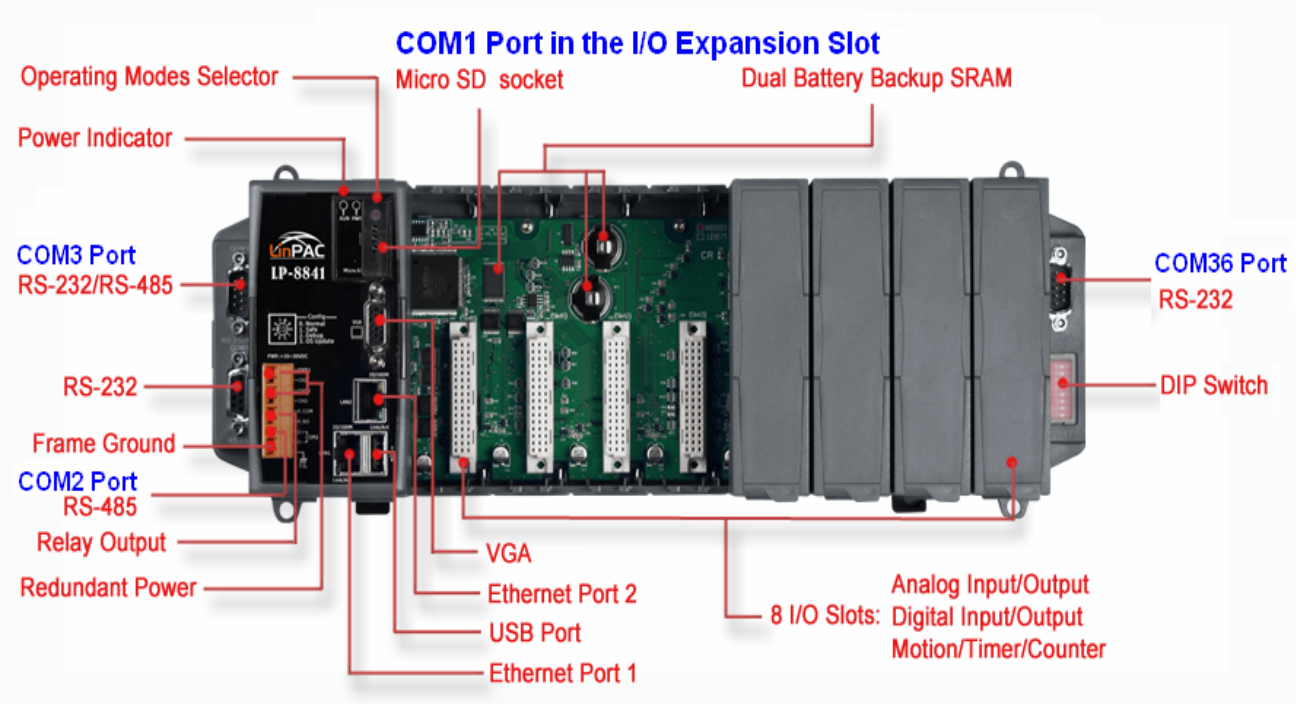


Fig. 8-1

User can try the **stty** command to query or setting COM port. For example, to modify baudrate 9600 to 115200 via COM3 port:

stty -F /dev/ttyS1 ispeed 115200 ospeed 115200

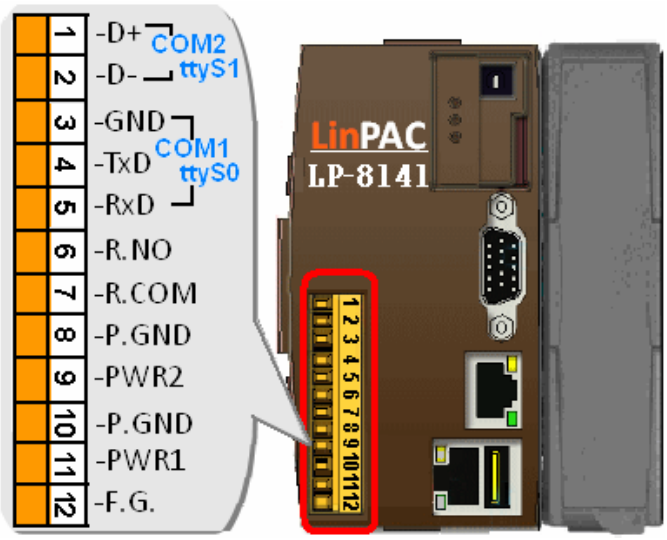


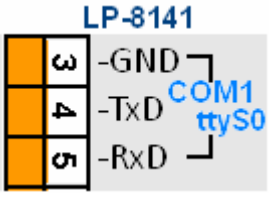
Fig. 8-2

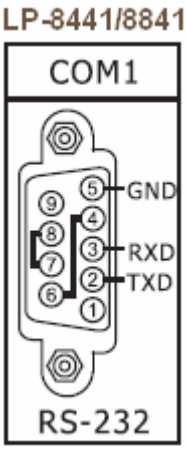
8.1 Introduction of COM1 Port of LinPAC-8x4x

COM1 is an internal I/O expansion port of the LinPAC-8x4x. This port is used to connect the I-87KW series module plugged into the LinPAC-8x4x embedded controller. Users must use the serial command to control the I-87KW series module. For controlling the I-87KW 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; char data=8, parity=0, stopbit=1 ;
unsigned char port=1;    // for all modules in com1 port of LinPAC-8x4x
DWORD baudrate=115200;
Open_Slot(slot);
Open_Com(port, baudrate, data, parity, stopbit);
ChangeSlotTol-87k(slot);
// send command...
Close_Com(port);
Close_Slot(slot);
```



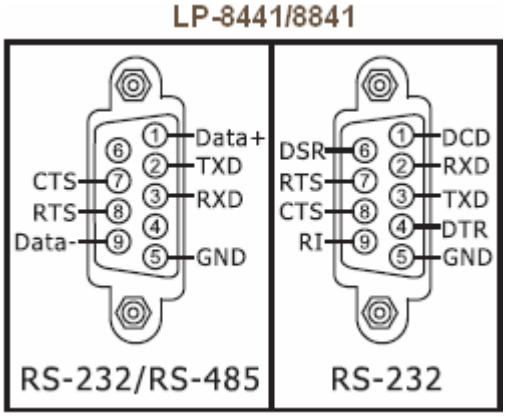


8.2 Introduction of COM3/COM36 Port of LinPAC-8x4x

This COM3/COM36 port is located on the right-upper corner on the LinPAC-8441/8841 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 COM3/COM36 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:

- Test by C language:

```
unsigned char port=3;
DWORD baudrate=9600;
char data=8;
char parity=0;
char stopbit=1;
Open_Com(port, baudrate, data, parity, stopbit);
// send command...
Close_Com(port);
```



- Test in command line: (PC <----------> COM3 of LinPAC-8x4x)

  A) Open "**Hyper Terminal**" of PC to monitor the process of update and the default COM3 port setting is 9600, 8, N, 1

  B) Send data via COM3 port:

  In LinPAC-8x4x:

  Type command: **echo test>/dev/ttyS1**

  And, user must be see the "test" in "Hyper Terminal" of PC

  C) Receive data via COM3 port:

  In LinPAC-8x4x:

  Type command: **cat /dev/ttyS1**

  In PC:

  User can input some words in "Hyper Terminal" of PC

  And, user can see some words in LinPAC-8x4x.

8.3 Introduction of COM2/COM3 Port of LinPAC-8x4x

This COM2/COM3 port provides **RS-485** serial communication (DATA+ and DATA-) and is located on bottom-right corner in the LinPAC-8x4x. You can connect to the RS-485 device with modules like the I-7000, I-87k and I-8000 serial modules(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.

- Test by C language:

```
unsigned char port=2; DWORD baudrate=9600; char data=8, parity=0, stopbit=1;
Open_Com(port, baudrate, data, char parity, stopbit);
// send command…
```

- Test in command line: (PC <-----> i-7520 <-----> COM2 of LinPAC-8x4x) (see Fig 8-3)

  A) Open "**Hyper Terminal**" of PC to monitor the process of update and the default COM2 port setting is 9600, 8, N, 1

  B) Send data via COM2 port:

  In LinPAC-8x4x:

  Type command: **echo test>/dev/ttyS0**

  And, user must be see the "test" in "Hyper Terminal" of PC

  C) Receive data via COM2 port:

  In LinPAC-8x4x:

  Type command: **cat /dev/ttyS0**

  In PC:

  User can input some words in "Hyper Terminal" of PC

  And, user can see some words in LinPAC-8x4x.

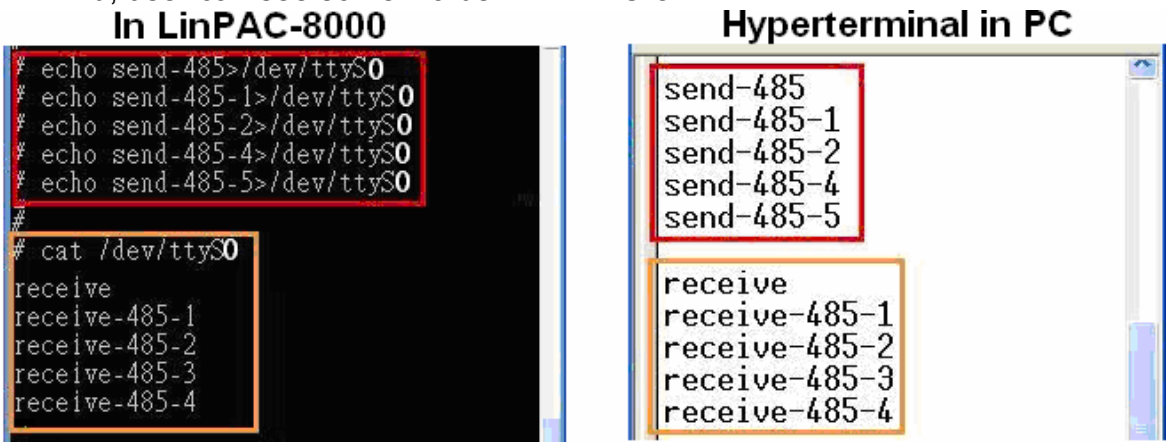


Fig. 8-3

9. LP-8x4x Library Reference in C Language

In this chaper, all the functions of **libi8k.a** 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)
void Read_SN(unsigned char serial_num[] )
int GetNameOfModule(char slot)
void setLED(unsigned int led)
int GetBackPlaneID()
int GetSlotCount()
int GetDIPswitch()
int GetRotaryID()
float GetSDKversion(void)
```

9.2 List Of Digital Input/Output Functions

9.2.1 For I-8000 modules via parallel port

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

void I8017_GetFirmwareVersion(int slot)

int I8017_Init(int slot)

void I8017_SetLed(int slot, unsigned int led)

void I8017_GetSingleEndJumper(int slot)

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 I8017_HEX_TO_FLOAT_Cal(int HexValue, int slot, int gain)

void I8017_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 I8017_CalHex_TO_FLOAT(int HexValue,int gain)

void I8017_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 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. Additional Support

In this chapter, ICP DAS provides extra module supported and instructions to enhance LP-8x4x functionality and affinity.

