Software Guide

ICP DAS LP-8x4x SDK

Implement industry control with Linux Technique

(Version 1.10)

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.....6

2. Installation of the LP-8x4x SDK 9

2.1 Quick Installation of the LP-8x4x SDK..... 9
2.2 Introduction of the LP-8x4x SDK 11
2.2.1 Introduction to Cygwin 12
2.2.2 Introduction to Cross-Compilation 12
2.2.3 Download the LP-8x4x SDK.....12

3. The Architecture of LIBI8K.A in the LP-8x4x 13

4. LP-8x4x System Settings.....14

4.1 LP-8x4x Network Settings..... 14
4.1.1 Configuring the IP、Netmask and Gateway Addresses 14
4.1.2 DNS Setting 16
4.2 Using a microSD Card 16
4.2.1 Mounting a microSD Card.....16
4.2.2 Unmounting the microSD Card.....17
4.2.3 Scanning and repairing a microSD Card 17
4.3 Using a USB Storage Device 18
4.3.1 Mounting a USB Storage Device 18
4.3.2 Unmounting the USB Storage Device 19
4.4 Adjusting the VGA Resolution 19
4.5 Automatically executing an applications at startup 20
4.5.1 Configuring a program to run at boot time 20
4.5.2 Disabling a program from running at boot time 23
4.6 Automatic login 23

5. Instructions for the LP-8x4x 24

5.1 Basic Linux Instructions 24
5.1.1 ls : lists the file information (Equivalent DOS Command: dir).....24
5.1.2 cd directory: Changes directory (Equivalent DOS Command: cd) 24
5.1.3 mkdir: creates a subdirectory (Equivalent DOS Command: md).....24

5.1.4 rmdir: deletes the subdirectory which must be empty (Equivalent DOS Command: rd) 24
5.1.5 rm: deletes (removes) the file or directory (Equivalent DOS Command: delete) 25
5.1.6 cp: copies one or more files (Equivalent DOS Command: copy) 25
5.1.7 mv: moves or renames a file or directory (Equivalent DOS Command: move).....25
5.1.8 pwd: displays the full path of the current working directory.....25
5.1.9 who: displays a list of the users current logged on 25
5.1.10 chmod: changes the access permissions for a file 25
5.1.11 uname: displays the Linux version information 26
5.1.12 ps: displays a list of the currently active procedures.....26
5.1.13 ftp: transfers a file using the file transfer protocol (FTP) 26
5.1.14 telnet: extablishes a connection to another PC via Telnet terminal 26
5.1.15 date: prints or sets the system date and time.....26
5.1.16 hwclock: queries and sets the hardware clock (RTC) 26
5.1.17 netstat: displays the current state of the network.....26
5.1.18 ifconfig: displays the ip and network mask information (Equivalent DOS Command: ipconfig).....26
5.1.19 ping: used to test whether the host in a network is reachable.....26
5.1.20 clear: clears the screen.....27
5.1.21 passwd: used to change the password.....27
5.1.22 reboot: reboots the LinPAC (or use 'shutdown –r now').....27
5.2 General GCC Instructions 27
5.2.1 Compile without linking to the LP-8x4x library 28
5.2.2 Compile by linking to the LP-8x4x library (libi8k.a) 29
5.3 A Simple Example – Helloworld.c..... 30
5.4 i-Talk Utility 35

6. LIBI8K.A 37

6.1 System Information Functions 38
6.2 Watch Dog Timer Functions..... 63
6.3 EEPROM Read/Write Functions..... 66
6.4 Digital Input/Output Functions 70
6.4.1 For I-8000 modules via parallel port.....70
6.4.2 For I-7000/I-8000/I-87000 modules via serial port 87
6.5 Analog Input Functions 127
6.5.1 For I-8000 modules via parallel port.....127
6.5.2 For I-7000/I-8000/I-87000 modules via serial port 143
6.6 Analog Output Functions 169
6.6.1 For I-8000 modules via parallel port.....169
6.6.2 For I-7000/I-8000/I-87000 modules via serial port 174

6.7 Error Code Explanation 204
6.8 3-axis Encoder Functions..... 205
6.9 2-axis Stepper/Servo Functions..... 213

7. Demos for LP-8x4x Modules With C Language.....245

7.1 DIO Control Demo for I-7k Modules..... 245
7.2 AIO Control Demo for I-7k Modules..... 250
7.3 DIO Control Demo for I-87KW Modules..... 252
7.3.1 I-87KW Modules in slots of LP-8x4x.....253
7.3.2 I-87KW Modules in slots on I-87KW I/O expansion unit 254
7.3.3 I-87KW Modules in slots on I-8000 Controller 256
7.4 AIO Control Demo for I-87KW Modules..... 257
7.4.1 I-87KW Modules in slots on an LP-8x4x.....257
7.4.2 I-87KW Modules in slots on an I-87KW I/O expansion unit 259
7.4.3 I-87KW Modules in slots on an I-8000 Controller 261
7.5 DIO Control Demo for I-8KW Modules..... 261
7.5.1 I-8KW Modules in slots on an LP-8x4x.....261
7.5.2 I-8KW Modules in slots on an I-8000 Controller 262
7.6 AIO Control Demo for I-8KW Modules..... 264
7.6.1 I-8KW Modules in the slots on an LP-8x4x 265
7.6.2 I-8KW Modules in slots on an I-8000 Controller 267
7.7 Overview of the Module Control Demo Program 269
7.8 Timer Function Demo..... 270

8. Overview of the Serial Ports on the LP-8x4x.....271

8.1 COM1 Port..... 272
8.2 COM3/COM36 Port..... 273
8.3 COM2/COM3 Port..... 274

9. LP-8x4x Library Reference in C Language.....275

9.1 List Of System Information Functions..... 275
9.2 List Of Digital Input/Output Functions 276
9.3 List Of Watch Dog Timer Functions..... 277
9.4 List Of EEPROM Read/Write Functions..... 277
9.5 List Of Analog Input Functions 278
9.6 List Of Analog Output Functions 279
9.7 List Of 3-axis Encoder Functions..... 280
9.8 List Of 2-axis Stepper/Servo Functions..... 280

10. Additional Support.....282

10.1 Support for N-Port Modules (I-8114W, I-8112iW, etc.) 282
10.2 Support for GUI Functionality 287
10.2.1 Booting the LP-8x4x without loading the X-window environment 288
10.2.2 Enabling the X-window environment to load at boot time.....289
10.3 Support for ScreensShot functionality 289
10.4 Support for WebCAM functionality..... 290
10.5 Support for Touch Screen Devices 291
10.5.1 USB Touch Screen interface.....291
10.5.2 Serial Touch Screen interface 293
10.6 Network Support..... 297
10.7 Support for USB to RS-232 Conversion 307
10.8 Additional Optional Functions..... 308

Appendix A. Service Information 312

Appendix B. Redundant Power 313

Manual Revision..... 314

1. Introduction

The Linux operating system has been widely adopted by many users because of its stability, and the face that it is open source and is free. Thanks to increased support from a wide range of companies, Linux has now become one of the most popular operating systems on the market. An additional advantage is that the hardware requirements for implementing the Linux OS in an embedded system are not high, with only a 386 CPU and 8 MB of RAM required. Consequently, besides Microsoft Windows CE, Linux has become an ideal alternative operating system for embedded systems.

The Linux OS places fewer demands on the system resources from the embedded controller, and is therefore an ideal fit since embedded controllers have a number of limitations with regards to system resources. For this reason, the Embedded-Linux OS has been adopted for a new generation product from ICP DAS - the LP-8x4x embedded controller. The main purpose of the LP-8x4x is to enable the numerous Linux users to easily control their embedded systems within the Linux environment.

LP-8x4x is a second generation PAC from ICP DAS, and is equipped with a powerful CPU module based on the Linux kernel 2.6 operating system, and includes a variety of interfaces (VGA, USB, Ethernet, RS-232/485), together with 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, not only has the CPU performance been improved and the OS upgraded from Linux kernel 2.4 to kernel 2.6, but also many reliability features have been added, including a dual LAN, redundant power input, and dual battery backup SRAM, etc., making the LP-8x4x one of the most powerful control systems.

ICP DAS also provides a library file, libi8k.a, which includes all the functions from I-7000/8000/87000 series modules used in the LP-8x4x Embedded Controller. The library is specifically designed for I-7000/8000/87000 series modules based on the Linux platform for use with the LP-8x4x controller. Custom applications can easily be develop for the LP-8x4x using either C or Java, and.NET applications will also be supported In the future. The various functions contained in the library are divided into sub-group functions for ease of use

within the different applications.

The powerful features of the embedded controller are depicted below, including a **VGA port, USB ports for Card Readers, Cameras, Mouse, or Keyboard, etc., a microSD/microSDHC card slot, RS-232/RS-485 serial ports, an Ethernet port,** together with **8 I/O slots**.

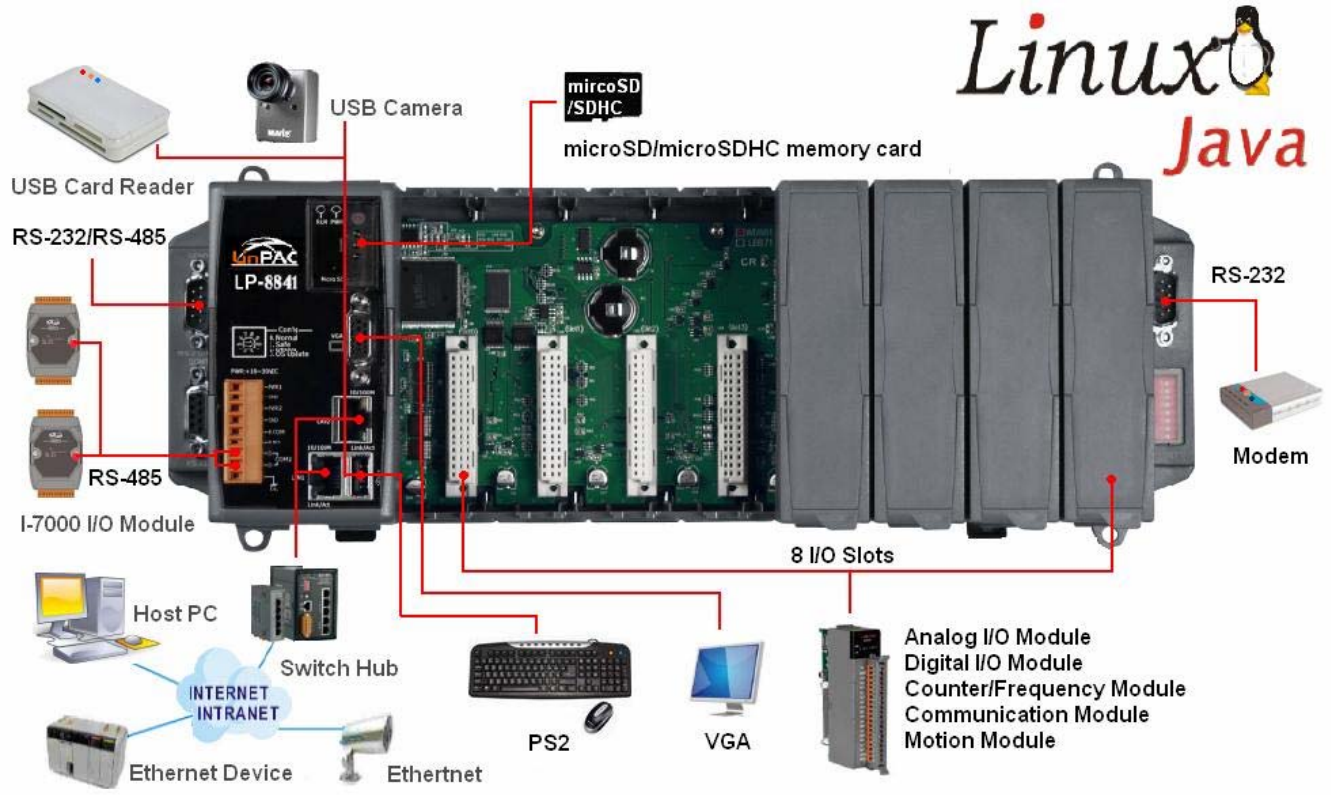


Fig. 1-1

The LP-8x4x controller contains built in HTTP, FTP, Telnet, SSH, and SFTP Servers, meaning that file transfer or remote control is much more convenient with the LP-8x4x. For network communication, **wireless**, **Bluetooth** transfer protocols and **Modem**, **GPRS, ADSL, Firewall** functions are also supported.

The architecture of the LP-8x4x hardware is illustrated in the figure below.

LP-8x41 Series

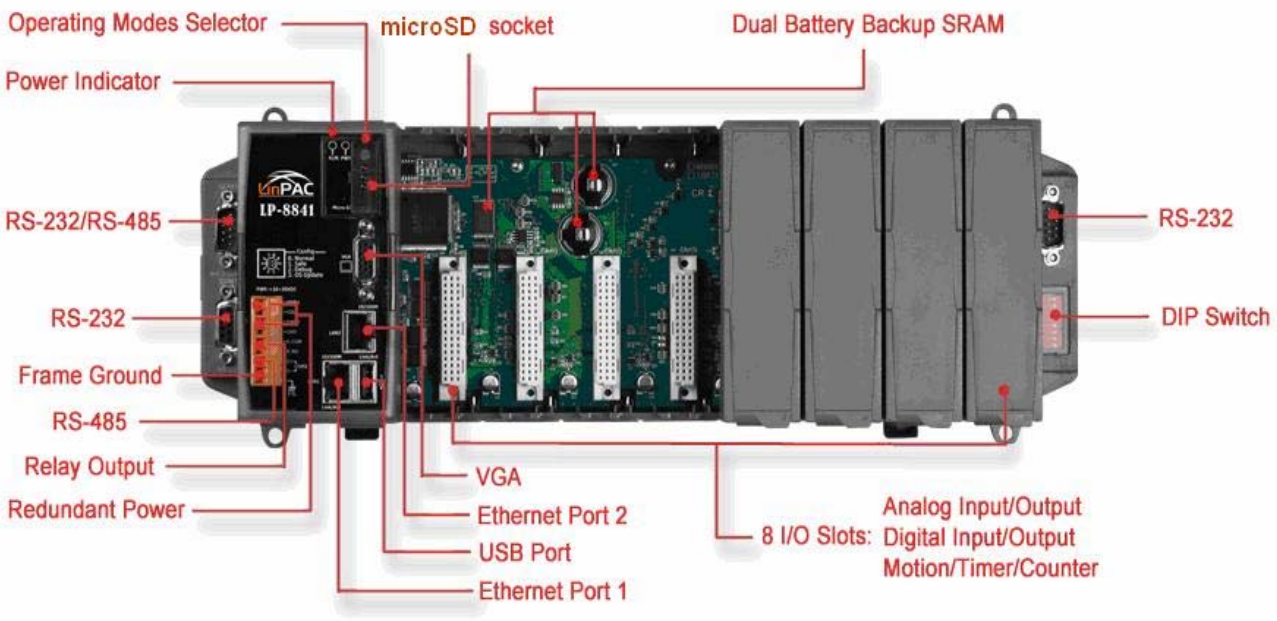


Fig. 1-2

Please note:

- **The flash and microSD disk have a finite number of program-erase cycles. Important information should always be backed up on other media or storage device for long-term safekeeping.**
- **The Li-batterie can continually supply power to the 512 KB SRAM to retain the data for 10 years (It is recommended that batteries are changed each 5~7 year.)**

2. Installation of the LP-8x4x SDK

The “LP-8x4x SDK” is a development toolkit provided by ICP DAS, which can be used to easily develop custom applications for the LP-8x4x embedded controller platform. The toolkit consists of the following items.

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

The latest version of the LP-8x4x SDK (hereinafter referred to as LP-8000 or LP-8K) can be downloaded from: ftp://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x4x/sdk/.

Install the SDK by following the instructions below.

2.1 Quick Installation of the LP-8x4x SDK

(1) Quick Installation Guide for Windows

1. Insert the installation CD into your CD-ROM driver.
2. Open the \napdos\lp-8x4x\SDK\ folder and double-click the icon for the “lp8x4x_sdk_for_windows.exe” file, when the Setup Wizard is displayed, click the “Next>” button to continue, refer to Fig. 2-1.
3. Click the “I accept the agreement” option and then click the “Next” button, refer to Fig. 2-2 below.

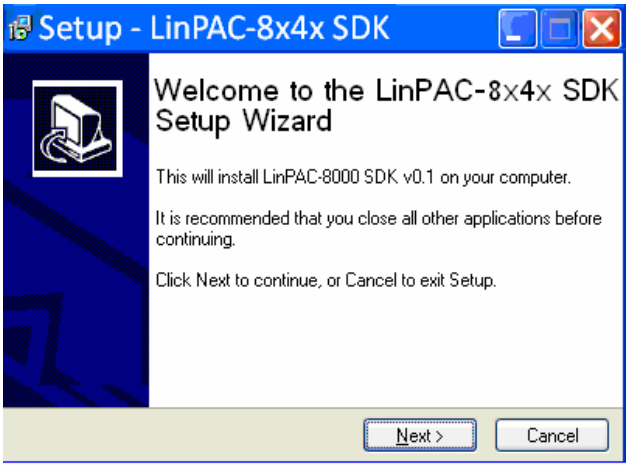


Fig. 2 -1

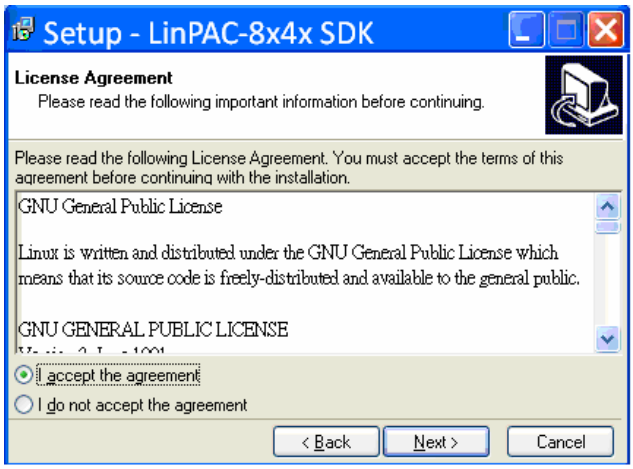


Fig. 2-2

4. The LP-8x4x SDK files will be extracted and installed and a progress bar will be displayed to indicate the status, refer to Fig 2-3.

5. Once the software has been successfully installed, click the "Finish" button to complete the development toolkit installation, refer to Fig. 2-4.

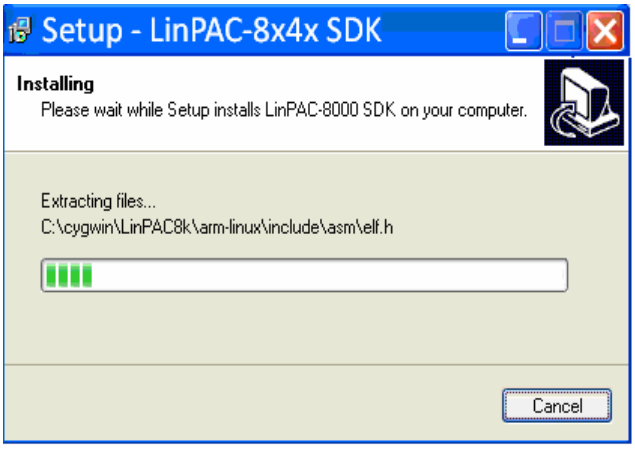


Fig. 2-3

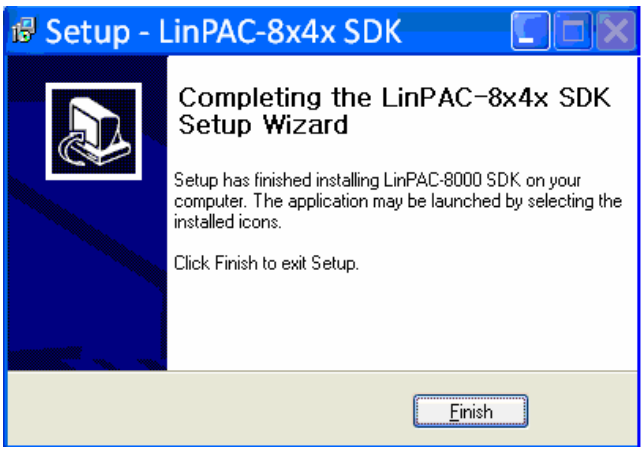


Fig. 2-4

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

Fig. 2-5

7. From the desktop, double-click the shortcut icon for the "LP-8x4x Build Environment" or click the "Start" > "Programs" > "ICPDAS" > "LP-8x4x SDK" > "LP-8x4x Build Environment". A Command Prompt window will then be displayed that allows applications for the LP-8x4x to be compiled. Refer to Fig. 2-6.

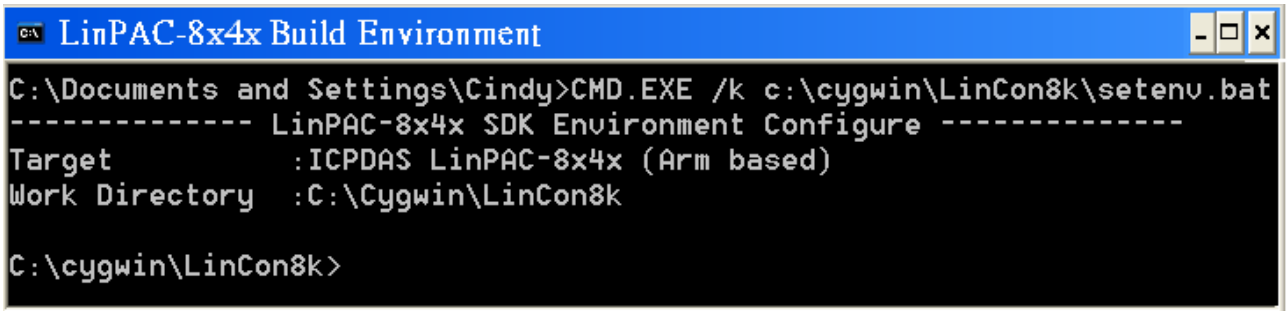


Fig. 2-6

Once the installation is complete, the library and demo files can be found in the following locations:

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

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

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

(2) Quick Installation Guide for Linux

1. Before installing the LP-8x4x SDK, several tasks must be completed, as the root user by ‘sudo’ or ‘su’ command.
2. Locate the “**lp8x4x_sdk_for_linux.tar.bz2**” file in the \napdos\lp-8x4x\SDK\ folder, or visit thre ICP DAS website to download the latest version of the LP-8x4x SDK.

   $ bzip2 -d lp8x4x_sdk_for_linux.tar.bz2

   $ tar zxvf lp8x4x_sdk_for_linux.tar

```
[root@localhost /]# bzip2 -d lp8x4x_sdk_for_linux.tar.bz2
[root@localhost /]#
[root@localhost /]# ls
ADK   etc          misc  opt   selinux   tmp      bin       home
     lp8x4x_sdk_for_linux.tar  mnt       proc     srv       usr
boot  lib          lost+found  net       root     sys       var
dev   media        nuwa  sbin  tftpboot
[root@localhost /]#
[root@localhost /]# tar zxvf lp8x4x_sdk_for_linux.tar
                    :
                    :
lincon/i8k/opt/lib/libmenu.so
lincon/i8k/opt/lib/libgdk_pixbuf-2.0.la
lincon/i8k/opt/lib/libiconv.so
lincon/i8k/opt/lib/libgobject-2.0.la
lincon/i8k/opt/lib/libgdbm.a
lincon/i8k/opt/lib/libjpeg.so
lincon/i8k/opt/lib/libexpat.a
[root@localhost /]#
[root@localhost /]# ls
ADK   etc      media   nuwa      sbin      tftpboot    bin
home  misc     opt     selinux   tmp       boot        lib
lp8x4x_sdk_for_linux.tar  mnt    proc      srv         usr
dev   lincon   lost+found        net       root        sys      var
[root@localhost /]#
```

3. To execute the shell startup script and set the environment variables, enter the following command:

   $. /lincon/linpac.sh

2.2 Introduction of the LP-8x4x SDK

This section will discuss some of the techniques that are adopted in the LP-8x4x SDK, including detailed explanations that describe how to easily use the LP-8x4x SDK. The LP-8x4x SDK is based on Cygwin and is also a Linux-like environment for Microsoft Windows systems, and provides a powerful GCC cross-compiler and an IDE (Integrated Development Environment) that enables LP-8x4x applications to be quickly developed. Therefore, once an application has been created, the LP-8x4x SDK can be used to compile it into an executable file that can be run on the LP-8x4x embedded controller.

2.2.1 Introduction to Cygwin

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. Cygwin is a Linux-like environment for Windows consisting 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 that provide users with the Linux look and feel.

2.2.2 Introduction to Cross-Compilation

Generally, program compilation is performed by running a compiler on the build platform. The compiled program will then run on the target platform. Usually these two processes are intended for use on the same platform. However, if the intended platform is different, the process is called **cross compilation**, where source code on one platform can be compiled into executable files to be used on other platforms. For example, if the "**arm-linux-gcc"** cross-compiler is used on an x86 windows platform, the source code can be compiled into an executable file that can run on an arm-linux platform.

So why use cross compilation? In fact, cross compilation is sometimes more complicated than normal compilation, and errors are easier to make. Therefore, this method is often only employed if the program cannot be compiled on the target system, or if the program being compiled is so large that it requires more resources than the target system can provide. For many embedded systems, cross compilation is the only possible approach.

2.2.3 Download the LP-8x4x SDK

- **For Windows systems : (**Extract the .exe file into to the **C:\ driver**.**)**
  Download the **linpacsdk_for_windows.exe** file from:

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

- **For Linux systems : (**Extract the .bz2 file into to the **root (/) directory**.**)**
  Download the **linpacsdk_for_linux.tar.bz2** file from:

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 library file **libi8k.a** is designed for I-7000/8000/87000 applications running on the LP-8x4x Embedded Controller based on the Linux operating system, and can be applied when developing custom applications **using the GNU C language**. ICP DAS provides a wide variety of demo programs that can be used to easily understand how to implement the functions and ensure that custom projects and applications can be quickly developed.

The relationship between the libi8k.a library and the custom applications is depicted in the figure below.

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

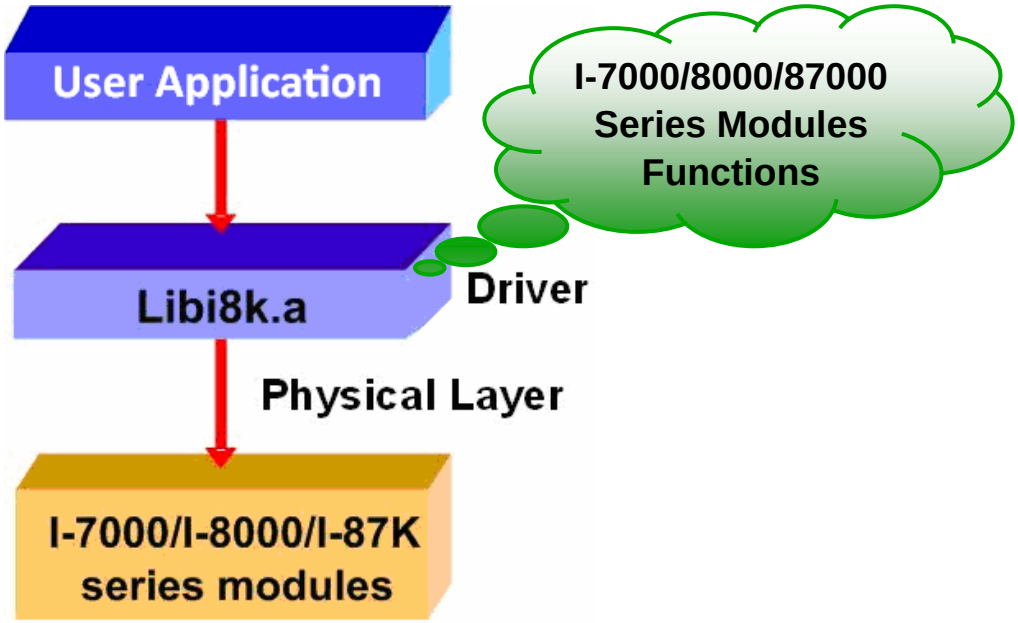


Fig. 3-1

Functions for the 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. Watchdog 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 contained in the libi8k.a library are specifically designed for the LP-8x4x controller, and those functions needed for specific applications can easily be determined from the descriptions provided in chapter 6 and from the demo programs described in chapter 7.

4. LP-8x4x System Settings

The following is a guide to easily configuration the LP-8x4x .

4.1 LP-8x4x Network Settings

There are two methods of configurting the network settings for the LP-8x4x. The first uses **DHCP** and the second is based on an "**Assigned IP**". The factor default configuration is DHCP. However, if a DHCP server is not available on the network system, then the network settings need to be configured using the "Assigned IP" method.

4.1.1 Configuring the IP、Netmask and Gateway Addresses

(1) Using DHCP :

Boot the LP-8x4x and click the **"start/ xterm"** to open a **"Command Prompt"**. Type in **"vi /etc/network/interfaces"** to open the network settings file. Once the network settings file is loaded, remove the **"#"** comment from each line in the dhcp block and add comment out the Assing IP block by adding **"#"** to each entry. Type **":wq"** to save the changes, and then type **"ifup eth0"** to activate the new settings. (Refer to the Fig 4-1 for more details)

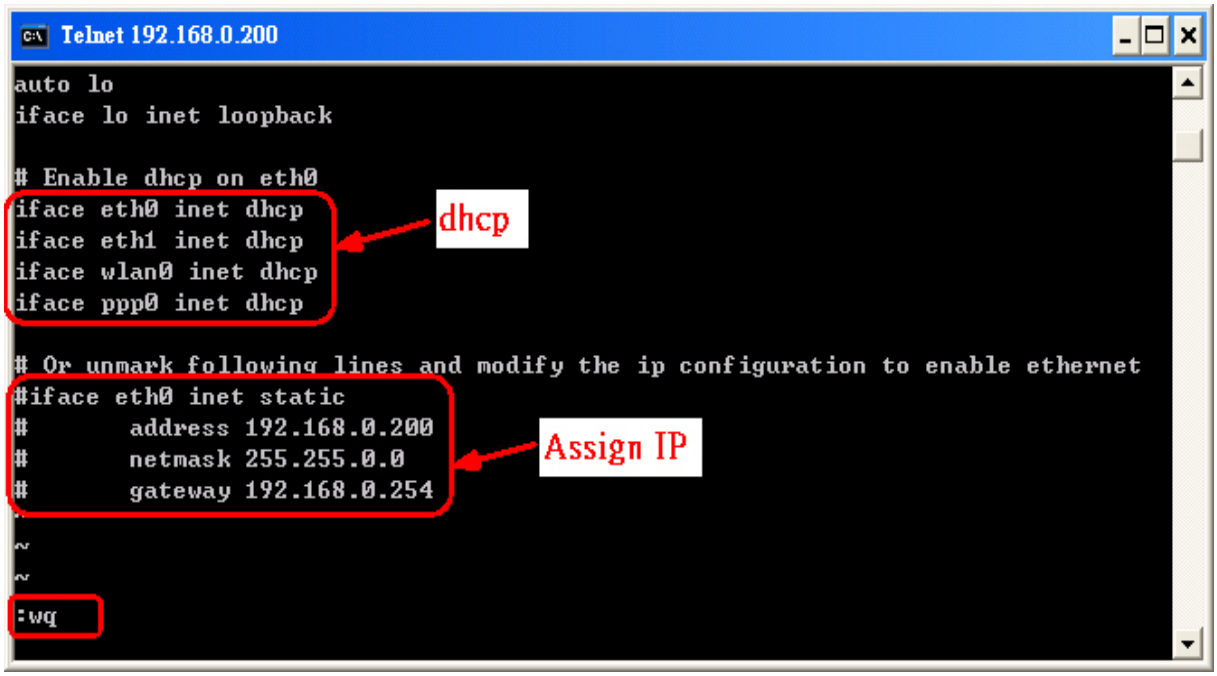


Fig 4-1

(2) Using "Assigned IP":

Boot the LP-8x4x and click the **"start/ xterm"** to open a "command line". Type **"vi /etc/network/interfaces"** to open the network settings file. Once the network settings file is loaded, remove the **"#"** comment from each line in the Assign IP block and comment out the dhcp block by adding **"#"** to each entry. Entry the relevant IP、Netmask and Gateway details in the respective Assign IP block entries. Type **":wq"** to save the changes, and the type **"ifup eth0"** to activate the new settings. (Refer to the Fig 4-2)

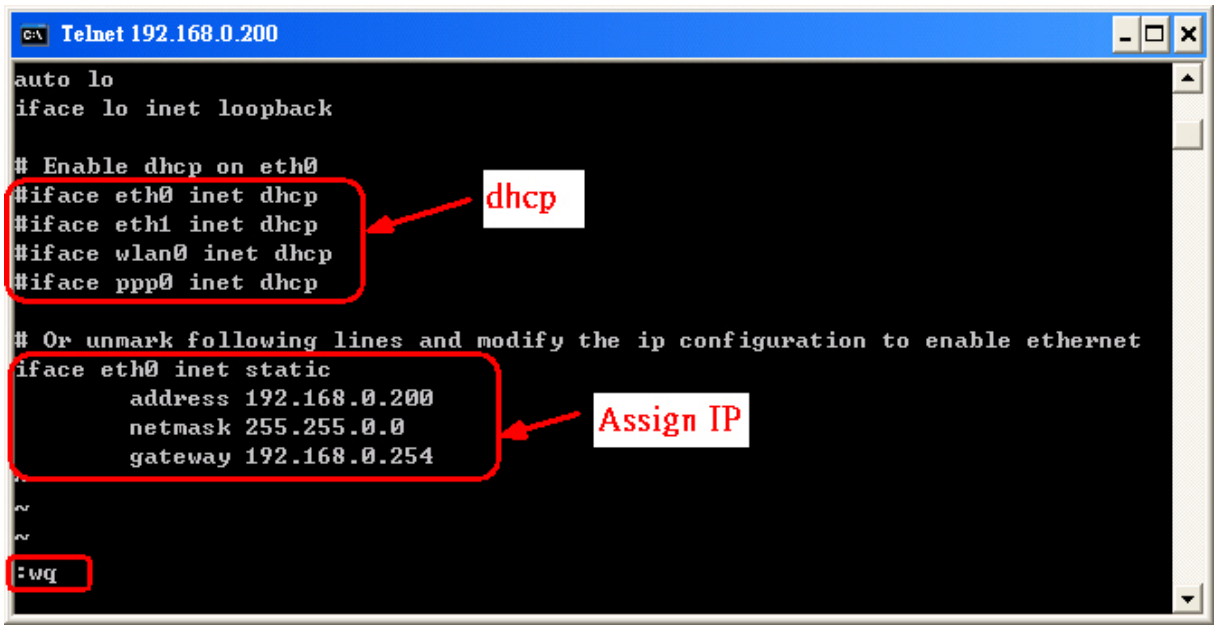


Fig 4-2

After configuring the LinPAC network settings, type **"ifconfig"** to verify that the network settings are correct. Refer to the Fig 4-3 for more details.

