Software Guide

ICP DAS LX-8X31 Series LinPAC

Implement industry control with Linux Technique

Contents

1. Introduction

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

The Linux OS demands less system resources from the embedded controller and is therefore the best fit for it because of the embedded controller has some limitations in system resources. It is for this reason that the LX-8X31 embedded controller has been published to be a new generation product from ICP DAS and the Embedded-Linux OS has been adopted into the LX-8X31. The main purpose of LX-8X31 is to allow the numerous enthusiastic Linux users to control their own embedded systems easily within the Linux Environment.

ICP DAS provides the library file - libi8k-r3600.a which includes all the functions from the I-7000/8000/87000 series modules which are used in the LX-8X31 Embedded Controller. The libi8k-r3600.a is designed especially for the I-7000/8000/87000 series modules on the Linux platform for use in the LX-8X31. Users can easily develop applications in the LX-8X31 by using either C Language In the future. The various functions of the libi8k-r3600.a are divided into the sub-group functions for ease of use within the different applications. The powerful functions of the LX-8X31 embedded controller, which includes a **VGA**, **USB** **(Card Reader,**

Camera …), <u>Mouse</u>, <u>Audio</u>, <u>Keyboard</u>, <u>Serial ports</u> (RS-232, RS-422/485), <u>Ethernet</u> (Hub…) and <u>many I/O slots</u> in the picture. Presently, HTTP、SSH Servers are built in and users can transfer files or use remote control with the LX-8X31 more conveniently In network communication. Fig. 1-1 illustrates hardware architecture of the LX-8X31.

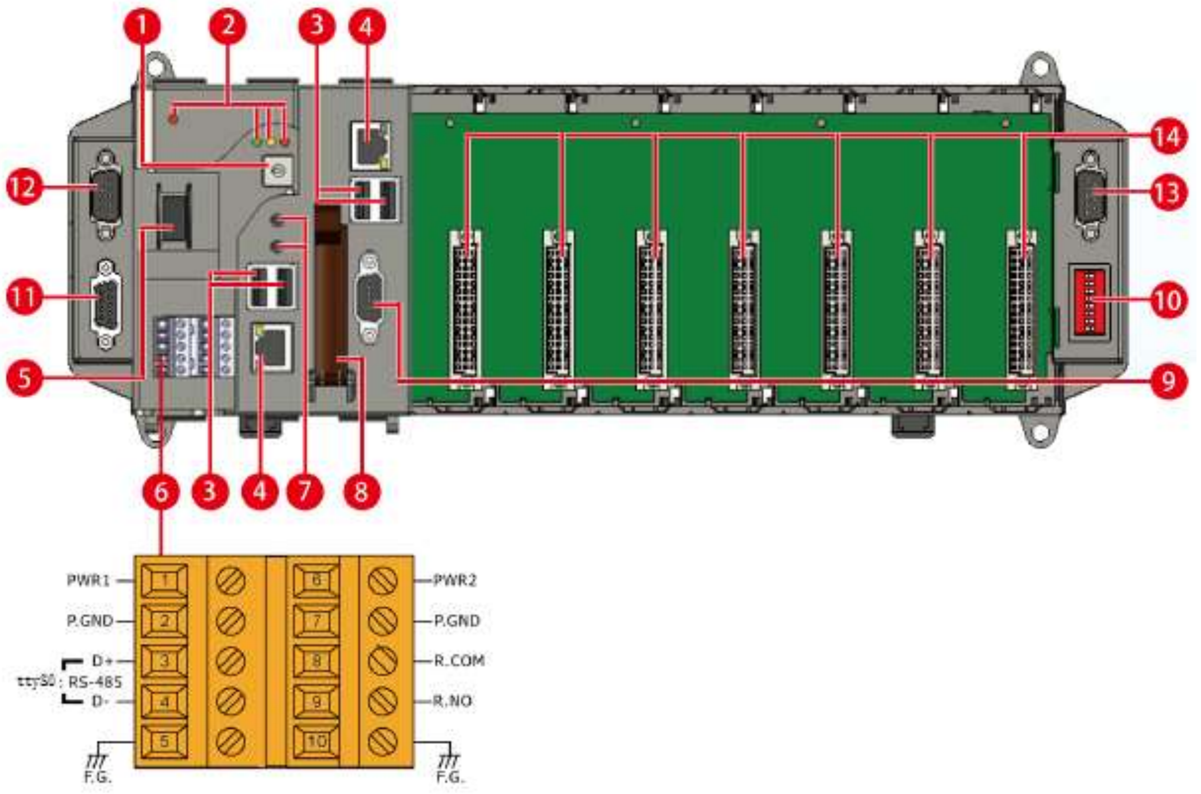


Fig. 1-1

1. Rotary Switch

The Rotary Switch is an operating mode selector switch which provides functions to configure with the selection of operating mode and authorization control.

2. LED Indicators

The LX-8X31 contains PWR LED indicators that located near power switch and show the power status.

The LX-8X31 contains four LED indicators. The first is labeled PWR, located near the power switch and show the power status. The three other are located next the rotary switch, denoted RUN and L1 and L2 and used for user defined.

LED Indicators	**Color**	**Meaning**
PWR	Red	Power is on
RUN	Green	User programmable LED
L1	Yellow	User programmable LED
L2	Red	User programmable LED

3. USB Ports

The LX-8X31 contains four USB ports that allow support for the USB devices such as mouse, keyboard or an external USB hard drive. These ports are denoted P1, P2, P3, P4.

4. Ethernet Ports

The LX-8X31 contains two Ethernet ports for use with network devices, and is denoted as LAN1 and LAN2.

5. Power Switch

The power switch is a small switch that enables or disables power to electric circuits and loads in the LX-8X31.

6. Connector

The connector has 10 pins arranged in 2 rows, as follows:
The pin assignments of the connector are as follows:

Pin	**Signal**	**Description**
1	PWR1	Power input 1
2	P.GND	

3	D+	ttyS0:RS485
4	D-	
5	F.G.	Frame ground
6	PWR2	Power input 2
7	P.GND	
8	R.COM	Relay Output
9	R.NO	
10	F.G.	Frame ground

7. Microphone and Earphone

The LX-8X31 contains the microphone and earphone jack to the input and output of sound system.

8. CF(Compact Flash) Card Slot

The CF card expansion slot is an interface that is used to access and download information on a CF card to a LX-8X31. The CF card can be used to expand the memory up to 32 GB.

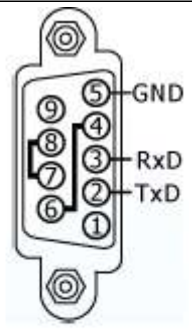
9. VGA Connector

A VGA connector is a 3-row 15-pin connector that can be used with a variety of supported VGA resolutions.

10. DIP Switch

The DIP switch can be used to set the Module ID to a number from 0 to 255. Do not use Module ID 0 for communication.

11. ttySA5(RS232)

 GND RxD TxD 1 2 3 4 5 6 7 8 9	
Port Type	Female
Baud Rate	115200, 57600, 38400, 19200, 9600, 4800, 2400, 1200 bps
Data Bits	7, 8
Parity	None, Even, Odd
Stop Bits	1
FIFO	1 byte

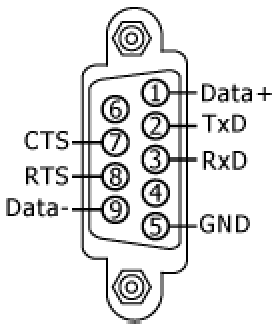
We set **ttySA5** to console port by default setting, if you want to set **ttySA5** to RS-232 port, you can follow below step after login.

```
#initctl stop ttySA5          //stop ttySA5 agetty, ttySA5 become RS-232 port
```

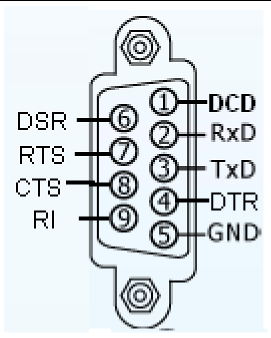
Set **ttySA5** from RS-232 port to console port.

```
#initctl start ttySA5
```

12. ttyS1(RS232/RS485)

 Data+ TxD RxD GND CTS RTS Data- 1 2 3 4 5 6 7 8 9	
Port Type	Male
Baud Rate	115200, 57600, 38400, 19200, 9600, 4800, 2400, 1200 bps
Data Bits	5, 6, 7, 8
Parity	None, Even, Odd, Mark (Always 1), Space (Always 0)

13. ttyS34(RS232)

	
Port Type	Male
Baud Rate	115200, 57600, 38400, 19200, 9600, 4800, 2400, 1200 bps
Data Bits	5, 6, 7, 8
Parity	None, Even, Odd, Mark (Always 1), Space (Always 0)
Stop Bits	1, 2
FIFO	16 bytes

14. PAC I/O Slots

The LX-8X31 contains some extra I/O slots.
The LX-8X31 uses I/O slots that can be expanded. They can serve in local and local expansion. The number of each type of the expansion I/O slot:
LX-8131: 1 I/O slot.
LX-8331: 3 I/O slots.
LX-8731: 7 I/O slots.

1.1 Packing list

- One LX-8X31 hardware module
- One 8GB CF card
- One Screw Driver
- One Quick start Guide

Note: If any of these items are missed or damaged, contact the local distributors for more information. Save the shipping materials and cartons in case you want to ship in the future.

1.2 Features

- x86 dual-core (1.0GHz)
- Linux kernel 3.2
- Embedded Service: Web Server, FTP Server, Telnet Server, SSH Server
- 64-bit Hardware Serial Number for Software Protection
- Rich I/O Expansion Ability (RS-232/RS-485, FRnet)
- 2 10/100/1000M Ethernet Ports
- Redundant Power Input
- Operating Temperature: -25 ~ +75°C

1.3 Specifications

Models	**LX-8131**	**LX-8331**	**LX-8731**
System Software			
OS	Linux kernel 3.2		
Embedded Service	SFTP server, Web server, SSH		
SDK Provided	LinPAC SDK (GCC based toolchain and LinPAC Libraries)		
CPU Module			
CPU	x86 dual-core (1.0 GHz)		
SDRAM	2 GB DDR3		
MRAM	512 KB		
Flash	mSATA slot with one 32 GB SSD		
EEPROM	16 KB		
CF	CF socket with one 8 GB CF card (support up to 32 GB)		
RTC (Real Time Clock)	Provide second, minute, hour, date, day of week, month, year		
64-bit Hardware Serial Number	Yes, for Software Copy Protection		
Dual Watchdog Timers	Yes		
Programmable LED Indicator	3 (L1, L2, L3)		
Rotary Switch	Yes (0 ~ 9)		
VGA & Communication Ports			
VGA Resolution	1280 x 1024 to 1920 x 1080 for 16:9 display; 640 x 480 to 1024 x 768 for 4:3 display		
Dual Ethernet Ports	RJ-45, 10/100/1000M Base-T (Auto-negotiating, Auto MDI/MDI-X, LED indicators)		

USB 2.0	4		
ttySA4	Internal communication with the high profile I-87K series modules in slots		
ttyS0	RS-485 (Data+, Data-); 3000 VDC isolated		
ttyS1	RS-232/485 (RxD, TxD and GND for RS-232; Data+, Data- for RS-485); 3000 VDC isolated		
ttySA5	RS-232 (RxD, TxD, CTS, RTS and GND for RS-232); 3000 VDC isolated		
ttyS34	RS-232 (RxD, TxD, CTS, RTS, DSR, DTR, CD, RI and GND); 3000 VDC isolated		
I/O Expansion Slots	1	3	7
Mechanical			
Dimensions (W x L x H)	169 mm x 132 mm x 125 mm	231 mm x 132 mm x 125 mm	355 mm x 132 mm x 125 mm
Installation	Wall Mounting		
Environmental			
Operating Temperature	-25 ~ +75°C		
Storage Temperature	-30 ~ +80°C		
Ambient Relative Humidity	10 ~ 90% RH (non-condensing)		
Power			
Input Range	+10 ~ +30 VDC		
Isolation	1 kV		
Redundant Power Inputs	Yes, with one power relay (1 A @ 24 VDC) for alarm		
Capacity	2.5 A, 5 V supply to CPU and backplane, 1.5 A, 5 V supply to I/O expansion slots, 20 W in total	2.8 A, 5 V supply to CPU and backplane, 4.2 A, 5 V supply to I/O expansion slots, 35 W in total	3 A, 5 V supply to CPU and backplane, 4 A, 5 V supply to I/O expansion slots, 35 W in total
Consumption	14.4 W (0.6 A @ 24 VDC)	15.6 W (0.65 A @ 24 VDC)	16.8 W (0.7 A @ 24 VDC)

2. The LX-8X31 SDK Introduction

LX-8X31 SDK consists of the following major items.

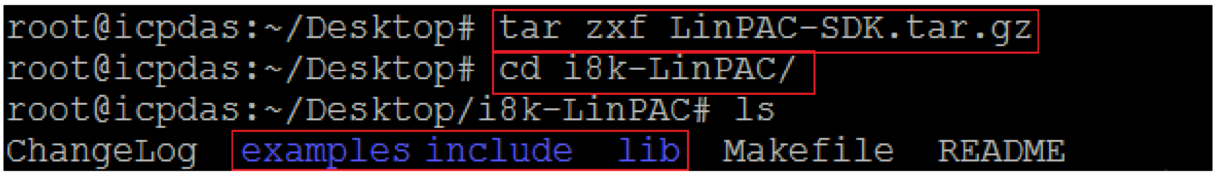
- LinPAC SDK library files
- LinPAC SDK include files
- Demo files

From http://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x8x/atom/, you can download the latest version of LX-8X31 SDK and the Manual. And then follows the below steps in order to get the development toolkit which has been provided by ICP DAS for the easy application of the LX-8X31 embedded controller platform.

(1)User can connect to LX-8X31 through **Ethernet 1**, **Ethernet 2** by using **"putty"** software (refer to Fig 4-3, Fig 4-4).

(2)After user connect to LX-8X31, user could type command "wget" http://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x8x/atom/sdk/linpac-8x81_sdk.tar.gz" to get the latest version of LX-8X31 SDK.

(3)To type "tar zxf LinPAC-SDK.tar.gz" to decompress tar file (refer to Fig 2-1).



```
root@icpdas:~/Desktop# tar zxf LinPAC-SDK.tar.gz
root@icpdas:~/Desktop# cd i8k-LinPAC/
root@icpdas:~/Desktop/i8k-LinPAC# ls
ChangeLog  examples include  lib  Makefile  README
```

Fig 2-1

Once user decompresses the SDK file, user can find the files for the library and demo in the following paths.

The libi8k.a and libi8k-3600.a path is "**i8k-LinPAC/lib**".

The include files path is "**i8k-LinPAC/include**".

The demo path is "**i8k-LinPAC/examples**".

3. The Architecture of LIBI8K-3600.A in the LX-8X31

The **libi8k-3600.a** is a library file that is designed for I7000/8000/87000 applications running in the LX-8X31 Embedded Controller using the Linux OS. Users can apply it to develop their own applications **with GNU C language**. In order to assist users to build their project quickly, we provide many demo programs. Based on these demo programs, users can easily understand how to use these functions and develop their own applications within a short period of time.