10.1 N-Port Module (I-8114W, I-8112iW, etc.) Support

The communication modules provide **four** and **two serial ports** respectively. Users can insert them into the LP-8x4x slots. In this way, users can use more serial ports in the LP-8x4x and the expanded maximum number of serial port in the LP-8x4x will be **thirty-four**. The LP-8x4x is a multi-tasking uint, therefore users can control all the serial ports simultaneously. **The serial port number** of I-8114W and I-8112iW modules are presented in the Fig.10-1 and10-2, Fig10-5 is for LP-8141 only, and it is **fixed** according to their slot position.

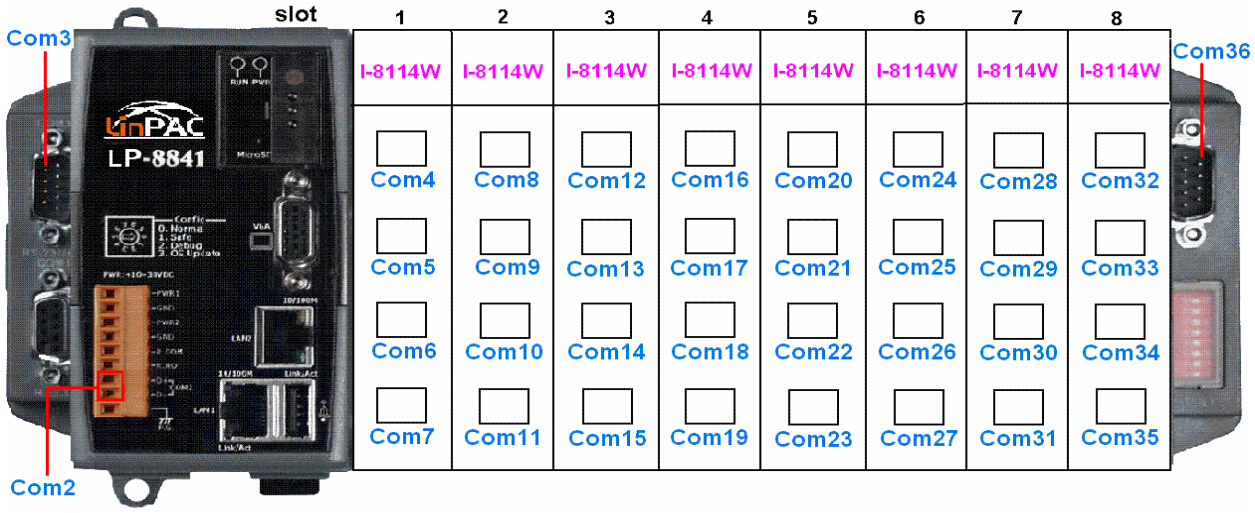


Fig.10-1

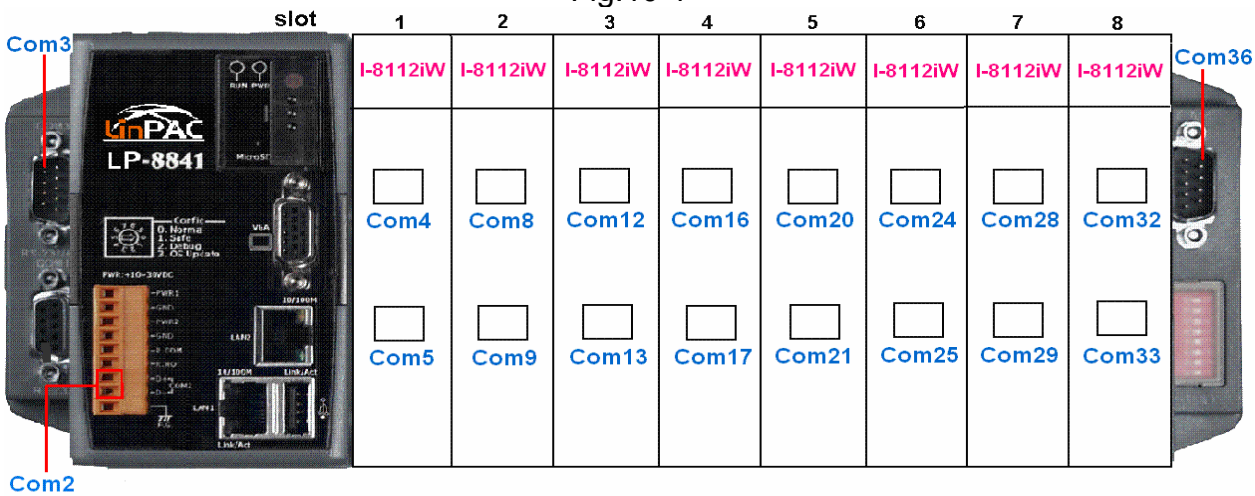


Fig.10-2

Fig.10-3 and 10-4 are the serial port number corresponding to the **device name** in the LP-8x4x.