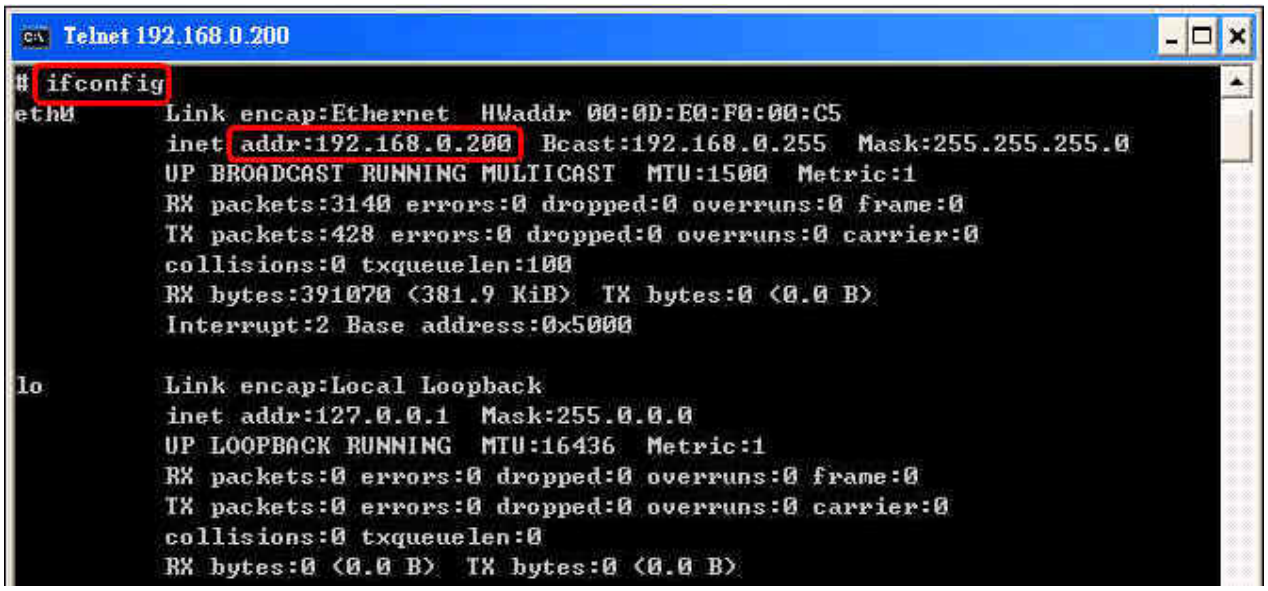


Fig 4-3

4.1.2 DNS Setting

Boot the LP-8x4x and click the **“start/ xterm”** to open a “command line”. Type **“vi /etc/resolv.conf”** to open the DNS setting file. Once the DNS settings file is loaded, type “DNS server” in the **“nameserver”** field. Type **“:wq”** to save the changes, and type **“reboot”** to reboot the LP-8x4x so that the new settings can take effect. Refer to the Fig 4-4 for more details.

Fig 4-4

4.2 Using a microSD Card

Files contained on a mounted microSD card can be accessed from 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/ram0 on /mnt/ramfs type minix (rw)
#
```

Fig 4-5

When using a microSD card, be sure to pay attention to the following items:

1. Unmount the microSD card before removing it.
2. Do not power off or reboot the LP-8x4x while data is being written to or read from the microSD card.
3. The microSD card must be formatted with the VFAT/EXT2/EXT3 file system.

4.2.1 Mounting a microSD Card

To use a microSD card, insert the microSD card into the socket on the LP-8x4x (Refer to Fig. 1-3), and it will be automatically mounted when the LP-8x4x is booted. The files of SD card

can then be access from the **/mnt/hda** directory.

If the card is not mounted automatically, type "**/etc/init.d/sd start** ", to mount it.

4.2.2 Unmounting the microSD Card

Before removing the microSD card from the LP-8x4x, unmount the card by entering the following steps:

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

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

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

The microSD card can then be safely removed to prevent damage to the card.

4.2.3 Scanning and repairing a microSD Card

After the LP-8x4x is booted, the microSD card will be named "**/dev/mmcblk0p1**". It is recommended that the microSD card is unmounted first before attempting to perform a scan or repair.

- **blockdev**: this command is used to call block device ioctls from the command line.
  e.g. blockdev --report /dev/mmcblk0p1 (print a report for device)
  blockdev -v --getra --getbz /dev/mmcblk0p1 (get readhead and blocksize)
- **fsck.minix**: this command is used to perform a consistency check for the Linux MINIX filesystem.
  e.g. fsck.minix -r /dev/mmcblk0p1 (performs interactive repairs)
  fsck.minix -s /dev/mmcblk0p1 (outputs super-block information)
- **fsck.vfat** : this command is used to check and repair MS-DOS file systems
  e.g. fsck.vfat -a /dev/mmcblk0p1 (automatically repair the file system)
  fsck.vfat -l /dev/mmcblk0p1 (list path names of files being processed)
- **mkfs**: this command is used to build a Linux file system on a device, usually a hard disk partition.
  e.g. 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**: this command is used to make a MINIX filesystem.
  e.g. 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**: this command is used to make an MS-DOS filesystem.

  e.g. mkfs.vfat -A /dev/mmcblk0p1 (use Atari variation of the MS-DOS filesystem)

  mkfs.vfat -v /dev/mmcblk0p1 (verbose execution)

4.3 Using a USB Storage Device

USB storage devices are not automatically mounted to the LP-8x4x, sot it must be manually mounted before attempting to access the USB storage device.

4.3.1 Mounting a USB Storage Device

To mount a USB storage devices follow the procedure described below:

(1) Type "**mkdir /mnt/usb**" to create a directory named "usb".

(2) Type "**mount /dev/sda1 /mnt/usb**" to mount the USB storage device to the usb directory and then type "**ls /mnt/usb**" to view the contents of the 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 Unmounting the USB Storage Device

Before removing the USB storage device from the LP-8x4x, the device must be unmounted to prevent any damage to the device. To unmount the device, type the "**umount /mnt/usb**" command and then remove the USB storage device.

4.4 Adjusting the VGA Resolution

The LinPAC supports two VGA resolutions -- **640x480** and **800x600**, and the **default setting is 800x600**. To change the VGA resolution, follow the procedure described below:

(1) Type **"vi /etc/init.d/fbman"** to open the VGA resolution configuration file.

(2) To set the resolution to be 640x480, comment out the reference to 800x600 by adding **"#"** to the entry, and then remove the **"#"** comment from the 640x480 entry. (Refer to Fig. 4-7)

❑ For example, to set the resolution to 800x600, open the file "/etc/init.d/fbman", the code should be as follows:

#/usr/sbin/fbset -n 640x480-60
/usr/sbin/fbset -n **800x600**-70

❑ To change the resolution to 640x480, the code should be as follows:

/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 **":wq"** to save the changes, and then type **"shutdown –r now"** to reboot the LP-8x4x, so that the new settings can take effect. (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 Automatically executing an applications at startup

A “run level” is an operation mode that is used to determine which programs are executed during system startup. The default run level for the LP-8x4x is level **2**.

The run level file is located in the /etc/init.d directory and contains the scripts that will be executed at boot time. These scripts are referenced by creating symbolic links in the /etc/rc2.d directory.

The format for the naming of these links is named S<2-digit-number><original-name>. The 2-digit number determines the order in which the scripts are executed. The valid range is from 00 to 99 and the lower numbered file will executed earlier. Scripts prefixed with an **S** are called at startup, and those prefixed with either a **K or an x** are called when the system is closed or shut down.

4.5.1 Configuring a program to run at boot time

To configure a program to run at boot time, create a startup script that runs the required commands to be automatically executed then save it in the “/etc/init.d” directory. The script must then be symbolically linked to the “**/etc/rc2.d”** directory.

The procedure for creating a script is described below:

(1) Create a script with the filename “hello” by typing “**vi /etc/init.d/hello**“. This will allow the script that executes the programs edit. Type “**:wq**“ to save the script and quit. (Refer to the Fig. 4-9)

Note: If necessary, the **PATH** and **LD_LIBRARY_PATH** environment variable should be included in the script. Check the “/etc/init.d/webcam” file for an example. Refer to the Fig. 4-10 for more details.

(2) Type “**chmod 755 /etc/init.d/hello**“ to change the access permissions for the file.

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

(4) Type ”**ln -s ../init.d/hello /etc/rc2.d/S85hello**“ to create a symbolic link to the script file so that it will be automatically executed at boot time. Refer to the Fig. 4-11 for more details.

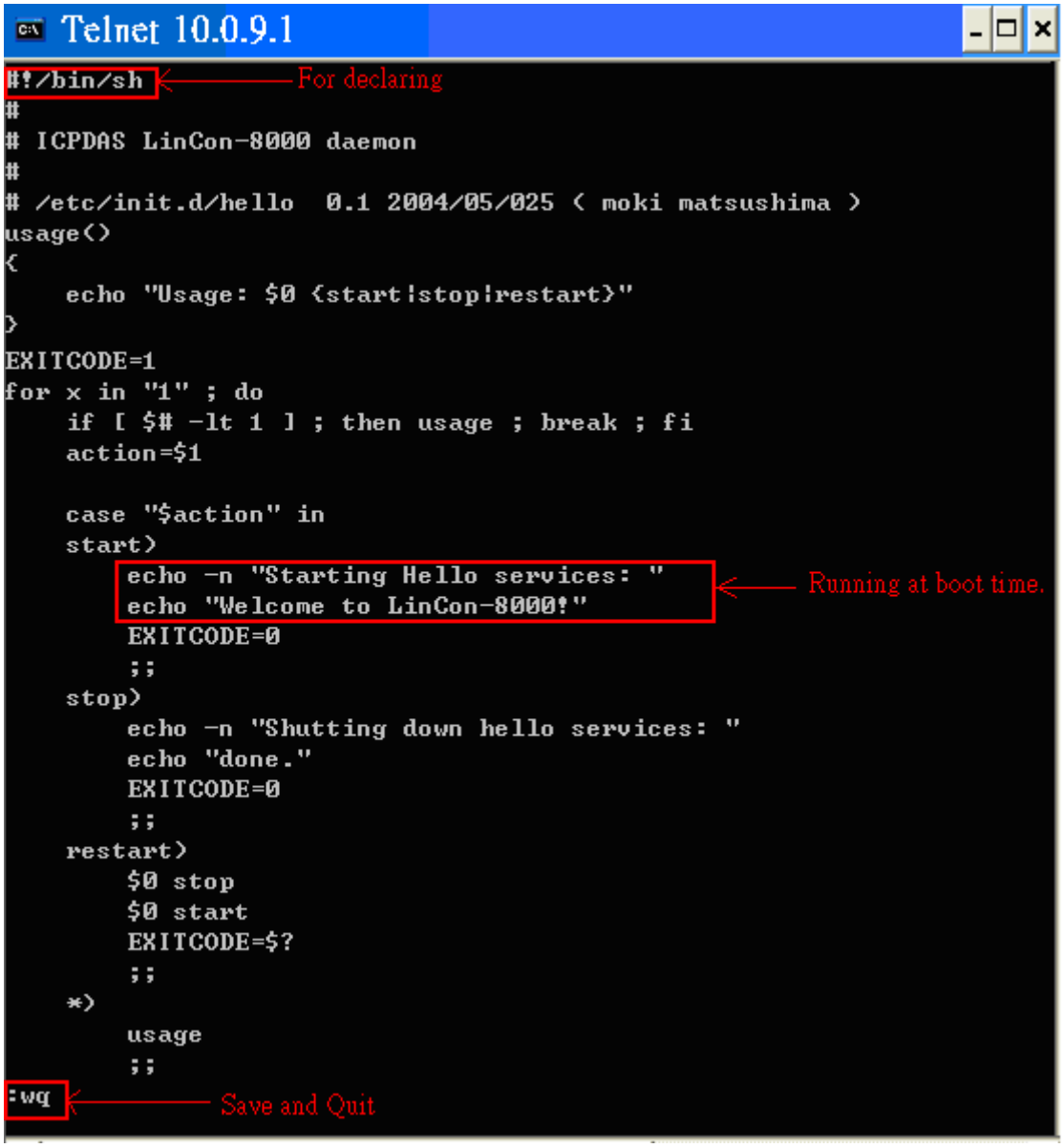


Fig. 4-9

```
# 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

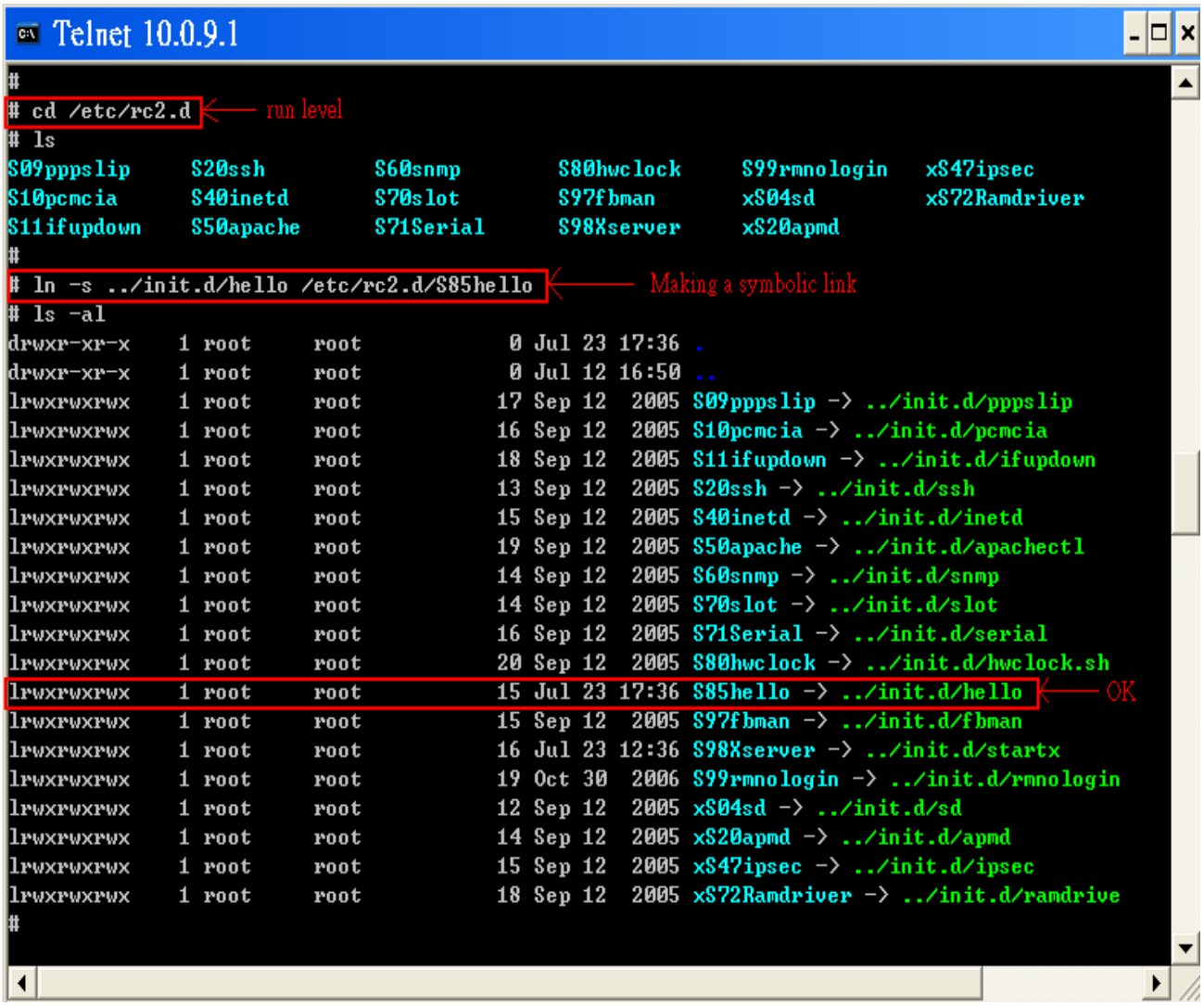


Fig. 4-11

4.5.2 Disabling a program from running at boot time

The procedure for disabling a script is described below:

(1) Type "**cd /etc/rc2.d**" to change directory to the default run level.

(2) Type "**mv S85hello xS85hello**" to rename the S85hello symbolic link and prevent the program from automatically executing at boot time.

4.6 Automatic login

Automatic Login allows a specified user to log into the console (normally /dev/tty1) when the system is first booted without displaying a prompt requesting a username or password. This is achieved using the **mingetty** command, and the setup procedure is as follows:

(1) Login as root and open the "**/etc/inittab**" file for the LP-8x4x.