The relationships among the libi8k-3600.a and user's applications are depicted as Fig. 3-1：

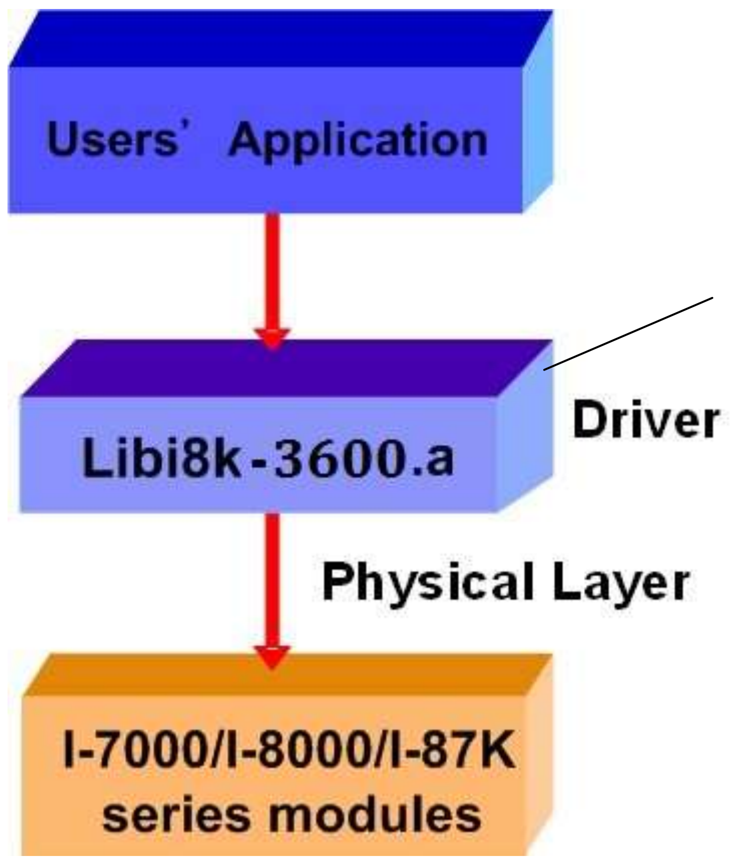


Fig. 3-1

Functions for LX-8X31 Embedded Controller are divided into sub-groups for ease of use within the different applications：

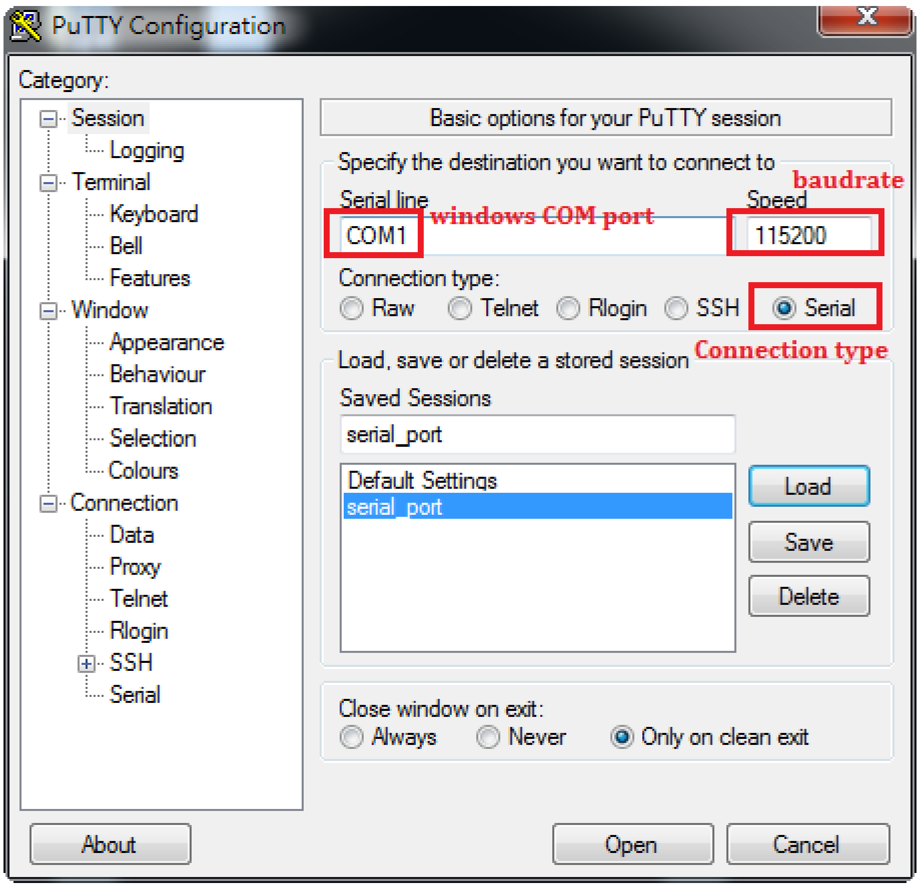
System Information Functions
Digital Input Functions
Digital Output Functions
Watch Dog Timer Functions
EEPROM Read/Write Functions
Analog Input Functions
Analog Output Functions

The functions in the Libi8k-3600.a are especially designed for LX-8X31 Series. Users can easily find the functions they need for their applications from the descriptions in chapter 6 and in the demo programs provided in chapter 7.

4. LX-8X31 System Settings

User can use two methods to connect to LX-8X31 to configure system, check system status and startup/stop system service:

(1)To connect to LX-8X31 through serial port "ttySA5" by using "**Putty**" software (please refer to Fig 4-1) (baud rate is 115200).


PuTTY Configuration

Category:

Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM1 — windows COM port

Speed: 115200 — baudrate

Connection type: Raw / Telnet / Rlogin / SSH / [✓ Serial] — Connection type

Load, save or delete a stored session

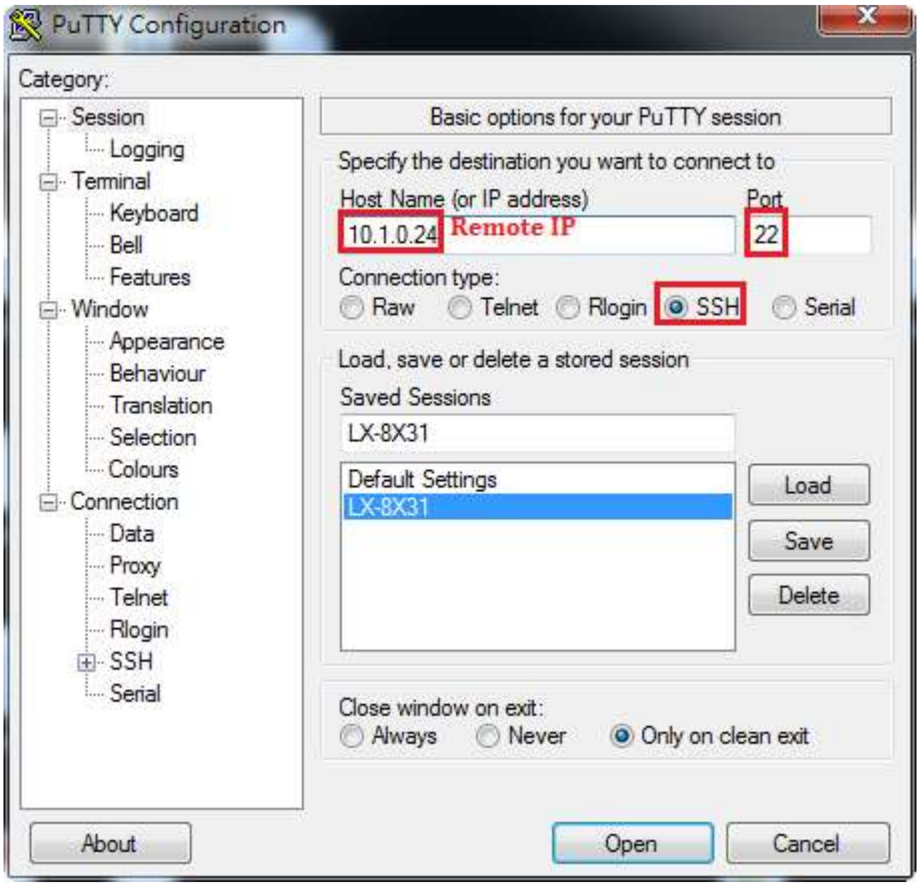
Saved Sessions: serial_port

Default Settings
serial_port

Close window on exit: Always / Never / [✓ Only on clean exit]

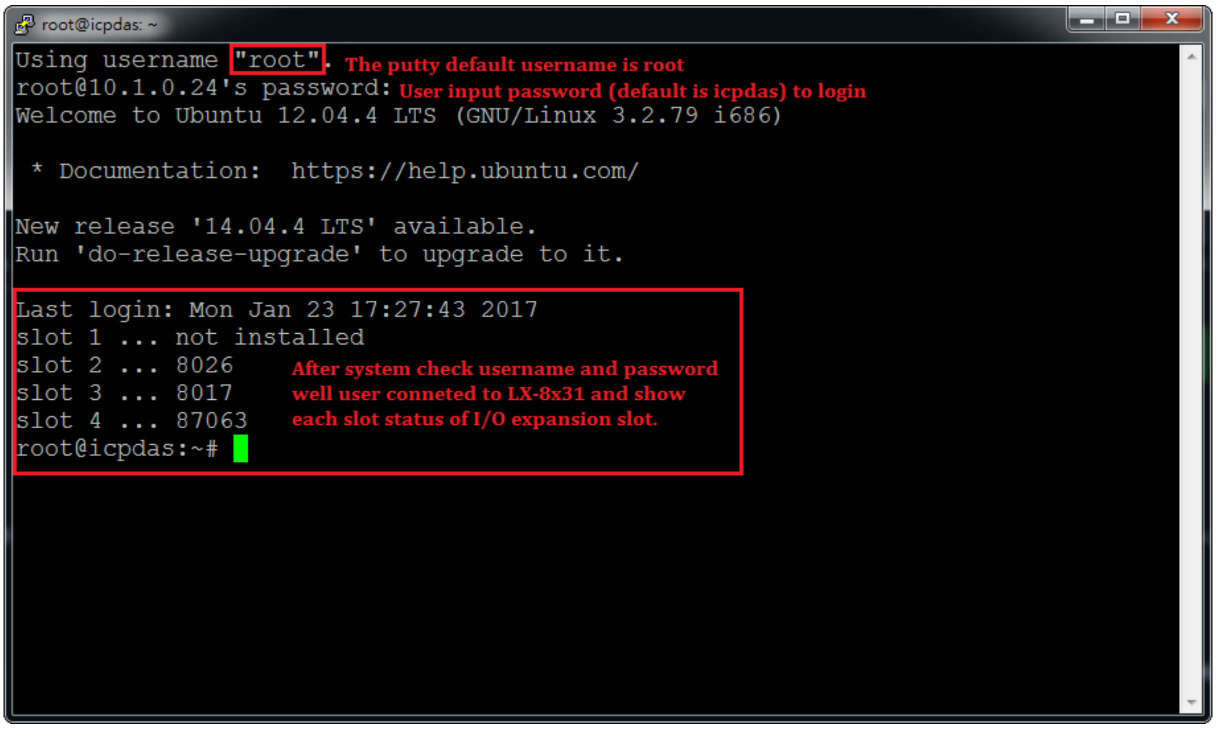
Fig 4-1

(2)To connect to LX-8X31 through Ethernet Port by using **"Putty"** software (please refer to Fig 4-2, Fig 4-3). The default ID is "**root**" and password is "**icpdas**".



Host Name (or IP address): 10.1.0.24 **Remote IP**
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Saved Sessions: LX-8X31
Close window on exit: Always Never [✓ Only on clean exit]

Fig 4-2



```
Using username "root".  The putty default username is root
root@10.1.0.24's password: User input password (default is icpdas) to login
Welcome to Ubuntu 12.04.4 LTS (GNU/Linux 3.2.79 i686)

 * Documentation:  https://help.ubuntu.com/

New release '14.04.4 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Mon Jan 23 17:27:43 2017
slot 1 ... not installed
slot 2 ... 8026
slot 3 ... 8017
slot 4 ... 87063
root@icpdas:~#
```

After system check username and password well user conneted to LX-8x31 and show each slot status of I/O expansion slot.

Fig 4-3

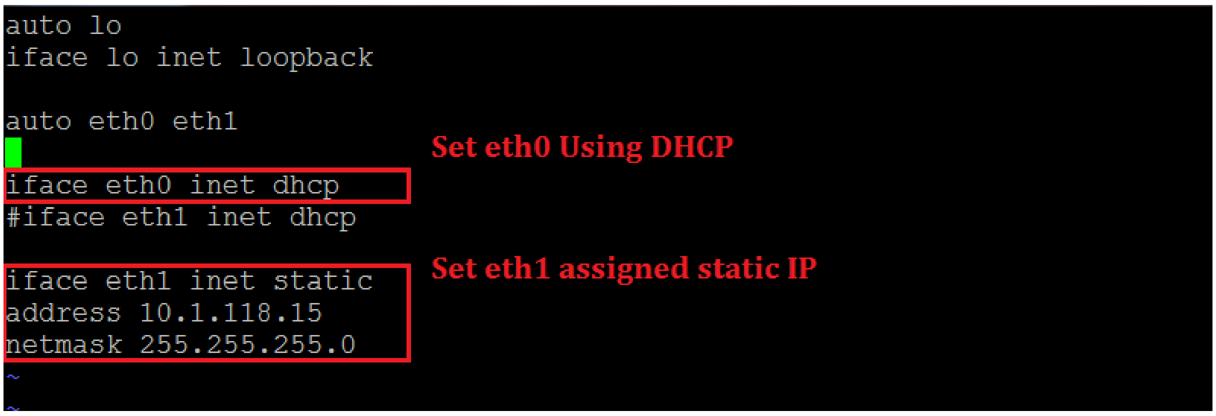
In this section, we will introduce how to setup the LX-8X31 configuration. Let users can use the LX-8X31 more easily.

4.1 Settings for the LX-8X31 Network

The LX-8X31 network setting includes two ways. One is **DHCP** and the other is "**Assigned Static IP**". DHCP is the default setting after the LX-8X31 is produced and this way is easy for users. However, if your network system is without DHCP server, then users need to configure the network setting by using "Assigned IP".