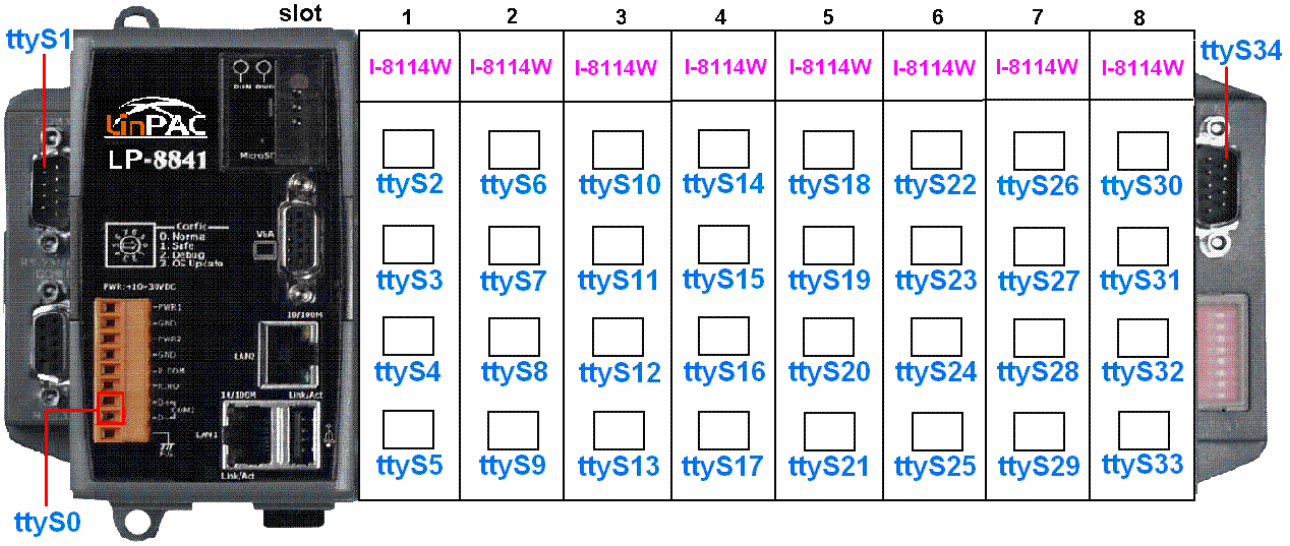


Fig.10-3

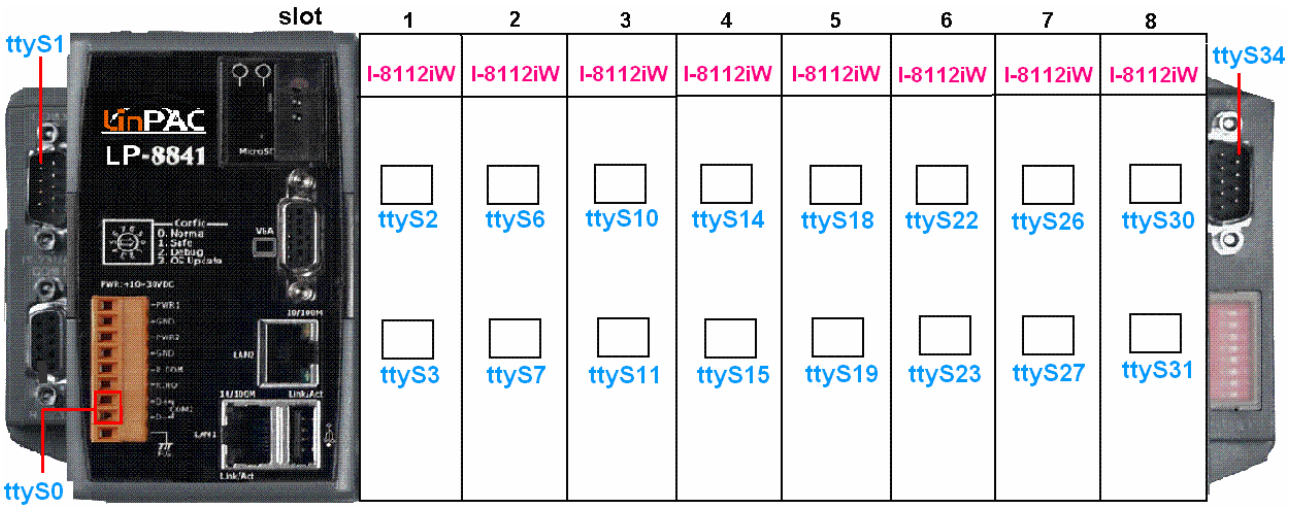


Fig.10-4

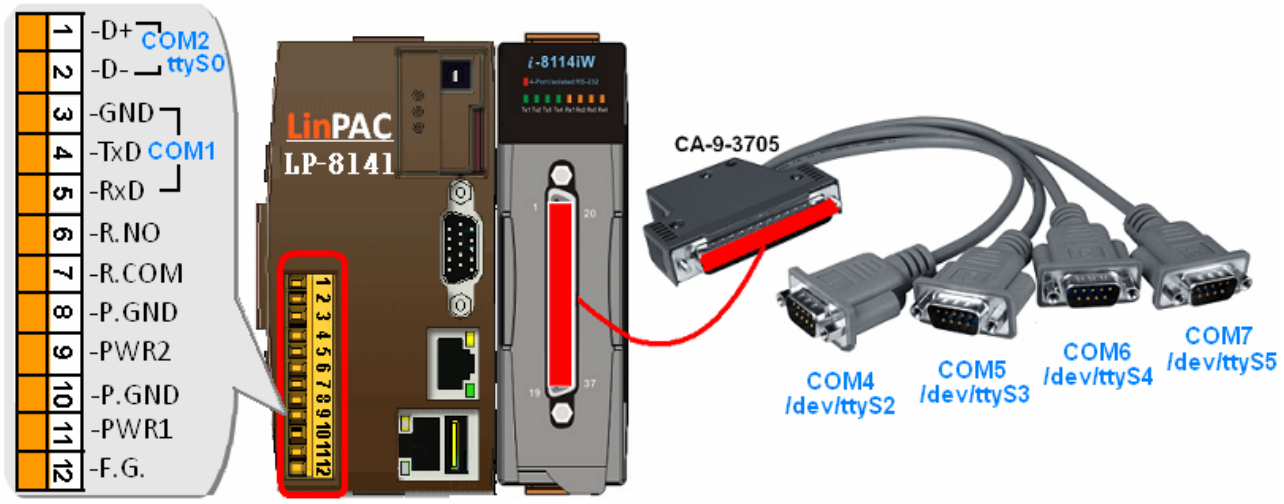


Fig.10-5

I-8K Hight Profile modules selection guide:

Module	Interface	Port	Max. Channels	Max. Speed (K bps)	Isolation
I-8112iW	RS-232	2	16	115.2	2500V
I-8114W	RS-232	4	32	115.2	—
I-8114iW	RS-232	4	32	115.2	2500V
I-8142iW	RS-422/RS-485	2	16	115.2	2500V
I-8144iW	RS-422/RS-485	4	32	115.2	2500V

Refer to : http://www.icpdas.com/products/Remote_IO/i-8ke/selection_rs232_i8k.htm

The demo - **i7kdio_8114.c** will illustrate how to use the I-8114W module in the LP-8x4x. 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-8114W plugged into the slot 2 on the LP-8x4x. The address and baudrate of the I-7044 module in the RS-485 network are 02 and 115200 separately. Fig.10-6 is the control diagram.

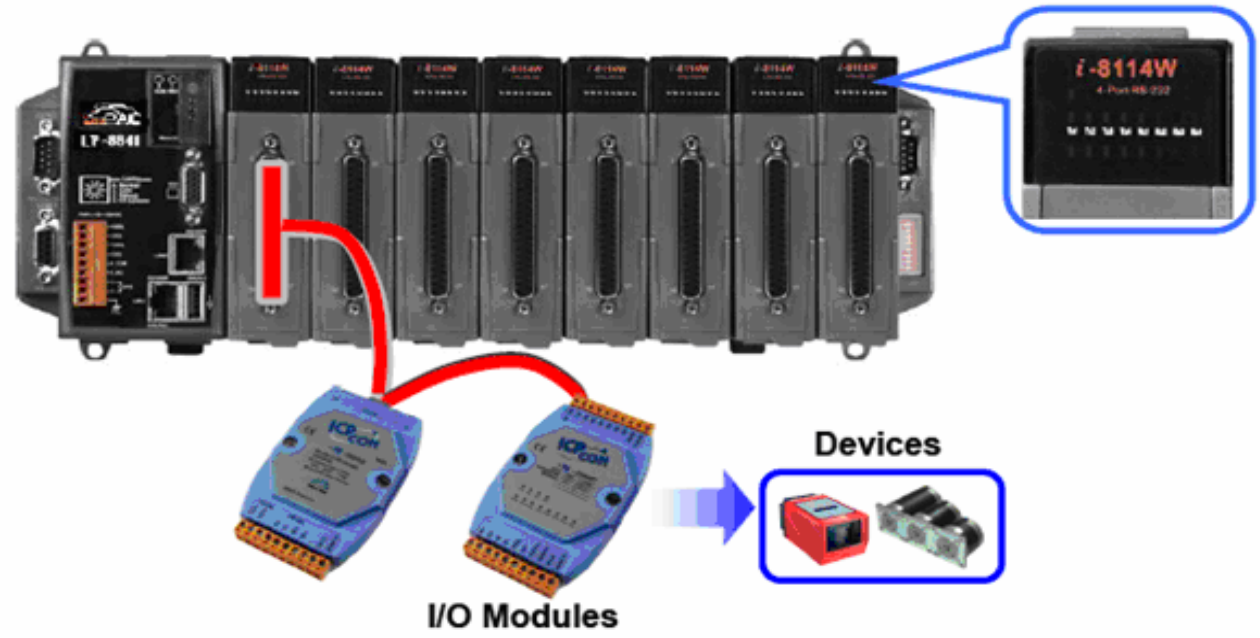
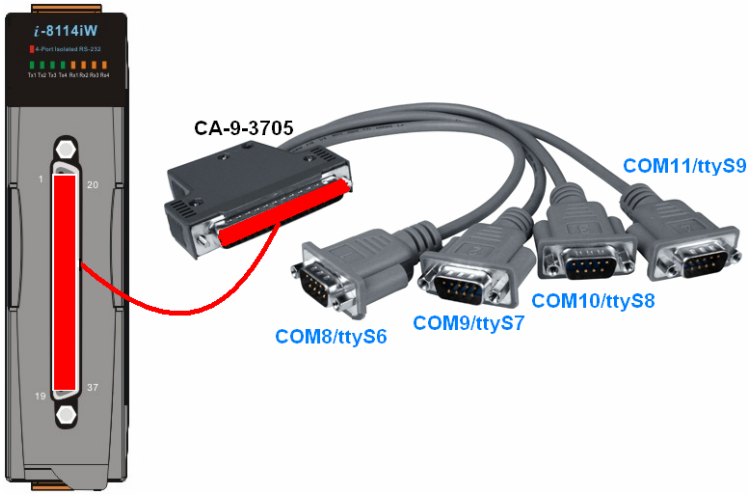


Fig.10-6

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-8114W
      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. 10-7.

Fig. 10-7

10.2 GUI Funtion Support

Now **“ X-window “** is supported in the LinPAC-8x4x and when the LinPAC-8x4x 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 LinPAC-8x4x. The GUI Library in the LinPAC-8x4x is provided with **GTK+ v1.2 & v2.0** Library. Therefore users can design their own **“ SCADA “** screen by the GTK+ Library in the LinPAC-8x4x. (Refer to the Fig. 10-7)

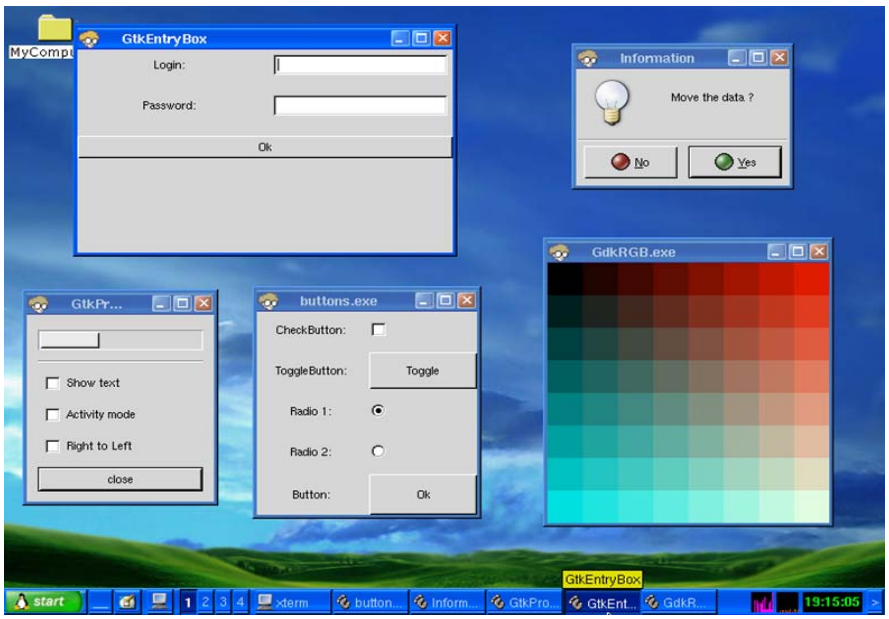


Fig. 10-7

In the meanwhile, we provide some GUI demo programs to control I/O modules of ICP DAS 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 LinPAC-8x4x SDK. (Refer to the Fig. 10-8)