(2) Modify the entry for the first terminal— **tty1**, as shown in Fig. 4-12, After rebooting the LP-8x4x, it will automatically login to the root account.

```
# /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

This section, provides an introduction to some of the more commonly used Linux instructions. These Linux instructions are similar to those used in DOS, and are generally expressed in lower case letters.

5.1 Basic Linux Instructions

5.1.1 ls : lists the file information (Equivalent DOS Command: dir)

Parameter	Description	Example
–l	Lists detailed information related to the files	ls –l
–a	Lists all files, including hidden files	ls –a
–t	Lists the files arranged in date/time order	ls –t

5.1.2 cd directory: Changes directory (Equivalent DOS Command: cd)

Parameter	Description	Example
..	Move to the parent directory	cd ..
~	Move back to the root directory	cd ~
/	Path component separator	cd /root/i8k

5.1.3 mkdir: creates a subdirectory (Equivalent DOS Command: md)

Parameter	Description	Example
-p	No error if the subdirectory exists, and creates parent directories as needed	mkdir -p directory

5.1.4 rmdir: deletes the subdirectory which must be empty (Equivalent DOS Command: rd)

Parameter	Description	Example
-p	Removes the specified DIRECTORY, then attempts to remove each parent directory component with the same path name	rmdir -p directory

5.1.5 rm: deletes (removes) the file or directory (Equivalent DOS Command: delete)

Parameter	Description	Example
–i	Displays a warning message before deleting	rm –i test.exe
–r	Deletes the directory even if it isn't empty	rm –r test.exe
–f	No warning message displayed when deleting	rm -f test.exe

5.1.6 cp: copies one or more files (Equivalent DOS Command: copy)

Parameter	Description	Example
–R	Performs a recursive copy	cp -R test bak
–i	Displays a confirmation prompt before overwriting	cp –i test bak
-l	Links the files instead of copying them	cp –l test bak

5.1.7 mv: moves or renames a file or directory (Equivalent DOS Command: move)

Parameter	Description	Example
-f	Does not display a confirmation prompt before overwriting	cp –f sour des
–i	Displays a confirmation prompt before overwriting	cp –i /sour /des

5.1.8 pwd: displays the full path of the current working directory

5.1.9 who: displays a list of the users current logged on

5.1.10 chmod: changes the access permissions for a file

Syntax→ 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: the **owner** can read and write and execute files

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

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

5.1.11 uname: displays the Linux version information

5.1.12 ps: displays a list of the currently active procedures

5.1.13 ftp: transfers a file using the file transfer protocol (FTP)

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

！ ：temporarily exits the FTP

exit：back to the ftp

bin：transfers files in “binary” mode

get：downloads a file from the LinPAC to the Host (For example: get /mnt/hda/test.exe c:/test.exe)

put：uploads a file from the host to the LinPAC (For example: put c:/test.exe /mnt/hda/test.exe)

bye：exits FTP

5.1.14 telnet: extablishes a connection to another PC via Telnet terminal

Syntax: telnet IPAddress

For example: telnet 192.168.0.200 (will allow remote control of the LP-8x4x)

5.1.15 date: prints or sets the system date and time

5.1.16 hwclock: queries and sets the hardware clock (RTC)

Parameters: (1) -r: reads the hardware clock and prints the time on a standard output.
(2) -w: sets the hardware clock to the current system time.

5.1.17 netstat: displays the current state of the network

Parameters: [-a]: list all states (For example: netstat -a)

5.1.18 ifconfig: displays the ip and network mask information (Equivalent DOS Command: ipconfig)

5.1.19 ping: used to test whether the host in a network is reachable

Syntax: ping IPAddress

For example：ping 192.168.0.1

5.1.20 clear: clears the screen

5.1.21 passwd: used to change the password

5.1.22 reboot: reboots the LinPAC (or use 'shutdown –r now')

5.2 General GCC Instructions

The GNU Compiler Collection (GCC) is a compiler system developed by the collaborative GNU Project that can be used to compile source code written using either ANSIC or Tranditional C into executable files. Executable files compiled using GCC can be executed within different operating system and different hardware systems. The GCC is distributed by the Free Software Foundation and is free of charge. Consequently, it has become very popular with users of Unix-based systems.

Fig. 5-1 below provides an illustration of the compilation procedure within Linux.

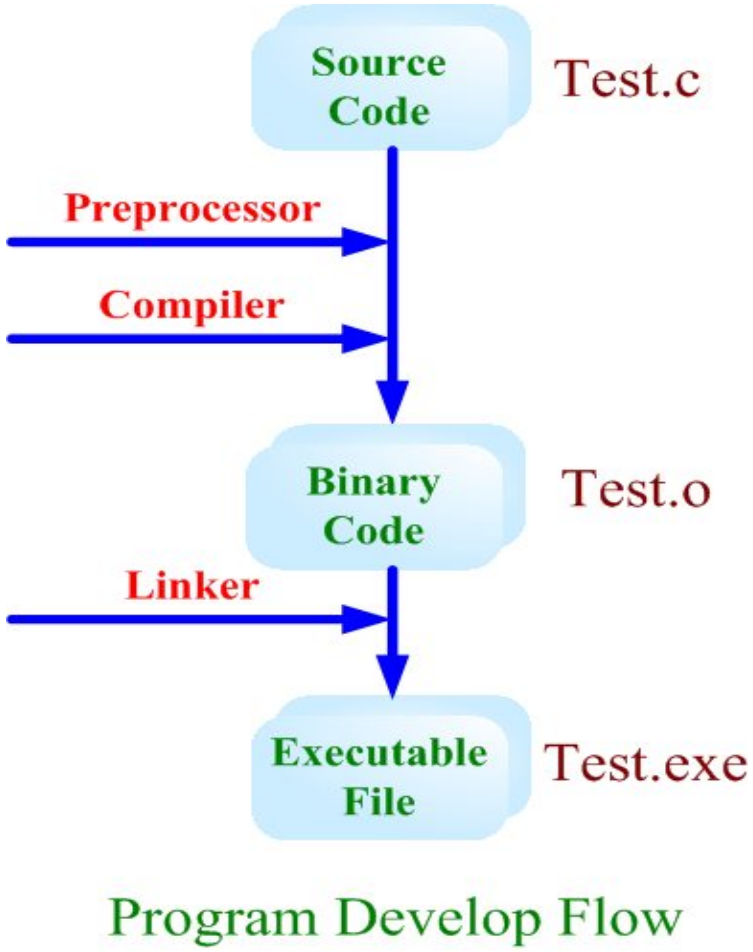


Fig. 5-1

The following is an overview of some of the common GCC instructions that will provide a guide for compiling *.c files to *.exe applications togegher with an explanation of the parameters used by the GCC during the compilation process.

5.2.1 Compile without linking to the LP-8x4x library

(1) Purpose: *. c to *. exe

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

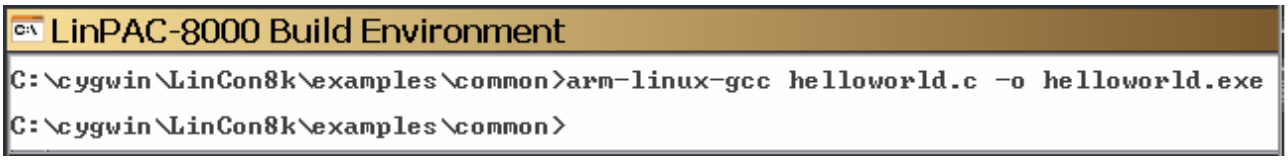
Parameter:

-o target: assigns the name of the output file

source.c: the C source code file

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: assigns the name of the output file

source.c: the C source code file

object.o: the 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 by linking to the LP-8x4x library (libi8k.a)

(1) Purpose: *. c to *. o

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

Parameters:

-lincludeDir: the path to the include files

-lm: includes the math library (libm.a)

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

-o target: assigns the name of the output file

source.c: the C source code file

library: the path of library

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

Output File: test.o

(2) Purpose: *. o to *. exe

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

Parameter:

-lincludeDir: the path of include files

-lm: includes the math library (libm.a)

-o target: assigns the name of the output file

source.o: the object file

library: the path to the library

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

Output File: test.exe

(3) Purpose: *. c to *. exe

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

Parameter:

-lincludeDir: the path of include files

-lm: include math library (libm.a)

-o target: assign the name of output file

source.c: source code of C

library: the path of library

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

Output File: test.exe

5.3 A Simple Example – Helloworld.c

This section describes how to 1) compile the helloworld.c file to the helloworld.exe executable file, 2) transfer the executable file (helloworld.exe) to the LP-8x4x using FTP, and 3) execute this file via the Telnet Server on the LP-8x4x.

This process can be accomplished using a single PC without requiring an additional monitor for the LP-8x4x. No ICP DAS modules are used in this example. However to use ICP DAS modules to control the system, refer to the demo described in the chapter 7.

The process can be divided into three steps, which are described below:

STEP 1: Compile helloworld.c to helloworld.exe

(1) Open the LP-8x4x SDK (refer to step 8 in section 2.1) and then type

"**cd examples/common**" to change the path to **C:/cygwin/LinCon8k/examples/common**.

Type "**dir/w**" and to display the contents of the directory and confirm that the helloworld.c file is present. Refer to Fig. 5-2 for more details.

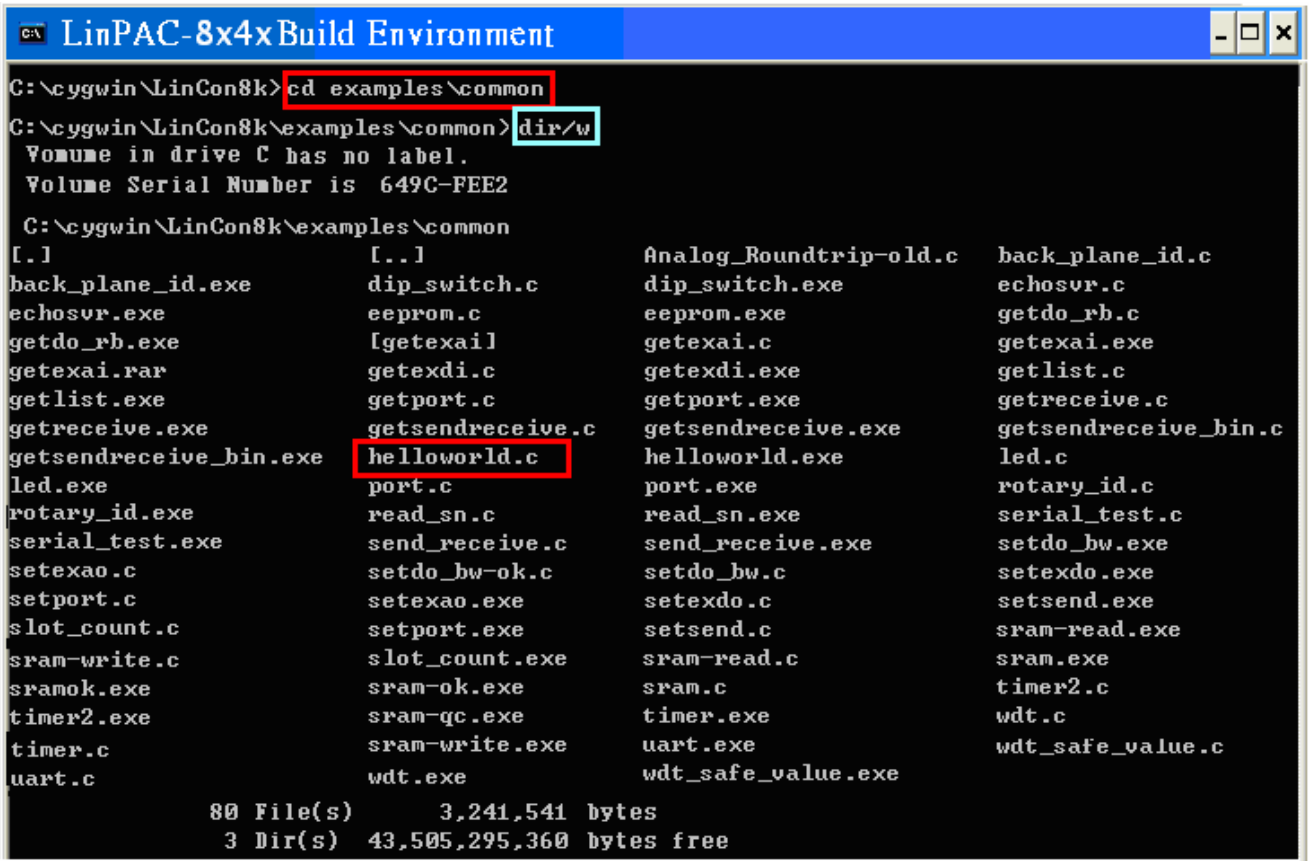


Fig. 5-2

(2) Type the command "**arm-linux-gcc –o helloworld.exe helloworld.c**" to compile helloworld.c into helloworld.exe, then type "**dir/w**" to display the contents of the directory and confirm that the helloworld.exe file has been created. (Refer to Fig. 5-3)

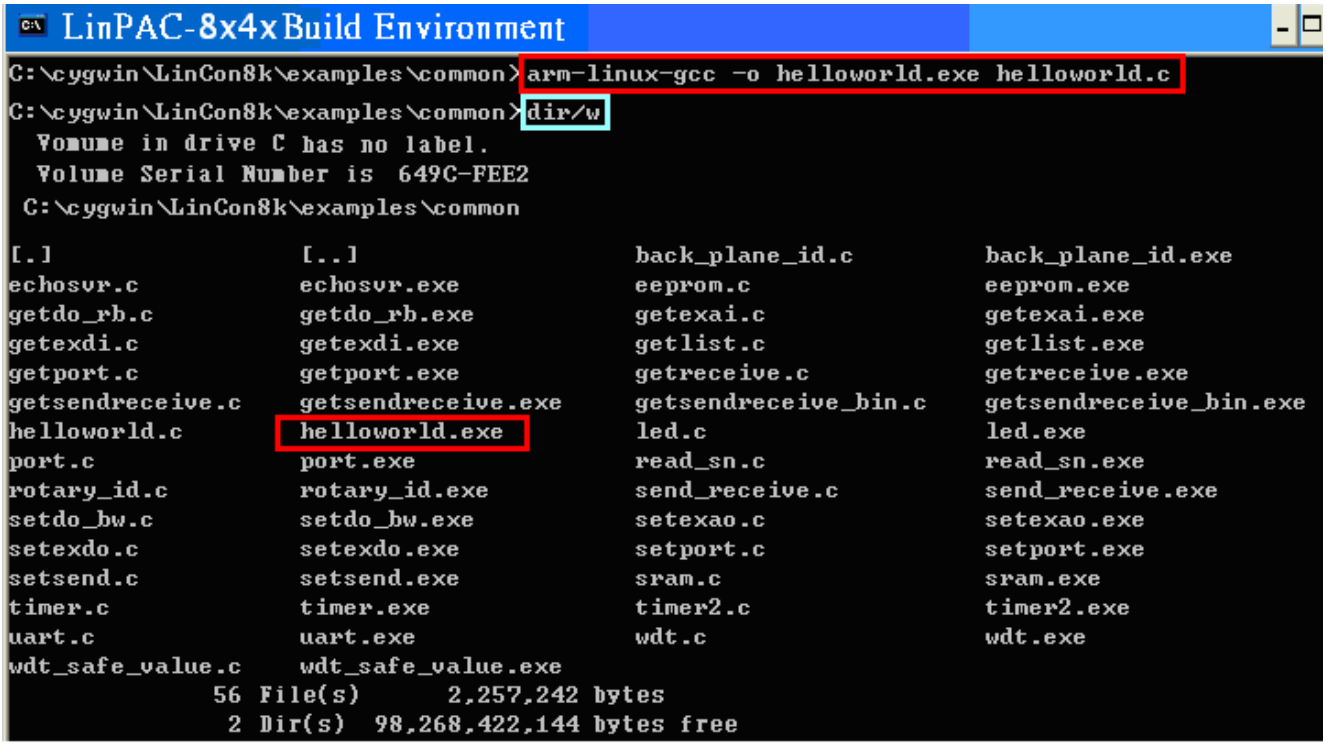


Fig. 5-3

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

Two methods can be used to transfer files to the LP-8x4x:

＜Method one＞ Using the "DOS Command Prompt"

(1) Open a "DOS Command Prompt" and type the ftp IP Address of the LP-8x4x, for example: **ftp 192.168.0.200** to establish a connection to the FTP Server on the LP-8x4x. When prompted, type the **User_Name** and **Password ("root" is the default value for both)** to establish a connection to the LP-8x4x.

(2) Before transferring the files to the LP-8x4x, type the "**bin**" command to ensure that the file is transferred to the LP-8x4x in **binary mode**. (refer to Fig. 5-4)

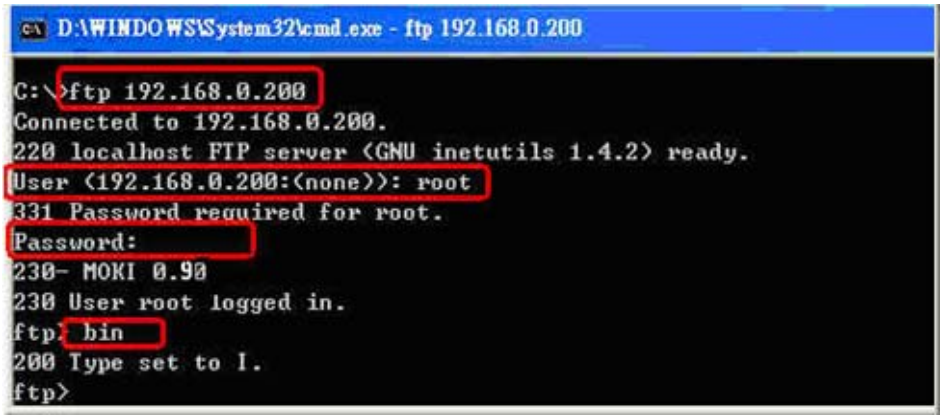


Fig.5-4

(3) Type the command “**put C:/cygwin/LinCon8k/examples/common/helloworld.exe helloworld.exe**” to transfer the helloworld.exe file to the LP-8x4x. Once the message “**Transfer complete**” is displayed, then transfer process has been completed. To disconnect from the LP-8x4x, type the “**bye**” command to return to the PC console. (refer to Fig. 5-5).

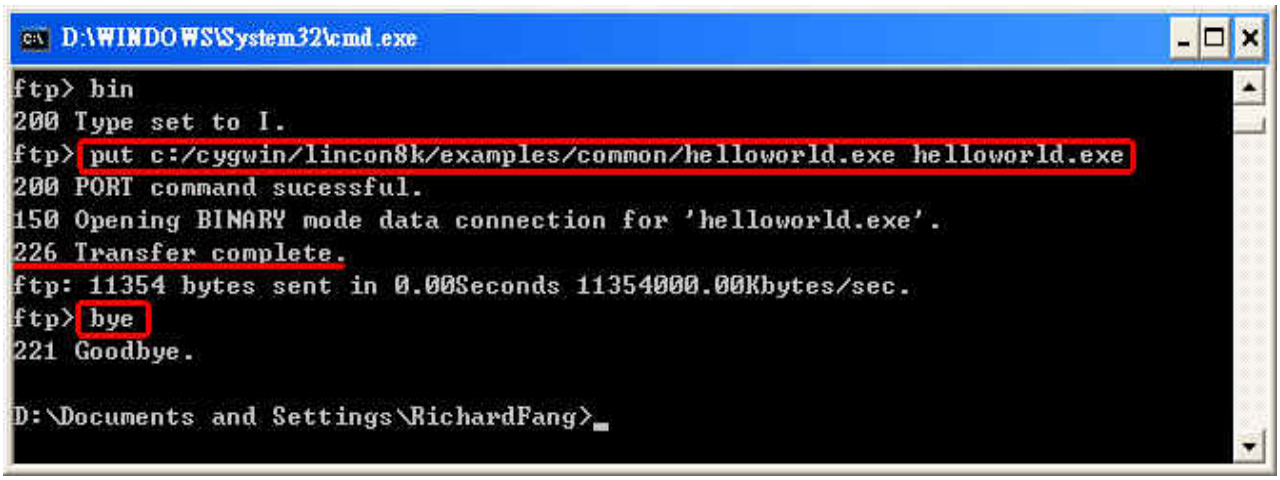


Fig.5-5

＜Method two＞ Using an FTP Client:

(1) Open the FTP Software and add an FTP Host to the LP-8x4x. The default value for both the **User_Name** and **Password** is **“root”**. Then click the **“Connect”** button to establish a connection to the ftp server on the LP-8x4x. (refer to Fig. 5-6).

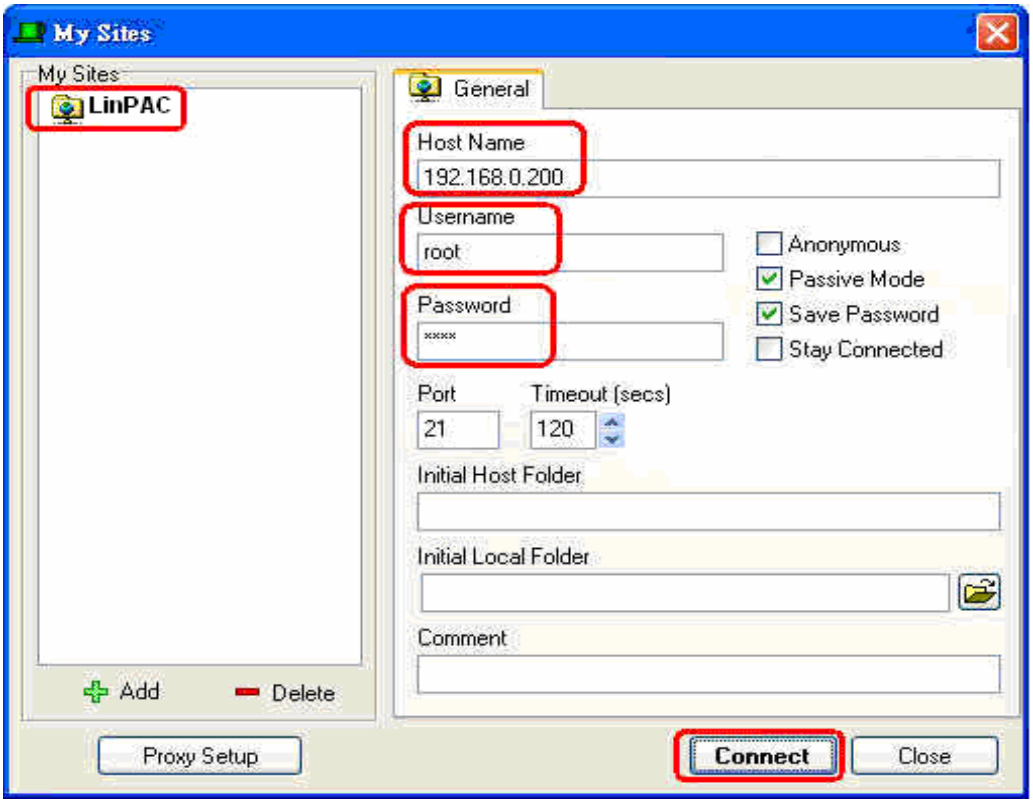


Fig.5-6

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

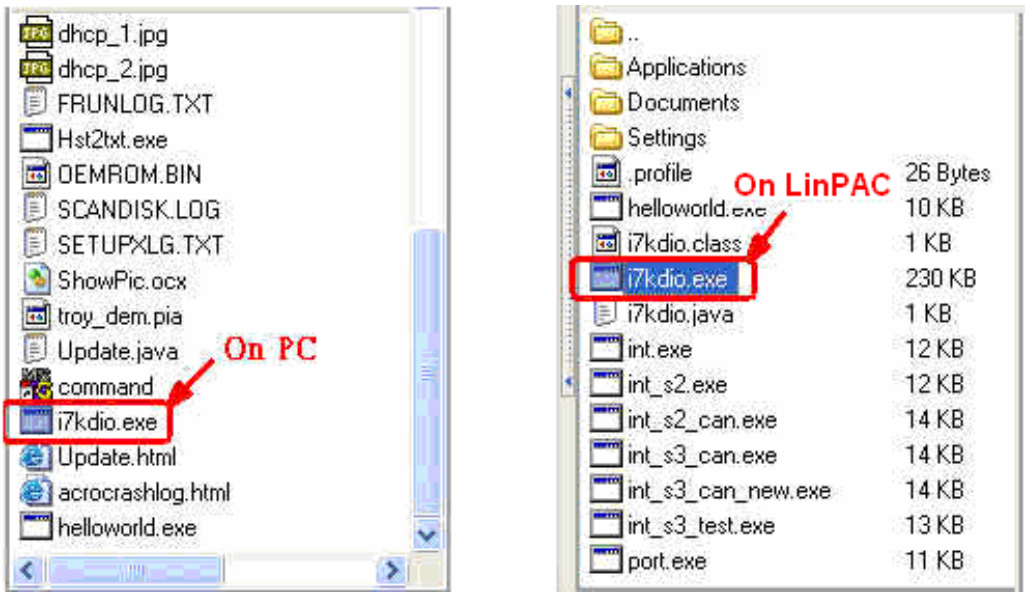


Fig. 5-7

(3) Click the helloworld.exe file in the LP-8x4x to select it and then right click the file icon and click the **“Permissions”** option. In the Properties dialog box, type 777 into the Numeric textbox, and then click the OK button. Refer to Fig. 5-8 and Fig. 5-9 for more details.

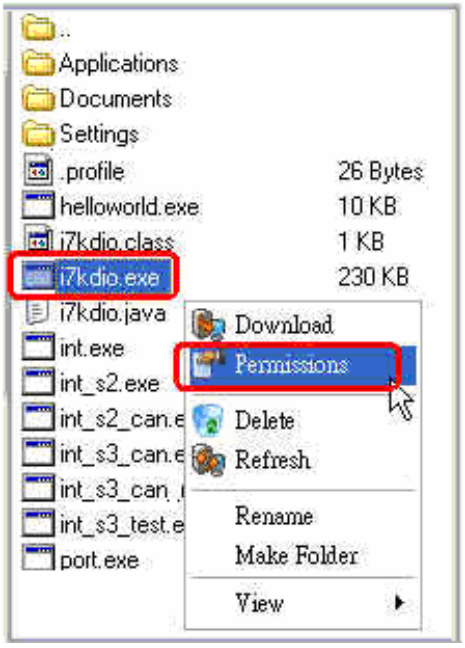


Fig.5-8

Fig.5-9

STEP 3: Use telnet to access the LP-8x4x and execute the program

(1) Open a "DOS Command Prompt" and type the IPAddress of the LP-8x4x, for example: **telnet 192.168.0.200**, to establish a connection to the Ttelnet Server on the LP-8x4x. When prompted, type the **User_Name** and **Password ("root" is the default value for both)** to establish a connection to the LP-8x4x. If the **"#"** prompt character is displayed, it signifies that a connection to the telnet server on the LP-8x4x has been successfully established. (refer to Fig. 5-10)

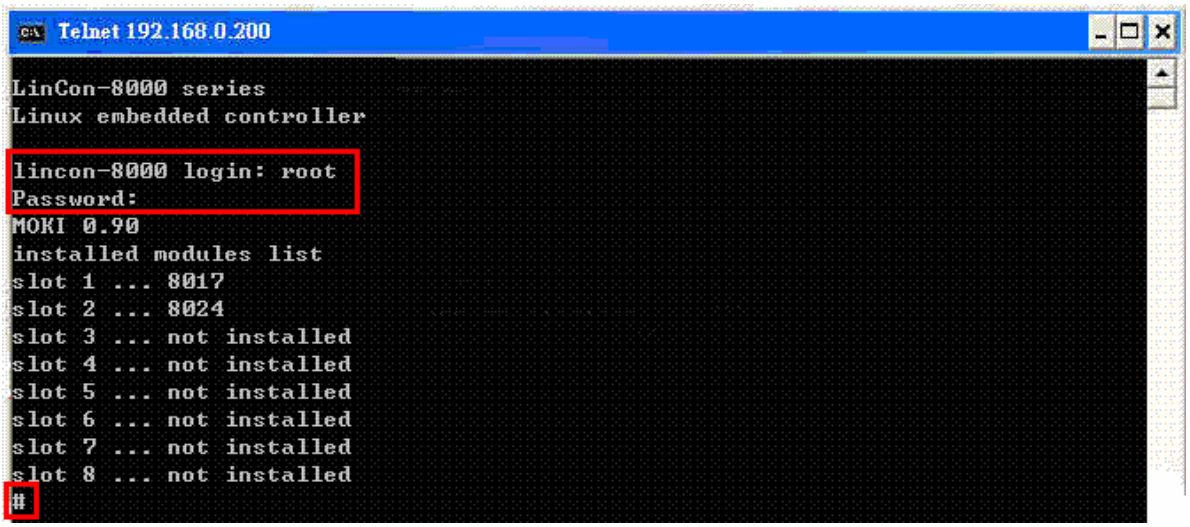


Fig.5-10

(2) Type the "**ls -l**" command to list all the files in the /root directory and verify that the helloworld.exe file is present. Type the "**chmod 777 helloworld.exe**" command to change the permissions for the helloworld.exe file. Type the "**ls -l**" command again to list all the files in the /root directory and verify the permissions assigned to the "helloworld.exe" file. This means that the file is executable. Execute the "**./helloworld.exe**" file by typing and the message "Welcome to LP-8x4x" will be displayed.

The **compile**, **transfer** and execution processes are now complete. (refer to Fig. 5-11)

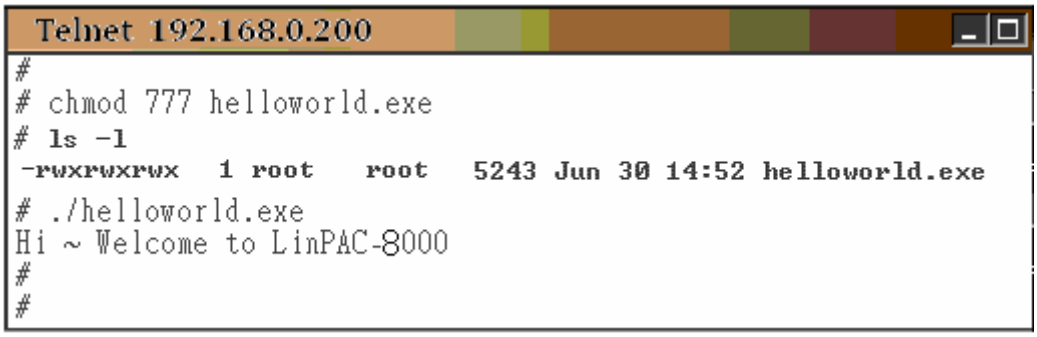


Fig.5-11

5.4 i-Talk Utility

The **i-Talk utility** provides **fifteen instructions** that make it convenient for users to access the modules and the hardware in the LP-8x4x and can be found in the path – **/usr/local/bin**. An overview of the i-Talk utility functions is given below: (Refer to Fig. 5-12)

Instruction	Description
getlist	Lists the names of all modules inserted in the LinPAC-8000
setdo	Sets the Digital Output value for I-8K modules
setao	Sets the Analog Output value for I-8K modules
getdi	Reads the Digital Input value for I-8K modules
getai	Reads the Analog Input value for I-8K modules
setexdo	Sets the Digital Output value for I-7K/87K modules
setexao	Sets the Analog Output value for I-7K/87K modules
getexdi	Reads the Digital Input value for I-7K/87K modules
getexai	Reads the Analog Input value for I-7K/87K modules
setport	Sets the Port offset value for the module
getport	Reads the Port offset value for the module
getsend	Sends a string from the LinPAC COM port
getreceive	Receives a string from the LinPAC COM port
getsendreceive	Sends/Receives a string from the LinPAC COM port
read_sn	Reads the hardware serial number for the LinPAC-8000

Fig. 5-12

The Fig. 5-13 provides details of the demo programs that illustrate how to use the I-talk utility. The **I-8024W** (AO Module), **I-8017HW** (AI Module) and the **I-8055W** (DIO Module) are used in the examples shown, and they are inserted in slots 1, 2 and 3 of the LinPAC respectively. Typing the name of the instruction will display usage details for the instruction.

Instruction	Example
getlist	*getlist* Lists the names of all modules inserted in the LinPAC-8000
setdo	*setdo {slot} {data}* ***setdo 3 3*** Sets channels 1 and 2 on the I-8055W module to ON
setao	*setao {slot} {channel} {data}* ***setao 1 0 2.2*** Sets the Analog Output value for channel 0 on the I-8024W module to 2.2 V.
getdi	*getdi {slot} {type}* ***getdi 3 8*** Reads the 8-bit Digital Input value from the I-8055W module
getai	*getai {slot} {channel} {gain} {mode}* ***getai 2 0 0 0*** Reads the Analog Input value from the I-8017HW module
setexdo	*setexdo {slot} {1} {data}* **setexdo 2 1 3** Sets the digital output value to the I-87K module at slot 2 *setexdo {slot} {comport} {data} {baudrate} {address}* **setexdo 0 3 55 9600 2** Sets the dec digital output value to the I-7K module at COM3 port
setexao	*setexao {slot} {1} {data} {channel}* **setexao 3 1 6.7 5** Sets channel 5 analog value 6.7 to the I-87K module at slot 3 *setexao {slot} {comport} {data} {channel} {baudrate} {address}* **setexao 0 3 6.7 5 9600 1** Sets channel 5 analog value 6.7 to the I-7K module at COM3 port
getexdi	*getexdi {slot} {1}* **getexdi 2 1** Get the dec digital input value from the I-87Kmodule at slot 2 *getexdi {slot} {comport} {baudrate} {address}* **getexdi 0 3 9600 2** Get the dec digital input value from the I-7K module at COM3 port
getexai	*getexai {slot} {1} {channel}* **getexai 2 1 5** Get channel 5 analog value from the I-87K module at slot 2 *getexai {slot} {comport} { channel } {baudrate} {address}* **getexai 0 3 5 115200 2** Get channel 5 analog value from the I-7K module at COM3 port
read_sn	*read_sn* Reads the hardware serial number for the LinPAC-8000 *Serial number = 9efebbebbe*

Fig. 5-13

6. LIBI8K.A

This section provides examples that focus on the description of and application of the functions found in the Libi8k.a library. The functions contained in the Libi8k.a library can be classified into three groups, as illustrated in Fig. 6-1.

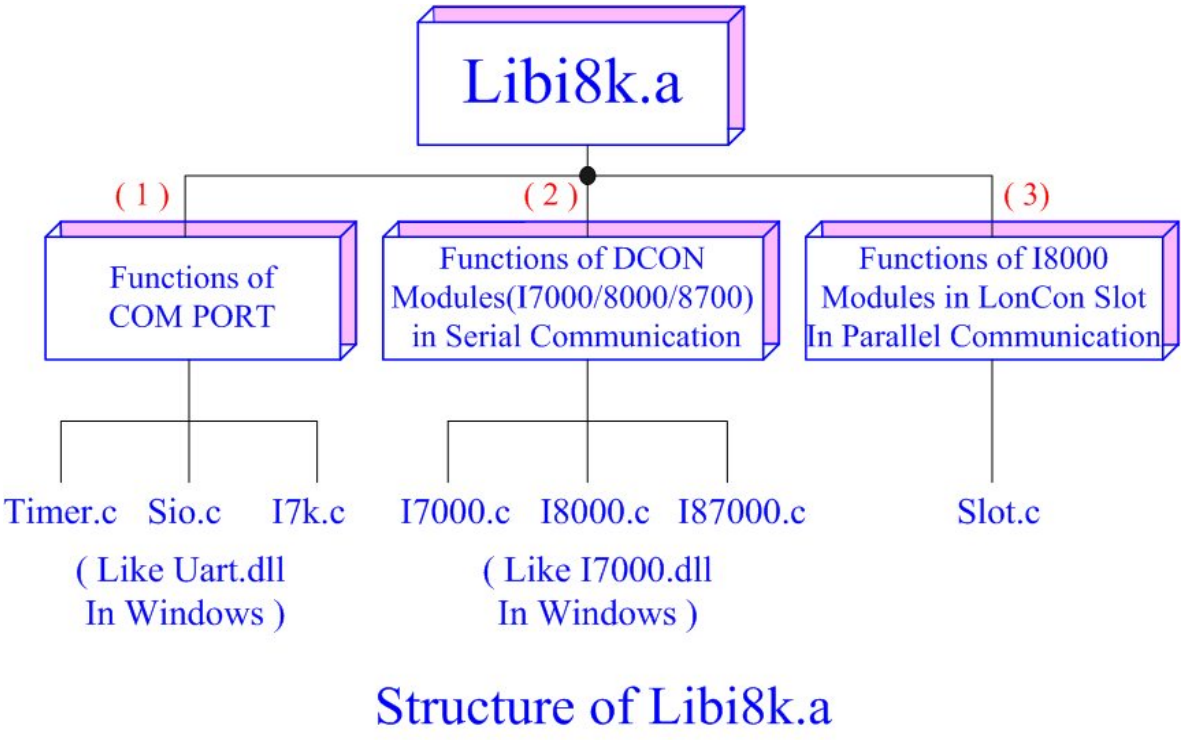


Fig. 6-1

The functions node (1) and (2) in the diagram for the Libi8k.a library are the same as those of the DCON.DLL Driver (including Uart.dll and I7000.dll) as used in the DCON modules (I-7000/ I-8000/ I-87000 for serial communication). For more details, refer to the DCON.DLL Driver manual which includes adescription of how to use the functions on DCON modules. The DCON.DLL Driver has now been included in the Libi8k.a library.

Functions (3) in the diagram contains the most important functions, as they are especially designed for I-8000 modules inserted in the LP-8x4x slots. They are different from functions (1) and (2) because the communication method for I-8000 modules inserted in the LP-8x4x is based on parallel mode rather than serial mode. Accordingly, the I8000.c library has been completely rewritten especially for I-8000 modules inserted in LP-8x4x slots and has been rename as slot.c. The following is an introduction to the funcitions for slot.c, which can be arranged into eight main categories:

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

Note that when using a development tool to create develop applications, the **msw.h** file must be included in the header of the source program, and the **Libi8k.a** library file must also be linked. To control ICP DAS remote I/O modules such as the I-7K, I-8K and I-87K series modules **via the COM2 or COM3 or COM4 ports on the LP-8x4x**, the functions are the same as those included in the DCON DLL. To control **I-8K series modules** that are inserted in the slots of the LP-8x4x, the functions are different and they are described in more detail below:

6.1 System Information Functions

■ Open_Slot

Description:

This function is used to open and initialize a specific slot on the LP-8x4x, and will be used by I-8K or I-87K modules inserted in the LP-8x4x. For example, to send or receive data from a specific slot, this function must be called first before any other functions can be used.

Syntax:

[C]

```
int Open_Slot(int slot)
```

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted.

Return Values:

0: The slot was successfully initialized.

Other: The initialization failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Example:

```
Int slot=1;
Open_Slot(slot);
// The first slot in the LP-8x4x will be open and initiated.
```

Remark:

■ Close_Slot

Description:

After using the of Open_Slot() function to open and initialize a specific slot on the LP-8x4x, the Close_Slot() function must also be used to close the slot. This function will be used by I-8K or I-87K series modules inserted in the LP-8x4x. For example, the Close_Slot() function should be called after sending or receiving data from the specified slot.

Syntax:

[C]

```
void Close_Slot(int slot)
```

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted.

Return Value:

None

Examples:

```
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 initialize **all slots** on the LP-8x4x. For example, to send or receive data from multiple slots, this function can be used to simplify the program, and other functions can be used.

Syntax:

[C]

```
int Open_SlotAll(void)
```

Parameter:

None

Return Value:

0: The slot was successfully initialized.

Other: The initialization failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
Open_SlotAll();
// All slots in the LP-8x4x will be open and initiated.
```

Remark:

■ Close_SlotAll

Description:

If you the Open_SlotAll() function was used to open and initialize all the slots on the LP-8x4x, the Close_SlotAll() function can be used to quickly close them simultaneously. For example, the Close_SlotAll() function can be called after sending or receiving data from multiple slots to close all the slots at the same time.

Syntax:

[C]

```
void Close_SlotAll(void)
```

Parameter:

None

Return Value:

None

Examples:

```
Close_Slot();
// All slots in the LP-8x4x will be closed.
```

Remark:

■ ChangeToSlot

Description:

This function is used to assign serial control to the specified slots for to allow control of the I-87K series. The serial bus on the backplane of the LP-8x4x is used for mapping through to COM1. For example, to send or receive data from a specified slot, this function should be called first, and then other series functions can be used.

Syntax:

[C]

```
void ChangeToSlot(char slot)
```

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted.

Return Value:

None

Examples:

```
char slot=2;
ChangeToSlot (slot);
// The first slot on the LP-8x4x is specified as the COM1 port.
```

Remark:

■ Open_Com

Description:

This function is used to open and configure the COM port, and must be **called at least once before** sending/receiving a command via the COM port. For example, to send or receive data from a specified COM port, this function should be called first, and then other series functions can be used.

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 Values:

0: The com port was successfully initialized.

Other: The initialization failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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

Examples:

```
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 inserted 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

Examples:

```
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

Examples:

```
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

Examples:

```
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.

Examples:

```
#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 inserted 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

Examples:

```
#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 inserted in 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] Specifies the slot where the I/O module is inserted.

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

Examples:

```
int slot=1;
int moduleType;
Open_Slot(slot);
printf("GetModuleType= 0x%X \n", GetModuleType(slot));
Close_Slot(slot);
// The I-8057W card is inserted 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] Specifies the slot where the I/O module is inserted.

Return Value:

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

Examples:

```
int slot=1;
int moduleID;
Open_Slot(slot);
moduleID=GetNameOfModule(slot);
Close_Slot(slot);
// The I-8017W module is inserted 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

Examples:

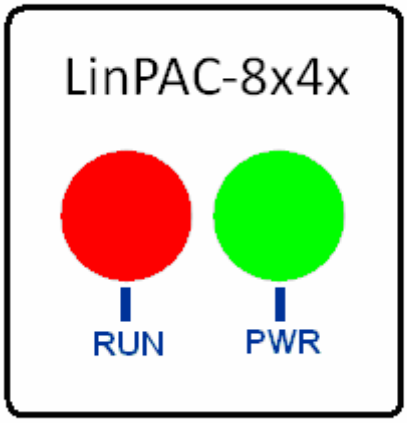
```
int slot ;
unsigned char serial_num[8];
Open_Slot(0);
Read_SN(serial_num);
printf("SN=%x%x%x%x%x%x%x%x\n",serial_num[0],serial_num[1], serial_ num[2]
      ,serial_num[3],serial_num[4],serial_num[5],serial_num[6],serial_num[7]);
```

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

Examples:

```
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.

Examples:

```
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.

Examples:

```
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.

Examples:

```
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] Type definition: LP-8x4x is type 1.

id: [Output] Rotary switch ID number.

Return Value:

0: The slot was successfully initialized.

Other: The initialization failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
int id, slot, type, wRetVal;
type= 1;     // for LP-8x4x
wRetVal= Open_Slot(slot);
if (wRetVal > 0) {
        printf("open Slot%d failed!\n",slot);
        return (-1);
}
GetRotaryID(type, &id);
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:

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

■ 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.

Examples:

```
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 milliseconds)
void DisableWDT(void)
```

Parameter:

milliseconds: LinPAC will reset in the assigned time if users don't reset WDT. The unit is mini-second.

Return Value:

None

Examples:

```
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)

Examples:

```
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 to bit 3

Return Value:

None

Examples:

ClearWDTSWEven(0xF); //Used to clear bit 0 to 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 64 blocks (block 0 to 63), and each block has 256 bytes in length from offset 0 to 255, with total size of 16,384 bytes (16K) capacity.

Syntax:

[C]

```
void Enable_EEP(void)
```

Parameter:

None

Return Value:

None

Examples:

```
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

Examples:

```
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 (0~0x3fff) EEPROM in the main control unit in the LP-8x4x system. This EEPROM is divided into 64 blocks (block 0 to 63), and each block has 256 bytes in length from offset 0 to 255. 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.

Examples:

```
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 (0~0x3fff) EEPROM in the main control unit of the LP-8x4x system. This EEPROM is divided into 64 blocks (block 0 to 63), and each block has 256 bytes in length from the offset of 0 to 255. 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

Examples:

```
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 to 7 bits of output data are mapped into the 0 to 7 channels of digital module output respectively.

Syntax:

[C]

```
void DO_8(int slot, unsigned char data)
```

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted.

data : [Input] output data.

Return Value:

None

Examples:

```
int slot=1;
unsigned char data=3;
DO_8(slot, data);
// The I-8064W is inserted 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 to 15 bits of output data are mapped into the 0 to 15 channels of digital output modules respectively.

Syntax:

[C]

```
void DO_16(int slot, unsigned int data)
```

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted..

data : [Input] output data.

Return Value:

None

Examples:

```
int slot=1;
unsigned int data=3;
DO_16(slot, data);
 // The I-8057W is inserted 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 to 31 bits of output data are mapped into the 0 to 31 channels of digital output modules respectively.

Syntax:

[C]

```
void DO_32(int slot, unsigned int data)
```

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted.

data : [Input] output data.

Return Value:

None

Examples:

```
int slot=1;
unsigned int data=3;
DO_32(slot, data);
// The I-8041W is inserted 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 to 7 bits of input data correspond to the 0 to 7 channels of digital input modules respectively.

Syntax:

[C]

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

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted.

Return Value:

Input data

Examples:

```
int slot=1;
unsigned char data;
data=DI_8(slot);
// The I-8058W is inserted 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 to 15 bits of input data correspond to the 0 to 15 channels of digital module's input respectively.

Syntax:

[C]

```
unsigned int DI_16(int slot)
```

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted.

Return Value:

Input data

Examples:

```
int slot=1;
unsigned int data;
data=DI_16(slot);
// The I-8053W is inserted 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 to 31 bits of input data correspond to the 0 to 31 channels of digital input module respectively.