4.1.1 Setting the IP、Netmask

Boot up LX-8X31 and type in **"vim /etc/network/interfaces "** to open the network setting file. Please refer to the Fig 4-4, show you how to use "DHCP" and "static IP" method.



```
auto lo
iface lo inet loopback

auto eth0 eth1

iface eth0 inet dhcp
#iface eth1 inet dhcp

iface eth1 inet static
address 10.1.118.15
netmask 255.255.255.0
~
~
```

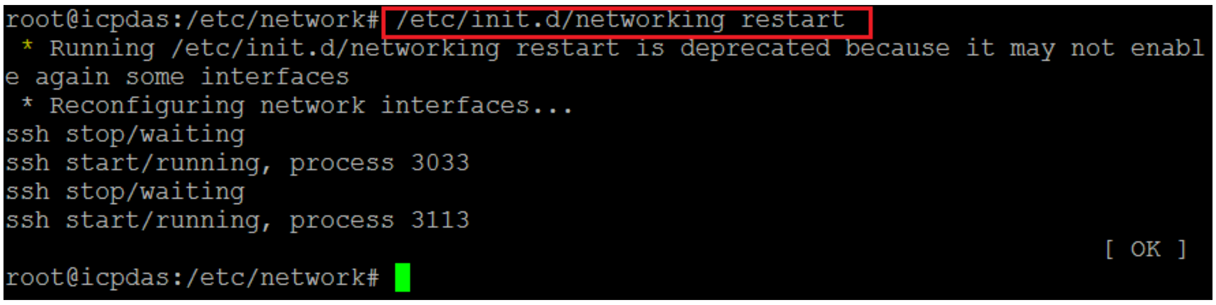
Fig 4-4

(1) Using DHCP:
In Fig 4-4, eth0 set as DHCP.

(2) Using "Assigned IP":

In Fig 4-4, eth1 set as static IP, assigned address "10.1.118.15" and set netmask.

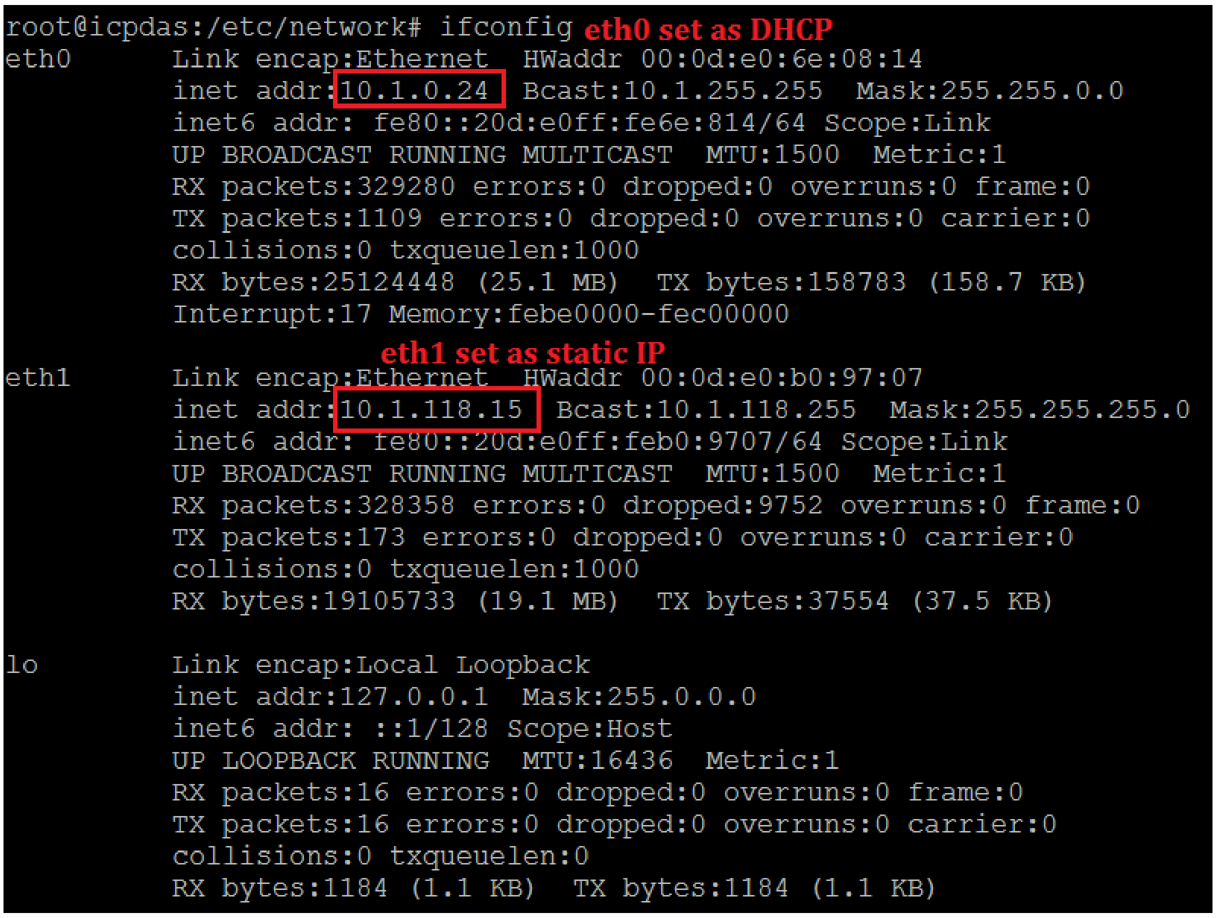
Use command **"/etc/init.d/networking restart"** to make the setting work. Please refer to the Fig 4-5:



```
root@icpdas:/etc/network# /etc/init.d/networking restart
 * Running /etc/init.d/networking restart is deprecated because it may not enabl
e again some interfaces
 * Reconfiguring network interfaces...
ssh stop/waiting
ssh start/running, process 3033
ssh stop/waiting
ssh start/running, process 3113
                                                                         [ OK ]
root@icpdas:/etc/network#
```

Fig 4-5

After finishing the LinPAC network setting, user can use command **" ifconfig "** to see the network setting. (Refer to the Fig 4-6)

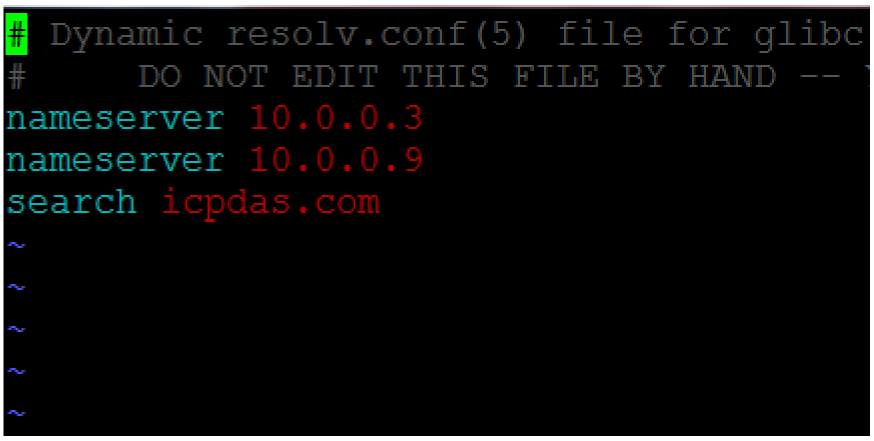


```
root@icpdas:/etc/network# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0d:e0:6e:08:14
          inet addr:10.1.0.24  Bcast:10.1.255.255  Mask:255.255.0.0
          inet6 addr: fe80::20d:e0ff:fe6e:814/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:329280 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1109 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:25124448 (25.1 MB)  TX bytes:158783 (158.7 KB)
          Interrupt:17 Memory:febe0000-fec00000

eth1      Link encap:Ethernet  HWaddr 00:0d:e0:b0:97:07
          inet addr:10.1.118.15  Bcast:10.1.118.255  Mask:255.255.255.0
          inet6 addr: fe80::20d:e0ff:feb0:9707/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:328358 errors:0 dropped:9752 overruns:0 frame:0
          TX packets:173 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:19105733 (19.1 MB)  TX bytes:37554 (37.5 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1184 (1.1 KB)  TX bytes:1184 (1.1 KB)
```

Fig 4-6

4.1.2 Setting of DNS

Boot up LX-8X31 and type in **“vim /etc/resolv.conf”** to open the DNS setting file. Type “DNS server” in “**nameserver**”,use command “reboot” to reboot LX-8X31 to make setting working. (Refer to the Fig 4-7)



```
# Dynamic resolv.conf(5) file for glibc
#     DO NOT EDIT THIS FILE BY HAND --
nameserver 10.0.0.3
nameserver 10.0.0.9
search icpdas.com
~
~
~
~
~
```

Fig 4-7

4.2 CF (Compact Flash) Card Usage

Before you startup the LX-8X31, you can insert the CF Card into the slot of CF Card in the LX-8X31. Then user can use the command **“fdisk –l”** to check the device name of CF card in the LX-8X31. Therefore, users can access the CF Card in the LX-8X31by using the command **“mount”** and **“umount”**. Please refer to the Fig 4-8:

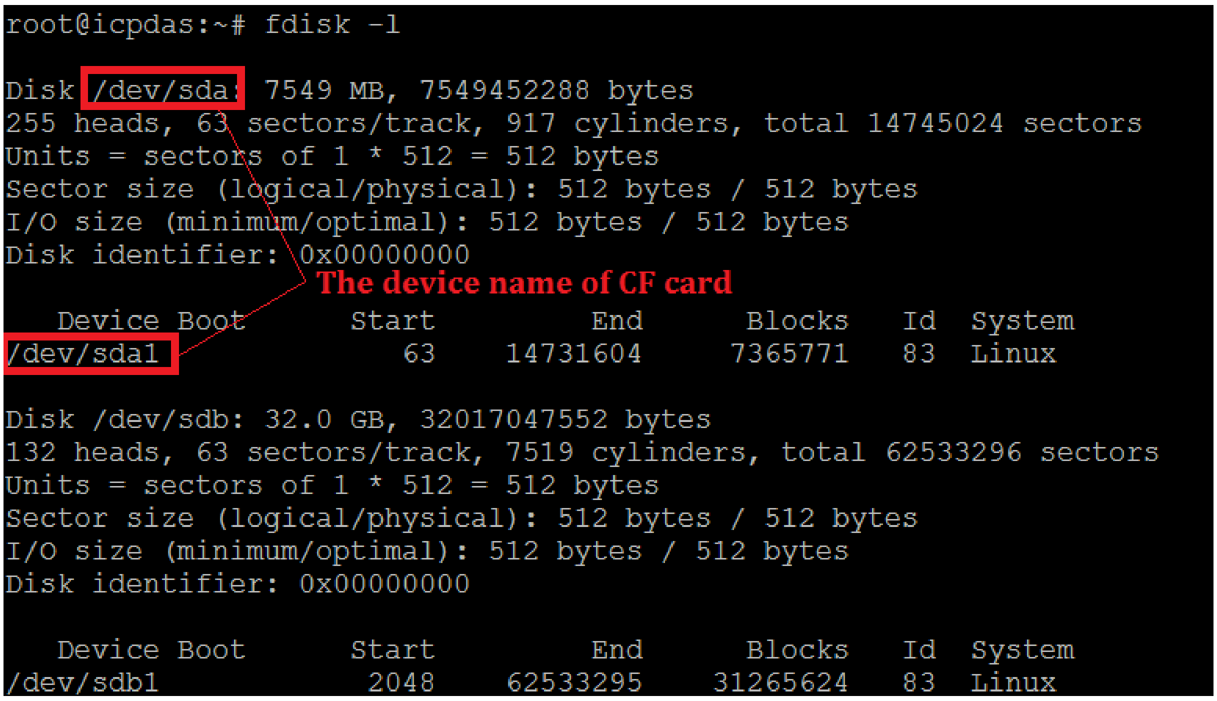
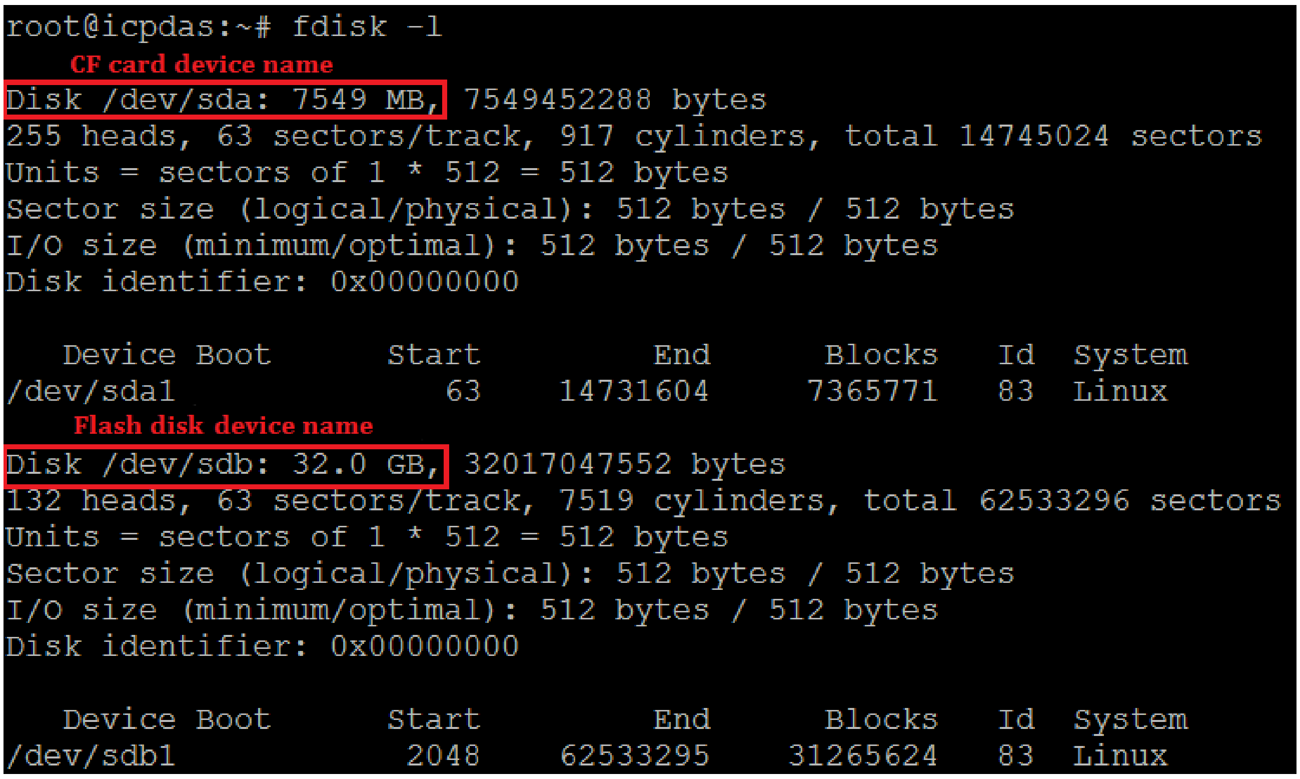

```
root@icpdas:~# fdisk -l

Disk /dev/sda: 7549 MB, 7549452288 bytes
255 heads, 63 sectors/track, 917 cylinders, total 14745024 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1              63    14731604     7365771   83  Linux

Disk /dev/sdb: 32.0 GB, 32017047552 bytes
132 heads, 63 sectors/track, 7519 cylinders, total 62533296 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1            2048    62533295    31265624   83  Linux
```

Fig 4-8

4.2.1 Flash disk device name

There’s two situation in your Flash device name between insert CF card and not insert CF card.