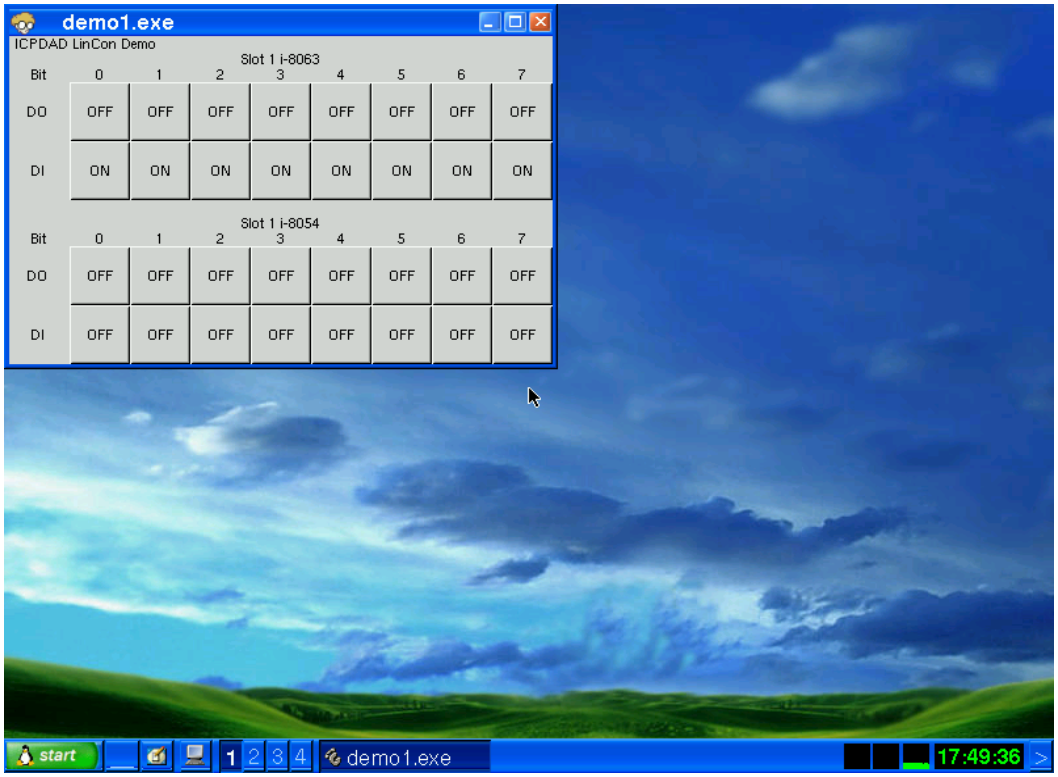


Fig. 10-8

Except GTK+ GUI Function, **“ Java GUI ”** is also supported in the LinPAC-8x4x. 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 LinPAC-8x4x. To execute Java GUI program – Stylepad.jar in the LinPAC-8x4x, users just type in **“ java -jar Stylepad.jar -cp .:Stylepad.jar ”.** Then it will take some time to run up the Java GUI program.

10.2.1 How to boot LinPAC-8x4x without loading X-window

LinPAC-8x4x 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 LinPAC-8x4x.

10.2.2 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 LinPAC-8x4x.

10.3 ScreenShot Support

There is a screenshot program — **" fbshot "** built in to let users to catch the LP-8x4x 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 SD card in the LinPAC first.) If users want to know the detailed parameters of fbshot, just type in **" fbshot --help "**. (Refer to the Fig. 10-9)

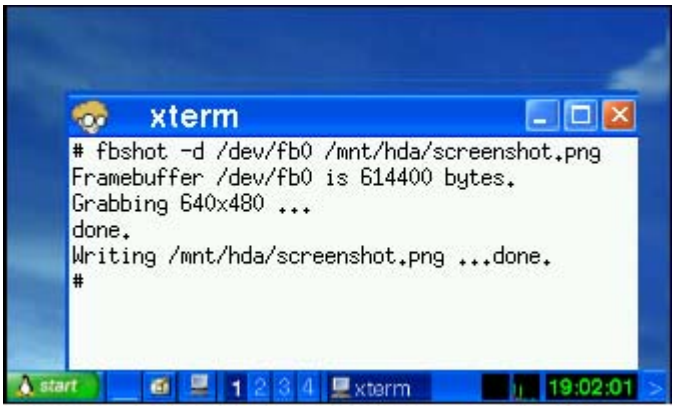


Fig. 10-9

10.4 WebCAM Support

WebCAM is also supported in the LP-8x4x 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 LP-8x4x with **“ USB Interface ”**.

(2) Reboot the LP-8x4x.

(3) Open a **“ Command Prompt ”**. Type in **“ insmod pwc.ko ”** to load the gqcam program decompressor (Refer to the Fig. 10-10) 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 ”**.

```
#
# insmod /lib/modules/2.6.19/pwc.ko
#
# lsmod
Module                  Size  Used by    Tainted: PF
pwc                    84996  0
pm9000                  2912  0
8250                   29204  2
8250_linpac             2656  0 [permanent]
slot                   35788  0
pxamci                  8352  0
dm9000x               276180  0
#
#
```

Fig. 10-10

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.

10.5 Touch Screen Support

10.5.1 USB Touch Screen interface

There are six steps to adjusting the USB touch screen calibration with LinPAC-8x41:

STEP1: Make sure the **usbtouchscreen.ko** and **penmount.ko** were mounted. (Refer to the Fig. 10-11)

```
# lsmod
Module                  Size  Used by    Tainted: P
tsdev                  10024  0
usbtouchscreen          9284  0
penmount                3968  0
8250                   29204  0
8250_linpac             2656  0 [permanent]
slot                   35788  0
pxamci                  8352  0
dm9000x               276180  0
#
```

Fig. 10-11

STEP2: Make sure the microSD card was mounted, which is include **opt** dictionary. (Refer to the Fig. 10-12)

```
# mount
rootfs on / type rootfs (rw)
/dev/root on / type jffs2 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
tmpfs on /var type tmpfs (rw)
shmfs on /dev/shm type tmpfs (rw)
usbfs on /proc/bus/usb type usbfs (rw)
/dev/mmcblk0p1 on /mnt/hda type vfat (rw,fmask=0022,dmask=0022,codepage=cp437,
iocharset=iso8859-1)
/dev/ram0 on /mnt/ramfs type minix (rw)
#
```

Fig. 10-12

STEP3: Users can edit the file : **/etc/init.d/fbman** to modify the setting as below:

- When users open the file : **/etc/init.d/fbman**, users can see the following lines :

  #/usr/sbin/fbset -n 640x480-60

  /usr/sbin/fbset -n **800x600**-70

  It means that the resolution setting is 800*600.

- If users want to change the setting to be **640*480**, just remove the "#" mark in line 2 and add the "#" mark in line 1. Please see the following setting result :

  /usr/sbin/fbset -n 640x480-60

  #/usr/sbin/fbset -n 800x600-70

STEP 4: Typing '**cat /proc/bus/input/devices**' to see a list of currently plugged in devices and associated device can be obtained. (Refer to the Fig. 10-13)

```
# cat /proc/bus/input/devices
I: Bus=0003 Vendor=04d9 Product=1702 Version=0101
N: Name="  USB Keyboard"
P: Phys=usb-pxa27x-1.3/input0
S: Sysfs=/class/input/input0
H: Handlers=kbd event0
B: EV=120003
B: KEY=10000 7 ff800000 7ff febeffdf f3cfffff ffffffff fffffffe
B: LED=7

I: Bus=0003 Vendor=04d9 Product=1702 Version=0101
N: Name="  USB Keyboard"
P: Phys=usb-pxa27x-1.3/input1
S: Sysfs=/class/input/input1
H: Handlers=kbd event1
B: EV=3
B: KEY=39fa d801d101 1e0000 0 0 0

I: Bus=0003 Vendor=15ca Product=00c3 Version=0512
N: Name="USB Optical Mouse"
P: Phys=usb-pxa27x-1.4/input0
S: Sysfs=/class/input/input2
H: Handlers=mouse0 event2
B: EV=7
B: KEY=70000 0 0 0 0 0 0 0
B: REL=103

I: Bus=0013 Vendor=0031 Product=0000 Version=0100
N: Name="Penmount Serial TouchScreen"
P: Phys=ttyS34/serio0/input0
S: Sysfs=/class/input/input3
H: Handlers=mouse1 event3
B: EV=b
B: KEY=400 0 10000 0 0 0 0 0 0 0
B: ABS=3

#
```

Fig. 10-13

STEP5: We are providing the calibration program to test and get the calibration data(Refer to the Fig. 10-14). For example, open a '**Xterm**' windows and run '**calibrator /dev/input/event3**', and then the calibration windows appers.

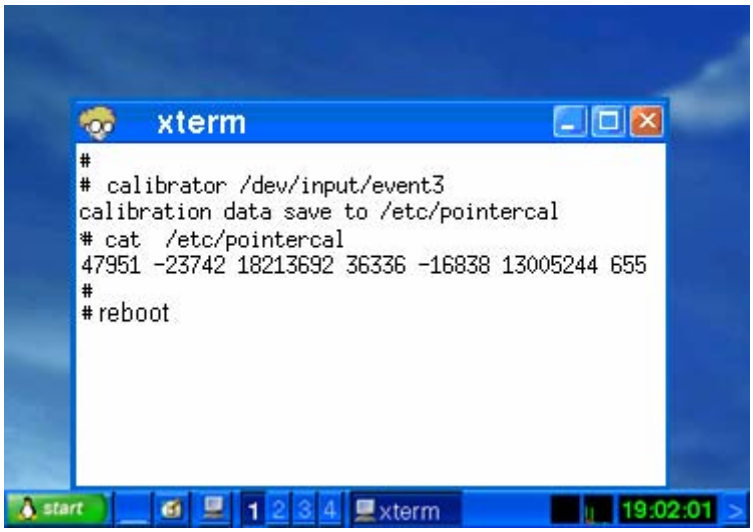


Fig. 10-14

STEP6: After rebooting the LP-8x4x, then the setting will work.

10.5.1 Serial Touch Screen interface

There are eight steps to adjusting the serial touch screen calibration with LP-8441 and LP-8841:

STEP 1: To execute script at startup and shutdown.