Syntax:

[C]

```
unsigned long DI_32(int slot)
```

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted.

Return Value:

Input data

Examples:

```
int slot=1;
unsigned long data;
data=DI_32(slot);
// The I-8040W is inserted 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 to 7 bits of output data are mapped onto the 0 to 7 output channels for their specific DIO modules respectively.

Syntax:

[C]

```
void DIO_DO_8(int slot, unsigned char data)
```

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted.

data : [Input] output data.

Return Value:

None

Examples:

```
int slot=1;
unsigned char data=3;
DIO_DO_8(slot, data);
// The I-8054W is inserted 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 to 15 bits of output data are mapped onto the 0 to 15 output channels for their specific DIO modules respectively.

Syntax:

[C]

```
void DIO_DO_16(int slot, unsigned int data)
```

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted.

data : [Input] output data.

Return Value:

None

Examples:

```
int slot=1;
unsigned int data=3;
DIO_DO_16(slot, data);
// The I-8042W is inserted 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 to 7 bits of intput data, are mapped onto the 0 to 7 iutput channels for their specific DIO modules respectively.

Syntax:

[C]

```
Unsigned char DIO_DI_8(int slot)
```

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted.

Return Value:

Input data

Examples:

```
int slot=1;
unsigned char data;
data=DIO_DI_8(slot);
// The I-8054W is inserted 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 to 15 bits of iutput data are mapped onto the 0 to 15 iutput channels for their specific DIO modules respectively.

Syntax:

[C]

```
Unsigned char DIO_DI_16(int slot)
```

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted.

Return Value:

Input data

Examples:

```
int slot=1;
unsigned char data;
data=DIO_DI_16(slot);
// The I-8042W is inserted 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] Specifies the slot where the I/O module is inserted.

Return Value:

all DO channel status

Examples:

```
int slot=1;
Open_Slot(slot);
printf("%u",DO_32_RB(slot));
Close_Slot(slot);
// The I-8041W module is inserted 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] Specifies the slot where the I/O module is inserted.

bit : [Input] channel of module.

data : [Input] channel status [on(1) / off(0)].

Return Value:

None

Examples:

```
int slot=1, bit=0, data=1;
Open_Slot(slot);
DO_32_BW(slot, bit, data);
Close_Slot(slot);
// The I-8041W module is inserted 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] Specifies the slot where the I/O module is inserted.

bit : [Input] channel of module.

Return Value:

None

Examples:

```
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 inserted 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] Specifies the slot where the I/O module is inserted.

config : [Input] channel status.[DO : 1 / DI : 0]

Return Value:

None

Examples:

```
int slot=1;
unsigned short config=0xffff;
UDIO_WriteConfig_16(slot, config);
// The I-8064W is inserted in slot 1 of LP-8x4x.
// WriteConfig: 0xffff (ch 0 to 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] Specifies the slot where the I/O module is inserted.

Return Value:

None

Examples:

```
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 inserted in slot 1 of LP-8x4x.
// WriteConfig: 0x0000 (ch 0 to 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 to 15 bits data to a universal DIO module according to the channel configuration. The 0 to 15 bits of output data are mapped onto the 0 to 15 output channels for their specific universal DIO modules respectively.

Syntax:

[C]

```
void UDIO_DO16(int slot, unsigned short config)
```

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted.

config : [Input] output data.

Return Value:

None

Examples:

```
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 to Ch7) of I-8050 in Slot %d = 0x%x\n\r",slot, data);
// The I-8050W is inserted in slot 1 of LP-8x4x
// WriteConfig: 0x00ff (ch0 to ch7 is DO mode and ch8 to ch15 is DI mode)
// Input DO value: 255
// DO(Ch0 to 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 to 15 bits data to a universal DIO module according to the channel configuration. The 0 to 15 bits of input data are mapped onto the 0 to 15 input channels for their specific universal DIO modules respectively.

Syntax:

[C]

```
unsigned short UDIO_DI16(int slot)
```

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted.

Return Value:

None

Examples:

```
int slot=1;
unsigned int data;
unsigned short config =0xff00;
UDIO_WriteConfig_16(slot, config);
data=UDIO_DI16(slot);
printf("DI(Ch0 to Ch7) of I-8055 in Slot %d = 0x%x\n\r",slot, data);
scanf("%d:",&data);
UDIO_DO16(slot, data);
printf("DO(Ch8 to Ch15) of I-8050 in Slot %d = 0x%x\n\r",slot, data);
// The I-8050W is inserted in slot 1 of LP-8x4x.
// WriteConfig: 0xff00 (ch0 to ch7 is DI mode and ch8 to ch15 is DO mode)
// DI(Ch0 to Ch7) of I-8055 in Slot 1 = 0xfbff
// Input DO value: 255
// DO(Ch8 to 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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds

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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds

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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 to 7 bits of output data are mapped into the 0 to 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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds

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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 to 7 bits of output data are mapped into the 0 to 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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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-87000 series modules.
szReceive : [Output] Result string receiving from I-87000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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] Specifies the slot where the I/O module is inserted. (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 inserted 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] Specifies the slot where the I/O module is inserted. (Range: 1 to 8)

Return Value:

The version of library

Example:

```
int slot=1,ver;
ver=I8017_Init(slot);
// The I-8017HW is inserted 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] Specifies the slot where the I/O module is inserted. (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 inserted 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] Specifies the slot where the I/O module is inserted. (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] Specifies the slot where the I/O module is inserted. (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:

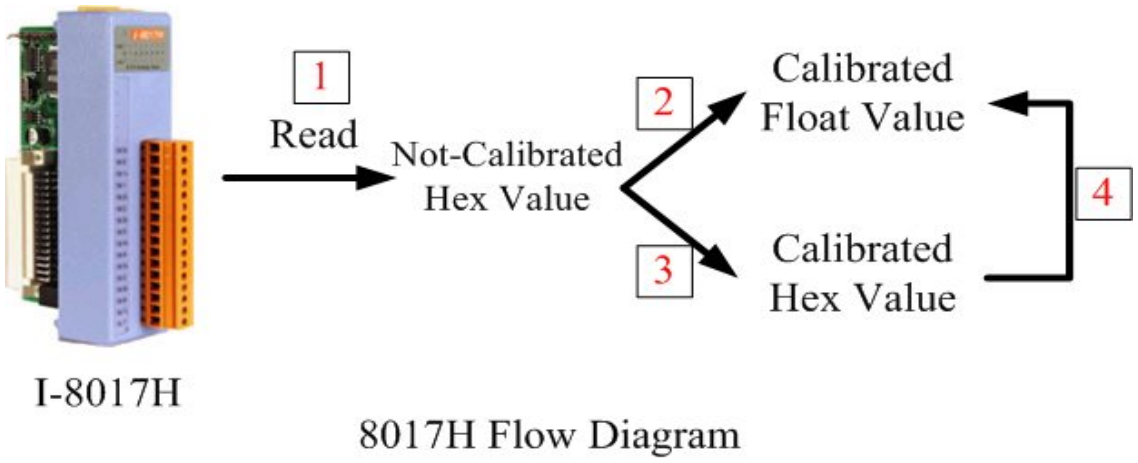
None

Example:

```
int slot=1,ch=0,gain=0;
I8017_SetChannelGainMode (slot, ch, gain,0);
// The I-8017HW card is inserted in slot 1 of LP-8x4x, and the range of the data // value from channel 0 for I-8017H will be -10 to +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] Specifies the slot where the I/O module is inserted. (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 inserted in 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] Specifies the slot where the I/O module is inserted. (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.

Examples:

```
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.

Examples:

```
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] Specifies the slot where the I/O module is inserted. (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

Examples:

```
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.

Examples:

```
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] Specifies the slot where the I/O module is inserted. (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.

Examples:

```
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.

Examples:

```
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.

Examples:

```
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] Specifies the slot where the I/O module is inserted. (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 inserted in slot 1 of LP-8x4x and the range of the
// data value from channel 0 in I-8017H is 0x0000 to 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] Specifies the slot where the I/O module is inserted. (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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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] Specifies the slot where the I/O module is inserted. (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 inserted in slot 1 of LP-8x4x and the range of the
// data value from channel 0 in I-8017H is –10V to +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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds

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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds

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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds

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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
float AI[12];
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] = 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:

6.6 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] Specifies the slot where the I/O module is inserted. (Range: 1 to 8)

Return Value:

None

Examples:

```
int slot=1;
I8024_Initial(slot);
// The I-8024W is inserted in 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] Specifies the slot where the I/O module is inserted. (Range: 1 to 8)

ch : [Input] Output channel (Range: 0 to 3)

data : [Input] Output data with engineering unit (Voltage Output: -10 to +10)

Return Value:

None

Examples:

```
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 W functions for current output.

Syntax:

[C]

```
void I8024_CurrentOut(int slot, int ch, float cdata)
```

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted. (Range: 1 to 8)

ch : [Input] Output channel (Range: 0 to 3)

cdata : [Input] Output data with engineering unit (Current Output: 0 to 20 mA)

Return Value:

None

Examples:

```
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 inserted in the slot in the LP-8x4x system.

Syntax:

[C]

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

Parameter:

slot : [Input] Specifies the slot where the I/O module is inserted. (Range: 1 to 8)

ch : [Input] Output channel (Range: 0 to 3)

hdata : [Input] Output data with hexadecimal
(data range: 0h to 3FFFh → Voltage Output: -10 to 10V)

Return Value:

None

Examples:

```
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] Specifies the slot where the I/O module is inserted. (Range: 1 to 8)

ch : [Input] Output channel (Range: 0 to 3)

hdata : [Input] Output data with hexadecimal

(data range: 0h to 3FFFh → Current Output: 0 to +20mA)

Return Value:

None

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds

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 to 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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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 milliseconds
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: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6.7: “Error Code Definitions” for details of other returned values.

Examples:

```
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] Specifies the slot where the I/O module is inserted. (Range: 1 to 8)

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

Return Value:

1: Success registration

0: Failure registration

Examples:

```
unsigned char slot=1;
unsigned int address=0x0;
I8090_REGISTRATION(slot, address);    // I-8090W is inserted 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] Specifies the slot where the I/O module is inserted. (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

Examples:

```
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] Specifies the slot where the I/O module is inserted. (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.

Examples:

```
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] Specifies the slot where the I/O module is inserted. (Range: 1 to 8)

axis : [Input] The selected axis. 1: X_axis; 2: Y_axis; 3: Z_axis

Return Value:

None

Examples:

```
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] Specifies the slot where the I/O module is inserted. (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.

Examples:

```
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] Specifies the slot where the I/O module is inserted. (Range: 1 to 8)

axis : [Input] The selected axis. 1: X_axis; 2: Y_axis; 3: Z_axis

Return Value:

None

Examples:

```
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] Specifies the slot where the I/O module is inserted. (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

Examples:

```
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 to 10ms) using the timer interrupt or manual method.

Syntax:

[C]

```
void I8090_ENCODER32_ISR(unsigned char cardNo)
```

Parameter:

cardNo : [Input] Specifies the slot where the I/O module is inserted. (Range: 1 to 8).

Return Value:

None

Examples:

```
unsigned char slot;
long data;
i8090_ENCODER32_ISR(slot);        // should be called in 2 to 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 to 19)

slot : [Input] Specifies the slot where the I/O module is inserted. (1 to 7)

Return Value:

1: card exist

0: card not exist

Examples:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
// The I-8091W card is inserted in 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 to 19, The selected card number.

Return Value:

None

Examples:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_RESET_SYSTEM(CARD1);
// The I-8091W card inserted in slot 1 of LP-8x4x
```

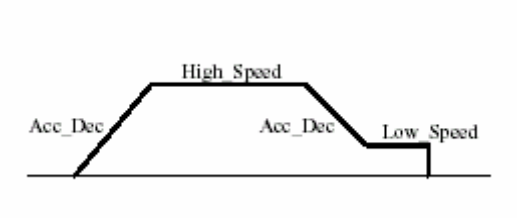
Remark:

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 to 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

Examples:

```
#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 to 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

Examples:

```
#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 to 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:

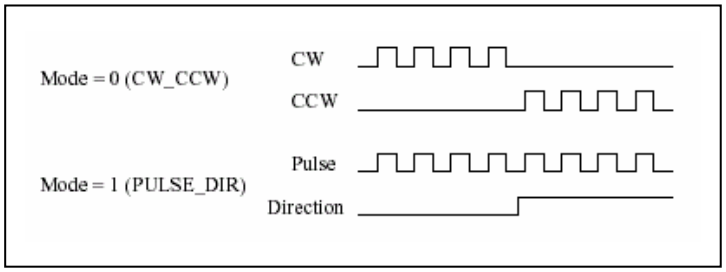
None

Examples:

```
#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 to 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

Examples:

```
#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 to 19)

sw : [Input] 0(NO) normally open (default), 1(YES) normally closed.

Return Value:

None

Examples:

```
#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 to 19)

Return Value:

None

Examples:

```
#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 to 19)

Return Value:

None

Examples:

```
#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 to 19)

Return Value:

None

Examples:

```
#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 to 19)

Return Value:

None

Examples:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_EMG_STOP(CARD1);
//The X-axis and Y-axis are stopped.
```

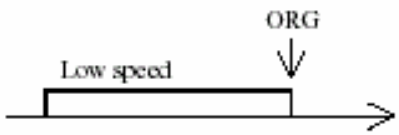
Remark:

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 to 19)

DIR : [Input] The moving direction. (0:CW, 1:CCW)

AXIS : [Input] The selected axis. (1: X_axis, 2: Y_axis)

Return Value:

None

Examples:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_LSP_ORG(CARD1, CCW, X_axis);
i8091_LSP_ORG(CARD1, CCW, Y_axis);
```

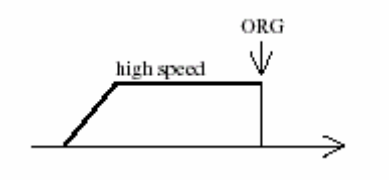
Remark:

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 to 19)

DIR : [Input] The moving direction. (0:CW, 1:CCW)

AXIS : [Input] The selected axis. (1: X_axis, 2: Y_axis)

Return Value:

None

Examples:

```
#define CARD1    1
int slot=1;
i8091_REGISTRATION(CARD1, slot);
i8091_HSP_ORG(CARD1, CCW, X_axis);
i8091_HSP_ORG(CARD1, CCW, Y_axis);
```

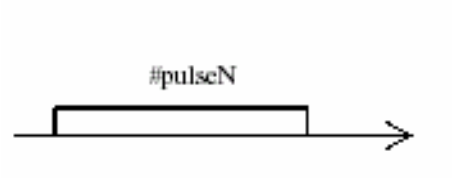
Remark:

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 to 19)

AXIS : [Input] The selected axis (1: X_axis, 2: Y_axis)

pulseN : [Input] The moving number of pulse.

Return Value:

None

Examples:

```
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
```

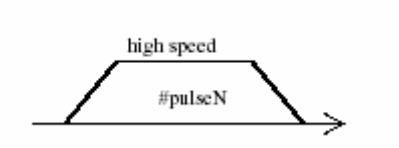
Remark:

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 to 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
```

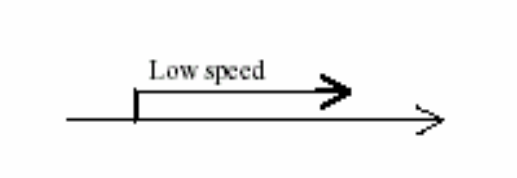
Remark:

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 to 19)

DIR : [Input] The moving direction. (0:CW, 1:CCW)

AXIS : [Input] The selected axis (1: X_axis, 2: Y_axis)

Return Values:

None

Example:

```
i8091_LSP_MOVE(CARD1, CW, X_axis);
i8091_LSP_MOVE(CARD1, CW, Y_axis);
```

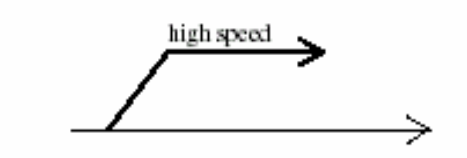
Remark:

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 to 19)

DIR : [Input] The moving direction. (0:CW, 1:CCW)

AXIS : [Input] The selected axis (1: X_axis, 2: Y_axis)

Return Value:

None

Examples:

```
i8091_HSP_MOVE(CARD1, CW, X_axis);
i8091_HSP_MOVE(CARD1, CW, Y_axis);
```

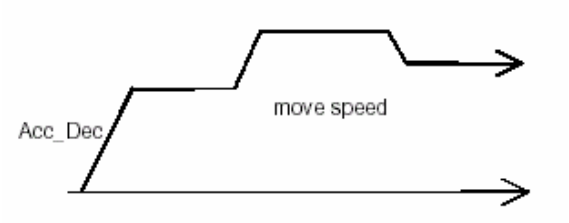
Remark:

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 to 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

Examples:

```
i8091_CSP_MOVE(CARD1, CW, X_axis, 10);    //Delay(10000);
i8091_CSP_MOVE(CARD1, CW, X_axis, 20);
```

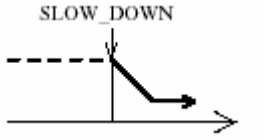
Remark:

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 to 19)

AXIS : [Input] The selected axis (1: X_axis, 2: Y_axis)

Return Value:

None

Examples:

```
i8091_HSP_MOVE(CARD1, CW, X_axis);
i8091_SLOW_DOWN(CARD1, X_axis);
i8091_STOP_X(CARD1);
```

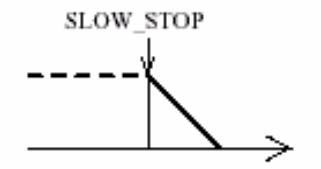
Remark:

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 to 19)

AXIS : [Input] The selected axis (1: X_axis, 2: Y_axis)

Return Value:

None

Examples:

```
i8091_HSP_MOVE(CARD1, CW, Y_axis);
i8091_SLOW_STOP(CARD1, Y_axis);
```

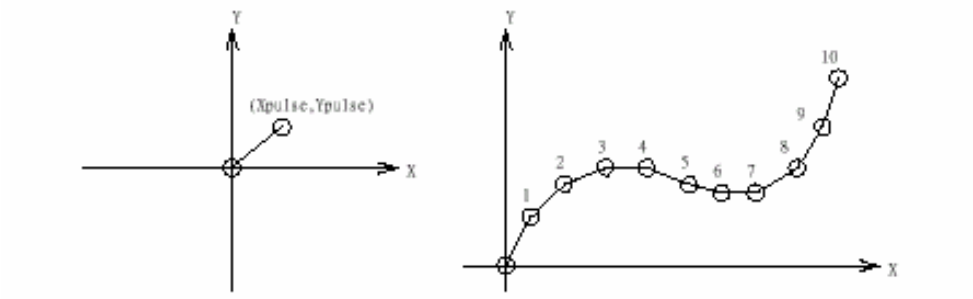
Remark:

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 to 19)

Xpulse : [Input] －2047<= # Xpulse <=2047

Ypulse : [Input] －2047<= # Ypulse <=2047

Return Value:

None

Examples:

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);

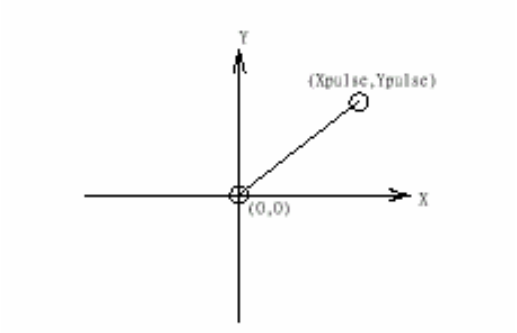
Remark:

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 to 19)

Xpulse : [Input] – 524287<= # Xpulse <=524287

Ypulse : [Input] – 524287<= # Ypulse <=524287

Return Value:

None

Examples:

```
i8091_INTP_LINE(CARD1, 2000, -3000);
i8091_INTP_LINE(CARD1, -500, 200);
```

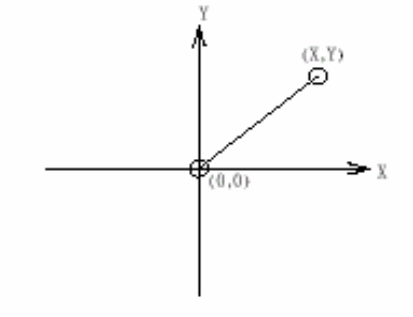
Remark:

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 to 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 to 2040
acc_mode :	[Input] 0: enables the acceleration and deceleration profiles 1: disables the acceleration and deceleration profiles

Return Value:

None

Examples:

```
i8091_INTP_LINE02(CARD1, 1000, 1000, 100, 0);
do { } while(i8091_INTP_STOP()!=READY) ; //call state machine
```

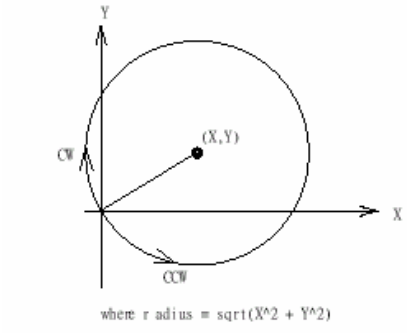
Remark:

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 to 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 to 2040
acc_mode :	[Input] 0: enable acceleration and deceleration profile 1: disable acceleration and deceleration profile

Return Value:

None

Examples:

```
i8091_INTP_CIRCLE02(CARD1, 2000, 2000, 100, 0);
do { } while(i8091_INTP_STOP()!=READY) ; //call state machine
```

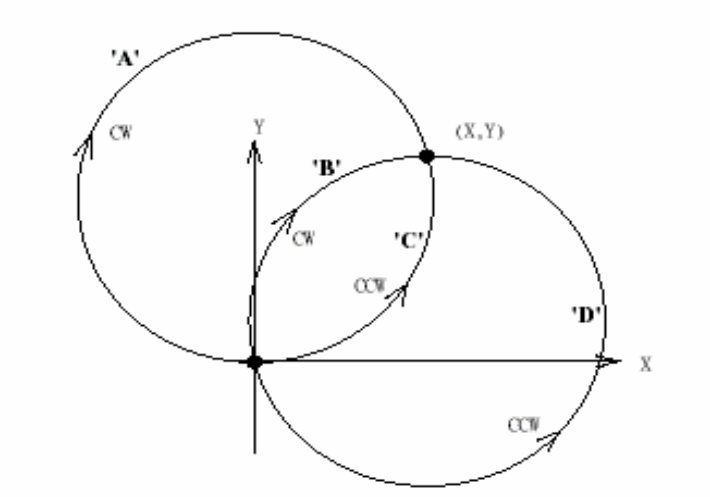
Remark:

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 to 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 to 2040

acc_mode : [Input] 0: enables the acceleration and deceleration profiles
1: disables the acceleration and deceleration profiles

Return Value:

None

Examples:

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

Examples:

```
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 to 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.

/EMG : emergency switch, low active.

/FFEF : active low, FIFO is empty

/FFFF : active low, FIFO is full

Examples:

```
unsigned char limit1;
limit1 = i8091_LIMIT_X(CARD1);
```

Remark:

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

I-8091W User's Manual:

http://www.icpdas.com/root/product/solutions/machine_automation/download%20data/i8091/pack9091.pdf

■ 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 to 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.

Xstop: 1:indicate X-axis is stop.

Ystop: 1:indicate Y-axis is stop.

Examples:

```
unsigned char limit2;
limit2 = i8091_LIMIT_Y(CARD1);
```

Remark:

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

I-8091W User's Manual:

http://www.icpdas.com/root/product/solutions/machine_automation/download%20data/i8091/pack9091.pdf

■ 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 to 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 to 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 to 19)

Return Value:

0 (NO) : not yet stop
1 (YES) : stop

Examples:

```
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 to 19)

Return Value:

0 (NO) : not yet stop

1 (YES) : stop

Examples:

```
unsigned char data;
data= i8091_IS_Y_STOP(CARD1);
```

Remark:

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

7. Demos for 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 with the LP-8x4x. After installing the LP-8x4x SDK, the demo programs provided below can be found in the "**c:/cygwin/LinCon8k/examples**" folder.

7.1 DIO Control Demo for I-7k Modules

The **i7kdio.c** demo application illustrates how to control DI/DO function using an I-7050 module (8 DO channels and 7 DI channels) connected to an RS-485 network. The address of the module is 02 and the Baud Rate is 9600 bps.

The result of executing this demo program is that DO channels 0 to 7 on the I-7050 module will be set as the output channels, and DI channel 2 on the I-7050 module will be set as the input channel. The source code for the 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 disabled
      wBuf[4] = 100;                // TimeOut , 100 milliseconds
```

```
    wBuf[5] = 0x0f;                 // Set 8 DO Channels to ON
    wBuf[6] = 0;                    // Debug string
    // 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 procedure below to achieve the desired results:

STEP 1: **Write i7kdio.c**

Copy the above source code above to a blank text file and save it using the name - i7kdio.c or open the file from the C:\cygwin\LinCon8k\examples\i7k folder.

STEP 2: **Compile i7kdio.c to an executable file - i7kdio.exe**

Two methods can be used to compile the program, each of which is introduced here:

Method One – Using a Batch File (lcc.bat)

Open the LP-8x4x Build Environment by clicking the Start > Programs > ICPDAS > LP-8x4x SDK > LP-8x4x Build Environment to open **LP-8x4x SDK** window, and change the path to C:\cygwin\LinCon8k\examples\i7k. To compile the i7kdio.c file to an executable file, type **lcc i7kdio**. (refer to Fig. 7-1)

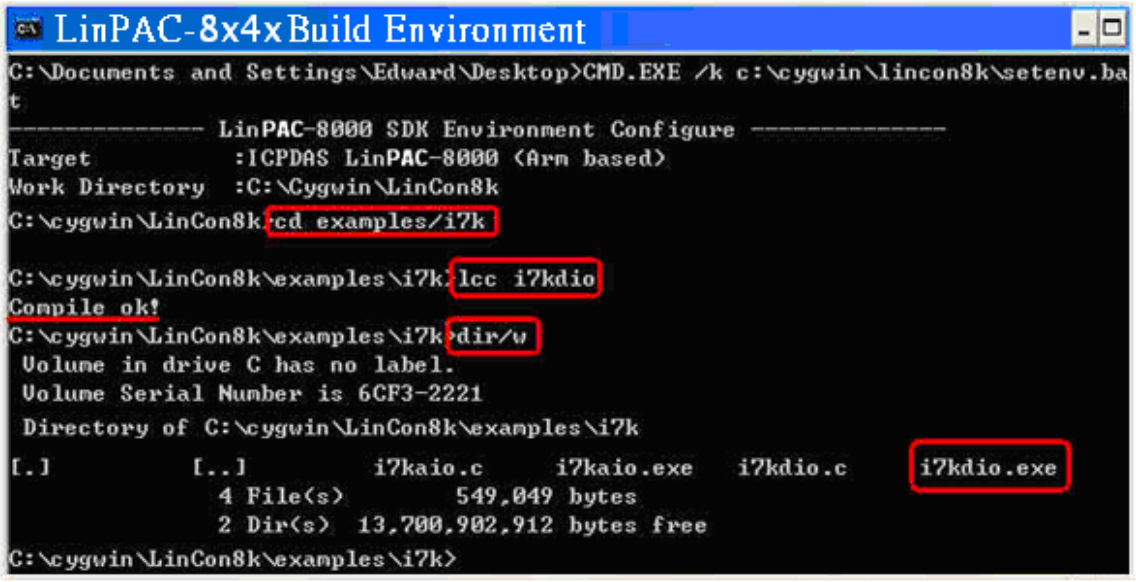


Fig. 7-1

Method Two – Using Compile Instructions

When using this method, type cd C:\cygwin\LinCon8k\examples\i7k command prompt to change the path. To compile i7kdio.c to an executable file, type arm-linux-gcc -I../../include –lm –o i7kdio.exe i7kdio.c ../../lib/libi8k.a (refer to Fig. 7-2).

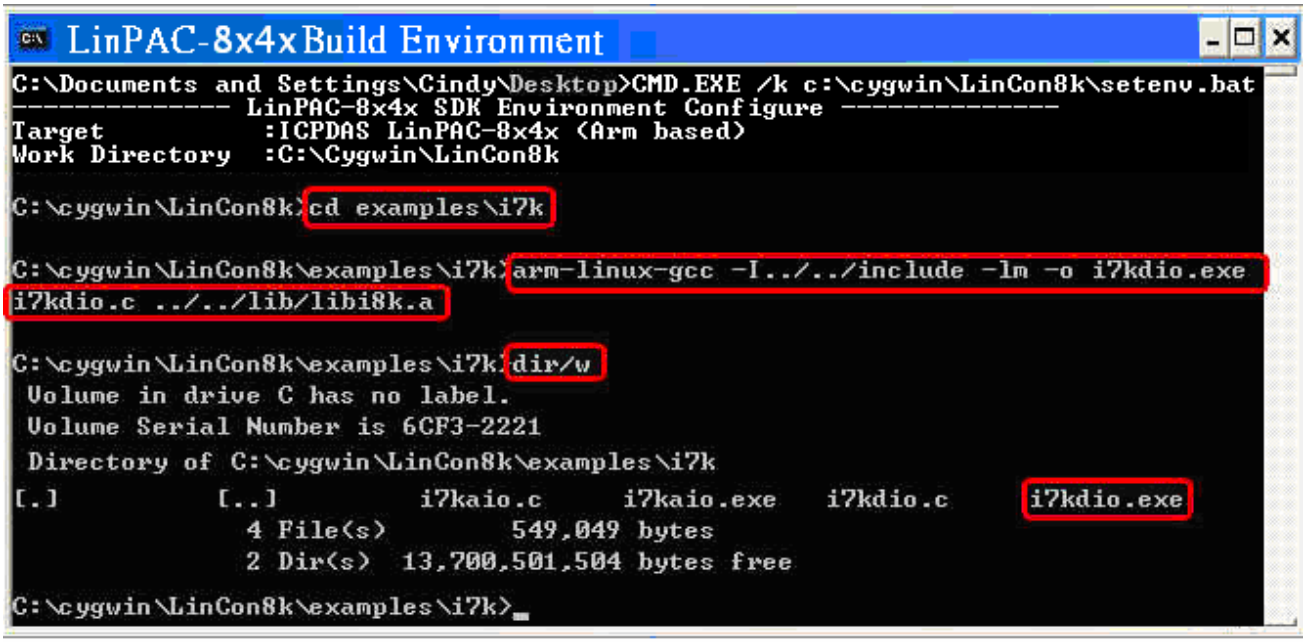


Fig. 7-2

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

Two methods can be used to transfer the executable file to the LP-8x4x, each of which is introduced here.

Method One – Using an FTP application

Step 1: Open a FTP application and create a new FTP connection. Enter the login details for the LP-8x4x, including the Host name (or IP address), Username and Passeord. The default value for both the **User_Name** and the **Password** is **"root"**. Click the **"Connect"** button to connect to the ftp server on the LP-8x4x. Refer to Fig.7-3 below for more details.

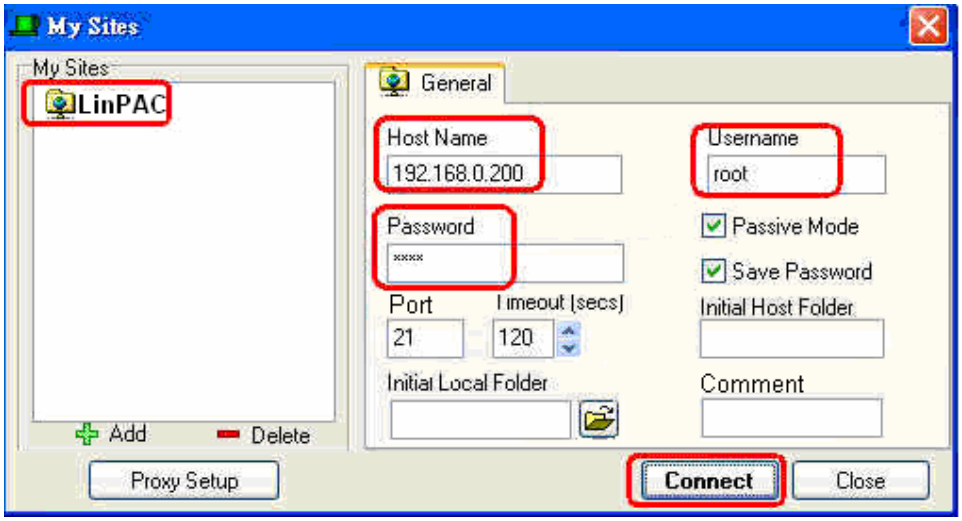


Fig.7-3

Step2: Upload the file **i7kdio.exe** file to the LP-8x4x. (refer to Fig.7-4).