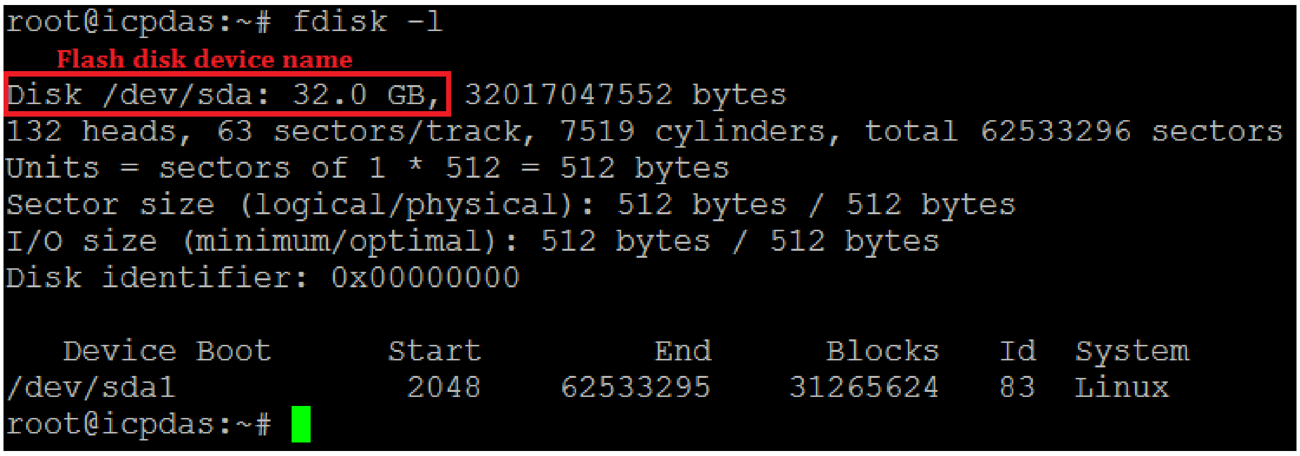
1.Insert CF card and startup LX-8X31, Please refer to Fig 4-9.
Flash disk device name is “/dev/sdb”, and CF card device name is “/dev/sda”



```
root@icpdas:~# fdisk -l
    CF card device name
Disk /dev/sda: 7549 MB, 7549452288 bytes
255 heads, 63 sectors/track, 917 cylinders, total 14745024 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1              63    14731604     7365771   83  Linux
    Flash disk device name
Disk /dev/sdb: 32.0 GB, 32017047552 bytes
132 heads, 63 sectors/track, 7519 cylinders, total 62533296 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1            2048    62533295    31265624   83  Linux
```

Fig 4-9

2.Not insert CF card and startup LX-8X31, Please refer to Fig 4-10.
Flash disk device name is “/dev/sda”.



```
root@icpdas:~# fdisk -l
   Flash disk device name
Disk /dev/sda: 32.0 GB, 32017047552 bytes
132 heads, 63 sectors/track, 7519 cylinders, total 62533296 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1            2048    62533295    31265624   83  Linux
root@icpdas:~#
```

Fig 4-10

If you insert CF card, then Flash disk device name is “/dev/sdb”, CF card device name is “/dev/sda”.
If you not insert CF card, then Flash disk device name is “/dev/sda”.

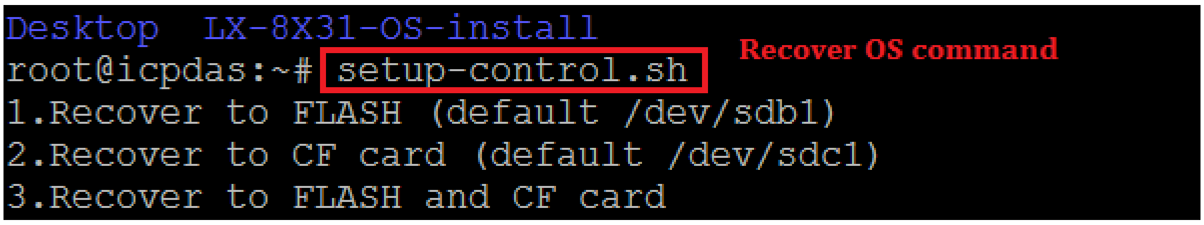
4.2.2 Recover OS by CF card

Before recover Flash disk OS, you should backup your data from your Flash disk OS first, after recover OS, old data will all covered.

Follow below steps to recover Flash OS.

1. Boot up LX-8X31 with CF card OS. (You can check if LX-8X31-OS-install directory exist or not)
2. #setup-control.sh (please refer to Fig 4-11)

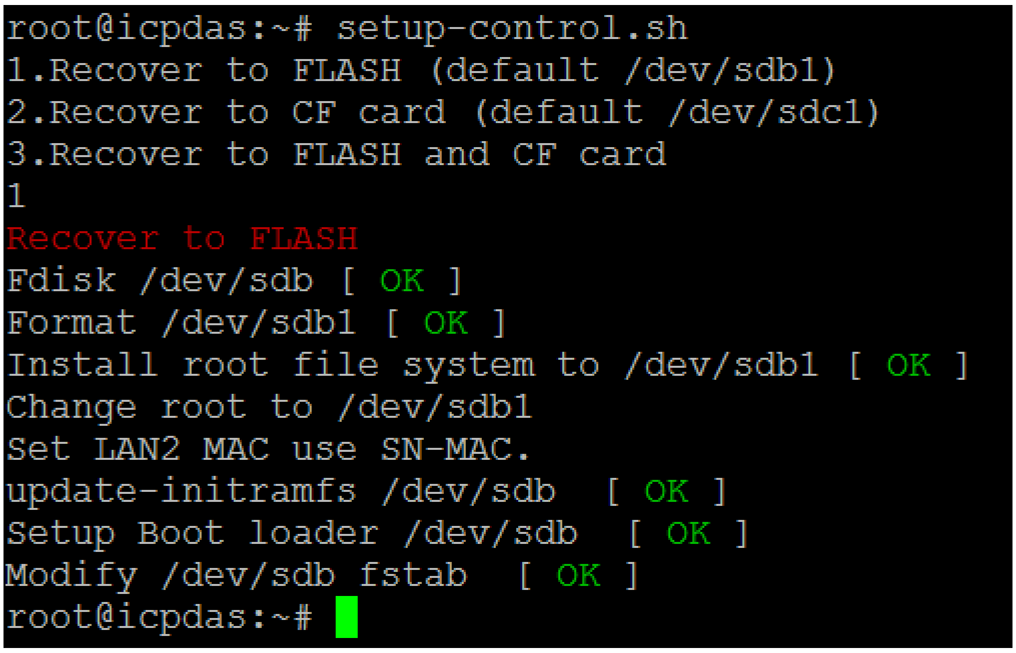
   You can choose option 1 to recover Flash OS, The default setting Flash OS device file name is /dev/sdb1.



```
Desktop  LX-8X31-OS-install
root@icpdas:~# setup-control.sh
1.Recover to FLASH (default /dev/sdb1)
2.Recover to CF card (default /dev/sdc1)
3.Recover to FLASH and CF card
```

Fig 4-11

3. After recovered (Refer Fig 4-12), you can use factory setting OS in Flash disk.



```
root@icpdas:~# setup-control.sh
1.Recover to FLASH (default /dev/sdb1)
2.Recover to CF card (default /dev/sdc1)
3.Recover to FLASH and CF card
1
Recover to FLASH
Fdisk /dev/sdb [ OK ]
Format /dev/sdb1 [ OK ]
Install root file system to /dev/sdb1 [ OK ]
Change root to /dev/sdb1
Set LAN2 MAC use SN-MAC.
update-initramfs /dev/sdb  [ OK ]
Setup Boot loader /dev/sdb  [ OK ]
Modify /dev/sdb fstab  [ OK ]
root@icpdas:~#
```

Fig 4-12

4.3 USB Device Usage

Before accessing the USB device, users need to mount the USB device to the LX-8X31. Because it will not auto-mount the USB device in the L-8X31. After Users mount the USB device (ex USB disk) to the LX-8X31, they can access the USB device.

4.3.1 Mount USB Device

The steps are as follows:

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

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

4.3.2 Umount USB Device

Before users pull out the USB device from the LX-8X31, users need to type the “ **umount /mnt/usb** “ command first. Then pull out the USB device to prevent any damage to usb device.

4.4 Running applications automatically at boot time

A “run level” determines which programs are executed at system startup. Run level 2 is default run level of LX-8X31. The contents of run level are in the “/etc/init.d” directory that directory contains the scripts executed at boot time. These scripts are referenced by symbolic links in the /etc/rc2.d.These links are named S<2-digit-number><original-name>. The numbers determine the order in which the scripts are run, from 00 to 99 — the lower number would earlier executed. Scripts named with an **S** are called with start, and named with a **K or x** are called with stop.

5. Instructions for the LX-8X31 Series

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

5.1 Basic Linux Instructions

5.1.1 Linux Command “ls”

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

Parameter：

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

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

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

5.1.2 Linux Command “cd”

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

Parameter：

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

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

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

5.1.3 Linux Command “mkdir”

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

mkdir –parameter subdirectory (Example：mkdir owner)

5.1.4 Linux Command “rmdir”

rmdir：delete(remove) the subdirectory and it must be empty －＞

(like rd in DOS)

mkdir –parameter subdirectory(Example：rmdir owner)

5.1.5 Linux Command “rm”

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

rm –parameter file (or directory)

Parameter：

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

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

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

5.1.6 Linux Command “cp”

cp：copy file －＞ (like copy in DOS)

cp –parameter source destination(Example：cp test.exe /root/Test/test.exe)

5.1.7 Linux Command “mv”

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

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

(Example：mv test.exe test1.exe)

(Example：mv test.exe /root/Test)

5.1.8 Linux Command “pwd”

pwd：show the current path

5.1.9 Linux Command “who”

who：show the on-line users

5.1.10 Linux Command “chmod”

chmod：change authority of file

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

For example：

chmod　754　test.exe

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

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

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

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

5.1.11 Linux Command “uname”

uname：show the version of linux

5.1.12 Linux Command “ps”

ps：show the procedures that execute now

5.1.15 Linux Command “date”

date：show date and time

5.1.16 Linux Command “netstat”

netstat：show the state of network

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

5.1.17 Linux Command “ifconfig”

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

5.1.18 Linux Command “wget”

wget : get the file from the web link.

5.1.19 Linux Command “ping”

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

ping IPAddress (Example：ping 192.168.0.1)

5.1.19 Linux Command “clear”

clear：clear the screen

5.1.20 Linux Command “passwd”

passwd：change the password

5.1.21 Linux Command “reboot”

reboot：reboot the LinPAC

5.2 A Simple Example – Helloworld.c

In this section, we will introduce how to compile the helloworld.c to helloworld and transfer the helloworld to the LX-8X31 Series by using WinSCP software or “scp” command (please refer to the Fig 5-1). Finally executes this file on the LX-8X31 Series. In this example, no ICP DAS modules are used.

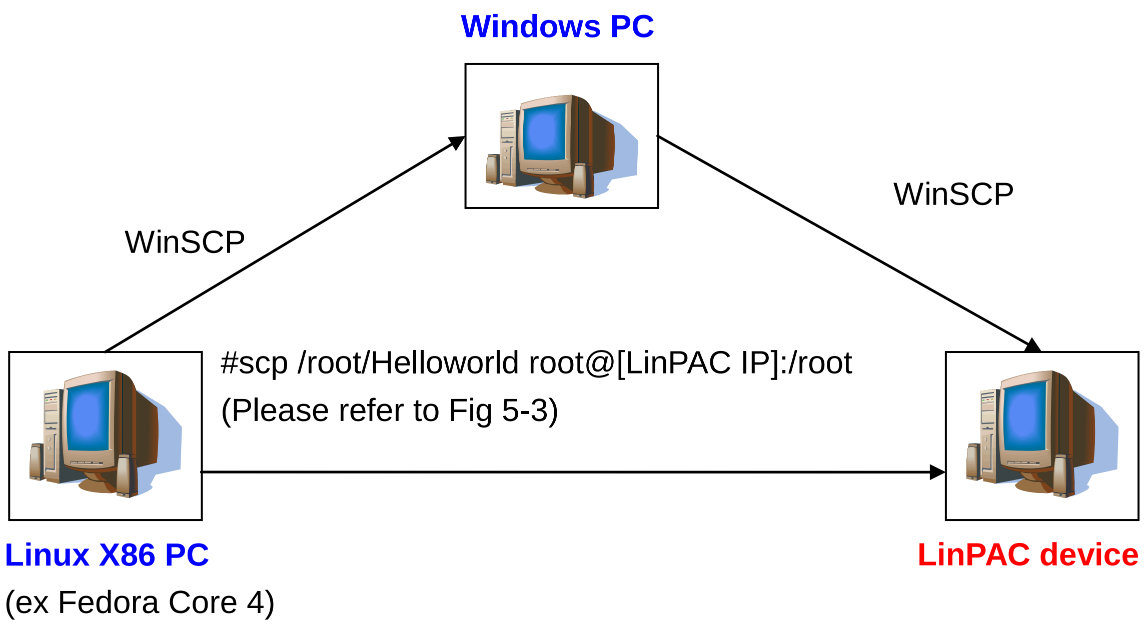

Windows PC
WinSCP
WinSCP
#scp /root/Helloworld root@[LinPAC IP]:/root
(Please refer to Fig 5-3)
Linux X86 PC
(ex Fedora Core 4)
LinPAC device

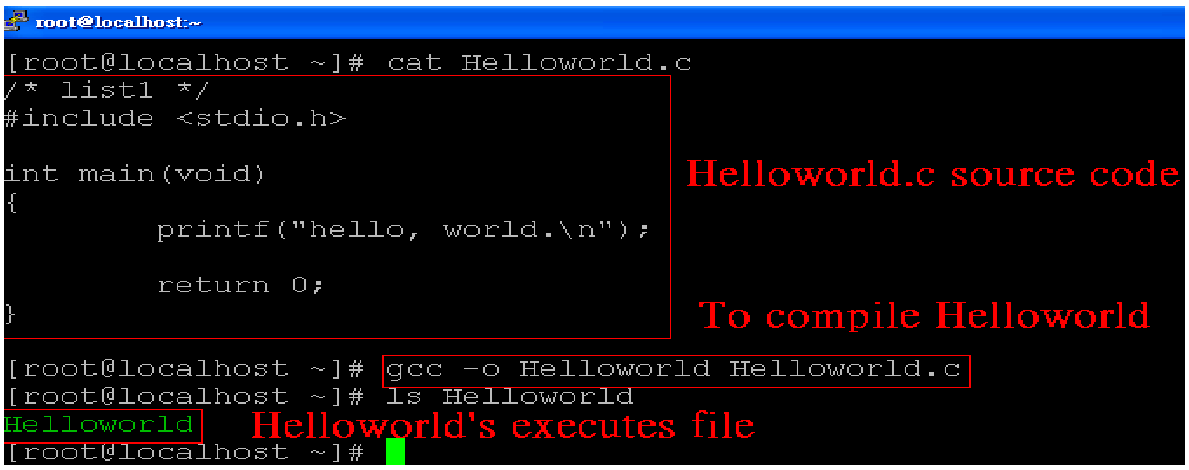
Fig 5-1

If you want to use the modules of ICP DAS to control your system, you can refer to demo in the chapter 7. These processes can be divided into three steps and thet are given as below：

STEP 1 ：(Compile helloworld.c to helloworld on the Linux X86 PC)