- By default, scrips of serial touch screen are disabled at startup, user can use 'mv' command to rename files in /etc/rc2.d. After reboot, it will be executed automatically at boot time (Refer to the Fig. 10-15).

```
# cd /etc/rc2.d
# ls
S04sd                   S70slot                 S98Xserver
S11ifupdown             S71Serial               S99rmnologin
S20ssh                  S72Ramdriver            old
S40inetd                S80hwclock              xS88penmount_serial
S50apache               S90tsdev_usb            xS91tsdev_serial
S60snmp                 S97fbman
# mv S90tsdev_usb xS90tsdev_usb
# mv xS88penmount_serial  S88penmount_serial
# mv xS91tsdev_serial   S91tsdev_serial
#
# ls
S04sd                 S70slot                 S97fbman
S11ifupdown           S71Serial               S98Xserver
S20ssh                S72Ramdriver            S99rmnologin
S40inetd              S80hwclock              old
S50apache             S88penmount_serial      xS90tsdev_usb
S60snmp               S91tsdev_serial
#
```

Fig. 10-15

STEP 2: Make sure the **pm9000.ko** were mounted (Refer to the Fig. 10-16).

```
# lsmod
Module                  Size  Used by    Tainted: PF
pm9000                  2912  0
8250                   29204  2
8250_linpac             2656  0 [permanent]
slot                   35788  0
pxamci                  8352  0
dm9000x               276180  0
#
```

Fig. 10-16

STEP 3: Make sure the microSD card was mounted, which is included **opt** dictionary. (Refer to the Fig. 10-17 and 10-18)

```
# mount  | grep mmc
/dev/mmcblk0p1 on /mnt/hda type vfat (rw,fmask=0022,dmask=0022,
codepage=cp437,iocharset=iso8859-1)
#
```

Fig. 10-17

```
# ls  /mnt/hda
boot    opt
#
```

Fig. 10-18

STEP 4: Modify device node in **/etc/init.d/tsdev_serial**, and setting up script to automatically start processes if necessary (Refer to the Fig. 10-19).

```
# /etc/init.d/tsdev_serial  0.1 2012/09/10 ( 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 "Starting Touch Device services: "
        /opt/bin/inputattach --penmount /dev/ttyS34 --daemon
        EXITCODE=0
        ;;
    stop)
        echo -n "Shutting down Touch Device services: "
        /usr/bin/killall inputattach
        echo "done."
        EXITCODE=0
        ;;
    restart)
        $0 stop
        $0 start
        EXITCODE=$?
        ;;
    *)
        usage
        ;;
    esac
done
```

COM 4

Fig. 10-19

STEP 5: Users can edit the file : **/etc/init.d/fbman** to modify the setting as below:

- When users open the file : **/etc/init.d/fbman**, users can see the following lines :

  #/usr/sbin/fbset -n 640x480-60

  /usr/sbin/fbset -n **800x600**-70

  It means that the resolution setting is 800*600.

- If users want to change the setting to be **640*480**, just remove the "#" mark in line 2 and add the "#" mark in line 1. Please see the following setting result :

  /usr/sbin/fbset -n 640x480-60

  #/usr/sbin/fbset -n 800x600-70

STEP 6: Typing '**cat /proc/bus/input/devices**' to see a list of currently plugged in devices and associated device can be obtained (Refer to the Fig. 10-20).

```
# cat /proc/bus/input/devices
I: Bus=0003 Vendor=04d9 Product=1702 Version=0101
N: Name="  USB Keyboard"
P: Phys=usb-pxa27x-1.3/input0
S: Sysfs=/class/input/input0
H: Handlers=kbd event0
B: EV=120003
B: KEY=10000 7 ff800000 7ff febeffdf f3cfffff ffffffff fffffffe
B: LED=7

I: Bus=0003 Vendor=04d9 Product=1702 Version=0101
N: Name="  USB Keyboard"
P: Phys=usb-pxa27x-1.3/input1
S: Sysfs=/class/input/input1
H: Handlers=kbd event1
B: EV=3
B: KEY=39fa d801d101 1e0000 0 0 0

I: Bus=0003 Vendor=15ca Product=00c3 Version=0512
N: Name="USB Optical Mouse"
P: Phys=usb-pxa27x-1.4/input0
S: Sysfs=/class/input/input2
H: Handlers=mouse0 event2
B: EV=7
B: KEY=70000 0 0 0 0 0 0 0 0
B: REL=103

I: Bus=0013 Vendor=0031 Product=0000 Version=0100
N: Name="Penmount Serial TouchScreen"
P: Phys=ttyS34/serio0/input0
S: Sysfs=/class/input/input3
H: Handlers=mouse1 event3
B: EV=b
B: KEY=400 0 10000 0 0 0 0 0 0 0 0
B: ABS=3
#
```

Fig. 10-20

STEP 7: We are providing the calibration program to test and get the calibration data (Refer to the Fig. 10-21). For example, open a '**Xterm**' windows and run '**calibrator /dev/input/event3**', and then the calibration windows appears. As show in Fig. 10-22.

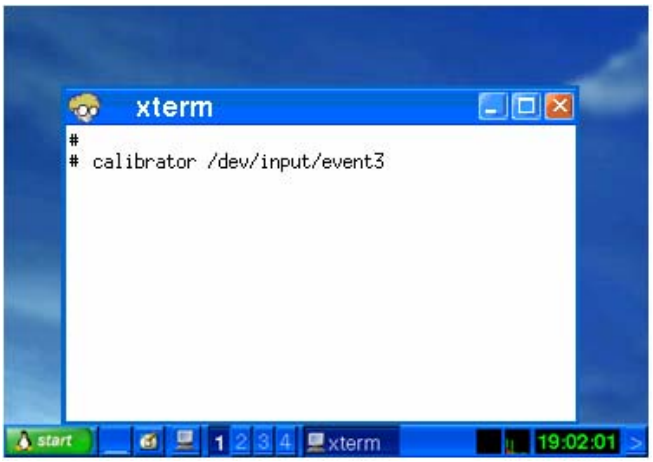


Fig. 10-21

Fig. 10-22

STEP 8: After rebooting the LP-8x4x, then the setting will work.

10.6 Network Support

There are many network functions already built in the LP-8x4x. Here are the network functions supported in the LP-8x4x :

(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 desired services, 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. 10-23)

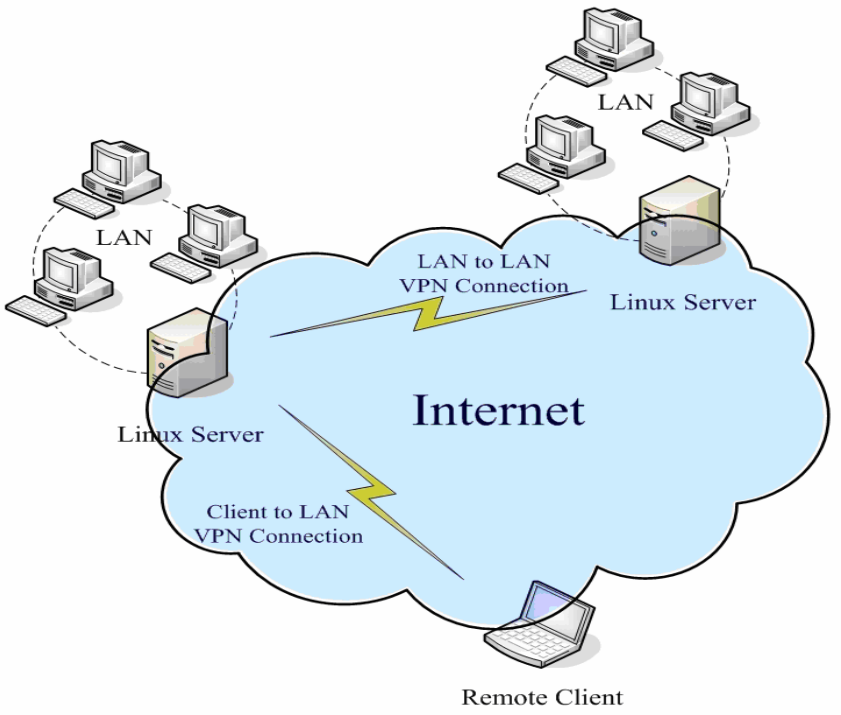


Fig. 10-23

(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 LP-8x4x, users need to modify **/etc/network/interfaces**.

(5) Support Dual LAN

Dual LAN means that users can combine wireless and cable network together through LP-8x4x. 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. 10-24)

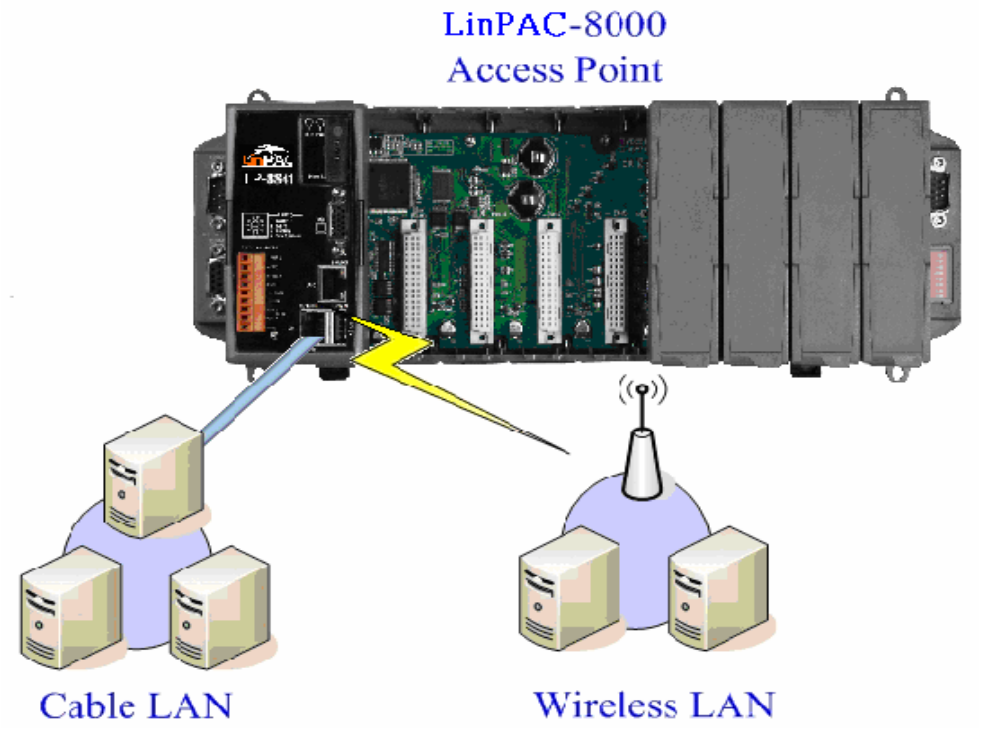


Fig. 10-24

(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 LP-8x4x and provides support for the core Bluetooth layers and protocols. It is flexible, efficient and uses a modular implementation.