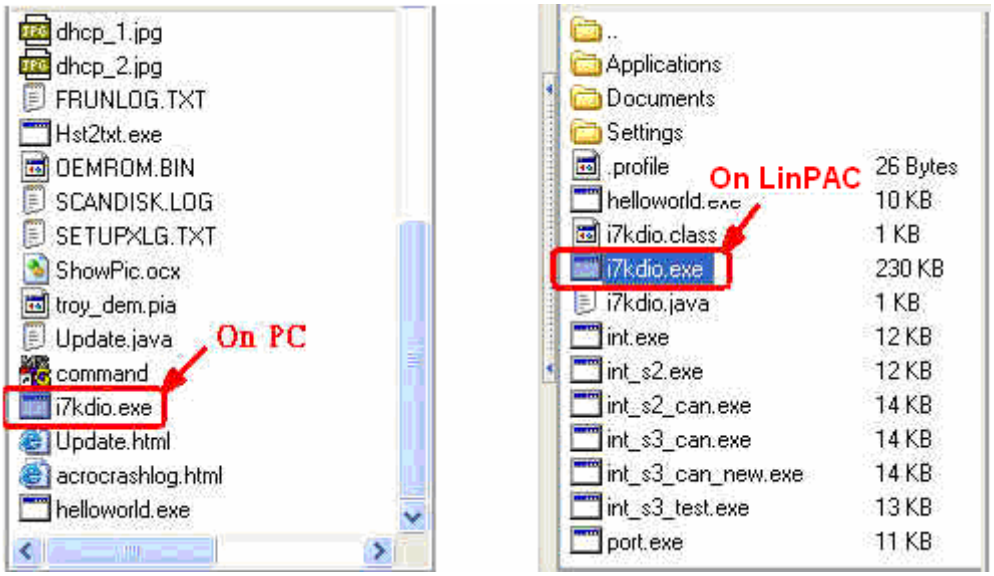


Fig.7-4

Step 3: Choose i7kdio.exe in the LP-8x4x and Click the right mouse button to select the "**Permissions**" option for the menu. Enter "777" in the Numeric textbox to set the file permissions to readable, writeable, and executable. Refer to FigS.7-5 and 7-6 below for more details.

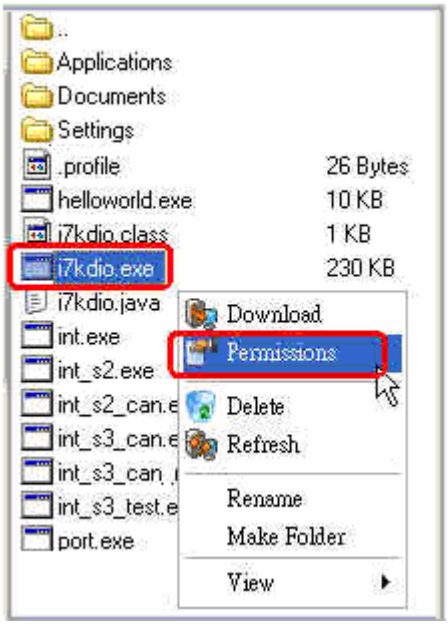


Fig.7-5

Fig.7-6

Method Two – Using a DOS Command Prompt

Open DOS Command Prompt and enter the IP Address of the server on the LP-8x4x in order to connect to the ftp server of the LP-8x4x. Enter the **User Name** and **Password** (the default value is root) to login to the LP-8x4x ftp server.

Files must be transferred in binary mode, so type "**bin**" to set the mode.

At Command Prompt, type put c:/cygwin/lincon8k/examples/i7k/i7kdio.exe i7kdio.exe to transfer the i7kdio.exe file to the LP-8x4x. Once the file has been transferred, the "Transfer complete" message will be displayed. Refer to Fig. 7-7 below for more details.

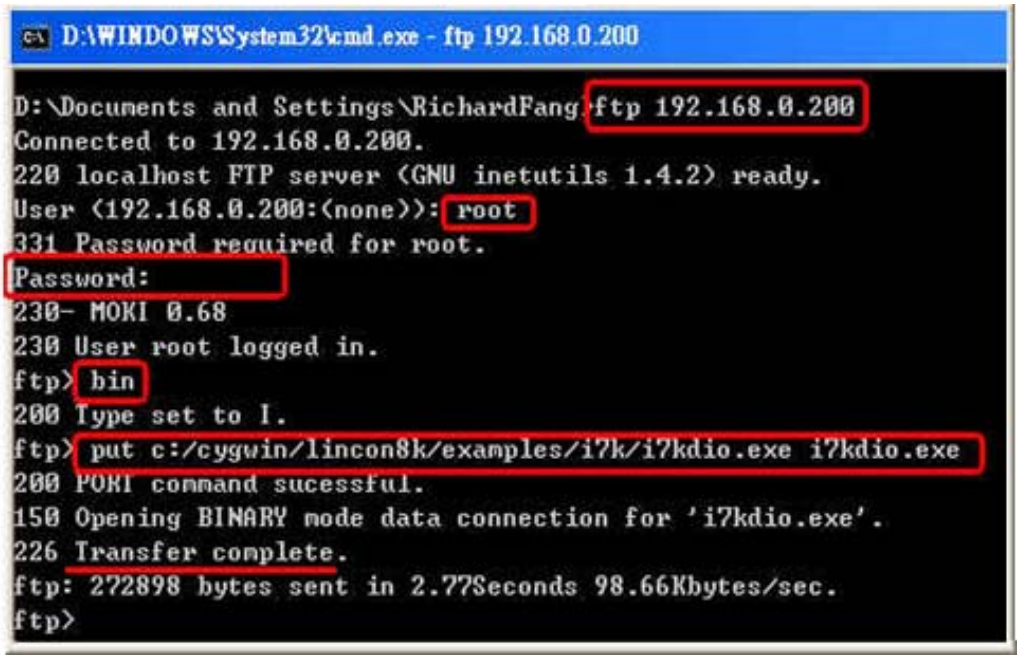


Fig. 7-7

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

At the Command Prompt, type telnet IP Address of the LP-8x4x to establish a connection to the LP-8x4x. Enter **User Name** and **Password** (the default value is **root**) to login to the LP-8x4x.

At Command Prompt, type chmod 777 i7kdio.exe to set the i7kdio.exe file to executable, and then type i7kdio.exe to execute the i7kdio.exe file. Refer to Figs. 7-8 and 7-9 below for more details.

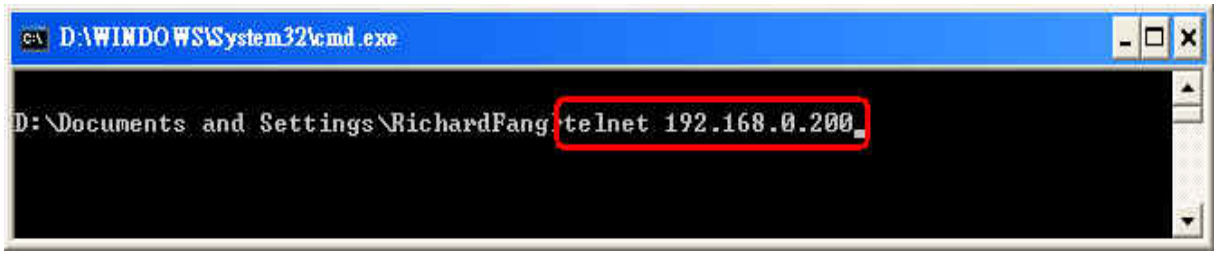


Fig. 7-8

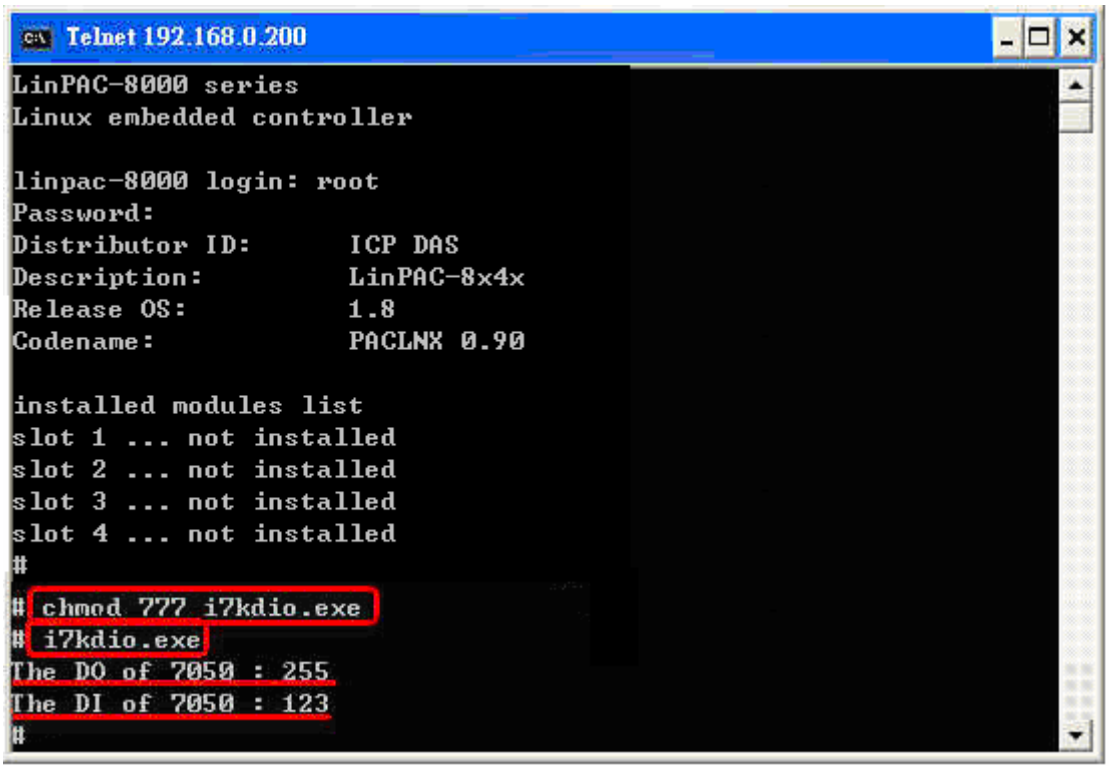


Fig. 7-9

The message "**The DO of I-7050：255** (=2 ^ **8** - 1)" indicates that DO channels 0 to 7 will be used to output data, and the message "**The DI of I-7050：123** (=127 - 2 ^ **2**)" indicates that DI channel 2 will be used as the input channel.

7.2 AIO Control Demo for I-7k Modules

The **i7kaio.c** demo application illustrates how to control the AI/AO functions using an I-7017 module (8 AI channels) and an I-7021 modules (1 AO channel) connected to an RS-485 network. The addresses for the I-7021 and I-7017 modules are 05 and 03, respectively, and the baudrate for both modules is 9600 bps.

The result of executing this demo program is that the AO channel on the I-7021 module will be set to output a voltage of 3.5V, and AI channel 2 on the I-7017 module will be set as the input channel. The source code for 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 milliseconds
	//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)                    // There was an error with the Analog Output on the I-7021
		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 disabled
	wBuf[4] = 100;                  // TimeOut , 100 milliseconds
	wBuf[5] = j;                    // Channel of AI
	wBuf[6] = 0;                    // Debug string

	wRetVal = AnalogIn(wBuf, fBuf, szSend, szReceive);
	if (wRetVal)                    // There was an error with the Analog Input on the I-7017
		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;
}
```

For this example, the programming and execution procedures are the same as those described in the section 7.1. Fig. 7-10 below illustrates the result of execution.

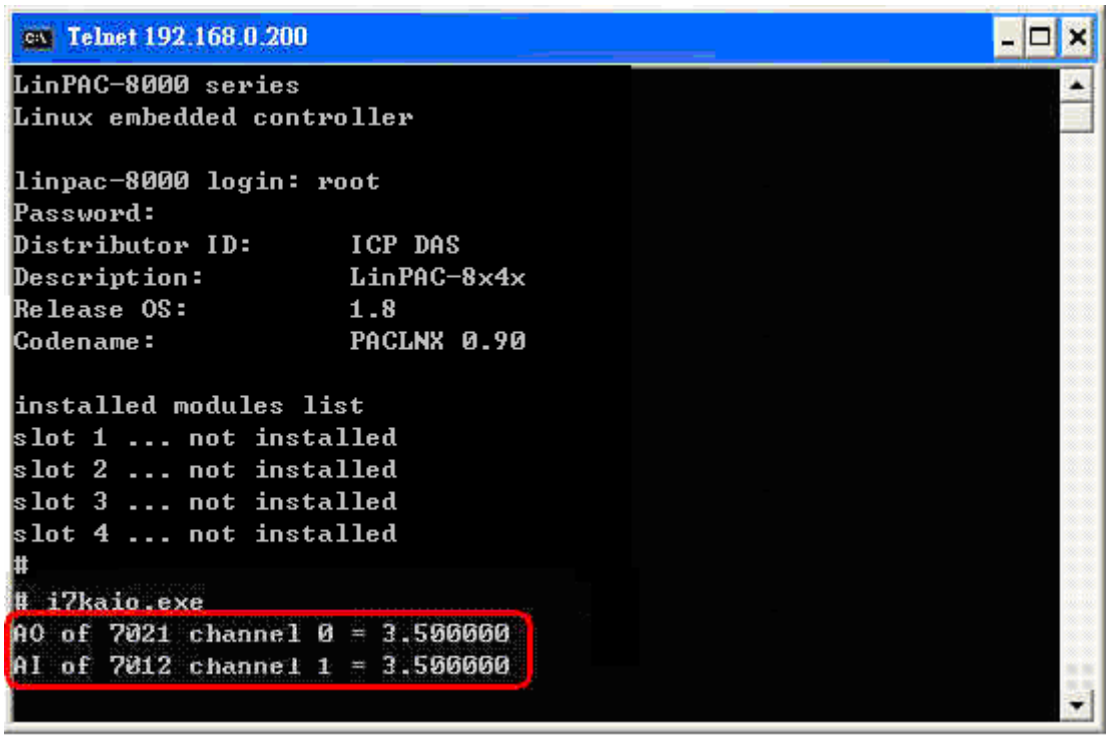


Fig. 7-10

7.3 DIO Control Demo for I-87KW Modules

When using I-87KW DIO modules to perform I/O control on the LP-8x4x, the program will be slightly different, depending on the location of the I-87KW modules. The code will need to be modified depending on the configuration of the I-87k modules. There are three possible variations, as described below:

(1) If there are I-87KW DIO modules inserted **in the slots on theLP-8x4x**, the “Open_Slot()” and “ChangeToSlot()” functions, must be called before other functions for the I-87KW modules and used, and the “Close_Slot()“ function also needs to be called at the end of the program. Refer to the demo code provides in section 7.3.1. for more details.

(2) If there are I-87KW DIO modules inserted **in the slots on I-87k I/O expansion unit**, refer to the demo code provided in section 7.3.2 for more information.

(3) If there are I-87KW DIO modules inserted **in the slots on the I-8000 controller**, then these modules will be regarded as I-8KW modules. For more details, refer to the instructions for performing I/O control on I-8KW modules provided in section 7.5.2.

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

The **i87kdio.c** demo program will illustrates how to control the DI/DO function using an I-87054W module (8 DO channels and 8 DI channels). The module is in slot 3 on the LP-8x4x. The address and baudrate in the LP-8x4x are 00 and 115200 respectively, they were fixed by library. The result of this demo program is that DO channels 0 to 7 on the I-87054W module will be se as the output channels, and DI channel 1 on the I-87054W module will be set as the input channel. The source code for 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 disabled
    dwBuf[4] = 100;                 // TimeOut , 100 milliseconds
    dwBuf[5] = 0xff;                // Set digital output
    dwBuf[6] = 0;                   // Debug string
     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 disabled
      dwBuf[4] = 100;                  // TimeOut , 100 milliseconds

      dwBuf[6] = 0;                    // Debug string
      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 disabled
      dwBuf[4] = 100;                  // TimeOut , 100 milliseconds
      dwBuf[5] = 0x00;                 // Digital output
      dwBuf[6] = 0;                    // Debug string
      getch();                        // Press 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 on I-87KW I/O expansion unit

If the I-87KW DIO modules are inserted in the slots on an I-87KW I/O expansion unit, three parts of the program illustrated in section 7.3.1 above will need to be modified as follows:

(1) The **Open_Slot(), ChangeToSlot()**, and **Close_SlotAll()** functions should be deleted.

(2) The **address** and **baudrate** of any I-87KW modules connected to the RS-485 network will need to be configured using the DCON Utility, which can be downloaded from http://www.icpdas.com/products/dcon/introduction.htm.

(3) The **Open com1** (i.e., the internal serial port on the LP-8x4x) will need to be change to **open com3** (i.e., the RS-485 port on the LP-8x4x).

The I-87054W is connected to an RS-485 network where the address is set to be 06 and the baudrate is 9600 bps, which must be configured using the DCON Utility. The source code for the **i87kdio_87k.c** 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, 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 milliseconds
      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 disabled
      dwBuf[4] = 100;               // TimeOut , 100 milliseconds
      dwBuf[6] = 0;                 // Debug string

      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 disabled
    dwBuf[4] = 100;                // TimeOut , 100 milliseconds
    dwBuf[5] = 0x00;               // Digital output
    dwBuf[6] = 0;                  // Debug string
    getch();                       // Press any key to continue
    wRetVal = DigitalOut_87k(dwBuf, fBuf, szSend, szReceive);

    Close_Com(COM3);
return 0;
}
```

For this example, the programming and execution procedures are the same as those described in the section 7.1. Fig. 7-11 below illustrates the result of execution.

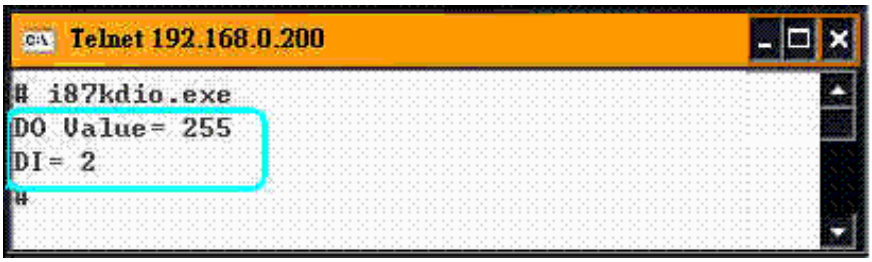


Fig. 7-11

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

If the I-87KW DIO modules are inserted in the slots on an I-8000 controller, the I-87KW modules will be regarded as I-8KW modules. For more details, refer to the description of how to perform DI/DO control on I-8KW modules provided in the section 7.5.

7.4 AIO Control Demo for I-87KW Modules

When using I-87KW modules to perform I/O control on the LP-8x4x, the program code will be slightly different depending on the location of the I-87KW modules.The code will need to be modified depending on the configuration of the I-8KW modules. There are three possible variations, as described below:

(1) If there are I-87KW AIO modules inserted **in the slots on the LP-8x4x**, the “Open_Slot” and “ChangeToSlot” functions must be called before other functions for the I-87KW modules are used, and the “Close_Slot()“ function also needs to be called at the end of the program. Refer to the demo code provided in section 7.4.1.

(2) If there are I-87KW AIO modules inserted **in the slots on the I-87k I/O expansion unit**, refert to the demo code provided in section 7.4.2 for more information.

(3) If there are I-87KW AIO modules inserted **in the slots on the I-8000 controller**, then these modules will be regarded as I-8KW modules. For more details, refer to the instructions for performing I/O control on I-8KW modules provided in section 7.6.2.

7.4.1 I-87KW Modules in slots on an LP-8x4x

The **i87kaio.c** demo program illustrates how to control the AI/AO using an the **I-87022W** module (2 AO channels) and an **I-87017W** module (8 AI channels). The I-87022W and I-87017W modules are inserted into slots 2 and 3 of the LP-8x4x reseparately. The addresses and baudrate fo both modules in the LP-8x4x are 00 and 115200 bps reseparately, they were fixed by library.

The result of executing this demo program is that AO channel 0 on the I-87022W module will be set to output a voltage of 2.5V, and AI channel 1 on the I-87017W module will be set as the input channel. The source code for 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 milliseconds
    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)                // There was an error with the Analog Output on the I-87022W
        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 disabled
    wBuf7[4] = 100;             // TimeOut , 100 milliseconds
    wBuf7[5] = j;               //Channel Number of AI
    wBuf7[6] = 0;               // Debug string
    wRetVal = AnalogIn_87k(wBuf7, fBuf, szSend, szReceive);
    if (wRetVal)                // There was an error with the Analog Output on the I-87017W
        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 on an I-87KW I/O expansion unit

If the I-87KW modules are inserted in slots on an I-87KW I/O expansion unit, the three parts of the program illustrated in Section 7.4.1 above will need to be modified, as follows:

(1)The **Open_Slot()**, **ChangeToSlot()**, and **Close_SlotAll()** functions should be deleted.

(2) The **address** and **baudrate** of any I-87KW modules connected to the RS-485 network will need to be configured using the DCON Utility, which can be downloaded from http://www.icpdas.com/products/dcon/introduction.htm.

(3) The **Open com1** (i.e,. the internal serial port on the LP-8x4x) will need to be changed to **open com3** (i.e., the RS-485 port on the LP-8x4x)

The I-87022W and I-87017W addresses are connected to the RS-485 network and the addresses are set to 01 and 02 reseparately, with the baudrate for both modules set to 9600 bps, which must be configured using the DCON Utility. The source code for the **i87kaio_87k.c** 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_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 disabled
```

```
    wBuf[4] = 100;                  // TimeOut , 100 milliseconds
    wBuf[5] = i;                    // Channel Number of AO
    wBuf[6] = 0;                    // Debug string
    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 disabled
    wBuf7[4] = 100;                 // TimeOut , 100 milliseconds
    wBuf7[5] = j;                   //Channel Number of AI
    wBuf7[6] = 0;                   // Debug string

    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;
}
```

For this example, the programming and execution procedures are the same as those described in section 7.1. Fig. 7-12. below illustrates the result of execution.

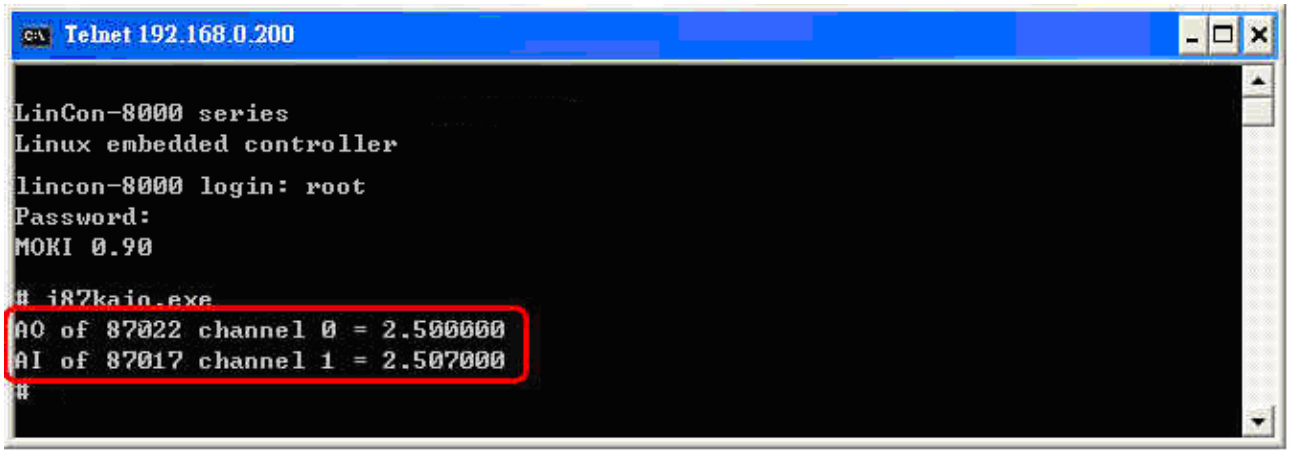


Fig. 7-12

7.4.3 I-87KW Modules in slots on an I-8000 Controller

If the I-87KW AIO modules are inserted in slots on an I-8000 controller, the modules will be regarded as I-8KW modules. For more details, refer to the description of how to perform AI/AO control on I-8KW modules provided in section 7.6.

7.5 DIO Control Demo for I-8KW Modules

The source file from the Libi8k.a library for -8KW modules that are inserted in the slots on an I-8000 controller is the **I8000.c** file. However, the source file from the Libi8k.a library for I-8KW modules that are inserted in the slots on the LP-8x4x is Slot.c file. Consequently, the functions used for I-8KW modules inserted in the slots on the LP-8x4x and in the slots on the I-8000 controller are completely different.

Two different demo programs an be used depend on the configuration of the I-8KW modules, as described below:

(1) If there are I-8KW DIO modules inserted **in the slots on the LP-8x4x**, refer to the demo code provided in section 7.5.1 for more information.

(2) If there are I-8KW DIO modules inserted **in the slots on the I-8000 controller**, refer to the demo code provided in section 7.5.2 for more information.

7.5.1 I-8KW Modules in slots on an LP-8x4x

The **i8kdio.c** demo program illustrates how to control the DI/DO functions using an I-8055W modules (8 DO channels and 8 DI channels) that is inserted into slot 3 on the LP-8x4x.

The address and baudrate for the LP-8x4x are 00 and 115200 bps reseparately, and they were fixed by library.

The result of executing this demo program is that DO channels 0 to 7 on the I-8055W module to will be set as the output channels, and DI channel 0 on I-8055W module will be set as the input channel. The source code for 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;
}
```

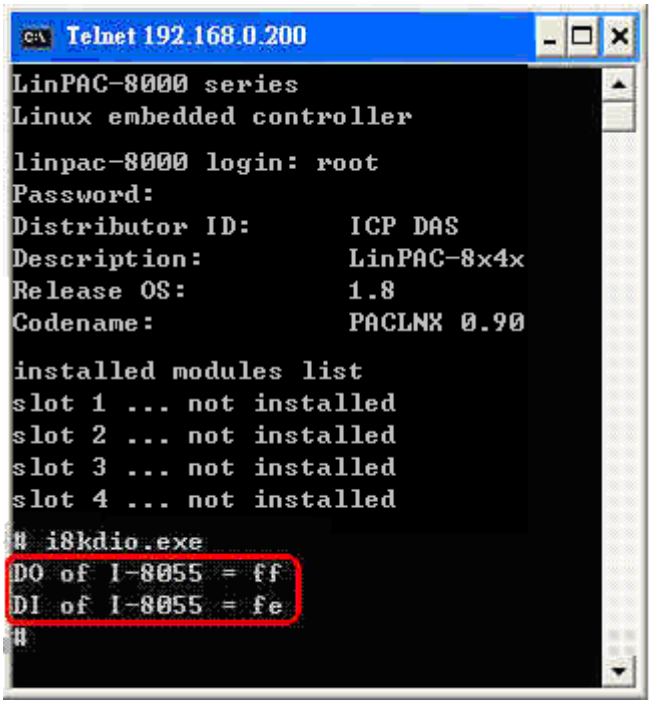


Fig. 7-13

For this example, the programming and execution procedures are the same as those described in section 7.1. Fig. 7-13 above illustrates the result of execution.

7.5.2 I-8KW Modules in slots on an I-8000 Controller

The **i8kdio_8k.c** demo program illustrates how to control the DI/DO using the I-8055W module (8 DO channels and 8 DI channels) on an I-8000 controller. Configure the hardware by following the procedure described below:

(1) Insert the I-8055W module into slot 0 on the I-8000 controller.

(2) Connect the **COM3** on the LP-8x4x to the COM1 on the I-8000 controller using an RS-232 cable.

The address of the I-8000 controller is 01 and the baudrate is 115200 bps, which must be configured using the DCON Utility. The result of executing this demo program is that DO

channels 0 to 7 on the I-8055W module to will be set as the output channel, and DI channel 0 on the I-8055W module will be set as input channel. The source code for 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 disabled
	dwBuf[4] = 100;                 // TimeOut , 100 milliseconds
	dwBuf[5] = 0xff;                // Digital output
	dwBuf[6] = 0;                   // Debug string
	dwBuf[7] = 1;                   // slot number
	wRetVal = DigitalOut_8K(dwBuf, fBuf, szSend, szReceive);
	if (wRetVal)                    // There was an error with the Analog Output on the I-8055
		printf("DO of I-8055 Error , Error Code=%d\n", wRetVal);
	else
		printf("DO of I-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 disableD
	dwBuf[4] = 100;                 // TimeOut , 100 milliseconds
	dwBuf[6] = 0;                   // Debug string
	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 disabled
    dwBuf[4] = 100;                 // TimeOut , 100 milliseconds
    dwBuf[5] = 0x00;                // Digital output
    dwBuf[6] = 0;                   // Debug string
    dwBuf[7] = 1;                   // slot number

    getch();                        // push any key to continue
    wRetVal = DigitalOut_8K(dwBuf, fBuf, szSend, szReceive);
    Close_Com(COM3);
    return 0;
}
```

For this example, the programming and execution procedures are the same as those described in section 7.1. Fig. 7-14 below illustrates the result of execution.

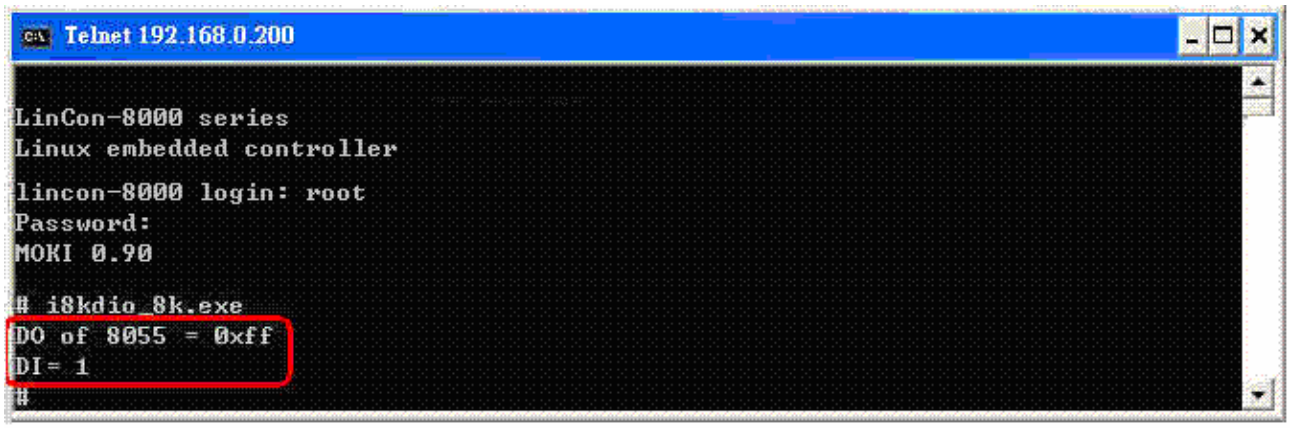


Fig. 7-14

7.6 AIO Control Demo for I-8KW Modules

The source file from the Libi8k.a library for I-8KW modules inserted in the slots on an I-8000 controller is the **I8000.c** file. However, the source file from the Libi8k.a library for I-8KW modules that are inserted in the slots on the LP-8x4x is Slot.c file. Consequently, the functions used for the I-8KW modules inserted in the slot on the LP-8x4x and in the slots on the I-8000 controller are completely different.

Two different demo programs can be used depending on the configuration of the I-8KW modules, as described below:

(1) If there are I-8KW AIO modules inserted in the slots on the LP-8x4x, refer to the demo code provided in section 7.6.1 for more information.

(2) If there are I-8KW AIO modules inserted in the slots on the I-8000 controller, refer to the demo code provided in section 7.6.2 for more information.

7.6.1 I-8KW Modules in the slots on an LP-8x4x

The **i8kaio.c** demo program illustrates how to control the AI/AO functions using the I-8024W (4 AO channels) and I-8017HW (8 AI channels) modules, which are inserted in slot 1 and slot 2 on the LP-8x4x reseparately.

The address and baudrate in the LP-8x4x are 00 and 115200 bps reseparately, and they were fixed by library. The result of executing this demo is that AO voltage channel 0 on the I-8024W module to will be set to output 5.5 V and AI channel 2 on the I-8017HW module to will be set as the input channel. The source code for 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 the uncalibrated AI Hex Value
	printf("8017_AI_not_Cal_Hex =%x\n",hAi);
	fAi = HEX_TO_FLOAT_Cal(hAi,2,0);          //Uncalibrated AI Hex Value and then
modify it to a 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 the Calibrated AI Hex Value
	printf("8017_AI_Cal_Hex =%x\n",hAi);
	fAi = CalHex_TO_FLOAT(hAi,0);
	//Modify the Calibrated AI Hex Value modify to a 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 the Calibrated AI Float Value
	printf("8017_AI_Cal_Float =%f\n\n\n",fAi);

	Close_Slot(2);
	return 0;
}
```

For this example, the programming and execution procedures are the same as those described in section 7.1. Fig. 7-15 below illustrates the result of execution.