(1) Create Helloworld.c on the linux PC(using linux distribution, for example Fedora Core 4, please refer to the Fig 5-2):

(2) Type in "**gcc –o Helloworld Helloworld.c**" to compile Helloworld.c into helloworld (please refer to Fig 5-2)



```
[root@localhost ~]# cat Helloworld.c
/* list1 */
#include <stdio.h>

int main(void)
{
        printf("hello, world.\n");

        return 0;
}
[root@localhost ~]# gcc -o Helloworld Helloworld.c
[root@localhost ~]# ls Helloworld
Helloworld
[root@localhost ~]#
```

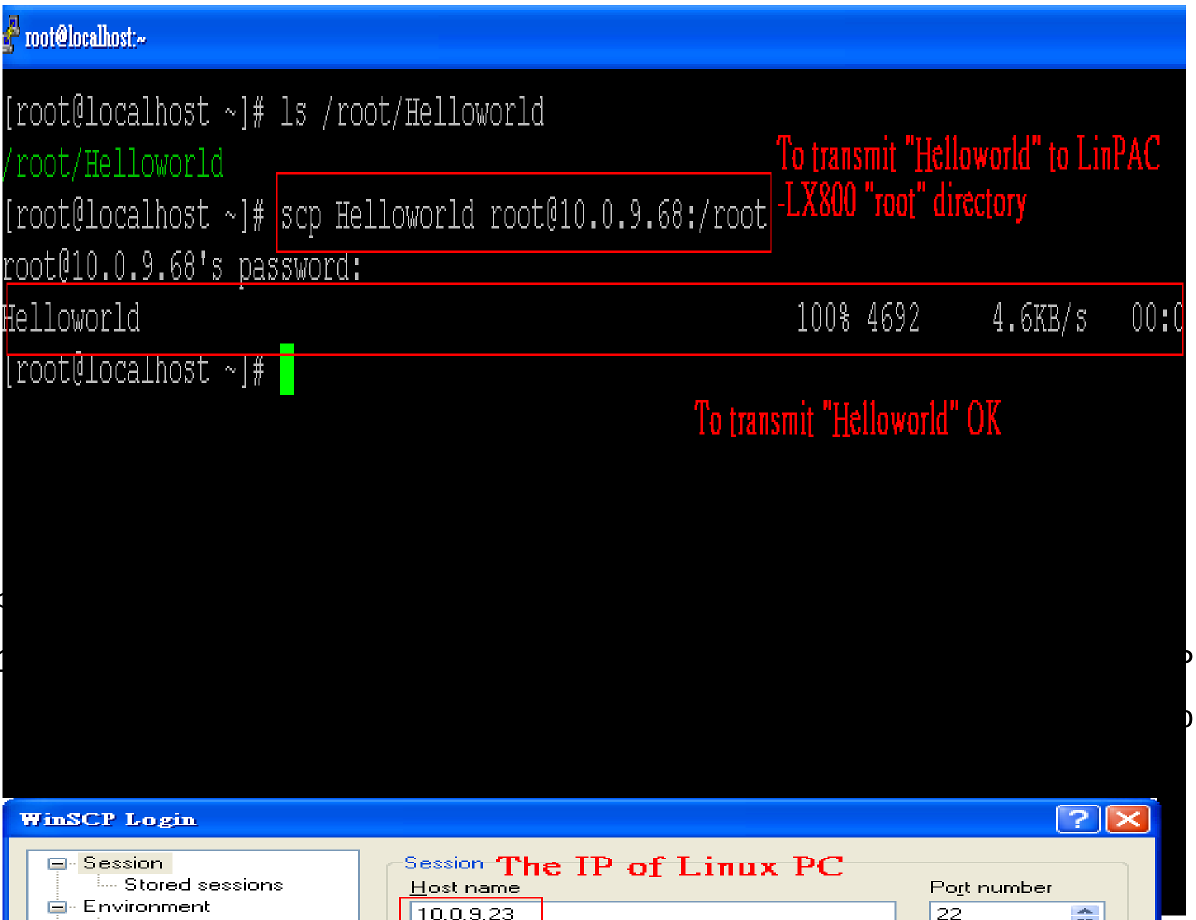
Fig 5-2

STEP 2 ：(Transfer Helloworld to the LX-8X31)

There are two methods for transferring files to the LX-8X31：

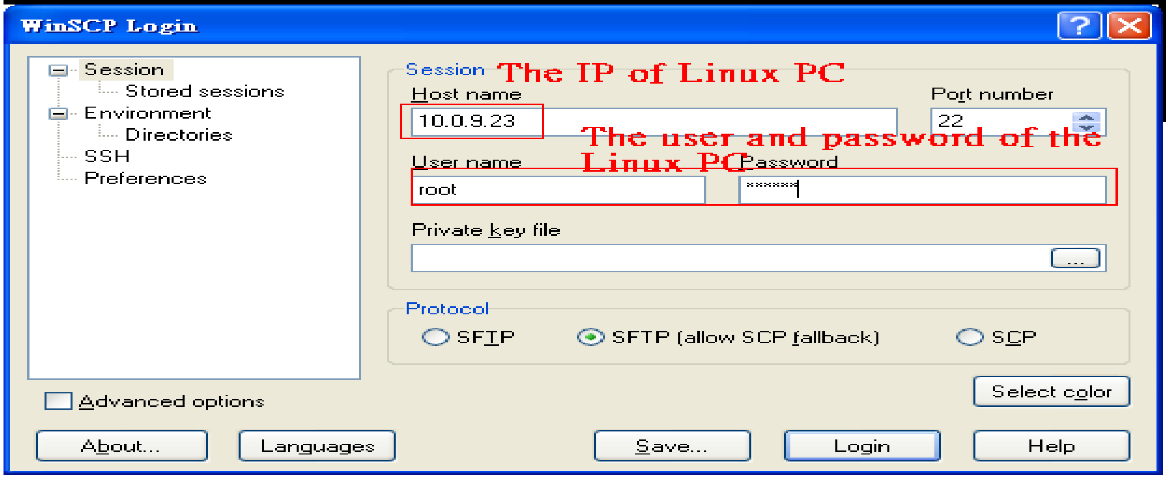
＜ Method one ＞ By Using the “Linux Command Prompt”：

Open a “Linux Command Prompt” and type “scp /root/Helloworld root@[LinPAC device IP] /root” command to connect and transmit the executes “Helloworld” to the LX-8X31. Then type the Password (“icpdas” is the default value.) to accomplish the connection and transmission from the linux PC to the LX-8X31.Please refer to Fig 5-3.



```
[root@localhost ~]# ls /root/Helloworld
/root/Helloworld
[root@localhost ~]# scp Helloworld root@10.0.9.68:/root
root@10.0.9.68's password:
Helloworld                                   100% 4692     4.6KB/s   00:0
[root@localhost ~]#
```

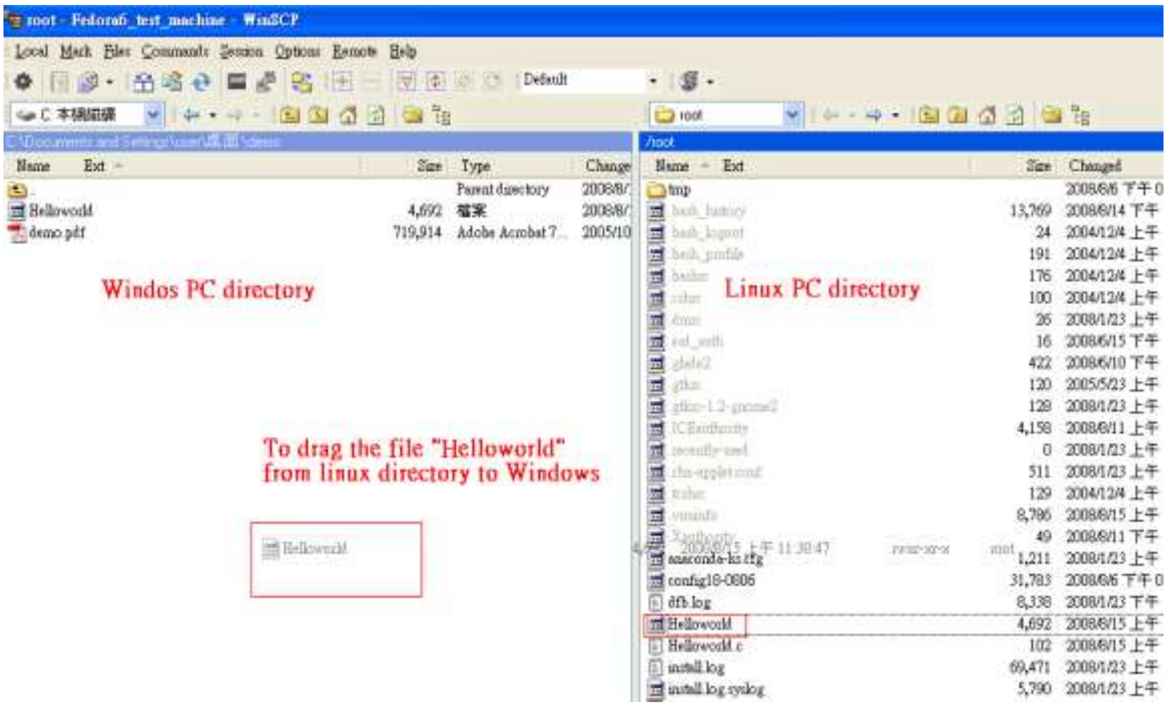
root@localhost:~
To transmit "Helloworld" to LinPAC -LX800 "root" directory
To transmit "Helloworld" OK



Host name: 10.0.9.23
Port number: 22
User name: root
Password: *******
Private key file:
Protocol: SFTP [✓ SFTP (allow SCP fallback)] SCP

Fig 5-4

(2) After user connect to the linux PC, user could find the “Helloworld” file in the directory that user compiled. Then user could drag the file “Helloworld” from the linux directory to window directory. (Refer to Fig.5-5).



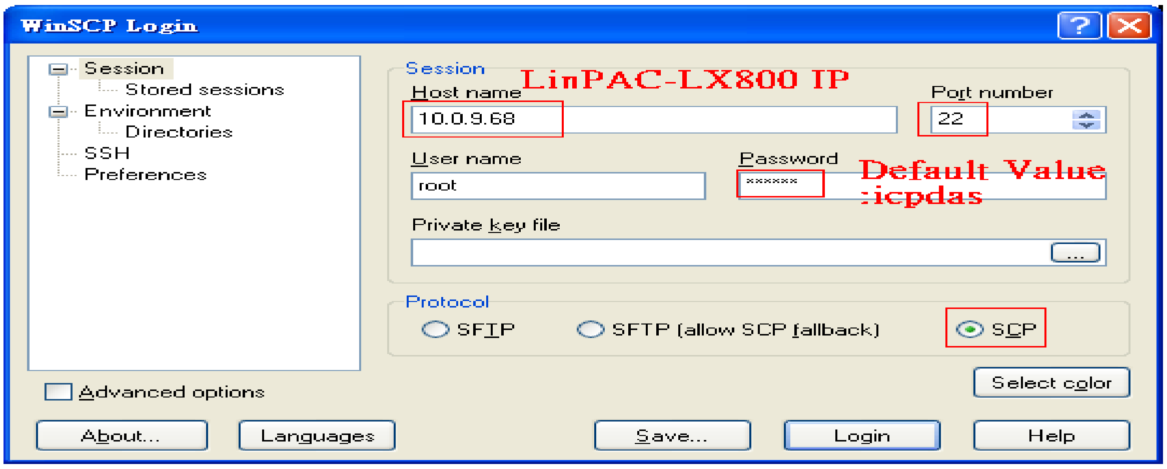
root - Fedora6_test_machine - WinSCP
Local Mark Files Commands Session Options Remote Help
Default
C 本機磁碟
C:\Documents and Settings\user\桌面\demo
root
/root
Windos PC directory
Linux PC directory
To drag the file "Helloworld" from linux directory to Windows
Helloworld

Name	Ext	Size	Type	Changed
..			Parent directory	2008/8/
Helloworld		4,692	檔案	2008/8/
demo.pdf		719,914	Adobe Acrobat 7...	2005/10

Name	Ext	Size	Changed
tmp			2008/6/6 下午 0
bash_history		13,769	2008/6/14 下午
bash_logout		24	2004/12/4 上午
bash_profile		191	2004/12/4 上午
bashrc		176	2004/12/4 上午
cshrc		100	2004/12/4 上午
dmrc		26	2008/1/23 上午
esd_auth		16	2008/6/15 下午
glade2		422	2008/6/10 下午
gtkrc		120	2005/5/23 上午
gtkrc-1.2-gnome2		128	2008/1/23 上午
ICEauthority		4,158	2008/6/11 上午
recently-used		0	2008/1/23 上午
zhu-applet.conf		511	2008/1/23 上午
tcshrc		129	2004/12/4 上午
viminfo		8,786	2008/6/15 上午
Xauthority		49	2008/6/11 下午
anaconda-ks.cfg		1,211	2008/1/23 上午
config18-0806		31,783	2008/6/6 下午 0
dfb.log		8,338	2008/1/23 下午
Helloworld		4,692	2008/6/15 上午
Helloworld.c		102	2008/6/15 上午
install.log		69,471	2008/1/23 上午
install.log.syslog		5,790	2008/1/23 上午

Fig.5-5

(3) After user get the file “Helloworld” from the linux PC, user could upload the file to the LX-8X31. Please create a new the login window of WinSCP, then type the LX-8X31 user and password, ssh port and the IP. After user set configure, user can click “login” button to connect to the LX-8X31(please refer to Fig 5-6).



Session

Host name: 10.0.9.68

Port number: 22

User name: root

Password: ******

Private key file:

Protocol: SFTP SFTP (allow SCP fallback) [✓ SCP]

Fig 5-6

(4) Upload the file - **Helloworld** to the LX-8X31. (Refer to Fig.5-7).