(7) Support Modem / GPRS / ADSL

LP-8x4x can be connected to the Internet with “Modem“, “GPRS“(with External RS-232/USB interface) or “ADSL“ mode. The setup method is described separately as follows :

[Modem]

[GPRS]

The following describes the operation of the GPRS modem for example: GTM-201-RS232.

❑ Part 1

If users want to connect the GPRS modem to the COM3 of LP-8x4x, users should modify **/etc/ppp/peers/wavecom** to define COM port first. Please follow the steps as below to connect the GTM-201-RS232(GPRS Modem) with RS-232 interface:

(1) Type **“ vi /etc/ppp/peers/wavecom ”**

(2) To find the “Serial device to which the GPRS phone is connected:” statement, and add device name of COM port.

(3) Type “ **:wq** “ to save and quit the script. (Refer to the Fig. 10-25)

Note: For support 2G GPRS Modem with, please install loadable kernel module—**ftdi_sio.ko** by **insmod** command.

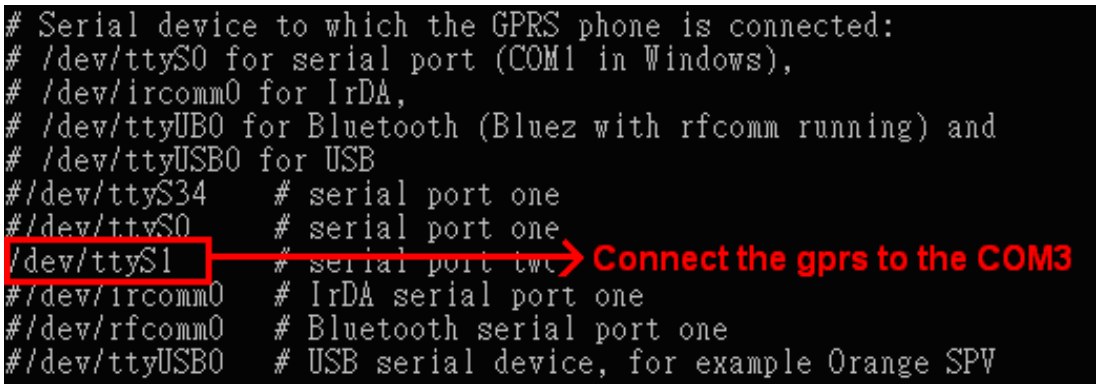


Fig. 10-25

❑ Part 2

Accourding to the GPRS chip default baud rate is “**115200**” bps, GPRS module and device node such as /dev/ttyS2, both of them should have same baudrate. User can use ‘**stty**’ command to set the input and output sepped of the device node. (Refer to the Fig. 10-26)

```
# login
linpac-8000 login: root
Password:
MOKI 0.90
Jan  3 18:01:25 login[1240]: root login on 'console'
-sh: can't access tty; job control turned off
installed modules list
slot 1 ... 8112
# insmod /lib/modules/2.6.19/ftdi_sio.ko
drivers/usb/serial/usb-serial.c: USB Serial support registered for FTDI USB
Serial Device
usbcore: registered new interface driver ftdi_sio
drivers/usb/serial/ftdi_sio.c: v1.4.3:USB FTDI Serial Converters Driver
# stty -F /dev/ttyS2 ispeed 115200 ospeed 115200
# stty -F /dev/ttyS2
speed 115200 baud;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W;
lnext = ^V; flush = ^O; min = 1; time = 0;
-brkint -imaxbel
```

Fig. 10-26

Before startup GPRS, you have to take down the network interface of eth0 and eth1, remove ethernet cable, and type in “**ifdown eth0**” or “**ifdown eth1**” to stop it.

Type in “**pppd call wavecom &**” and then LP-8x4x will be connected to the internet automatically. Remember that the network interface of LinPAC should stop first. If users type in “**ifconfig**” will see the “**ppp0**” section. (Refer to the Fig. 10-27)

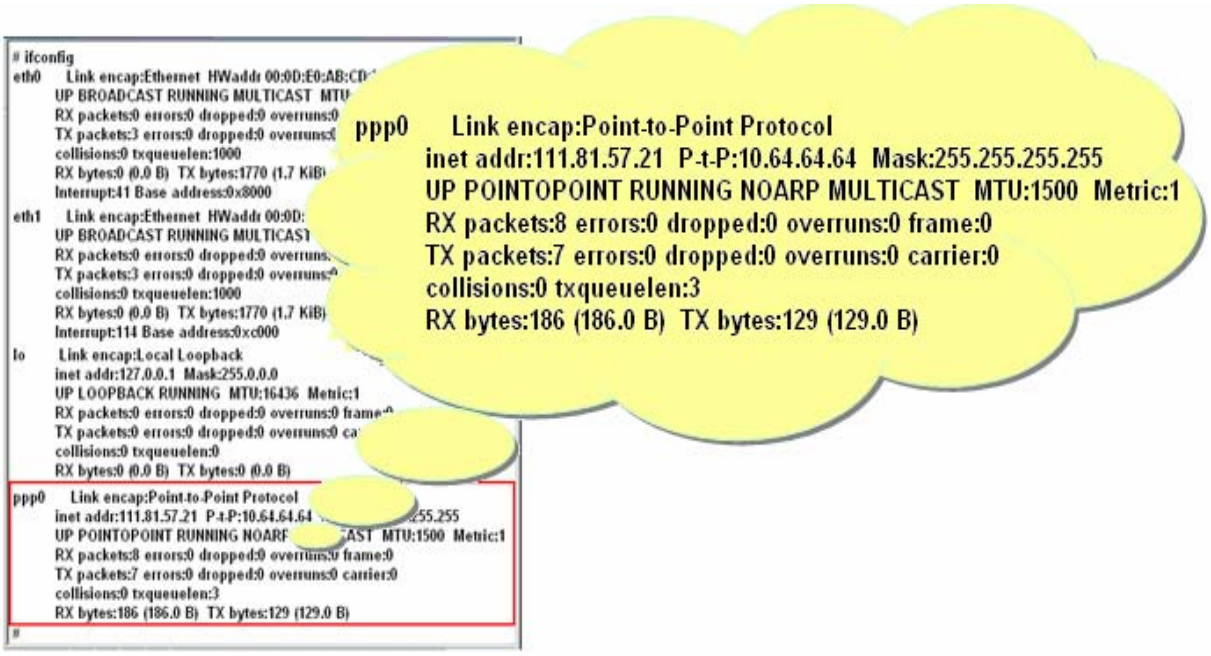
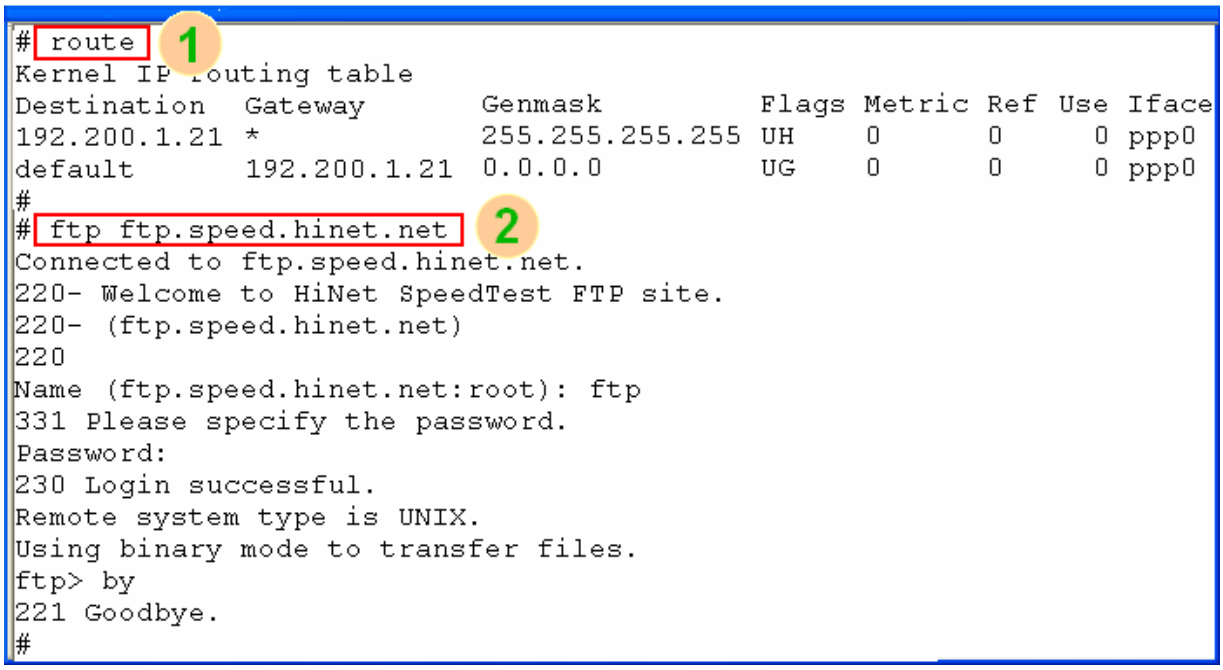


Fig. 10-27

See the routing table below and let's try connecting (Refer to the Fig. 10-28):




```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.200.1.21    *               255.255.255.255 UH    0      0        0 ppp0
default         192.200.1.21    0.0.0.0         UG    0      0        0 ppp0
#
# ftp ftp.speed.hinet.net
Connected to ftp.speed.hinet.net.
220- Welcome to HiNet SpeedTest FTP site.
220- (ftp.speed.hinet.net)
220
Name (ftp.speed.hinet.net:root): ftp
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> by
221 Goodbye.
#
```

Fig. 10-28

GPRS modem selection guide:

Module	Install loadable kernel module	Execute command
GTM-201-USB	/lib/modules/2.6.19/**usbserial.ko** /lib/modules/2.6.19/**sim5218.ko**	pppd call 3g &
GTM-201-RS232 I-8212W	/lib/modules/2.6.19/ **ftdi_sio.ko**	pppd call wavecom &

For more information, please refer to : http://m2m.icpdas.com/m2m_layer2_gprs.html

Note: If user want to try GTM-201-USB, please type "insmod usbserial.ko" first, and "insmod sim5218.ko" to load the program decompressor.