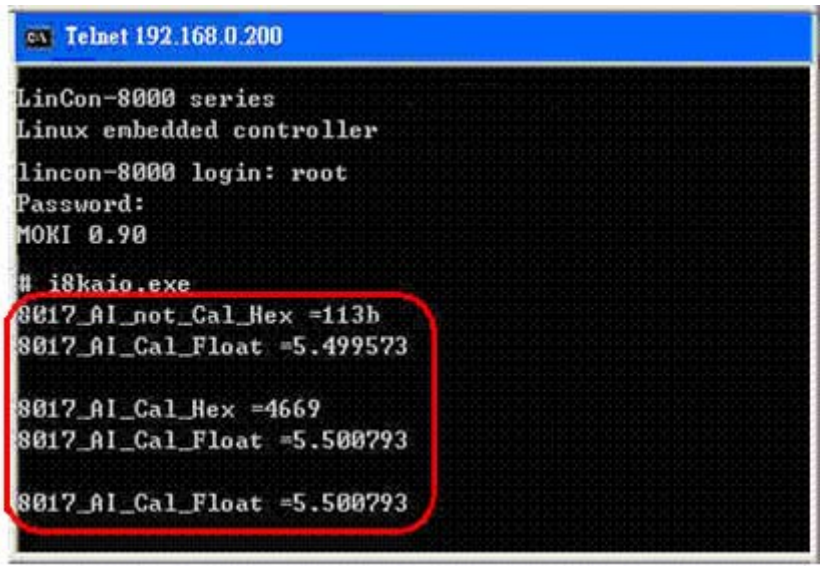


Fig. 7-15

7.6.2 I-8KW Modules in slots on an I-8000 Controller

The **i8kaio_8k.c** demo program illustrates how to control the AI/AO functions using the I-8024W (4 AO channels) and I-8017HW (8 AI channels) modules, which are inserted into slot 0 and slot 1 on the I-8000 controller. Configure the hardware by following the procedure described below:

(1) Insert the I-8024W and I-8017HW modules in slot 0 and slot 1 on the I-8000 controller respectively.

(2) Install 8k232.exe or R232_300.exe to flash memory of I-8000 controller as firmware.

(3) Connect **COM3** on the LP-8x4x to COM1 on the I-8000 controller using an RS-232 cable.

The address of the I-8000 controller is 01 and baudrate is 115200 bps, which must be configured using the DCON Utility. The result of executing this demo program is that AO voltage channel 0 on the I-8024W module to will be set to output 3.5 V, and AI channel 2 on the I-8017HW module will be set as the input channel. The source code for 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 disabled
	wBuf[4] = 100;                  // TimeOut , 100 milliseconds
	wBuf[5] = i;                    // Channel No. of AO
	wBuf[6] = 0;                    // Debug string
	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 disabled
	wBuf[4] = 100;                  // TimeOut , 100 milliseconds
	wBuf[5] = j;                    // Channel of AI
	wBuf[6] = 0;                    // Debug string
	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;
}
```

For this example, the programming and execution procedures are the same as those described in section 7.1. Fig. 7-16 below illustrates the result of execution.

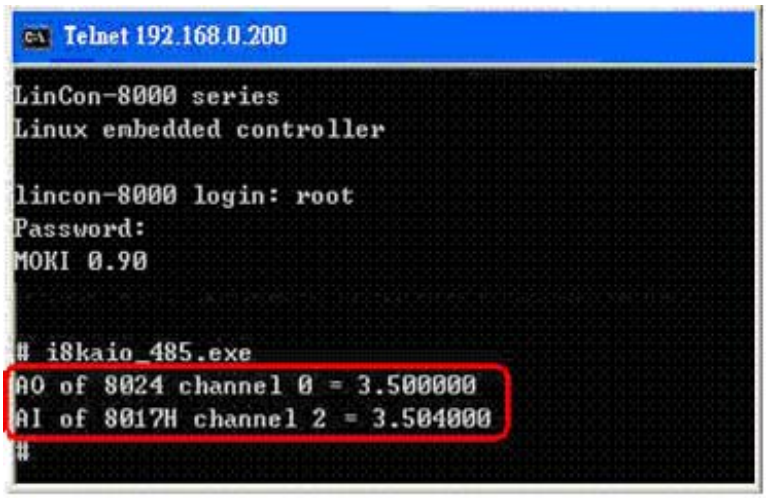


Fig. 7-16

7.7 Overview of the Module Control Demo Program

Fig. 7-17 provides a summary of the various communication functions that can be used depending on the for the different locations of the I-7000/I-8000/I-87000 modules when using the ICP DAS modules in conjunction with the LP-8x4x, which can be helpful in understanding which communication functions should be used.

Note that the Open_slot()/Close_Slot() and sio_open()/sio_close() functions cannot be used for the 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 provides an overview of the source files from the libi8k.a library that can be used depending on the different locations of I-7000/I-8000/I-87000 modules when using ICP DAS modules in conjunction with the LP-8x4x, which can be helpful in understanding which source files should be calld.

Source File / Module Location	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

For an example of how to use the "**Timer**" function in conjunction with the LP-8x4x, refer to the – timer.c and time2.c demo programs provided in the LinPAC SDK - (C:\cygwin\LinCon8k\examples\common) folder. The **timer.c** program can be used when the execution period is between 0.5 to 10 ms (Real-Time) and the **timer2.c** program can be used when the execution is period greater than 10 ms (General).

8. Overview of the Serial Ports on the LP-8x4x

The following is a description of the functionlity for the three serial ports contained in the LP-8x4x embedded controller, and are based on the RS-232 or RS-485 interfaces. Fig 8-1 illustrates the ports contained on the LP-8841 and Fig 8-2 illustrates those on the 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	Definition 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

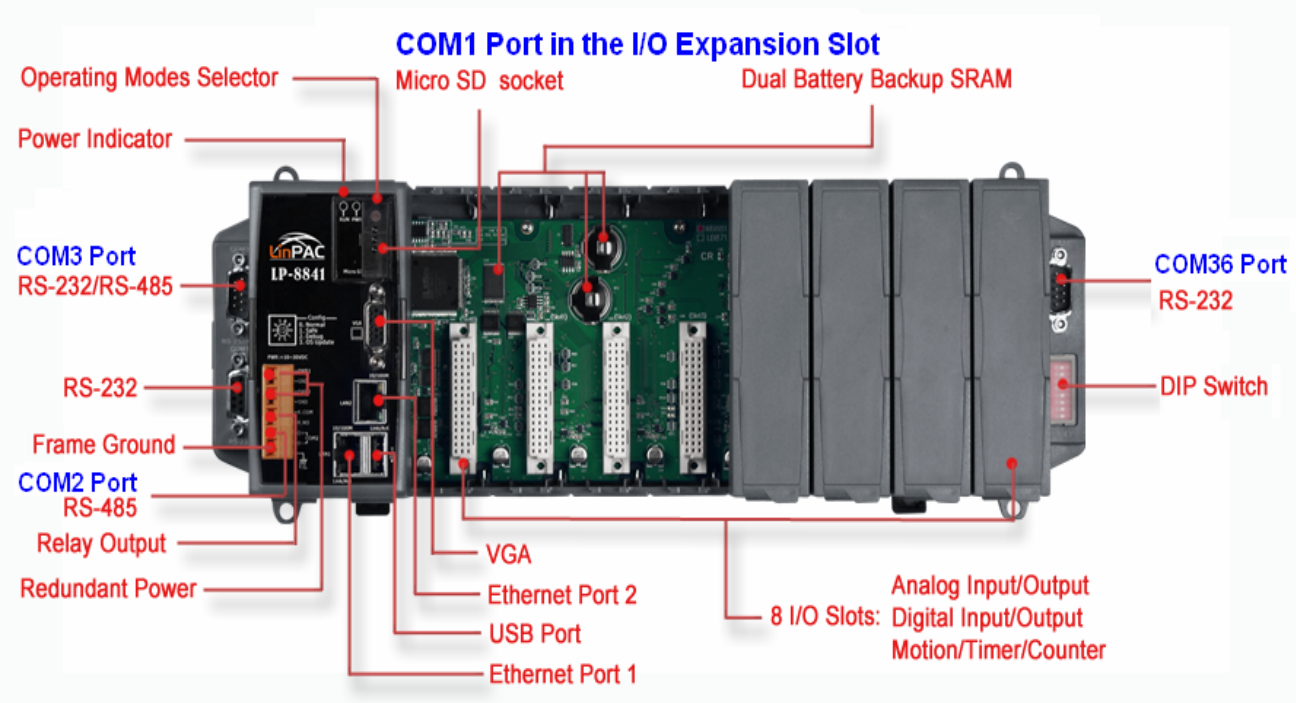


Fig. 8-1

Use the **stty** command to query or configure the COM port. For example, to modify the baudrate 9600 to 115200 bps via COM3 port:

stty -F /dev/ttyS1 ispeed 115200 ospeed 115200

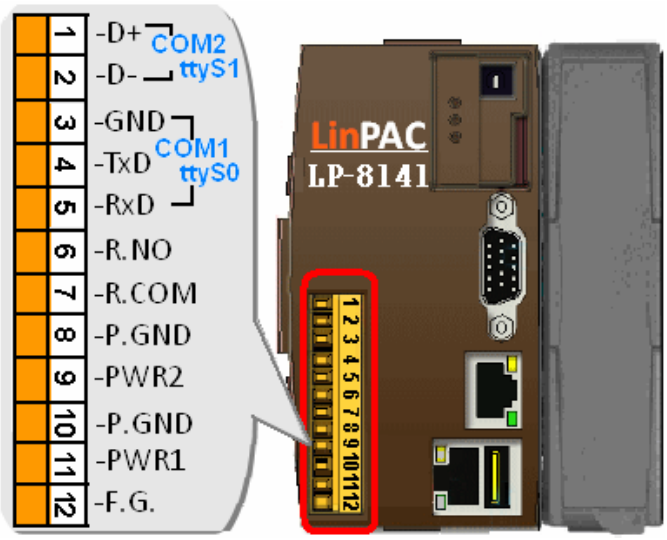


Fig. 8-2

8.1 COM1 Port

COM1 is an the internal I/O expansion port on the LP-8x4x, and is used to connect to an I-87KW series module inserted into the LP-8x4x embedded controller. A serial command must be used to control the I-87KW series module.

To control the I-87KW module, the Com port parameters and call the **Open_Com()** function to open the COM1 port based on the appropriate settings. Finally, call the **ChangeToSlot(slot)** function to specify which slot will be controlled. This is like the serial address, meaning that control commands can be sent to an I/O module that is inserted in the specified slot. Consequently, the serial address for the slot that contains the module 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 LP-8x4x
DWORD baudrate=115200;
Open_Slot(slot);
Open_Com(port, baudrate, data, parity, stopbit);
ChangeSlotToI-87k(slot);
// send command...
Close_Com(port);
Close_Slot(slot);
```

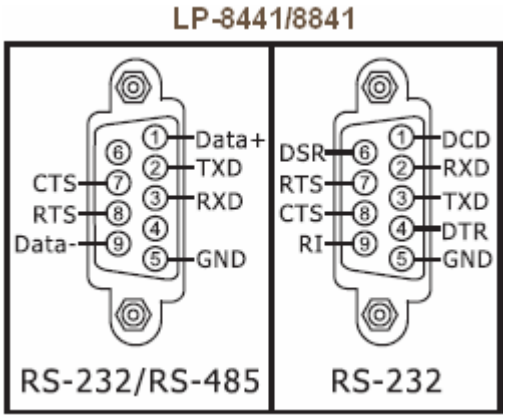
8.2 COM3/COM36 Port

This COM3/COM36 port is located on the right-upper corner on the LP-8441/8841, and is a standard **RS-232** serial port that provides TxD, RxD, RTS, CTS, GND, non-isolated and a maximum speed of 115200 bps.

This port can also be used to connect to an I-7520 module in order to provide general RS-485 communication functionality. The COM3/COM36 port can also be used to connect to a wireless modem so that the module can be controlled from a remote device. The application example and code is demonstrated below:

- Test using 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 a command...
Close_Com(port);
```



- Test using the command line interface: (PC connected to COM3 on the LP-8x4x)

  A) Open "**Hyper Terminal**" on the Host PC to monitor the test process. The default settings for COM3 port are 9600, 8, N, 1

  B) Send data via COM3 port:

  On the LP-8x4x:

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

  Check that the word "test" is displayed on the "Hyper Terminal" screen on th PC

  C) Receive data via the COM3 port:

  On the LP-8x4x:

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

  On the PC:

  Enter some text in the "Hyper Terminal" screen on the PC

  Check that the some words on the LP-8x4x.

8.3 COM2/COM3 Port

The COM2/COM3 port provides **RS-485** serial communication functionality (DATA+ and DATA-) and is located on the bottom-right corner on the LP-8x4x. This port allow a connection to be made to modules that contain an RS-485 interface such as the I-7000, I-87k and I-8000 serial modules (DCON Module), meaning that ICP DAS I-7000/I-87k/I-8000 series modules can be directly controlled via this port with any converter. ICP DAS provides a very easy to use library of functions (libi8k.a) that can used to easily communicate with I-7000/I-87k/I-8000 series modules. Below is an application example of the program code demo.

- Test using 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 using command line: (PC <-----> i-7520 <-----> COM2 on the LP-8x4x - see Fig 8-3)

  A) Open "**Hyper Terminal**" on the Host PC to monitor the test process. The default settings for the COM2 port are 9600, 8, N, 1

  B) Send data via COM2 port:

  On the LP-8x4x:

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

  Check that the word "test" is displayed on the "Hyper Terminal" screen on the PC.

  C) Receive data via the COM2 port:

  On the LP-8x4x:

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

  On the PC:

  Enter some words in the "Hyper Terminal" screen on the PC

  Check that the same text displayed on the LP-8x4x.

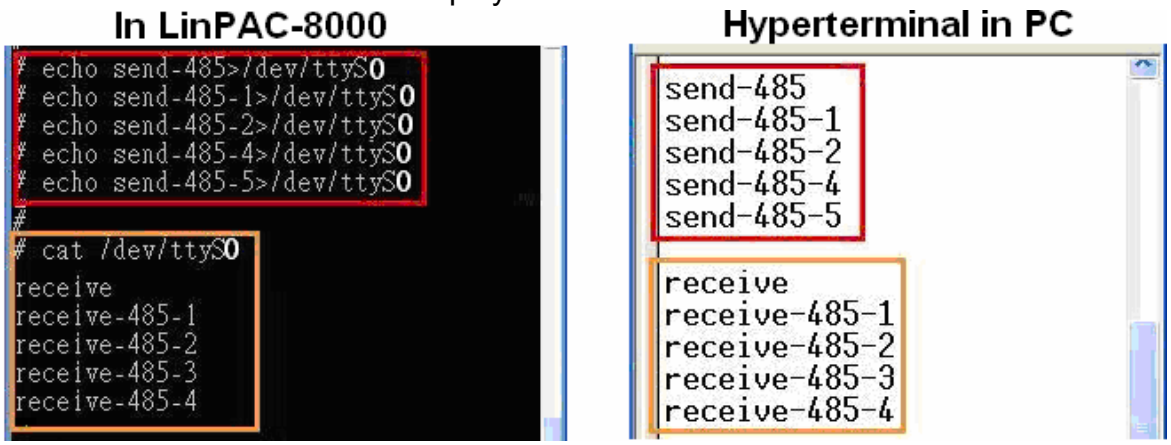


Fig. 8-3

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 milliseconds)

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

This chapter provides additional information related to the modules supported, together with instructions that can be used to enhance the functionality and efficiency of the LP-8x4x module.

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

N-port communication modules provide **two** or **four serial ports** and can be inserted into the slot of an LP-8x4x embedded controller. In this way, additional serial ports can be used on the LP-8x4x embedded controller, meaning that the maximum number of serial ports available on the LP-8x4x will be expanded to **thirty-four**.

The LP-8x4x embedded controller is a multi-tasking uint, meaning that all the serial ports can be controlled simultaneously. **The number** of **each serial port on the** I-8114W and I-8112iW modules are presented in Figs.10-1 and10-2. The information illustrated in Fig. 10-5 is for the LP-8141 only, and is **fixed** based on their slot position.

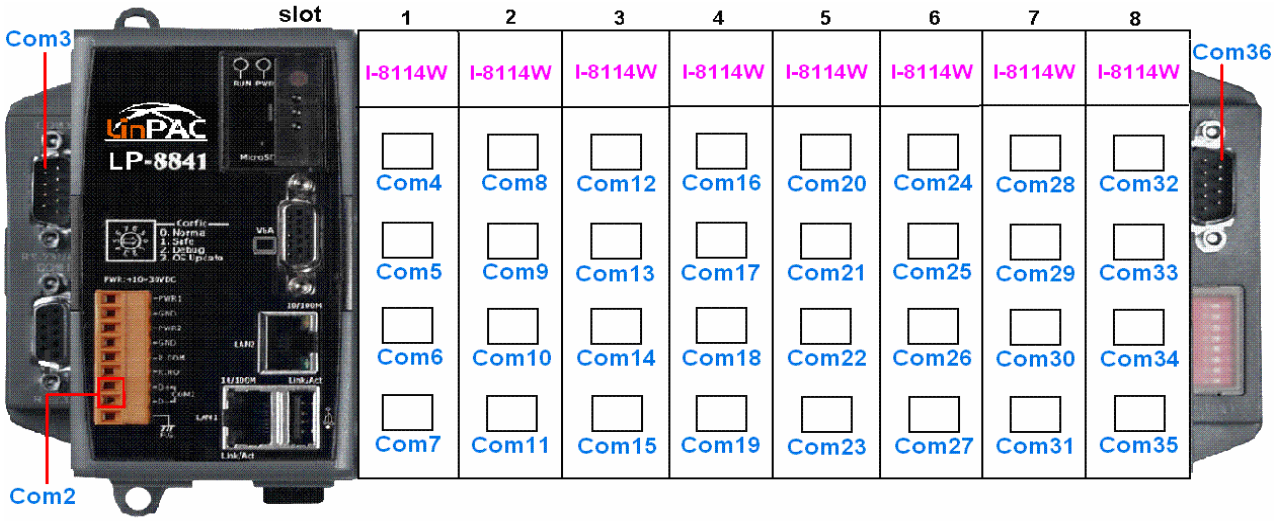


Fig.10-1

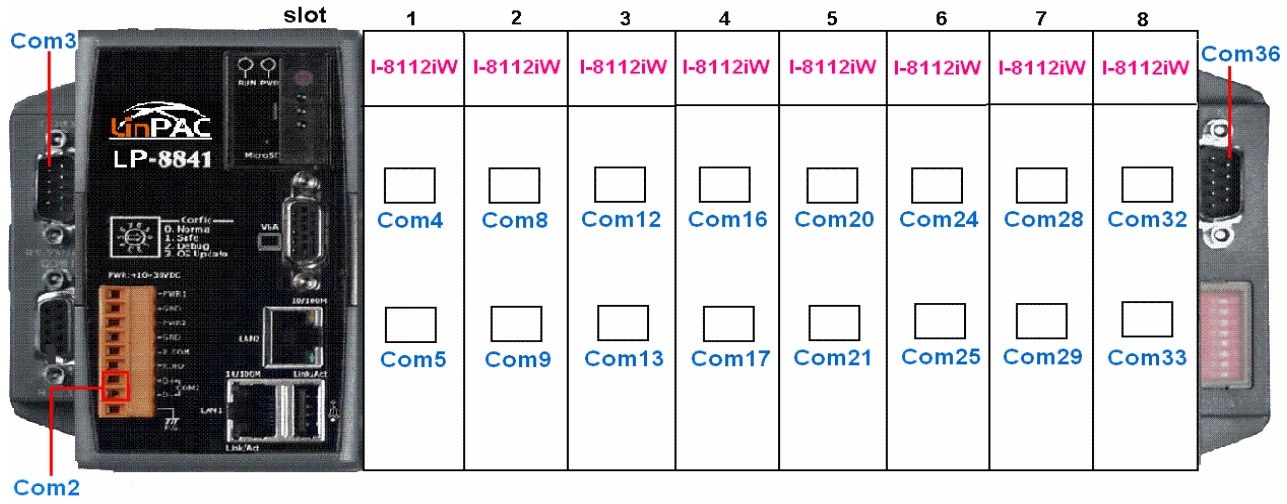


Fig.10-2

Figs.10-3 and 10-4 illustrated the serial port numbers that correspond to the **device name** on the LP-8x4x.

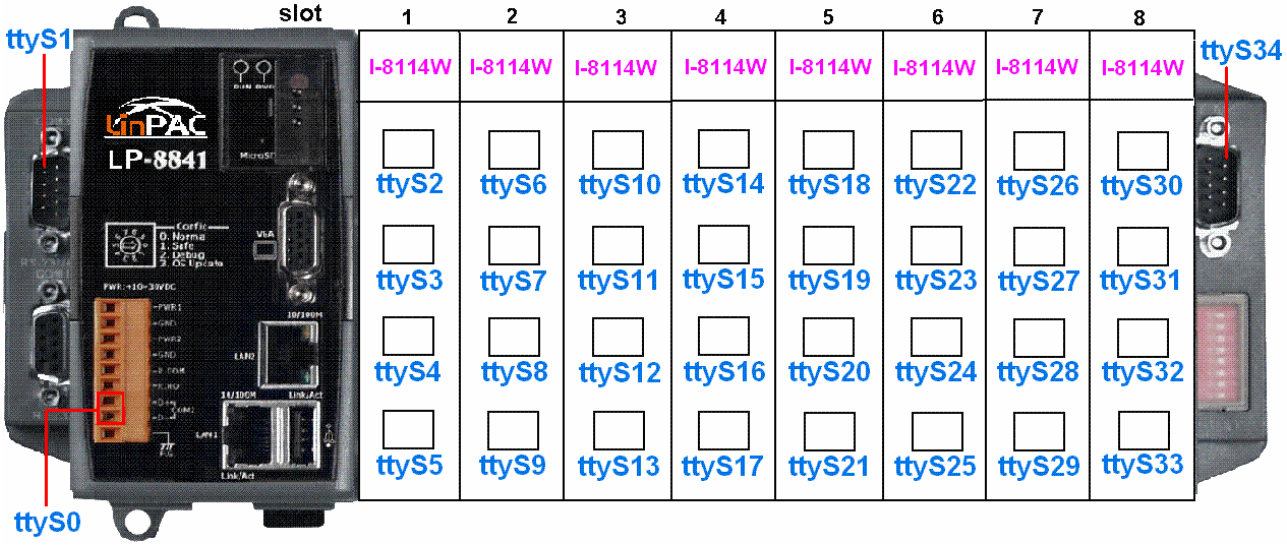


Fig.10-3

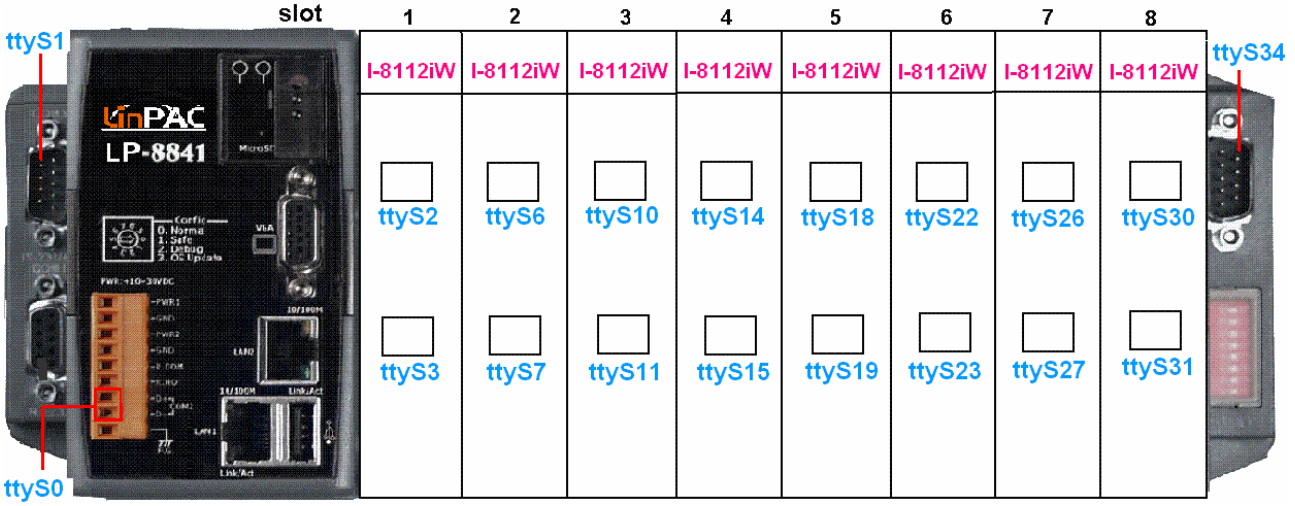


Fig.10-4

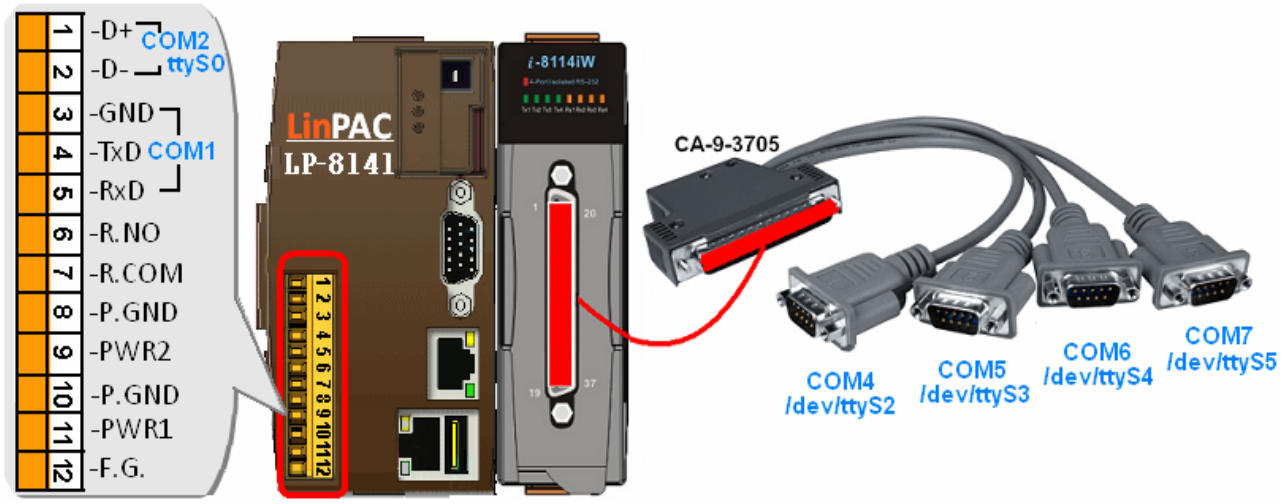


Fig.10-5

Selection guide for Hight-profile I-8K modules:

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

For more information relating to these modules, refer to:
http://www.icpdas.com/products/Remote_IO/i-8ke/selection_rs232_i8k.htm

The **i7kdio_8114.c** demo program illustrates how to use an I-8114W module that is inserted into an LP-8x4x embedded controller. In this demo program, the I-7044 module (8 DO and 4 DI channels) is controlled through the second serial port on the I-8114W module that is inserted into the slot 2 on the LP-8x4x, which, in turn, is connected to an RS-485 network. The address of the I-7044 module is 02 and the baudrate is 115200 bps. Fig.10-6 provides an illustration of the control diagram.

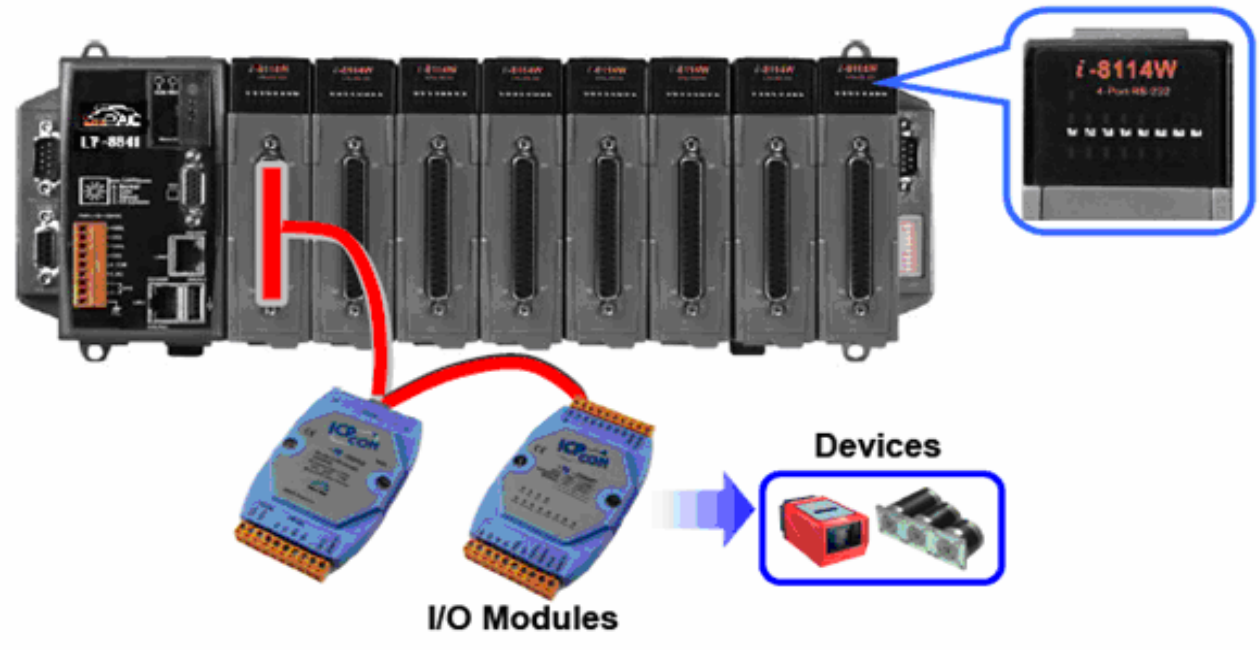
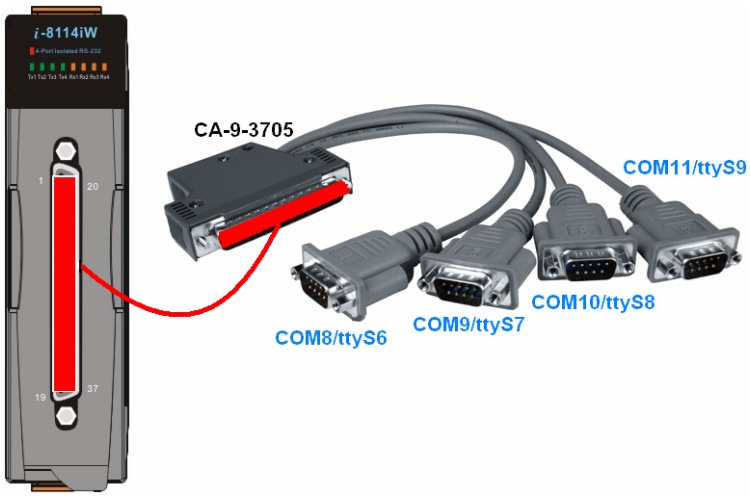


Fig.10-6

The result of executing this demo program is that the state of the DO channels can be controlled, and the program returns the state of the DI channels. The source code for the 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 on the I-8114W
      wRetVal = Open_Com(COM9, 115200, Data8Bit, NonParity, OneStopBit);

      if (wRetVal > 0) {
            printf("Failed to open port. \n");
            return (-1);
      }
```

```
    //   *****   7044 DO & DI Parameters *******
    wBuf[0] = 9;                    // COM Port
    wBuf[1] = 0x02;                 // Address
    wBuf[2] = 0x7044;               // ID
    wBuf[3] = 0;                    // Checksum disable
    wBuf[4] = 100;                  // Timeout , 100 milliseconds
    wBuf[6] = 0;                    // Debug string

  // 7044 DO
  while(j!=113) {
        printf("Enter the 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)          // There was an error with the Digital Output on the I-7044
            printf("Digital Output of 7044 is error, Error Code=%d\n", wRetVal);
        else
            printf("The DO value of 7044 is: %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;
}
```

For this example, the programming and execution procedures are the same as those described in Section 7.1. Fig. 10-7 below illustrates the result of the execution.