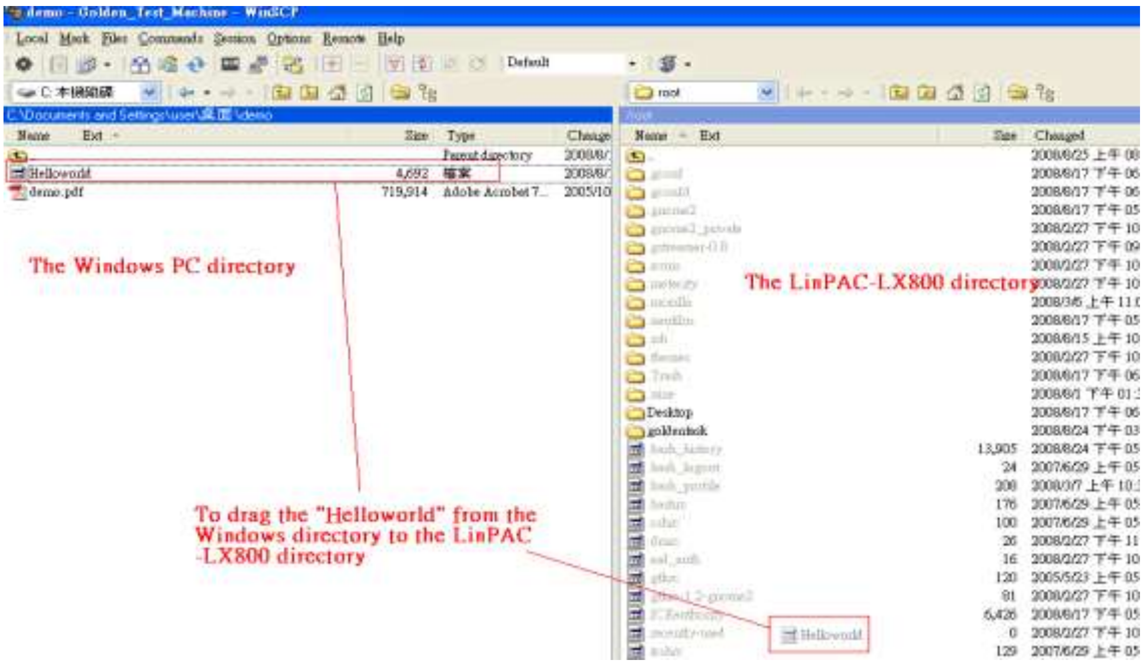
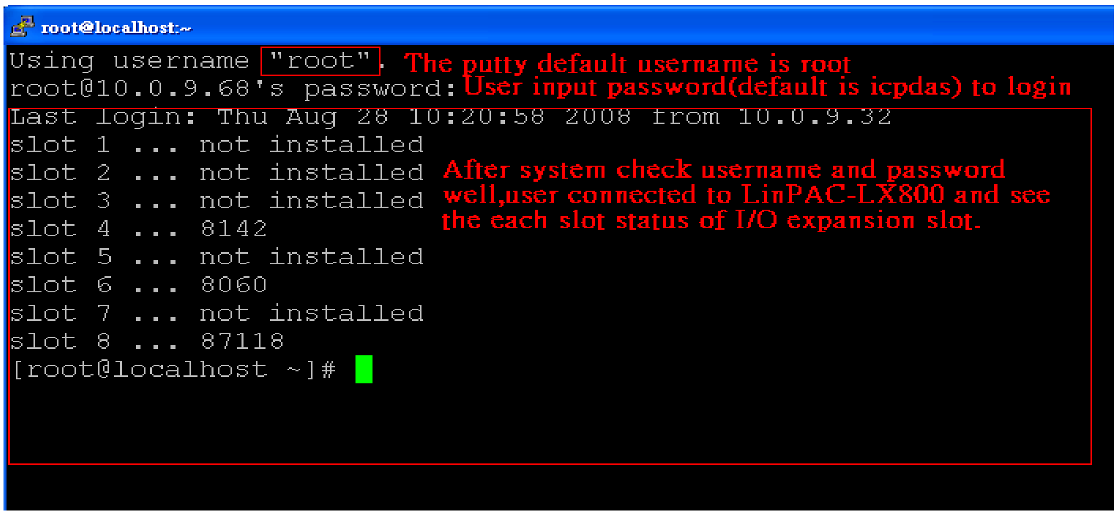

demo - Golden_Test_Machine - WinSCP
Local Mark Files Commands Session Options Remote Help
Helloworld
demo.pdf
The Windows PC directory
The LinPAC-LX800 directory
To drag the "Helloworld" from the Windows directory to the LinPAC -LX800 directory
Desktop
goldentook
Helloworld

Fig.5-7

STEP 3 ：(SSH to the LX-8X31and execute program)

(1) To use “putty.exe” software (please refer to chapter 4) or “telnet” command (please refer to chapter 4) to connect to the remote server of the LX-8X31. Then type the **Password (“<u>root</u>” is the default value.)**. If it shows the “ # “ prompt character, the process of connecting from your PC to the remoter server of the LX-8X31 is finished. (refer to Fig.5-8)

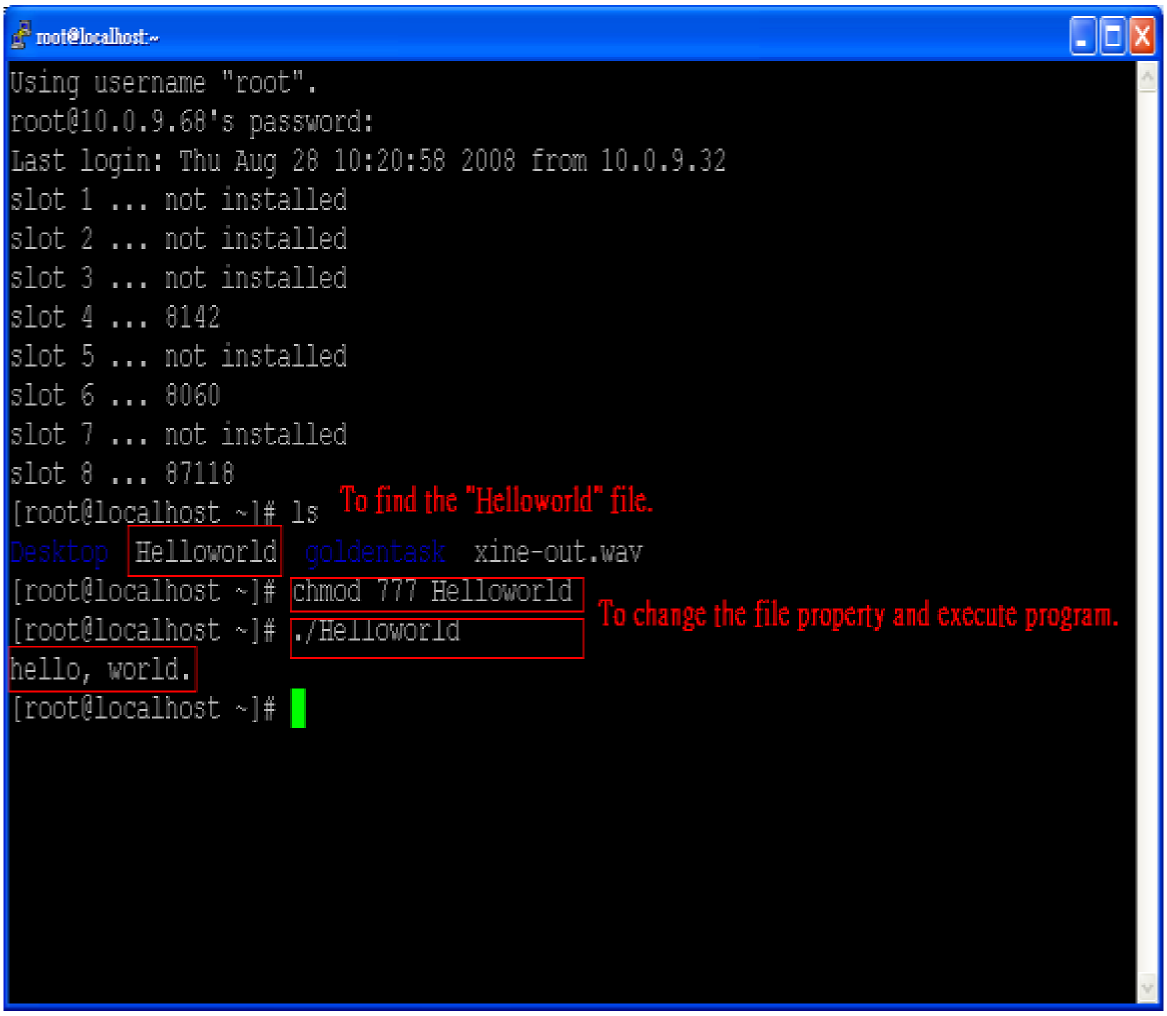

root@localhost:~
The putty default username is root
User input password(default is icpdas) to login
After system check username and password well,user connected to LinPAC-LX800 and see the each slot status of I/O expansion slot.

```
Using username "root".
root@10.0.9.68's password:
Last login: Thu Aug 28 10:20:58 2008 from 10.0.9.32
slot 1 ... not installed
slot 2 ... not installed
slot 3 ... not installed
slot 4 ... 8142
slot 5 ... not installed
slot 6 ... 8060
slot 7 ... not installed
slot 8 ... 87118
[root@localhost ~]#
```

Fig.5-8

(2) Type in the “**ls**“ command in order to list all the files in /root and to see the

"Helloworld" file. Then type in the "**chmod 777 Helloworld**" command to change the authority of Helloworld. This means that the file is executable. Type in "**./Helloworld**" to execute the file and it will show "hello, world.". Then all the steps from **compile**、 **transfer** to **connect to execute program** will be completed. (Refer to Fig.5-9)


root@localhost:~
To find the "Helloworld" file.
To change the file property and execute program.

```
Using username "root".
root@10.0.9.68's password:
Last login: Thu Aug 28 10:20:58 2008 from 10.0.9.32
slot 1 ... not installed
slot 2 ... not installed
slot 3 ... not installed
slot 4 ... 8142
slot 5 ... not installed
slot 6 ... 8060
slot 7 ... not installed
slot 8 ... 87118
[root@localhost ~]# ls
Desktop  Helloworld  goldentask  xine-out.wav
[root@localhost ~]# chmod 777 Helloworld
[root@localhost ~]# ./Helloworld
hello, world.
[root@localhost ~]#
```

5.3 i-Talk Utility

The **i-Talk utility** provides **eightteen instructions** that make it convenient for users to access the modules and hardware in the LX-8X31 and they are placed in the path - **/usr/local/sbin**. Table 5-1 describes the functions of i-Talk utility.

Instruction	Function Description
getlist	List all module name in the LX-8X31
setdo	Set digital output value to 8k module
setao	Set analog output value to 8k module
getdi	Get digital input value from 8k module
getai	Get analog input value from 8k module
setexdo	Set digital output value to 7k/87k module
setexao	Set analog output value to 7k/87k module
getexdi	Get digital input value from 7k/87k module
getexai	Get analog input value from 7k/87k module
setport	Set port value by offset to a module
getport	Get port value by offset from a module
setsend	Send string from LinPAC COM port
getreceive	Receive string from LinPAC COM port
getsendreceive	Send/Receive string from LinPAC COM port
read_sn	Get Hardware Serial Number of LX-8X31
LX-8x31-rotary_id	Get Rotary Switch ID of LX-8X31
ttyS1-config	Set ttyS1 communication mode
iztconfig	Zigbee protocol communicate, Use ZT-USBC commute with ZT-2000 series device

Table 5-1

ZT-USBC introduce
http://www.icpdas.com/root/product/solutions/industrial_wireless_communication/wireless_solutions/zt-usb.html
ZT-2000 series inreoduce
http://www.icpdas.com/root/product/solutions/industrial_wireless_communication/wireless_solutions/wireless_selection.html

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

Instruction	**Demo**
getlist	**Command:** getlist → getlist **Description**: List all module names in the LX-8X31 Series.
setdo	**Command:** setdo [slot] [data] → setdo 3 3 **Description**: Set i-8055 channel 1 and 2 on.
setao	**Command:** setao [slot] [channel] [data] → setao 1 0 2.2 **Description**: Set the i-8024 channel 0 output 2.2V.
getdi	**Command:** getdi [slot] [type] → getdi 3 8 **Description**: Get the 8 bits DI value from i-8055.
getai	**Command:** getai [slot] [channel] [gain] [mode] → getdi 2 0 0 0 **Description**: Get the AI value from i-8017HW.
setexdo	**Command:** (1)setexdo [slot] 1 [data]

	→ setexdo 2 1 55 (2)setexdo [slot] [comport] [data] [baudrate] [address] → setexdo 0 3 55 9600 2 **Description**: (1)Set the dec digital output value to the module at slot 2 at COM1. (2)Set the dec digital output value to the module at slot 0 at COM3.
setexao	**Command:** (1)setexao [slot] 1 [value] [channel] → setexao 2 1 6.7 5 (2)setexao [slot] [comport] [value] [channel] [baudrate] [address] → setexao 0 3 6.7 5 9600 2 **Description**: (1)Set channel 5 analog value 6.7 to the module at slot 2. (2)Set channel 5 analog value 6.7 to the module at COM3.
getexdi	**Command:** (1)getexdi [slot] 1 → getexdi 2 1 (2)getexdi [slot] [comport] [baudrate] [address] → getexdi 0 3 9600 2 **Description**: (1)Get the dec digital input value from the module at slot 2. (2)Get the dec digital input value from the module at COM3.
getexai	**Command:** (1)getexai [slot] 1 [channel] → getexai 2 1 5 (2)getexai [slot] [comport] [channel] [baudrate] [address] → getexai 0 3 5 9600 2 **Description**: (1)Get channel 5 analog value from the module at slot 2. (2)Get channel 5 analog value from the module at COM3.

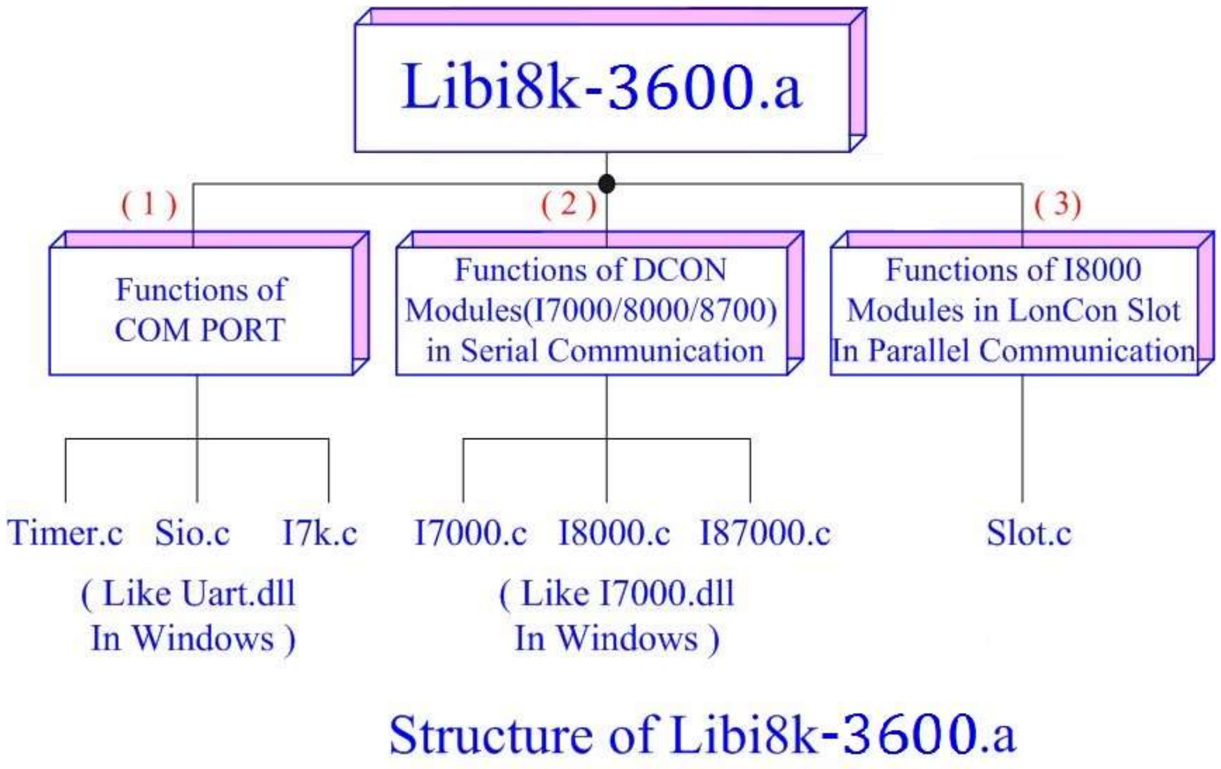
read_sn	**Command:** read_sn → read_sn **Description**: Show the serial number.
LX-8x31-rotary_id	**Command:** LX-8x31-rotary_id → LX-8x31-rotary_id **Description**: Read rotary switch ID of LX-8X31
ttyS1-config	**Command:** LX-8x31-rotary_id → LX-8x31-rotary_id **Description**: Read rotary switch ID of LX-8X31
iztconfig	**Command:** iztconfig → iztconfig **Description**: Using ZT-USBC communicate with ZT-2000 series 1.Loaded ftdi_sio driver first #modprobe ftdi_sio 2.Add ZT-USBC device id # echo "1b5c 0210" > /sys/bus/usb-serial/drivers/ftdi_sio/new_id