[ADSL]

Users need to type in **" adsl-setup "** first to setup ADSL options. After that, users need to type in **" adsl-connect "** to make LP-8x4x 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 LP-8x4x. Just type in **“ dillo ”** to open the web browser and input the web site address. (Refer to Fig. 10-29) (Note : dillo is placed in the path : **/mnt/hda/opt/bin** so users need to plug microSD card in the LinPAC first.)

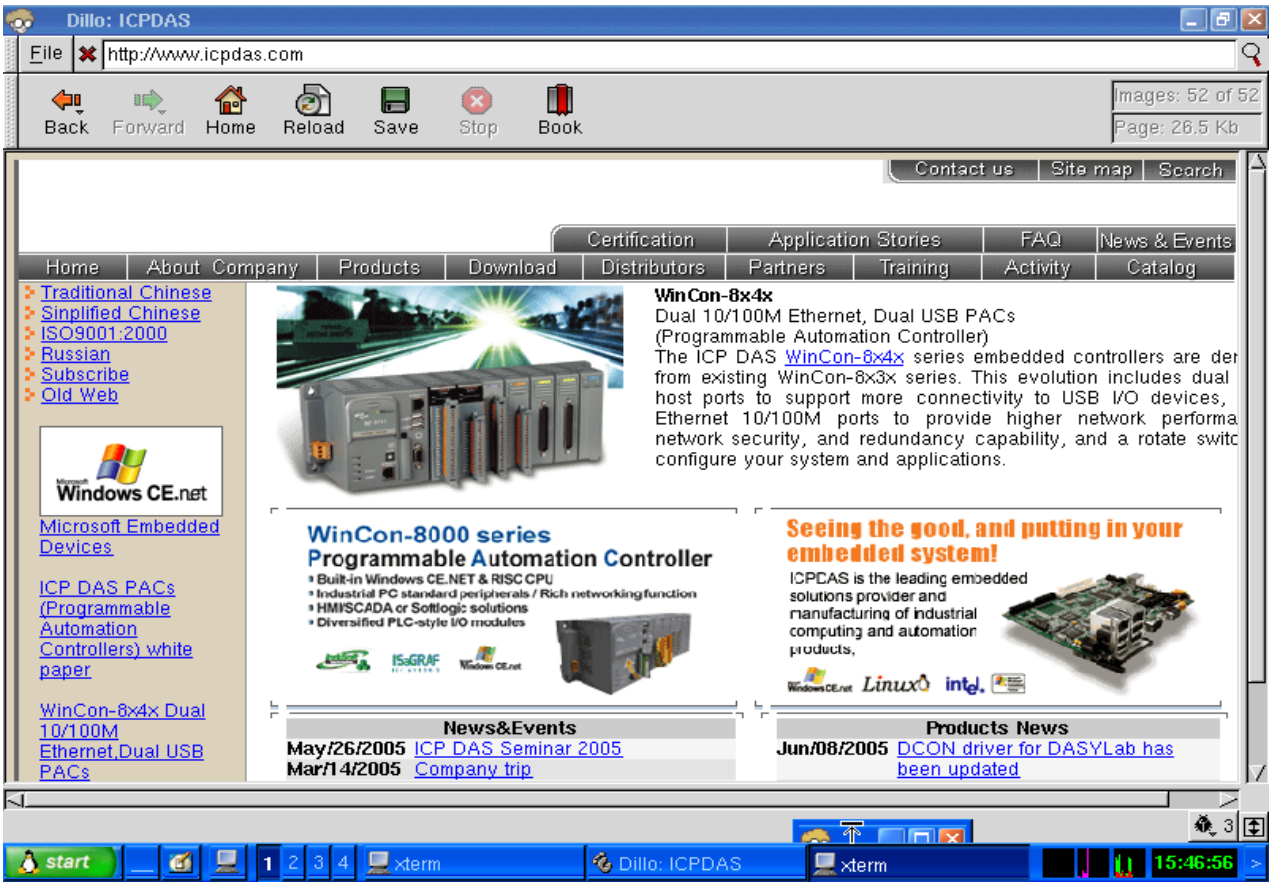


Fig. 10-29

(10) Provide Apache Server

The Web Server — **“ Apache Server ”** has been built in the LP-8x4x 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 LP-8x4x. If it returns a successful web page, it means that the web server in the LP-8x4x has been started. The index web page of Apache Server is in the path : “ **/opt/apache2/htdocs/** “.

These files placed in the microSD card are full functions of Apache Server. So if users want to use other function of Apache Server that are not supported in the LP-8x4x, users just copy them to the path : **/opt/apache2** and reboot.

10.7 USB to RS-232 Support

LP-8x4x support USB to RS-232 converter— I-7560 for example. The I-7560 contains a Windows serial com port via it's USB connection and is compatible with new and legacy RS-232 devices. USB Plug-and-Play allows easy serial port expansion and requires no IRQ, DMA, or I/O port resources.(http://www.icpdas.com/products/Remote_IO/i-7000/i-7560.htm)

Please follow the steps to make the USB to RS-232 converter work smoothly :

(1) Connect the I-7560 to the LP-8x4x with **"USB Interface"**. (Refer to Fig. 10-30)

Fig 10-30

(2) Power on.

(3) Open a **"Command Prompt"**. Type in **" insmod usbserial.ko"** first, and **"insmod pl2303.ko"** to load the program decompressor. (Refer to the Fig. 10-31)

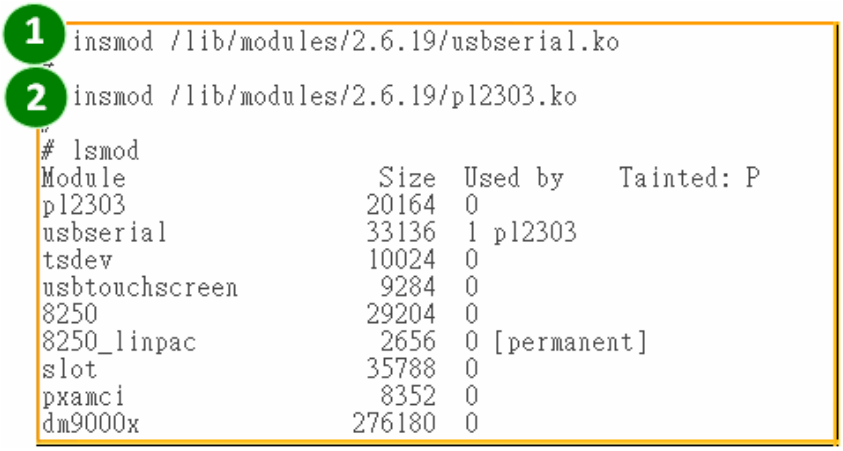


Fig. 10-31

(4) Upon successfully insmodding, a new **/dev/ttyUSB0** serial device is created, user can use "**echo**" and "**cat**" command to send and receive message as below. (Refer to the Fig. 10-32)

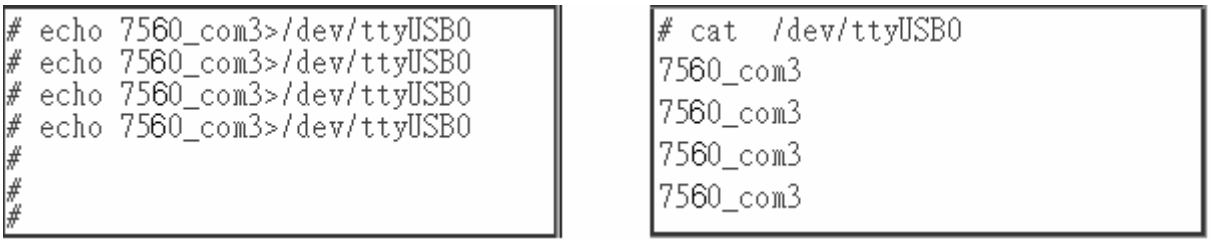


Fig. 10-32

10.8 Other Optional Function

These optional functions are listed below all supported in the LP-8x4x. Users can choose which function to be used in the LP-8x4x and just copy the corresponding file directory to the "**opt**" directory of microSD card. Then reboot LP-8x4x, and the function 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.