Fig. 10-7

10.2 Support for GUI Functionality

Now **'X-Window'** Utility is now supported on the LP-8x4x. After the LP-8x4x boots, a GUI environment similar to a standard **'Windows screen'** desktop will be displayed, as illustrated in Fig.10-7. This means that custom GUI applications can be created and then executed on the LP-8x4x. The GUI Library in the LP-8x4x is provided by the **GTK+** Library **(v1.2 and v2.0)**, which is a multi-platform toolkit for creating graphical user interface, allowing custom **SCADA** screens to be designed using the GTK+ Library on the LP-8x4x. For more information regarding the GTK+ project, refer to the GTK+ project website at http://www.gtk.org.

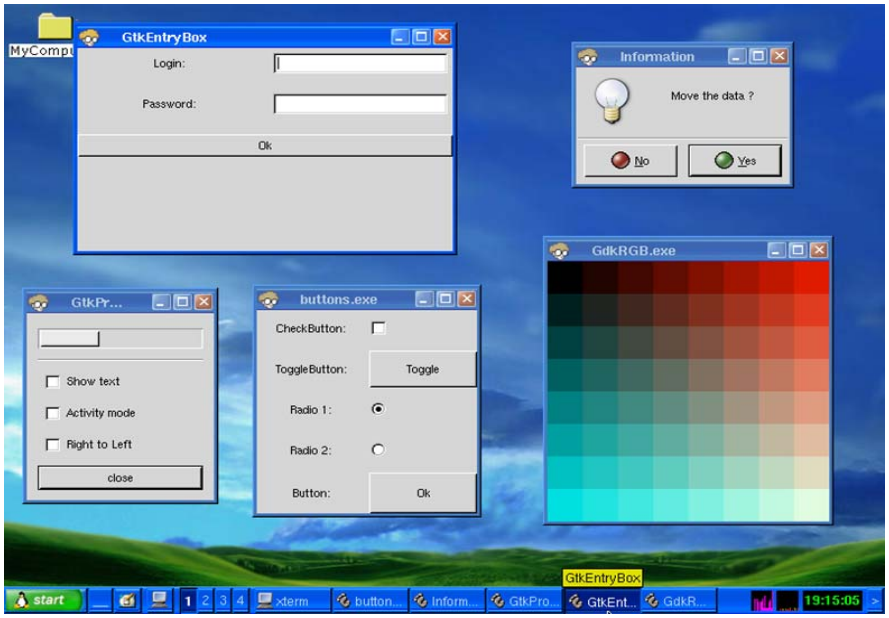


Fig. 10-7

ICP DAS provides a number of demo programs that illustrate how to use the GUI to control I/O modules and assist in quickly developing custom GUI programs, as illustrated in Fig. 10-8 below. Once the LP-8x4x SDK has been installed, these demo programs can be found in the **C:\cygwin\LinCon8k\examples\gui** folder.

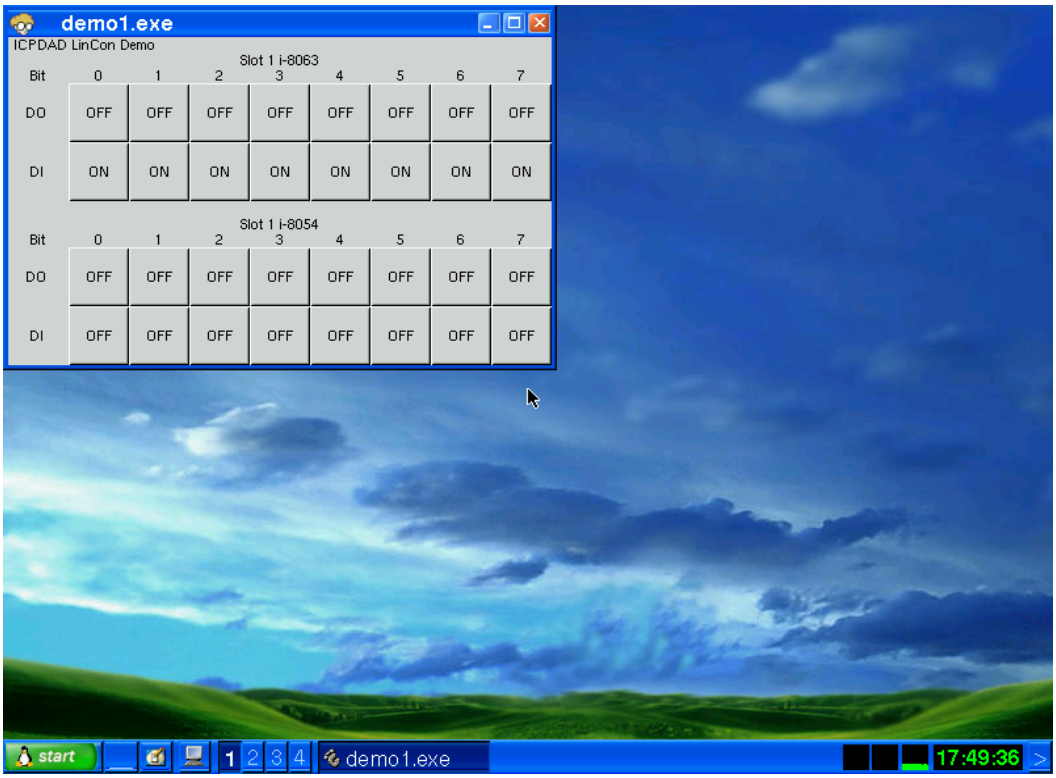


Fig. 10-8

In addition to the GTK+ GUI functionality, the **"Java GUI"** is also supported on the LP-8x4x. This means that developers familiar with the Java environment can also develop custom GUI applications. However, only the Awe and Swing v1.1 elements are supported on the LP-8x4x.

To execute the Stylepad.jar Java GUI program on the LP-8x4x, open an Xterm window by clicking the 'Start' button and choose 'Xterm'. At the Xterm window, enter **"java -jar Stylepad.jar -cp .:Stylepad.jar"**. Note that it may take some time to execute the Java GUI program.

10.2.1 Booting the LP-8x4x without loading the X-window environment

The LP-8x4x can be configurated to boot without loading the X-window environment by following the procedure described below:

(1) Enter the command "**cd /etc/rc2.d** " to switch to the default run level.

(2) Enter the command "**ls -al** " to list the S98Xserver link information into ../init.d/startx.

(3) To disable X-windows environment, enter the command " **mv S98Xserver xS98Xserver** " to rename the S98Xserver then exit the Xterm window and reboot the LP-8x4x to apply the new configuration.

10.2.2 Enabling the X-window environment to load at boot time

To enable the X-windows environment, enter the command "**ls -al /etc/rc2.d**" at the Command Prompt to view the S98Xserver link information ../init.d/startx, and then enter the command "**mv xS98Xserver S98Xserver**" to rename and enable the xS98Xserver. If the S98Xserver link is not listed, follow the procedure described below:

(1) Enter the command "**cd /etc/rc2.d** " to switch to the default run level.

(2) Enter the command "**ln -s ../init.d/startx /etc/rc2.d/S98Xserver**" to create a symbolic link into the X-window script file, which will enable the X-window environment and then exit the Command Prompt and reboot the LP-8x4x to apply the new configuration.

10.3 Support for ScreensShot functionality

The **"fbshot"** screenshot application is an embedded program that enables an image of the screen for the LP-8x4x to be conveniently captured, as illustrated in Fig. 10-9 below.

Open a Xterm window by clicking the Start button and then clicking Xterm. At the Command Prompt, enter the command **"fbshot -d /dev/fb0 /mnt/hda/catch1.png"** and an image of the current screen will be captured and saved to a file — catch1.png, and will be stored in the /mnt/hda/.directory, as illustrated in Fig. 10-9 below.

To view the image, enter the command **"vi /mnt/hda/catch1.png"**. Note that vi application is included in the path: /mnt/hda/opt/bin directory, so an microSD card must be inserted into the LinPAC before attempting to save the file. To view details of the parameters that can be used in conjunction with the **fbshot** application, enter the command "**fbshot --help**" at the Command Prompt.

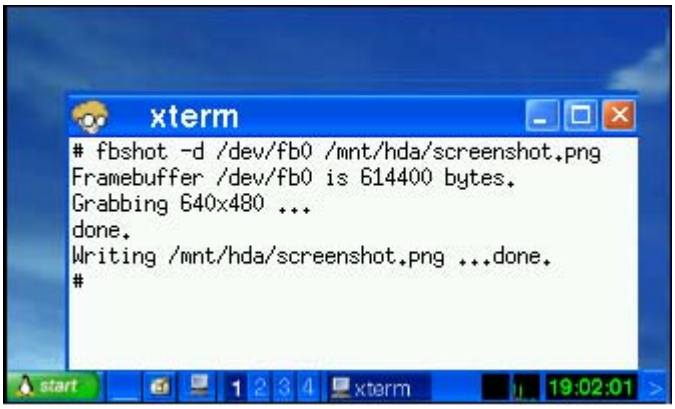


Fig. 10-9

10.4 Support for WebCAM functionality

The LP-8x4x embedded Controller provides support for WebCAM functionality. Logitech brand cameras have been tested and have been found to work successfully. If a different brand of camera is to used, testing should be performed first to ensure compatibility .

Follow the procedure described below to ensure that the Webcam is configured correctly:

(1) Connect the webcam to the LP-8x4x using the **"USB Interface"**.

(2) Reboot the LP-8x4x.

(3) Open a **"Xterm windows"** by clicking the Start button and then clicking Xterm. At the Command Prompt, enter the command **"insmod pwc.ko"** to load the gqcam program decompressor, as illustrated in Fig. 10-10 and then enter the command **"gqcam"** to view the webcam screen. To view details of the parameters that can be used in conjunction with the gqcam application, enter the command **"gqcam --help"** at the Command Prompt.

```
#
# 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

The gqcam program can also be used to capture an image via a webcam. To capture an image, follow the procedure described below:

(1) Click **"File/ Save Image…"**

(2) On the **"Gqcam: Save Image"** screen, enter the path for the folder where the image is to be store in the **"File Field"**, together with the file name, and then click the "**OK** " button.

10.5 Support for Touch Screen Devices

The LP-8x4x embedded Controller provides support for both USB and Serial Touch Screen devices, each of which is discussed in more detail below:

10.5.1 USB Touch Screen interface

Before a USB touch screen can be used, it must first be calibrated. There are six steps involved in adjusting the calibration for a touch screen connected to an LP-8x41 via the USB interface, as follows:

Step 1: Open a **"Xterm windows"** by clicking the Start button and then clicking Xterm. At the Command Prompt, ensure that the **usbtouchscreen.ko** and **tsdev.ko** files have been mounted, enter the command '**lsmod**' as illustrated in Fig. 10-11.

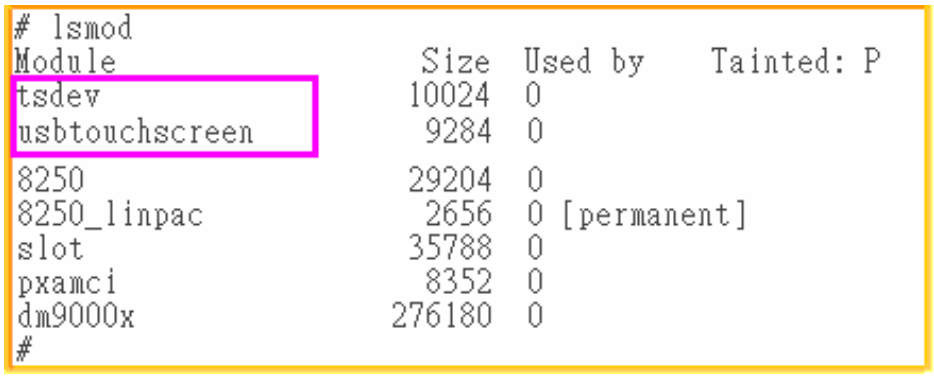


Fig. 10-11

Step 2: At the Command Prompt, ensure that a microSD card has been mounted, enter the command '**mount**' as illustrated in Fig. 10-12.

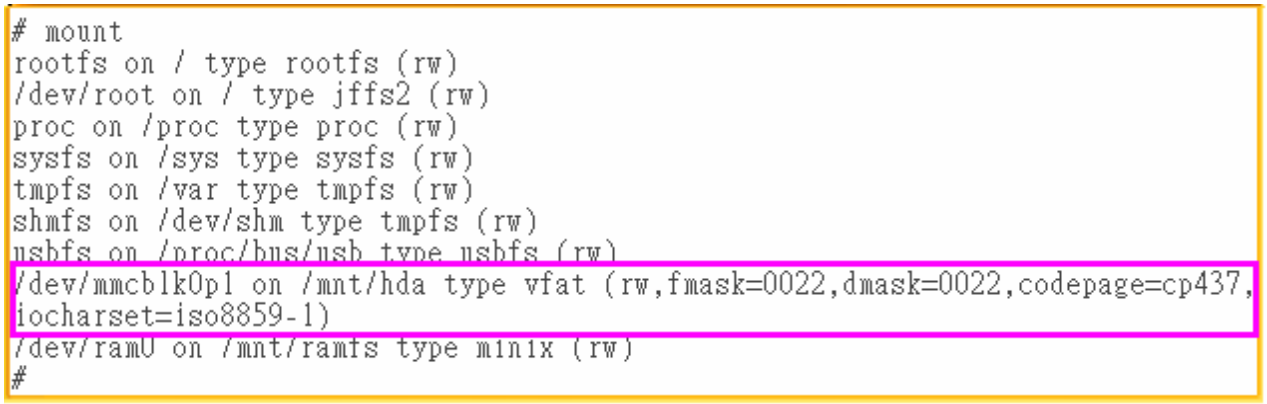


Fig. 10-12

Step 3: At the Command Prompt, edit the **/etc/init.d/fbman** file by modifying the settings so that they are the same as below:

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

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

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

  These lines indicate that the resolution is currently set to 640*480. The # character indicates that a setting is not currently being used.

- To change the resolution settings to **800*600**, remove the "#" character in line 2 and add the "#" character in line 1 as indicated below:

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

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

Step 4: At the Command Prompt, enter the command '**cat /proc/bus/input/devices**' to view a list of devices that are currently connected and the associated device can be obtained as illustrated in 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-1.1/input0
S: Sysfs=/class/input/input4
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-1.1/input1
S: Sysfs=/class/input/input5
H: Handlers=kbd event1
B: EV=3
B: KEY=39fa d801d101 1e0000 0 0 0

I: Bus=0003 Vendor=14e1 Product=6000 Version=a4b4
N: Name="DIALOGUE INC PenMount USB"
P: Phys=usb-pxa27x-1.2/input0
S: Sysfs=/class/input/input6
H: Handlers=event2
B: EV=b
B: KEY=70000 0 0 0 0 0 0 0 0
B: ABS=3

I: Bus=0003 Vendor=15d9 Product=0a33 Version=0100
N: Name="USB Mouse"
P: Phys=usb-pxa27x-1.3/input0
S: Sysfs=/class/input/input7
H: Handlers=mouse0 event3 ts0
B: EV=7
B: KEY=70000 0 0 0 0 0 0 0 0
B: REL=103
#
```

Fig. 10-13

Step 5: We are providing the calibration program to test and get the calibration data. For example, open an '**Xterm** windows' and execute the command '**calibrator /dev/input/event2**', and then the calibration windows displayed as illustrated in Fig. 10-14)

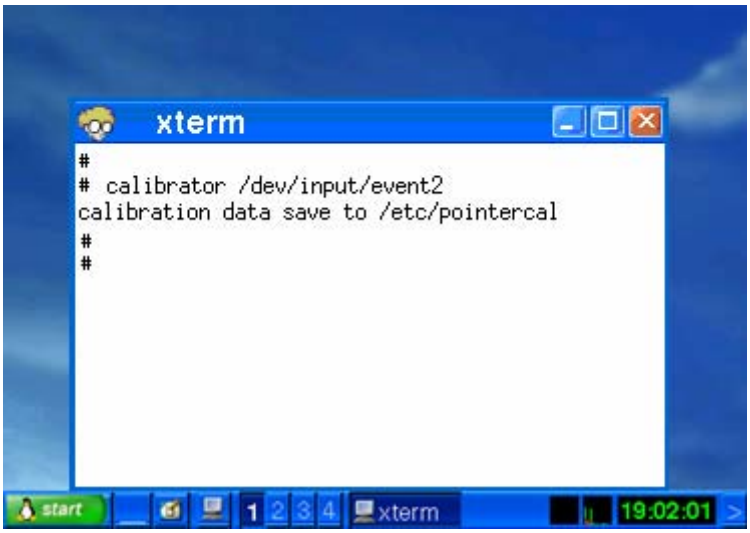


Fig. 10-14

Step 6: Rebooting the LP-8x4x to apply the new configuration.

10.5.2 Serial Touch Screen interface

There are three kinds of Touch Screen LCD monitor, so the relevant driver needs to be installed before it can be used. An overview of the respective device drives and the installation location is provided below:

Module	Loadable kernel module to Install
ADP-1080T	/lib/modules/2.6.19/**pm9000.ko**
TPM-4100 / TP-4100 / TP-6150 / TP-2070	/lib/modules/2.6.19/**pm6000.ko**

Before a Serial touch screen device an be used, it must first be calibrated. There are nine steps involved in adjusting the calibration for a touch screen calibratied to an LP-8x41 via the serial interface, as follows:

Step 1: Open a **"Xterm windows"** by clicking the Start button and then clicking Xterm. At the Command Prompt, enter the command '**cat /etc/init.d/penmount_serial**' to check that the penmount serial driver has been mounted from **/etc/init.d/penmount_serial**, as illustrated in Fig. 10-15.

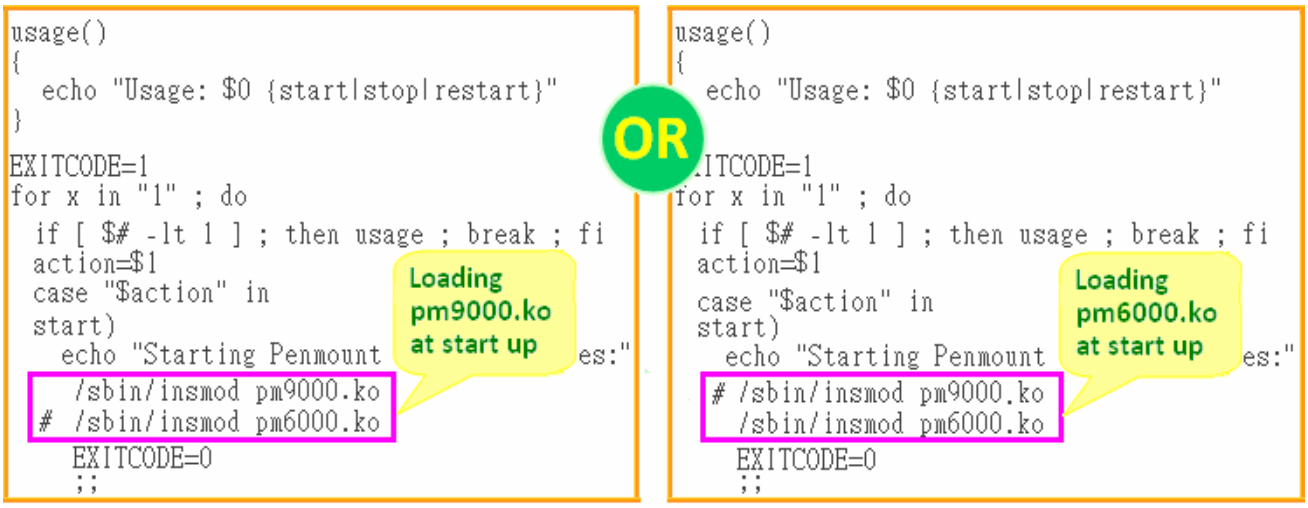


Fig. 10-15

Step 2: Edit the **/etc/init.d/tsdev_serial** script to modify the device mode. By default, serial interface is COM4 port, and device mode is ttyS34, as illustrated in Fig. 10-16.

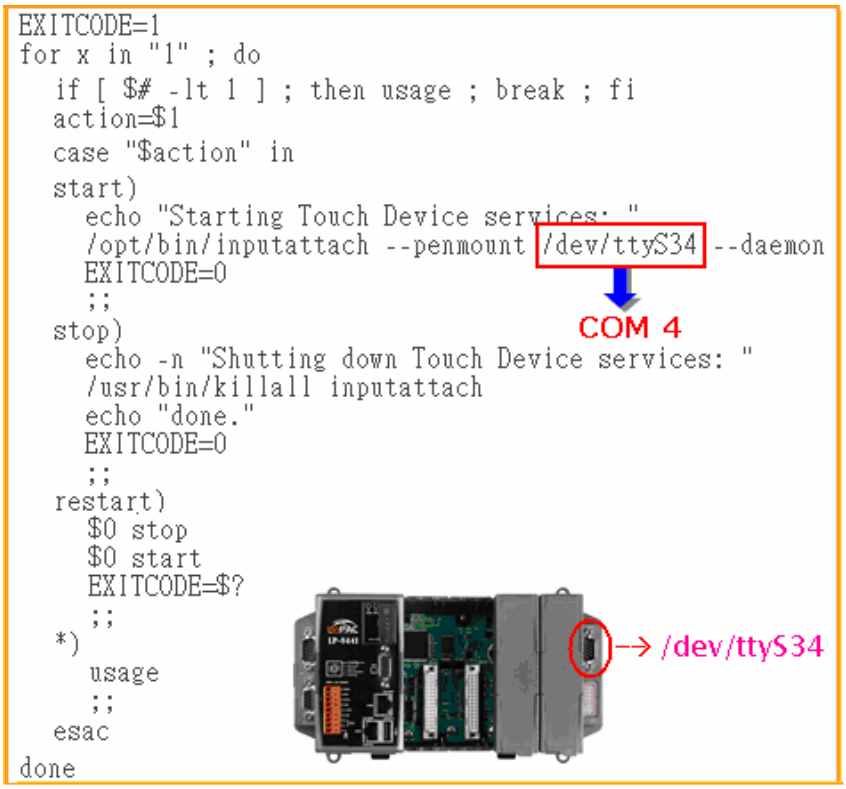


Fig. 10-16

Step 3: Configure the script to be executed at startup and shutdown.

- By default, the scripts for serial touch screens are disabled at startup. The '**mv**' command can be used to rename the files in **/etc/rc2.d**, which is the file containing instructions used to start processes. The file will be automatically executed when LP-8x4x is rebooted, as illustrated in Fig. 10-17.

```
# 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. 10-17

Step 4: At the Command Prompt, enter the command '**lsmod**' to check that the **pm9000.ko** or **pm6000.ko** have been mounted, as illustrated in Fig. 10-18.

```
# 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
#
```

```
# lsmod
Module          Size  Used by    Tainted: PF
pm6000          2912  0
8250           29204  2
8250_linpac     2656  0 [permanent]
slot           35788  0
pxamci          8352  0
dm9000x       276180  0
#
```

Fig. 10-18

Step 5: At the Command Prompt, enter the command '**mount | grep mmc**' and '**ls /mmt/had**' to check the microSD card has been mounted, as illustrated in Fig. 10-19 and 10-20.

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

Fig. 10-19

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

Fig. 10-20

Step 6: At the Command Prompt, enter the command '**vi /etc/init.d/fbman**' to edit the **/etc/init.d/fbman** file by modifying the setting so that that are the sam as below:

- After opening the file locate the following lines:

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

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

  These lines indicate that the resolution is currently set to 640*480. The # character indicates that a setting is not currently being used.

- To change the resolution setting to be **640*480**, remove the "#" character in line 1 and add it to line 2, as indicated below:

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

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

Step 7: At the Command Prompt, enter the command '**cat /proc/bus/input/devices**' to view a list of devices that are currently connected and the associated device can be obtained, as illustrated in Fig. 10-21.

```
# 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
B: ABS=3

#
```

Fig. 10-21

Step 8: We are providing the calibration program to test and get the calibration data, as illustrated in Fig. 10-22. For example, open an '**Xterm**' windows and execute the command '**calibrator /dev/input/event3**', and then the calibration windows will be displayed, correct 4 point locations on screen with the panel, as illustrated in Fig. 10-23.

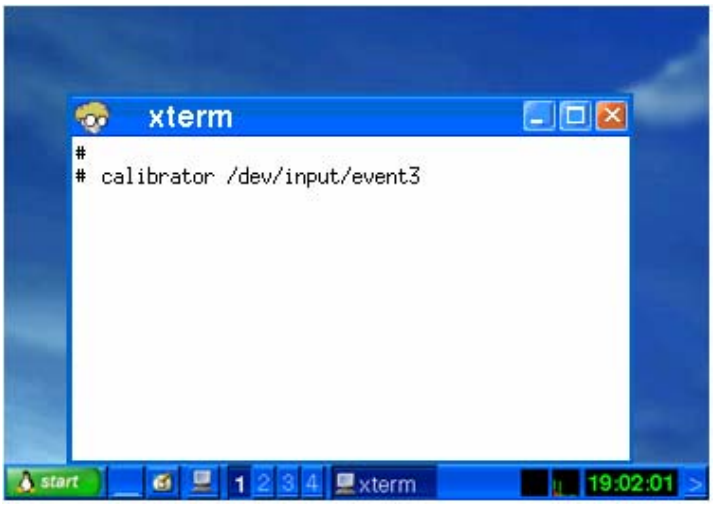


Fig. 10-22

Fig. 10-23

Step 9: Reboot the LP-8x4x to apply the new configuration.

10.6 Network Support

The LP-8x4x embedded controller already includes a variety of network functions. The following is an overview of the network functions supported in the LP-8x4x:

(1) UPnP

UPnP refers to **"Universal Plug and Play"** which is a set of networking protocols that allows other devices to automaticlly discover and control of services available on a network, without the need for user intervention. Devices that act as a servers can advertise their services to clients. Client systems, known as control points, are able to search for devices that provide specific services on the network. When a device that provides the desired service is discovered, the Control Points is able to retrieve a detailed descriptions of the devices and services allowing the server and client devices to interact from that point on.

(2) VPN

VPN refers to **“Virtual Private Network”** and is used to securely extend a private network across a public network, as illustrated in Fig. 10-24. VPN describes a network that includes secure remote access for client devices, enabling data to be securely sent and received across shared or public networks as if the device was directly connected to the private network.

The term **“Virtual”** refers to the fact that the devices on the network do not need to be physically connected.

The term **“Private”** implies that the data is encrypted and can only be viewed by a defined group connected to the VPN, meaning that it is extremely difficult for unauthorized user to access confidential information.

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.

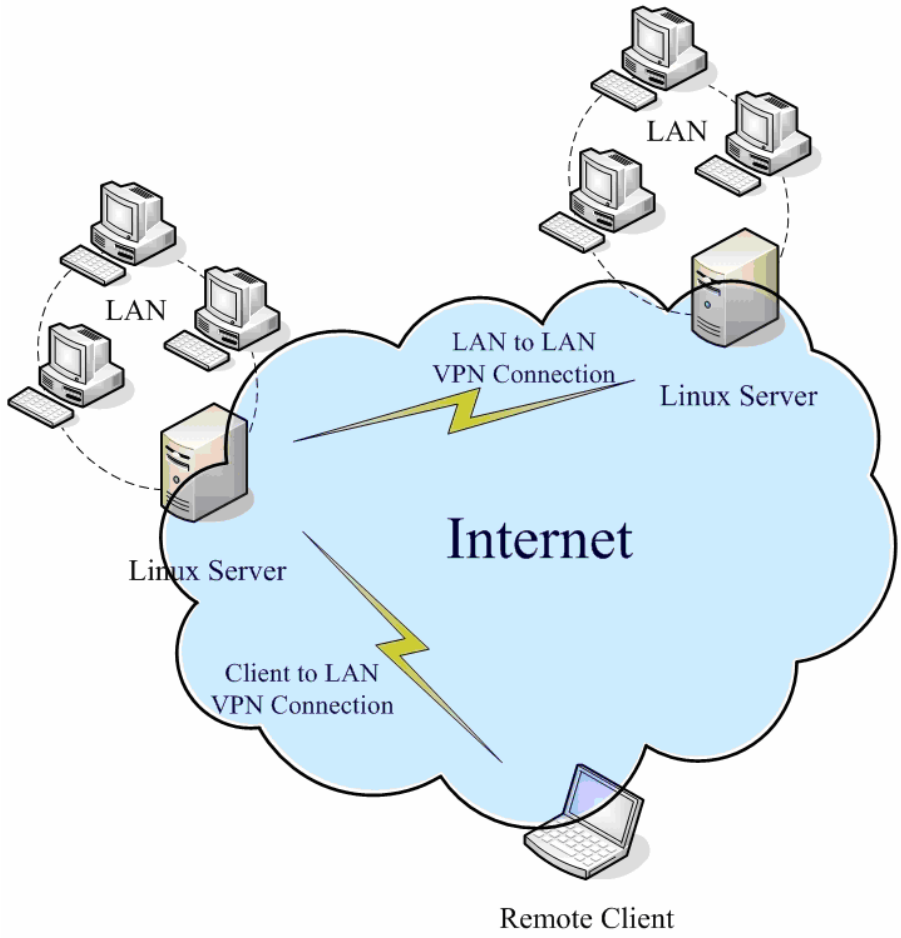
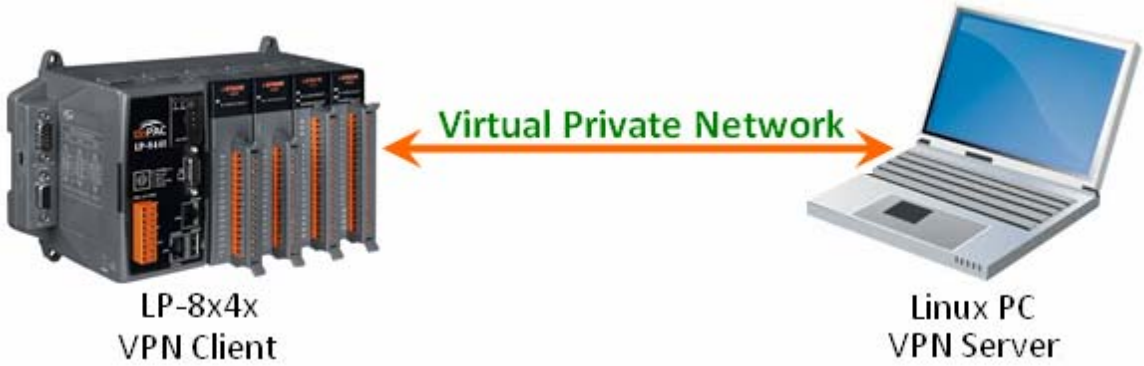


Fig. 10-24

The following is an explanation of the process used to set up and configure a VPN server and client for both the server side and the client side.



[VPN Server Side Configuration: Linux PC]

To configure the server side of the VPN, follow the procedure described below.

Step 1: Install PPTP package. At the command prompt, enter the command '**apt-get install pptpd**' to install the PPTP package file.

Step 2: At the command prompt, enter the command **'vi /etc/pptpd.conf'**, and comment out the following lines:

```
localip 192.168.0.1
remoteip 192.168.0.234-238,192.168.0.245
# or
#localip 192.168.0.234-238,192.168.0.245
#remoteip 192.168.1.234-238,192.168.1.245
```

Fig. 10-25

Step 3: At the command prompt, enter the command **'vi /etc/ppp/pptpd-options'** to ensure that the following lines are set:

```
lock
debug
name vpn
refuse-pap
refuse-chap
refuse-mschap
rrequire-mschap-v2
#require-mppe-128   ——> need to remarked
proxyarp
noccp               ——> append to the end of the file
```

Fig. 10-26

Step 4: At the command prompt, enter the command '**vi /ppp/chap-secrets**' and enter the username and password for the VPN.