Table 5-2

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

6. LIBI8K-3600.A

In this section, we will focus on examples for the description of and application of the functions found in the Libi8k.a. The Libi8k-3600.a functions can be clarified into 3 groups which are listed in Fig. 6-1



Libi8k-3600.a
(1)
(2)
(3)
Functions of COM PORT
Functions of DCON Modules(I7000/8000/8700) in Serial Communication
Functions of I8000 Modules in LonCon Slot In Parallel Communication
Timer.c
Sio.c
I7k.c
I7000.c
I8000.c
I87000.c
Slot.c
(Like Uart.dll In Windows)
(Like I7000.dll In Windows)
Structure of Libi8k-3600.a

Fig. 6-1

Functions (1) and (2) in the Libi8k-3600.a are the same as with the DCON.DLL Driver (including Uart.dll and I7000.dll) as used in the DCON modules (I-7000 / I-8000 / I-87000 in serial communication). You can refer to the DCON.DLL Driver manual which includes the functions on how to use DCON modules. The DCON.DLL Driver has already been wrapped into the Libi8k-3600.a. Functions (3) of the Libi8k-3600.a consist of the most important functions as they are especially designed for I-8000 modules in the LX-8X31 slots. They are different from functions (1) and (2) because the communication of I-8000 modules in the LX-8X31 slots are parallel and not serial. Therefore ICP DAS rewrote I8000.c to “slot.c” especially for I-8000 modules in the LX-8X31 slots. Here we will introduce all the functions for “slot.c” and they can be divided into eight parts for ease of use.

1. System Information Functions;
2. Digital Input Functions;
3. Digital Output Functions;
4. Watch Dog Timer Functions;
5. EEPROM Read/Write Functions;
6. Analog Input Functions;
7. Analog Output Functions;

When using the development tools to develop applications, the “**msw.h”** file must be included in front of the source program, and when building applications, **Libi8k-3600.a** must be linked. If you want to control ICP DAS I/O remote modules like i7k, i8k and i87k **through ttyS0 or ttyS1 or ttyS34 of the LinPAC**, the functions are all the same with DCON DLL. And if you want to control **i8k modules** that are plugged in the slots of the LinPAC, then the functions are different and they are described as below:

6.1 System Information Functions

ChangeToSlot

Description:

This function is used to dedicate serial control to the specified slots for the control of the I-87k series. The serial bus in the LX-8X31 Series backplane is for mapping through to ttySA4. For example, if you want to send or receive data from a specified slot, you need to call this function first. Then you can use the other series functions.

Syntax:

```
[ C ]

void ChangeToSlot(char slot)
```

Parameter:

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

Return Value:

None

Example:

char slot=1;

ChangeToSlot (slot);

// The first slot is specified as ttySA4 port in LX-8X31.

Open_Slot

Description:

This function is used to open and initiate a specified slot in the LX-8X31. The 8k or I-87k modules in the LX-8X31 will use this function. For example, if you want to send or receive data from a specified slot, this function must be called first. Then the other functions can be used later.

Syntax:

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

Parameter:

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

Return Value:

0 is for Success

Not 0 is for Failure

Example:

Int slot=1;

Open_Slot(slot);

// The first slot in the LX-8X31 will be open and initiated.

Remark:

Close_Slot

Description:

If you have used the function of Open_Slot() to open the specified slot in the LX-8X31, you need to use the Close_Slot() function to close the specified slot in the LX-8X31. The 8k or I-87k modules in the LX-8X31 will use this function. For example, once you have finished sending or receiving data from a specified slot, this function would then need to be called.

Syntax:

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

Parameter:

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

Return Value:

None

Example:

```
Int slot=1;
Close_Slot(slot);
// The first slot in the LX-8X31 will be closed.
```

Remark:

Open_SlotAll

Description:

This function is used to open and initiate **all slots** in the LX-8X31. For example, if you want to send or receive data from multiple slots, you can call this function to simplify your program. Then you can use the other functions later.

Syntax:

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

Parameter:

None

Return Value:

0 is for Success

Not 0 is for Failure

Example:

Open_SlotAll();

// All slots in the LX-8X31 will be open and initiated.

Remark:

Close_SlotAll

Description:

If you have used the function Open_SlotAll() to open all the slots in the LX-8X31, you can use the Close_SlotAll() function to close all the slots in the LX-8X31. For example, once you are finish sending or receiving data from many slots, this function can be called to close all the slots rapidly.

Syntax:

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

Parameter:

None

Return Value:

None

Example:

Close_Slot();

// All slots in the LX-8X31 will be closed.

Remark:

GetModuleType

Description:

This function is used to retrieve which type of 8000 series I/O module is plugged into a specific I/O slot in the LX-8X31. This function performs a supporting task in the collection of information related to the system's hardware configurations.

Syntax:

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

Parameter:

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

Return Value:

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

Example:

```
char slot=1;
int moduleType;
Open_Slot(slot);
moduleType=GetModuleType(slot);
Close_Slot(Slot);
// The I-8057 card is plugged in slot 1 of LX-8X31 and has a return Value : 20
```

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 LX-8X31 . This function supports the collection of system hardware configurations.

Syntax:

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

Parameter:

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

Return Value:

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

Example:

```
char slot=1;
int moduleName;
Open_Slot(slot);
moduleID=GetNameOfModule(slot);
Close_Slot(Slot);
// The I-8017 card plugged in slot 1 of LX-8X31
// Returned Value: moduleName=" 8017 "
```

Remark:

6.2 Digital Input/Output Functions

DO_8

Description:

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

Syntax:

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Example:

```
int slot=1;
unsigned char data=3;
DO_8(slot, data);
// The I-8064 card is plugged in slot 1 of LX-8X31 and can turn on channel 0
// and 1.
```

Remark:

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

DO_16

Description:

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

Syntax:

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Example:

int slot=1;

unsigned int data=3;

DO_16(slot, data);

// The I-8057 card is plugged in slot 1 of LX-8X31 and can turn on channel 0 // and 1.

Remark:

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

DO_32

Description:

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

Syntax:

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Example:

```
int slot=1;
unsigned int data=3;
DO_32(slot, data);
// The I-8041 card is plugged in slot 1 of LX-8X31 and can turn on channel 0
// and 1.
```

Remark:

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

DI_8

Description:

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

Syntax:

```
[ C ]
unsigned char DI_8(int slot)
```

Parameter:

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

Return Value:

Input data

Example:

```
int slot=1;
unsigned char data;
data=DI_8(slot);
```

```
// The I-8058 card is plugged in slot 1 of LX-8X31 and has inputs in
// channel 0 and 1.
// Returned value: data=0xfC
```

Remark:

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

DI_16

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

Input data

Example:

```
int slot=1;
unsigned int data;
data=DI_16(slot);
// The I-8053 card is plugged in slot 1 of LX-8X31 and has inputs in
// channel 0 and 1.
// Returned value: data=0xfffC
```

Remark:

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

DI_32

Description:

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

Syntax:

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

Parameter:

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

Return Value:

Input data

Example:

```
int slot=1;
unsigned long data;
data=DI_32(slot);
// The I-8040 card plugged is in slot 1 of LX-8X31 and has inputs in
// channels 0 and 1.
// Returned value: data=0xfffffffC
```

Remark:

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

DIO_DO_8

Description:

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

Syntax:

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Example:

```
int slot=1;
unsigned char data=3;
DIO_DO_8(slot, data);
// The I-8054 card is plugged in slot 1 of LX-8X31 and can turn on channels 0
// and 1.
// It not only outputs a value, but also shows 16LEDs.
```

Remark:

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

DIO_DO_16

Description:

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

Syntax:

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Example:

```
int slot=1;
unsigned int data=3;
DIO_DO_16(slot, data);
// The I-8042 card is plugged in slot 1 of LX-8X31 and can turn on the
// channels 0 and 1.
// It not only outputs a value, but also shows 32LEDs.
```

Remark:

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

DIO_DI_8

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

Input data

Example:

int slot=1;

unsigned char data;

data=DIO_DI_8(slot);

// The I-8054 card is plugged in slot 1 of LX-8X31 and has inputs in

// channel 0 and 1.

// Returned value: data=0xfC

Remark:

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

DIO_DI_16

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

Input data

Example:

int slot=1;

unsigned char data;

data=DIO_DI_16(slot);

// The I-8042 card is plugged in slot 1 of LX-8X31 and has inputs in

// channel 0 and 1.

// Returned value: data=0xfffC

Remark:

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

DO_8_RB、DO_16_RB、DO_32_RB

DIO_DO_8_RB、DIO_DO_16_RB

Description:

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

Syntax:

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

Parameter:

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

Return Value:

all DO channel status

Example:

```
int slot=1;
Open_Slot(slot);
printf("%u",DO_32_RB(slot));
Close_Slot(slot);
// The I-8041 module is plugged in slot 1 of LX-8X31 and return all DO channel status.
(Return value：From 0 ~ 4294967295 for 32 channel)
```

Remark:

These functions can be applied on modules:

DO 8 channel：I-8060, I-i8064, I-8065, I-8066, I-8068 and I-8069.

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

DO 32 channel：I-8041

DO_8_BW、DO_16_ BW、DO_32_ BW

DIO_DO_8_ BW、DIO_DO_16_ BW

Description:

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

Syntax:

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

Parameter:

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

bit : [Input] channel of module.

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

Return Value:

None

Example:

```
int slot=1, bit=0, data=1;
Open_Slot(slot);
DO_32_BW(slot, bit, data);
Close_Slot(slot);
// The I-8041 module is plugged in slot 1 of LX-8X31 and just turn on channel 0 of
```

I-8041.

Remark:

These functions can be applied on modules:

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

DO 16 channel：I-i8037, I-8056 and I-8057

DO 32 channel：I-8041

DI_8_BW、DI_16_ BW、DI_32_ BW

Description:

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

Syntax:

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

Parameter:

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

bit : [Input] channel of module.

Return Value:

None

Example:

```
int slot=1, bit=0;
Open_Slot(slot);
printf("DI channel %d = %d\n", bit, DI_32_BW(slot, bit));
Close_Slot(slot);
// The I-8040 module is plugged in slot 1 of LX-8X31 and return channel 0
// status. ( 0：ON ; 1：OFF ).
```

Remark:

These functions can be applied on modules:

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

DI 16 channel：I-8051 and I-8053

DI 32 channel：I-8040

UDIO_WriteConfig_16

Description:

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

Syntax:

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

Parameter:

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

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

Return Value:

None

Example:

```
int slot=1;
unsigned short config=0xffff;
UDIO_WriteConfig_16(slot, config);
// The I-8064 card is plugged in slot 1 of LX-8X31 Series.
// WriteConfig: 0xffff (ch 0~ch15 is DO mode)
```

Remark:

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

UDIO_ReadConfig_16

Description:

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

Syntax:

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

Parameter:

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

Return Value:

None

Example:

```
int slot=1;
unsigned int ret;
unsigned short config=0x0000;
UDIO_WriteConfig_16(slot, config);
ret=UDIO_ReadConfig_16(slot);
printf("Read the I/O Type is : 0x%04lx \n\r",ret);
// The I-8050 card is plugged in slot 1 of LX-8X31.
// WriteConfig: 0x0000 (ch 0~ch15 is DI mode)
// Read the I/O Type is: 0x0000
```

Remark:

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

UDIO_DO16

Description:

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

Syntax:

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

Parameter:

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

config : [Input] output data.

Return Value:

None

Example:

```
int slot=1;
unsigned int data;
unsigned short config =0x00ff;
UDIO_WriteConfig_16(slot, config);
scanf("%d:",&data);
UDIO_DO16(slot, data);
printf("DO(Ch0~Ch7) of I-8050 in Slot %d = 0x%x\n\r",slot, data);
// The I-8050 card is plugged in slot 1 of LX-8X31.
// WriteConfig: 0x00ff (ch 0~ch7 is DO mode and ch8~ch15 is DI mode)
// Input DO value: 255
// DO(Ch0~Ch7) of I-8050 in Slot 1 = 0xff
```

Remark:

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

UDIO_DI16

Description:

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

Syntax:

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

Parameter:

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

Return Value:

None

Example:

```
int slot=1;
unsigned int data;
unsigned short config =0xff00;
UDIO_WriteConfig_16(slot, config);
data=UDIO_DI16(slot);
printf("DI(Ch0~Ch7) of I-8055 in Slot %d = 0x%x\n\r",slot, data);
scanf("%d:",&data);
UDIO_DO16(slot, data);
printf("DO(Ch8~Ch15) of I-8050 in Slot %d = 0x%x\n\r",slot, data);
// The I-8050 card is plugged in slot 1 of LX-8X31.
// WriteConfig: 0x00ff (ch 0~ch7 is DI mode and ch8~ch15 is DO mode)
// DI(Ch0~Ch7) of I-8055 in Slot 1 = 0xfbff
// Input DO value: 255
// DO(Ch8~Ch15) of I-8050 in Slot 1 = 0xff
```

Remark:

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

6.3 Watch Dog Timer Functions

EnableWDT

Description:

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

Syntax:

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

Parameter:

msecond:LinPAC will reset in the assigned time if users don’t reset WDT.

The unit is mini-second.

Return Value:

None

Example:

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

Remark:

DisableWDT

Description:

This function is used to disable WDT.

Syntax:

```
[C]
void DisableWDT(void)
```

Parameter:

None

Return Value:

None

Example:

Remark:

WatchDogSWEven

Description:

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

Syntax:

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

Parameter:

None

Return Value:

Just see the last number of the return value – **RCSR** (Reset Controller Status

Register). For example: RCSR is “20009a4”, so just see the last number “4”. 4 is **0100** in bits and it means:

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

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

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

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

Example:

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

Remark:

ClearWDTSWEven

Description:

This function is used to clear RCSR value.