☑ Startup MySQL service

If users want to use MySQL in the LP-8x4x, check the "**mysql**" directory in the /opt directory of microSD card, and user can choose one of the following:

a) **Manual**	b) **Auto**
# mysql_install_db # mysqld_safe --user=root & # mysql	# cd /etc/rc2.d # ln -s ../init.d/mysql.server S88mysql # cd /etc/rc0.d # ln -s ../init.d/mysql.server K15mysql # cd /etc/rc6.d # ln -s ../init.d/mysql.server K15mysql # shutdown –r now

Fig. 10-33

```
# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 4.1.10

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
mysql>
```

Fig. 10-34

☑ Compile a mysql demo program

Please refer to the following steps to compile a mysql demo program by LinPAC SDK:

1) Copy mysql directory from **/opt** of microSD card to **C:\cygwin\opt** (Refer to the Fig. 10-35)

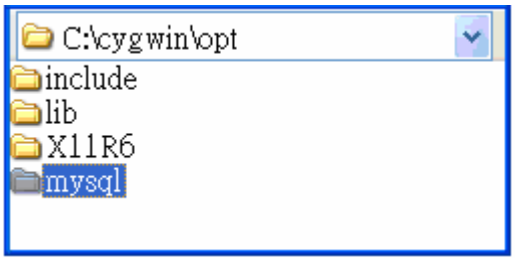


Fig. 10-35

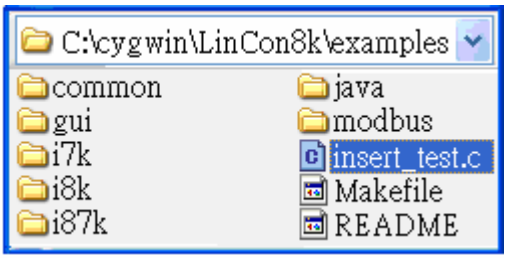


Fig. 10-36

2) Coding a demo program in C:\cygwin\LinCon8k\examples (Refer to the Fig. 10-36)

3) Double click the "LP-8x4x Build Environment" to compile applications.

4) To compile: (Refer to the Fig. 10-37)

C:\cygwin\LinCon8k\examples> **arm-linux-gcc -I..\..\opt\mysql\include\mysql\ -L..\..\opt\mysql\lib\mysql\ insert_test.c -o insert_test.exe -lmysqlclient**

```
C:\cygwin\LinCon8k>cd examples

C:\cygwin\LinCon8k\examples>arm-linux-gcc  -I..\..\opt\mysql\include\mysql\
-L..\..\opt\mysqlf\lib\mysql\ insert_test.c -o insert_test.exe -lmysqlclient

C:\cygwin\LinCon8k\examples> ls ins*
insert_test.c  insert_test.exe

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

Fig. 10-37

(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 10-38)

PHP has been built in the LP-8x4x Kernel so users just boot up LP-8x4x and can use PHP directly in the LP-8x4x.

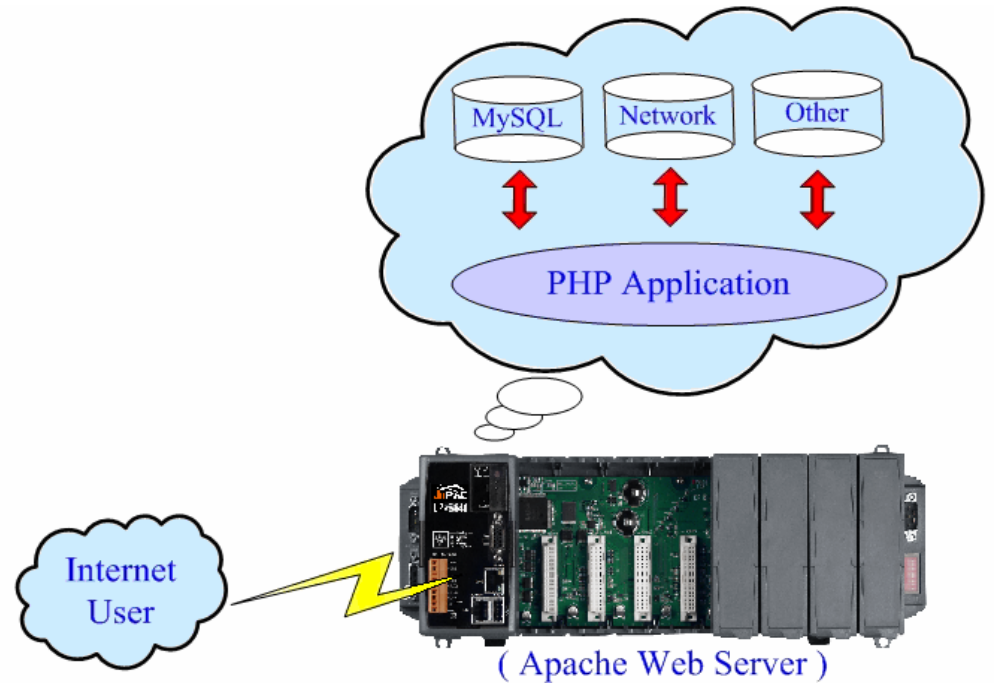


Fig 10-38

(3) Support Perl

Perl（Practical Extraction and Report Language）is also a “open source script language“ and has been built in the LP-8x4x Kernel so users just boot up LP-8x4x and can use Perl directly in the LP-8x4x.

Appendix A. Service Information

This appendix will show how to contact ICP DAS when you have problems in the LP-8x4x or other products.

Internet Service :

The internet service provided by ICP DAS will be satisfied and it includes Technical Support, Driver Update, OS_Image, LinPAC_SDK and User's Manual Download etc. Users can refer to the following web site to get more information :

1. ICP DAS Web Site :

http://www.icpdas.com

2. Software Download :

http://www.icpdas.com/download/index.htm

3. Java Supported Document :

http://www.icpdas.com/download/java/index.htm

4. E-mail for Technical Support :

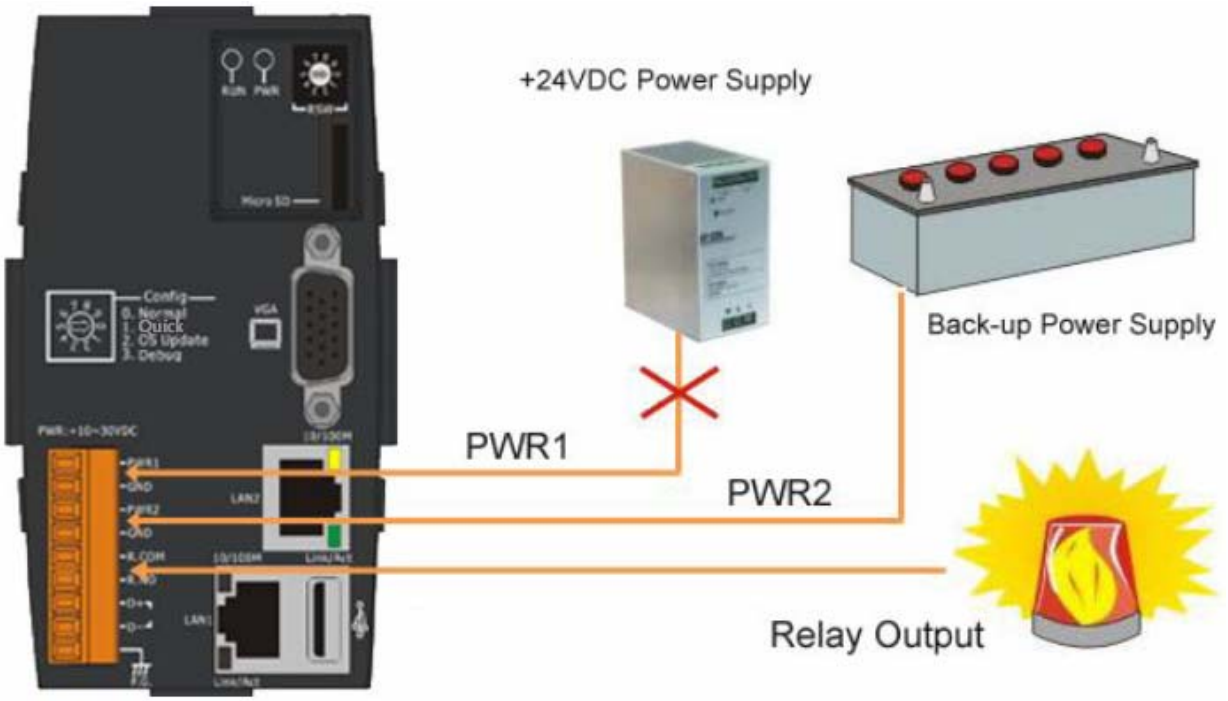
service@icpdas.com

service.icpdas@gmail.com

Appendix B. Redundant Power

The LinPAC provides two power inputs that can be connected simultaneously to live DC power sources. If one of the power inputs fails, the other live source acts as a backup to automatically support the LinPAC's power needs.

The LinPAC provides relay contact outputs to warn technicians on the shop floor when the power fails.



Manual Revision :

Manual Edition	Revision Date	Revision Details
V 0.1	2008. 04	1. Modifiy the LinPAC_SDK installation path 2. Add demo description in chapter 7
V1.0	2008. 08	1. Add sio_open Function 2. Add sio_close Function
V1.1	2008. 12	1. Add settled Function 2. Add GetBackPlaneID Function 3. Add GetSlotCount Function 4. Add GetDIPswitch Function 5. Add GetRotaryID Function 6. Add GetSDKversion Function 7. Add DIO Function for I-8K/7K/87K modules via serial port. 8. Add AIO Function for I-8K/7K/87K modules via serial port.
V1.2	2009. 02	1. Add Read_SN Function 2. Add Send_Binary Function 3. Add Receive_Binary Function
V1.3	2009.07	1. Modify ftp download path. 2. Add descrption of GPRS usage 3. Add descrption of serial port usage 4. Add Modbus API user manual 5. Add Java API user manual
V1.4	2010.06	1. Add I-8017W API Function 2. Add Mysql user guide 3. Add snaphot for I-8112iW device node 4. Add microSD card instruction 5. Add 4.2.3 scan and repair microSD card
V1.5	2010.12	1. Add e-mail account (gmail) 2. Add quick installation guide for Linux 3. Modify Java refer website 4. Add error code explanation
V1.6	2012.04	1. Add sio_open Function 2. Add sio_close Function 3. Add detailed description for GPRS usage 4. Add snaphot for demo1.exe of GTK
V1.7	2012.07	1. Add description for Touch screen support
V1.8	2013.03	1. Add ‘Appendix B’ for redundant power. 2. Add snaphot for mount USB storage.