```
# Secrets for authentication using CHAP
# client      server     secret      IP addresses
lp8x4x * test *
```

Fig. 10-27

Step 5: Restart the pptpd services by entering the command '**sudo /etc/init.d/pptpd restart**' (or sudo service restart pptpd).

Step 6: Confirm the VPN server is active and is listening for client connections:

```
root@LinuxPC-ICPDAS:/home# netstat  -an |grep -r 1723
tcp       0      0 0.0.0.0:1723      0.0.0.0:*       LISTEN

root@LinuxPC-ICPDAS:/home#
```

Fig. 10-28

[VPN Client Side Configuration: LP-8x4x]

To configure the Client side of the VPN, follow the procedure described below.

Step 1: At Command Prompt, enter the command '**vi /etc/ppp/peers/vpn**' and modify the contents according to the syntax below:

```
#There are VPN server configuration.
#LP-8x4x as VPN client, connect to VPN server.
pty "pptp 10.1.0.74 --nolaunchpppd"
name lp8x4x
password test
logfd 2
nodetach
remotename pptpd
```

Fig. 10-29

Note: The address "**10.1.0.74**" indicated in the figure is the IP address of the VPN server.

Step 2: Create a VPN connection by entering the '**pppd call vpn &**' command at the command Prompt, and verify that the details are correct, as illustrated in Fig 10-30.

```
Using interface ppp0
Connect: ppp0 <--> /dev/ttyp1
local  IP address 192.168.0.234
remote IP address 192.168.0.1
```

Fig. 10-30

Step 3: At the Command Prompt, enter the '**ifconfig**' command to verify the status of the VPN interface, as illustrated in Fig. 10-31.

```
# ifconfig ppp0
ppp0 Link encap:Point-to-Point Protocol
     inet addr:192.168.0.234  P-t-P:192.168.0.1  Mask:255.255.255.255
     UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
     RX packets:3 errors:0 dropped:0 overruns:0 frame:0
     TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:3
     RX bytes:42 (42.0 B)  TX bytes:63 (63.0 B)
```

Fig. 10-31

Step 4: At the Command Prompt, enter the command "ping 192.168.0.1" to ping the VPN host. To stop the tunnel, enter the '**poff**' command at the Command Prompt.

(3) QoS

QoS refers to **"Quality of Service"** and describes the overall performance of a network, particularly that send by users of the network. QoS is based on a set of techniques that are used to manage network resources. For example, if there are a number of packets that need to be sent over a network device, the device has to first determine a transmission priority, i.e., which packets to send first, which ones to delay, and, if necessary, which ones to drop. The transmission is then performed based on that priority, thereby achieving the required QoS by meanaging the delay, jitter, bandwidth, and packet loss parameters on the network. Using the Linux QoS subsystem, it is possible to create a higly flexible traffic control system that ensures the flow rate to an assigned port can be effectively controlled, improving network quality in the process.

(4) Wireless LAN (WLAN)

A **"Wireless Local Area Network (WLAN)"** is a network technology that allows the connection of two or more devices without requiring the installation of any wires or cables, mostly using **radio** technology, and sometimes **infrared**. The range of the WLAN is targeted on a limited area, generally within an office building, a commercial area, a small campus, or a home, etc. As technology has become more prevalent, Linux has adopted many of the technologies and tools that take advantage of wireless networking.

If a wireless card is inserted into a slot on the LP-8x4x, the parameters contained in the '**/etc/network/interfaces**' file need to be modify.

(5) Dual LAN

The Dual LAN functionality provided on the LP-8x4x allows a wireless network and a cable network to be combined through the LP-8x4x, meaning that it is possible to establish communication between the cabled LAN and the wireless LAN. This ensures that as long as either of the two LANs can connect to the Internet, then all devices connected to the network will be able to asscess the Internet as illustrated in Fig. 10-32.

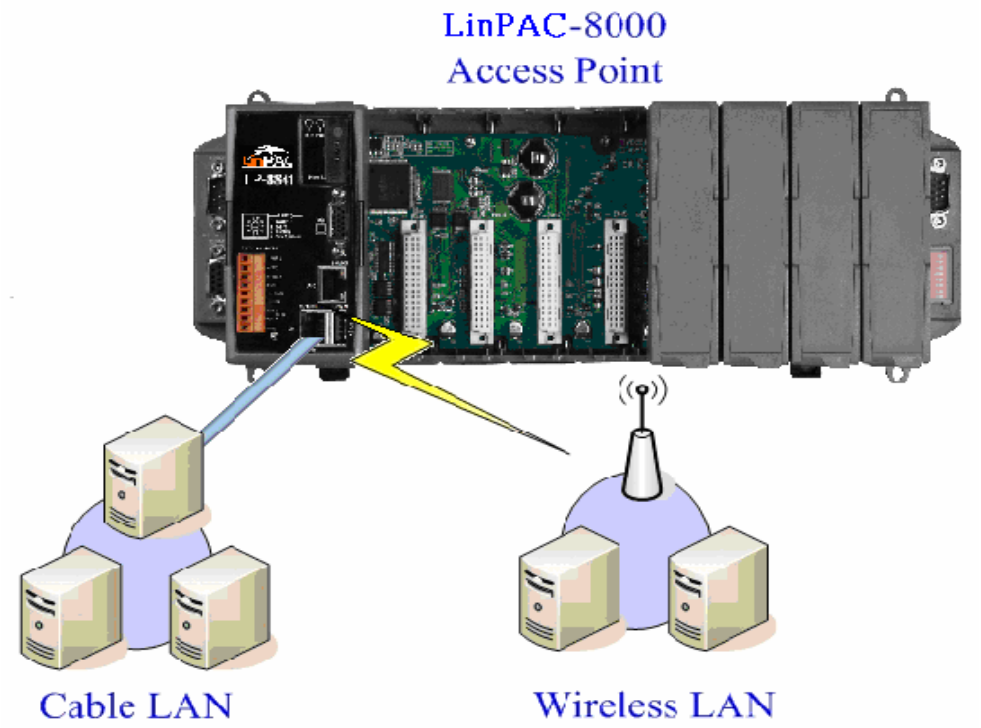


Fig. 10-32

(6) Bluetooth

Bluetooth is a worldwide specification for a small-form factor, low-cost wireless technology that is used for exchanging data over short distances between both fixed and mobile computers, mobile phones, and other portable handheld devices, as well as providing connectivity to the Internet. **"BlueZ"** is a Bluetooth stack designed for Linux operating system that is now embedded in the LP-8x4x, providing support for the core Bluetooth layers and protocols. Bluetooth technology is flexible and efficient, and based on a modular implementation.

(7) Modem / GPRS / ADSL Connectivity

LP-8x4x can be connected to the Internet using one of three modes, "Modem", "GPRS" in combination with an external RS-232/USB interface or "ADSL". The setup and configuration method for each is described below:

[Modem]

[GPRS]

The following is a description of the procedure for configuring the GPRS modem, an GTM-201-RS232 GPRS Modem for example.

❑ Part 1

In order to connect a GPRS modem to the COM3 port on an LP-8x4x, the **/etc/ppp/peers/wavecom** file must first be modified to define the COM port. Connect the GTM-201-RS232 (GPRS Modem) using an RS-232 interface by follow the instructions below:

(1) Open a **"Xterm windows"** by clicking the Start button and then clicking Xterm. At the Command Prompt, enter the command **"vi /etc/ppp/peers/wavecom"** to edit the file.

(2) Locate the statement "Serial device to which the GPRS phone is connected:", and add the device name for the COM port, as illustrated in Fig. 10-33.

(3) At the Command Prompt, enter the command "**:wq** " to save the changes and close the script.

Note: In order to provide support for 2G GPRS Modems the — **ftdi_sio.ko** loadable kernel module must be installed using the **insmod** command.

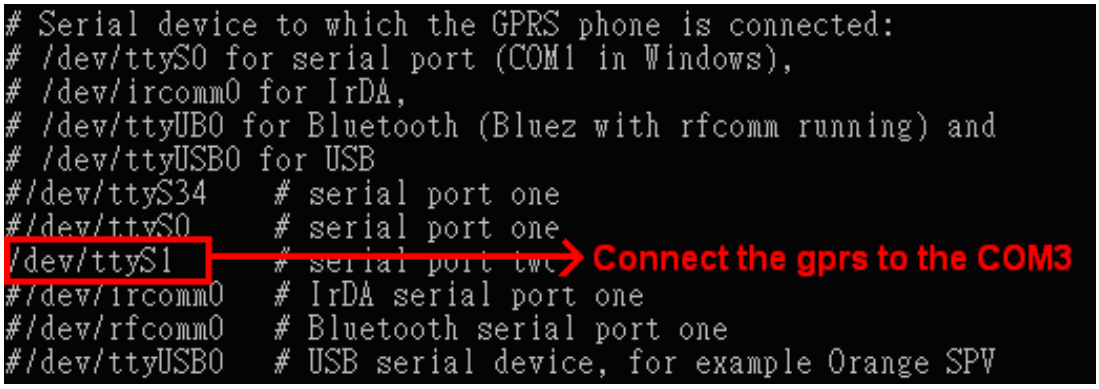


Fig. 10-33

❑ Part 2

The default baud rate for GPRS chip is "**115200**" bps. Consequently, both the GPRS module and the device node, such as /dev/ttyS2, should be configured to use the same baudrate. User the '**stty**' command to set the input and output sepped of the device node, as illustrated in Fig. 10-34.

```
# 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. 10-34

Before starting the GPRS modem, the network interface for both eth0 and eth1 msut first be deactivated. Remove the Ethernet cable, and then enter the command "**ifdown eth0**" and "**ifdown eth1**" at the Command Prompt to deactivate the interfaces.

At the Command Prompt, enter the command "**pppd call wavecom &**". The LP-8x4x will then be automatically connected to the internet. It should be remembered that the network interface for the LinPAC device must be deactivated first. Enter the command "**ifconfig**" at the Command Prompt to display the "**ppp0**" section, as illustrated in Fig. 10-35.

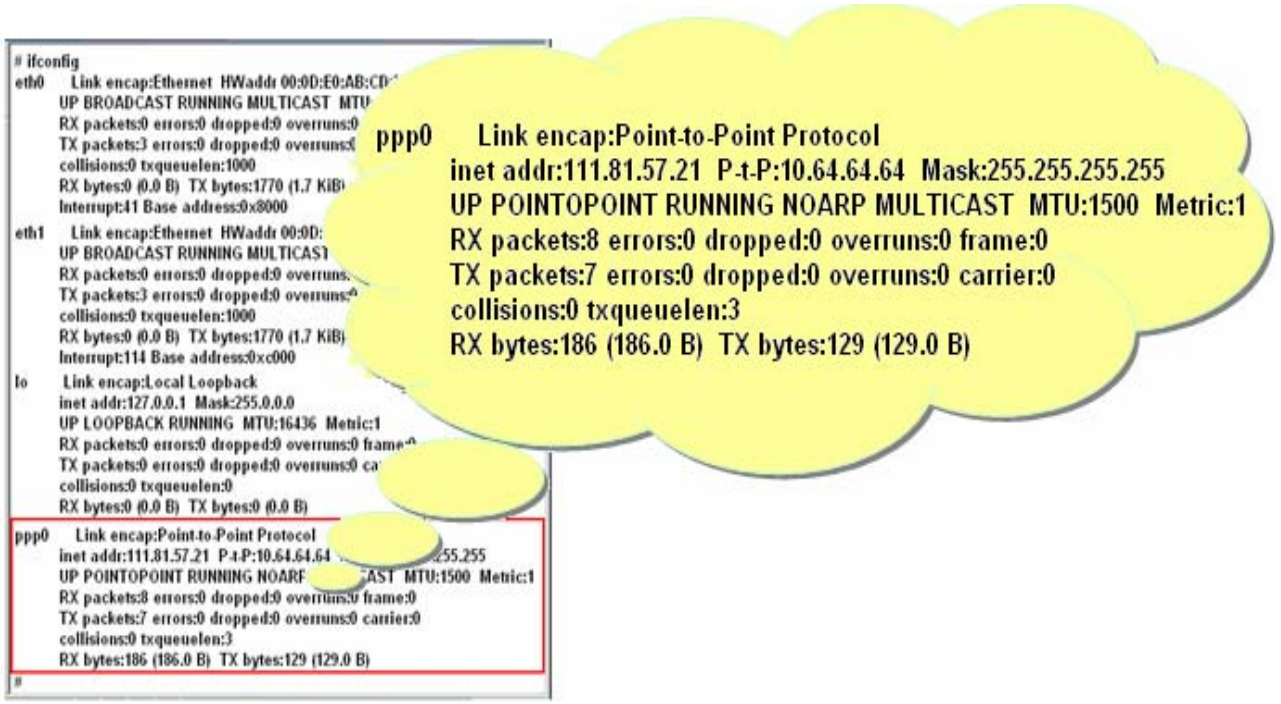
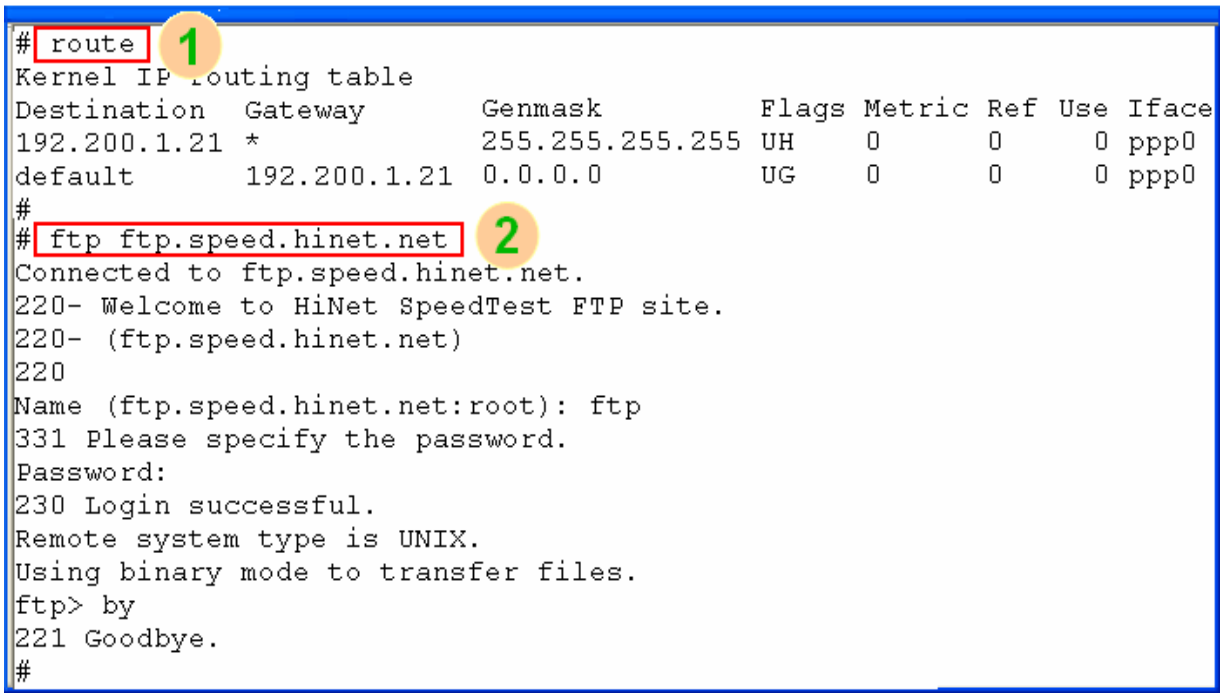


Fig. 10-35

Fig. 10-36 below provides an example of a routing table.




```
# 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. 10-36

GPRS modem selection guide:

Module	Loadable kernel module to Install	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, refer to : http://m2m.icpdas.com/m2m_layer2_gprs.html

Note: To perform the same operations using an alternative modem, such as the GTM-201-USB, a different loadable kernel module must first be installed. At the Command Prompt, enter the command “insmod usbserial.ko”, and then enter the command “insmod sim5218.ko” to load the program decompressor.

[ADSL]

To use an ADSL modem, the ADSL option musbe be configured first. To do this, enter the command **“adsl-setup”** at the Command Prompt, then enter the command **“adsl-connect ”** to establish an Internet connection on the LP-8x4x. To disable the ADSL connection, enter the command **“adsl-stop”** at the Command Prompt.

(8) Firewall (iptables function)

A firewall is used to control unauthorized access to a local network, even against an intentional attack from an external network, locking out intruders and ensuring that both the system, and the data is safe

(9) Web Browser

The LP-8x4x contains an embedded Web Browser that can be used to access the Internet. Open a **"Dillo windows"** by click the "Start" > "Programs" button and then clicking 'Dillo'. At the Command Prompt to open the web browser and then enter the address of a web site, as illustrated in Fig. 10-37. Note that the dillo command is located in the path: **/mnt/hda/opt/bin,** so a microSD card must first be inserted into the SD socket on the LinPAC .

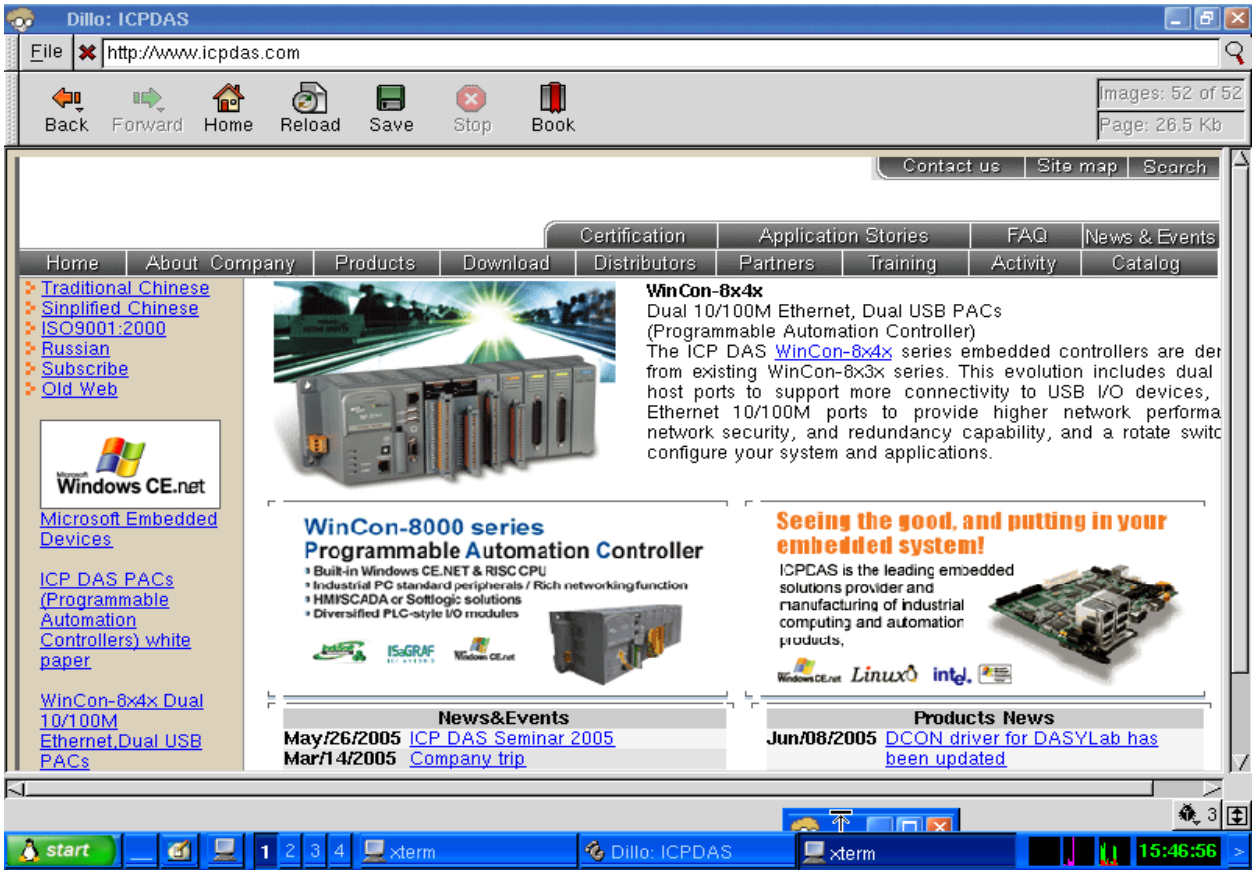


Fig. 10-37

(10) Apache Server

The LP-8x4x contains an embedded **"Apache" Web Server**, which will be automatically loaded when the module is booted up. The files for the server can be found in the **/opt/apache2** directory. To connect to the web server embedded on the LP-8x4x, enter the URL **"http://192.168.0.200"**. If a web page is successfully displayed, it means that the web page on the LP-8x4x has been started. The index page for the Apache Server can be found in the "**/opt/apache2/htdocs/** " directory.

The files that provide the full functions of the Apache Server are located on the microSD card. This means it other Apache Server function need to be used that are not supported on the LP-8x4x, this files can be copies from the microSD card to the **/opt/apache2** directory from the microSD card. The new or additional functions will then be available once the LP-8x4x is rebooted.

10.7 Support for USB to RS-232 Conversion

The LP-8x4x provides support for USB to RS-232 converter modules, such as the I-7560 for example. The I-7560 module contains a Windows serial COM port that is implemented via its USB connection and is compatible with both 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. For more details related to the I-7560 module, refer to http://www.icpdas.com/products/Remote_IO/i-7000/i-7560.htm.
Follow the procedure described below to ensure the successful operation of the USB to RS-232 converter:

(1) Connect the I-7560 to the LP-8x4x using a **"USB Interface"**, as illustrated in Fig. 10-38.

Fig 10-38

(2) Power on the LP-8x4x.

(3) Open a **"Xterm windows"** by clicking the Start button and then clicking Xterm. At the Command Prompt, enter the command Open a **"Command Prompt"** by clicking **"insmod usbserial.ko"**, and then enter the command **"insmod pl2303.ko"** to load the program decompressor, as illustrated in Fig. 10-39 below.

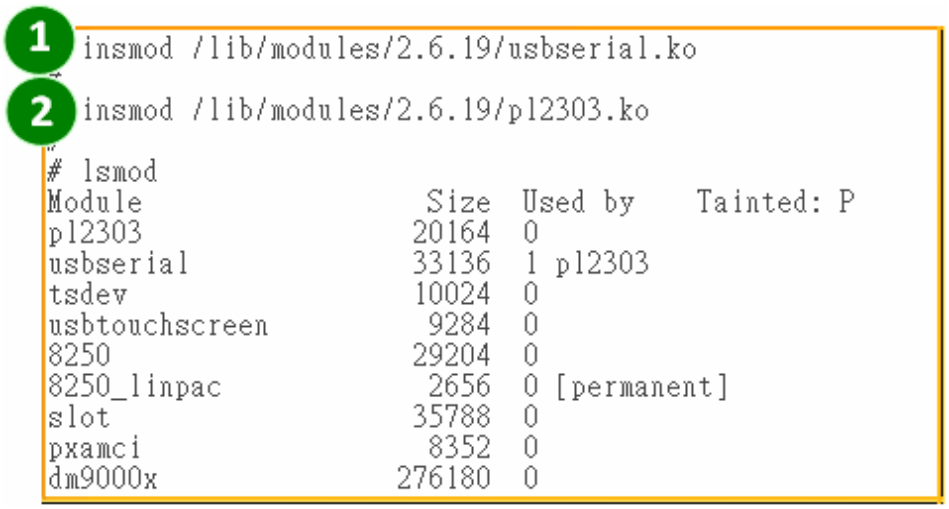


Fig. 10-39

(4) After successfully executing the **insmod** command, a new **/dev/ttyUSB0** serial device will be created. Use the "**echo**" and "**cat**" commands to send and receive messages, as illustrated in Fig. 10-40 below.

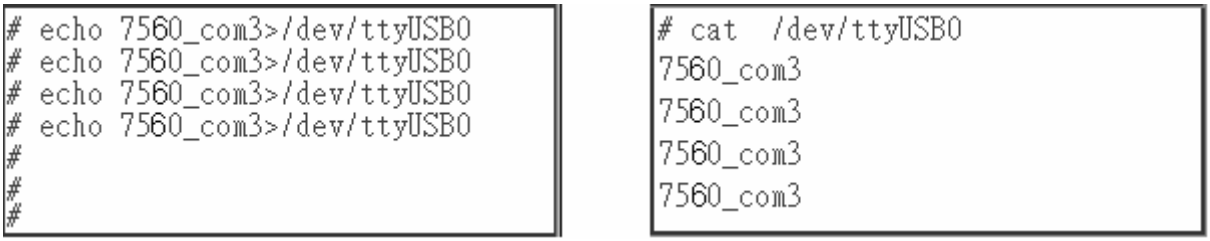


Fig. 10-40

10.8 Additional Optional Functions

The LP-8x4x provides support for a number of additional functions, a description of which is listed below. To activate any of these functions so that they can be used in combination with the LP-8x4x, copy the relevant directory to the "**opt**" directory on the microSD card. After rebooting LP-8x4x, the new function will be loaded automatically.

(1) MySQL

MySQL is a small open source "Relational DataBase Management System" RDBMS that provides support for a wide range of platforms, including UNIX, Linux or Windows, allowing data to be easily added or deleted.

☑ Start the MySQL service

To install MySQL for use in combination with the LP-8x4x, check the "**mysql**" directory in the /opt directory of microSD card, and then choose one of the following installation methods:

At the Command Prompt, enter the following commands:	
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. 10-41

```
# 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. 10-42

☑ Compile a mysql demo program

Follow the procedure described below to compile a MySQL demo program using the LinPAC SDK:

1) Copy the mysql directory from the ***/opt*** directory on the microSD card to **C:\cygwin\opt** , as illustrated in Fig. 10-43.

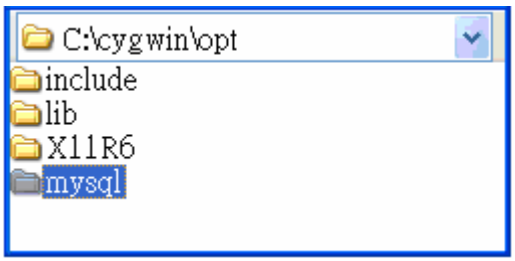


Fig. 10-43

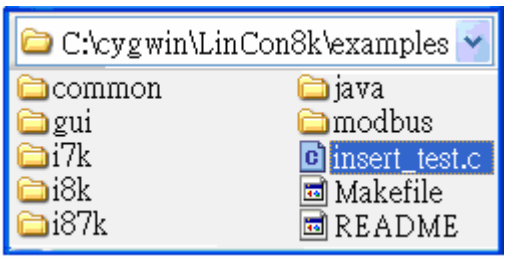


Fig. 10-44

2) Coding a demo program in the C:\cygwin\LinCon8k\examples directory, as illustrated in Fig. 10-34.

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

4) Compile the demo program, as illustrated in Fig. 10-45:

At the Command Prompt, enter the following:

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. 10-45

(2) PHP

PHP is a server-side open source scripting language that can be used to design dynamic web pages. When PHP is implemented in combination with MySQL, the resulting applications are cross-platform, which means that applications can be developed on a Windows-based system and served on a Linux platform, as illustrated in Fig 10-46 below.

PHP functionality has been embedded into the kernel on the LP-8x4x, meaning that PHP can be used on the LP-8x4x directly after booting the device.

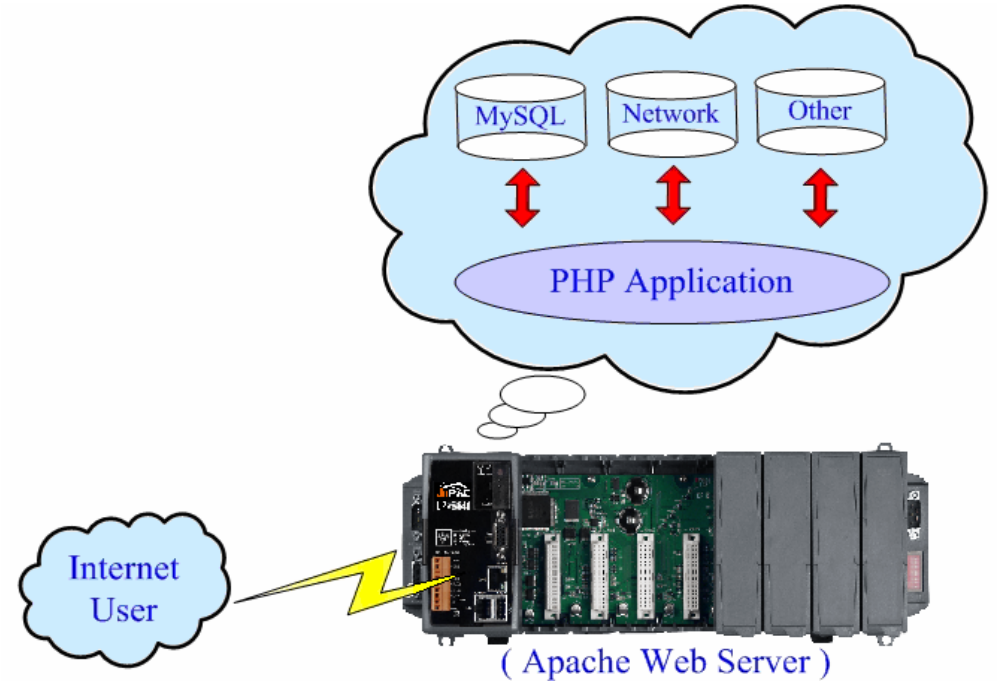


Fig 10-46

(3) Perl Support

Perl（Practical Extraction and Report Language）is also an “open source scripting language“ and has been embedded into the kernel on the LP-8x4x, meaning that Perl can be used on the LP-8x4x directly after booting 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

1. 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 :

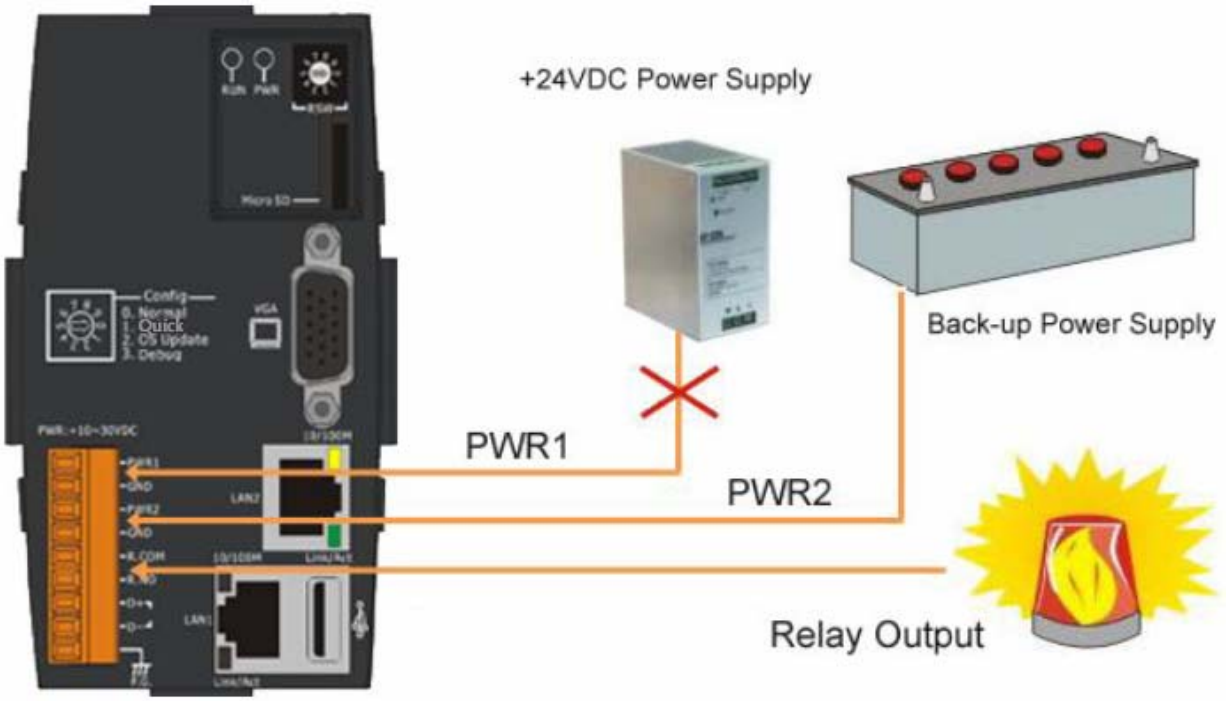
service@icpdas.com

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 website for Java 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.
V1.9	2014.01	1. Add VPN usage 2. Add pm6000.ko for Touch LCD monitor
V1.10	2016.01	1. Add remark for flash, microSD disk and battery. 2. Modify demo of AnalogInAll_87k().