Syntax:

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

Parameter:

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

1 : clear bit 0

2 : clear bit 1

4 : clear bit 2

8 : clear bit 3

F : clear bit 0 ~ bit 3

Return Value:

None

Example:

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

Remark:

6.4 EEPROM Read/Write Functions

Enable_EEP

Description:

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

Syntax:

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

Parameter:

None

Return Value:

None

Example:

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

Remark:

Disable_EEP

Description:

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

Syntax:

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

Parameter:

None

Return Value:

None

Example:

Disable_EEP();

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

Remark:

Read_EEP

Description:

This function will read one byte data from the EEPROM. There is a 16K-byte EEPROM in the main control unit in the LX-8X31 system. This EEPROM is divided into 256 blocks (0 to 255), and each block is 64 bytes in length from offset 0 to 63. This EEPROM with its accessing APIs provides another mechanism for storing critical data inside non-volatile memory.

Syntax:

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

Parameter:

block : [Input] the block number of EEPROM.

offset : [Input] the offset within the block.

Return Value:

Data read from the EEPROM.

Example:

int block, offset;

unsigned char data;

data= ReadEEP(block, offset);

// Returned value: data= read an 8-bit value from the EEPROM (block & offset)

Remark:

Write_EEP

Description:

To write one byte of data to the EEPROM. There is a 16K-byte EEPROM in the main control unit of the LX-8X31 system. This EEPROM is divided into 256 blocks (0 to 255), and each block is 64 bytes in length from the offset of 0 to 63. This EEPROM with its accessing APIs, provides another mechanism for storing critical data inside non-volatile memory.

Syntax:

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

Parameter:

block : [Input] the block number of EEPROM.

offset : [Input] the offset within the block.

Data : [Input] data to write to EEPROM.

Return Value:

None

Example:

int block, offset;

unsigned char data=10;

WriteEEP(block, offset, data);

// Writes a 10 value output to the EEPROM (block & offset) location

Remark:

6.5 Analog Input Functions

I8017_Init

Description:

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

Syntax:

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

Parameter:

slot : [Input] specified slot of the LX-8X31 system (Range: 1 to 7)

Return Value:

The version of library

Example:

```
int slot=1,ver;
ver=I8017_Init(slot);
// The I-8017H card is plugged in slot 1 of LX-8X31 and initializes the module I-8017H.
```

Remark:

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

I8017_SetLed

Description:

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

Syntax:

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

Parameter:

slot : [Input] specified slot of the LX-8X31 system (Range: 1 to 7)

led : [Input] range from 0 to 0xffff

Return Value:

None

Example:

int slot=1;

unsigned int led=0x0001;

I8017_SetLed (slot, led);

// There will be a L/A-LED light on channel 0 of the I-8017H card which is plugged in slot 1 on the LX-8X31.

Remark:

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

I8017_SetChannelGainMode

Description:

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

Syntax:

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

Parameter:

slot : [Input] Specify the slot in the LX-8X31 system (Range: 1 to 7)

ch : [Input] Specify the I-8017H channel (Range: 0 to 7)

Specify the I-8017HS channel (Range: 0 to 15)

gain : [Input] input range:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

mode : [Input] **0: normal mode** (polling)

Return Value:

None

Example:

```
int slot=1,ch=0,gain=0;
I8017_SetChannelGainMode (slot, ch, gain,0);
// The I-8017H card is plugged in slot 1 of LX-8X31, and the range of the data
// value from channel 0 for I-8017H will be -10 ~ +10 V.
```

Remark:

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

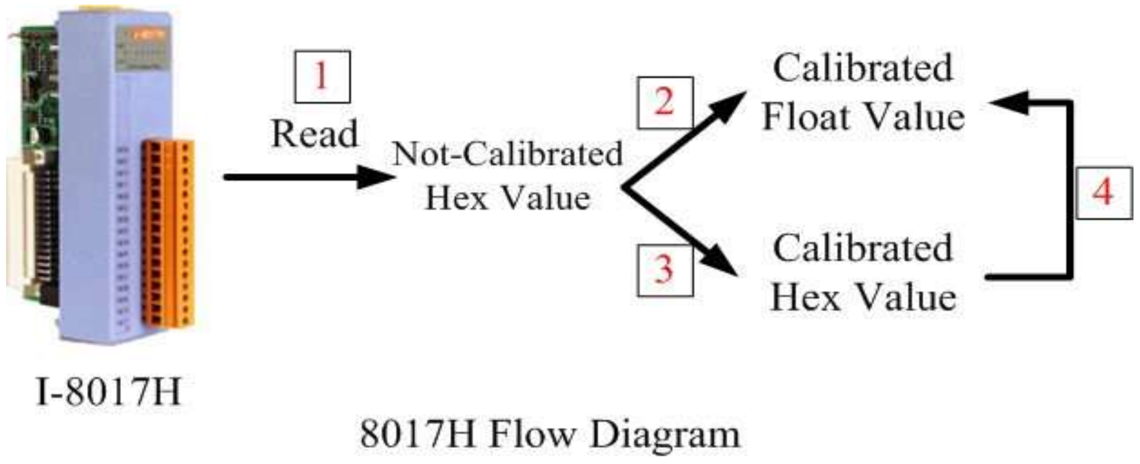


Fig.6-2

Function of [1]

I8017_GetCurAdChannel_Hex

Description:

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

Syntax:

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

Parameter:

slot : [Input] specified slot of the LX-8X31 Series system (Range: 1 to 7)

Return Value:

The analog input value in Hex format.

Example:

```
int slot=1,ch=0,gain=4;
int data;
I8017_SetChannelGainMode (slot, ch, gain,0);
data= I8017_GetCurAdChannel_Hex (slot);
// The I-8017H card is plugged into slot 1 of LX-8X31 and the range of the data
// value from channel 0 in I-8017H is +/- 20mA
```

I8017_AD_POLLING

Description:

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

Syntax:

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

Parameter:

slot : [Input] Specified slot in the LX-8X31 system (Range: 1 to 7)

ch : [Input] Specified channel for I-8017H (Range: 0 to 7)

Specified channel for I-8017HS (Range: 0 to 15)

gain : [Input] Input range:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

datacount : [Input] Range from 1 to 8192, total ADCs number

*DataPtr : [Output] The starting address of data array[ ] and the array size must be equal to or bigger than the datacount.

Return Value:

0 : indicates success.

Not 0 : indicates failure.

Example:

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

Function of [2]

HEX_TO_FLOAT_Cal

Description:

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

Syntax:

[C]

```
float HEX_TO_FLOAT_Cal(int HexValue, int slot, int gain)
```

Parameter:

HexValue : [Input] specified not-calibrated HexValue before converting

slot : [Input] specified slot of the LX-8X31 system (Range: 1 to 7)

gain : [Input] Input range:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

Return Value:

The Calibrated Float Value.

Example:

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

Remark:

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

ARRAY_HEX_TO_FLOAT_Cal

Description:

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

Syntax:

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

Parameter:

*HexValue : [Input] data array in not-calibrated Hex type before converting

*FloatValue : [Output] Converted data array in calibrated float type

slot : [Input] specified slot of the LX-8X31 system (Range: 1 to 7)

gain : [Input] Input range:

len : [input] ADC data length

Return Value:

None

Example:

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

Remark:

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

Function of [3]

I8017_Hex_Cal

Description:

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

Syntax:

```
[ C ]
```

```
int I8017_Hex_Cal(int data)
```

Parameter:

data : [Input] specified not-calibrated hex value

Return Value:

The Calibrated Hex Value.

Example:

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

Remark:

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

I8017_Hex_Cal_Slot_Gain

Description:

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

Syntax:

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

Parameter:

slot : [Input] specified slot of the LX-8X31 system (Range: 1 to 7)

gain : [Input] Input range:

data : [Input] specified not-calibrated hex value

Return Value:

The Calibrated Hex Value.

Example:

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

Remark:

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

Function of [4]

CalHEX_TO_FLOAT

Description:

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

Syntax:

[C]

```
float CalHex_TO_FLOAT(int HexValue,int gain)
```

Parameter:

HexValue : [Input] specified not-calibrated HexValue before converting

gain : [Input] Input range:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

Return Value:

The Calibrated Float Value.

Example:

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

Remark:

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

ARRAY_CalHEX_TO_FLOAT

Description:

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

Syntax:

[C]

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

Parameter:

*HexValue : [Input] data array in calibrated Hex format

*FloatValue : [Output] Converted data array in calibrated float format

gain : [Input] Input range:

len : [input] ADC data length

Return Value:

The Calibrated Float Value.

Example:

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

Remark:

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

Function of [1] + [3]

I8017_GetCurAdChannel_Hex_Cal

Description:

Obtain the calibrated analog input values in the Hex format directly from the analog input modules, I-8017H. This function is a combination of the “I8017_GetCurAdChannel_Hex” function and the “I8017_Hex_Cal”. Please refer to Fig 6-2

Syntax:

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

Parameter:

slot : [Input] specified slot of the LX-8X31 system (Range: 1 to 7)

Return Value:

The analog input value in Calibrated Hex format.

Example:

```
int slot=1,ch=0,gain=0, data;
I8017_SetChannelGainMode (slot, ch, gain,0);
data = I8017_GetCurAdChannel_Hex_Cal (slot);
// The I-8017H card is plugged into slot 1 of LX-8X31 and the range of the
// data value from channel 0 in I-8017H is 0x0000 ~ 0x3fff.
```

Remark:

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

I8017_AD_POLLING_Cal

Description:

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

Syntax:

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

Parameter:

slot : [Input] Specified slot in the LX-8X31 system (Range: 1 to 7)

ch : [Input] Specified channel for I-8017H (Range: 0 to 7)

Specified channel for I-8017HS (Range: 0 to 15)

gain : [Input] Input range:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

datacount : [Input] Range from 1 to 8192, total ADCs number

*DataPtr : [Output] The starting address of data array[ ] and the array size must be equal to or bigger than the datacount.

Return Value:

0 : indicates success.

Not 0 : indicates failure.

Example:

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

Function of [1]+[2]

I8017_GetCurAdChannel_Float_Cal

Description:

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

Syntax:

[C]

```
int I8017_GetCurAdChannel_Float_Cal(int slot)
```

Parameter:

slot : [Input] specified slot of the LX-8X31 system (Range: 1 to 7)

Return Value:

The analog input value in Calibrated Float format.

Example:

```
int slot=1,ch=0,gain=0;
float data;
I8017_SetChannelGainMode (slot, ch, gain,0);
data = I8017_GetCurAdChannel_Float_Cal (slot);
// The I-8017H card is plugged into slot 1 of LX-8X31 and the range of the
// data value from channel 0 in I-8017H is –10V ~ +10V.
```

Remark:

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

6.6 Analog Output Functions

I8024_Initial

Description:

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

Syntax:

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

Parameter:

slot : [Input] Specify the LX-8X31 system slot (Range: 1 to 7)

Return Value:

None

Example:

```
int slot=1;
I8024_Initial(slot);
// The I-8024 card is plugged into slot 1 of LX-8X31 and initializes the I-8024 module.
```

Remark:

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

I8024_VoltageOut

Description:

This function is used to send the voltage float value to the I-8024 module with the specified channel and slot in the LX-8X31 system.

Syntax:

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

Parameter:

slot : [Input] Specified the LX-8X31 system slot (Range: 1 to 7)

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

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

Return Value:

None

Example:

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

Remark:

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

I8024_CurrentOut

Description:

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

Syntax:

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

Parameter:

slot : [Input] Specify the LX-8X31 system slot (Range: 1 to 7) ch : [Input] Output channel (Range: 0 to 3)

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

Return Value:

None

Example:

int slot=1, ch=0;

float cdata=10.0f;

I8024_CurrentOut(slot, ch, data);

// Output the 10.0mA current from the channel 0 of I-8024 module.

Remark:

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

I8024_VoltageHexOut

Description:

This function is used to send the voltage value in the Hex format to the specified channel in the I-8024 module, which is plugged into the slot in the LX-8X31 system.

Syntax:

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

Parameter:

slot : [Input] Specify the LX-8X31 system slot (Range: 1 to 7)

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

hdata : [Input] Output data with hexadecimal

(data range: 0h ~ 3FFFh→ Voltage Output: -10. ~ +10. V)

Return Value:

None

Example:

int slot=1, ch=0; data=0x3000;

I8024_VoltageHexOut(slot, ch, data);

// The I-8024 module output the 5.0V voltage from the channel 0.

Remark:

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

I8024_CurrentHexOut

Description:

This function is used to send the current value in the Hex format to the specified channel in the analog output module I-8024, which is plugged into the slot in the LX-8X31 system.

Syntax:

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

Parameter:

slot : [Input] Specify the LX-8X31 system slot (Range: 1 to 7)

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

hdata : [Input] Output data with hexadecimal

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

Return Value:

None

Example:

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

Remark:

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

6.7 The Software Develop Toolkit Error Code

Error Code	Error ID
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	UnderRange
20	ExceedRange

21	InvalidateCounterNo
22	InvalidateCounterValue
23	InvalidateGateMode
24	InvalidateChannelNo
25	ComPortInUse

7. Demo of LX-8X31 Modules With C Language

In this section, we will focus on examples for the description and application of the control sections on the I-7000/I-8000/I-87K series modules for use in the LX-8X31. After user downloading the latest version of LX-8X31 SDK “LinPAC-SDK.tar.gz” from the ftp path (Please refer to chapter 2), then user could use command “**tar zxvf LinPAC-SDK.tar.gz**” to unzip SDK “**LinPAC-SDK.tar.gz**" in the LX-8X31 , user can find all demo programs (include demo source code and executes file) in the “examples” directory.

7.1 I-7k Modules DIO Control Demo

This demo – **i7kdio.c** will illustrate how to control DI/DO with the I-7050 module (8 DO channels and 7 DI channels). The address and baudrate of the I-7050 module in the RS-485 port are 02 and 9600 separately.

The result of this demo allows the DO channels 0 ~ 7 output and DI channel 2 input. The source code of this demo program is as follows:

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80], ans;
WORD wBuf[12];
float fBuf[12];

/* ------------------------------------------------------------------ */
int main()
{
      int   wRetVal;

      // Check Open_Com3
      wRetVal = Open_Com(ttyS1, 9600, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
          printf("open port failed!\n");
          return (-1);
      }
      //    *****    7050 DO && DI Parameter *******
      wBuf[0] = ttyS1;                    // COM Port
      wBuf[1] = 0x02;                     // Address
      wBuf[2] = 0x7050;                   // ID
      wBuf[3] = 0;                        // CheckSum disable
      wBuf[4] = 100;                      // TimeOut , 100 msecond
      wBuf[5] = 0x0ff;                    // 8 DO Channels On
      wBuf[6] = 0;                        // string debug

      // 7050 DO Output
          wRetVal = DigitalOut(wBuf, fBuf, szSend, szReceive);
          if (wRetVal)
              printf("DigitalOut_7050 Error !, Error Code=%d\n", wRetVal);
          printf("The DO of 7050 : %u \n", wBuf[5]);

      // 7050 DI Input
          DigitalIn(wBuf, fBuf, szSend, szReceive);
          printf("The DI of 7050 : %u \n", wBuf[5]);
      Close_Com(ttyS1);
  return 0;
}
```

Follow the below steps to achieve the desired results：

STEP 1：(To transmit "LinPAC-SDK.tar.gz" SDK to LX-8X31)

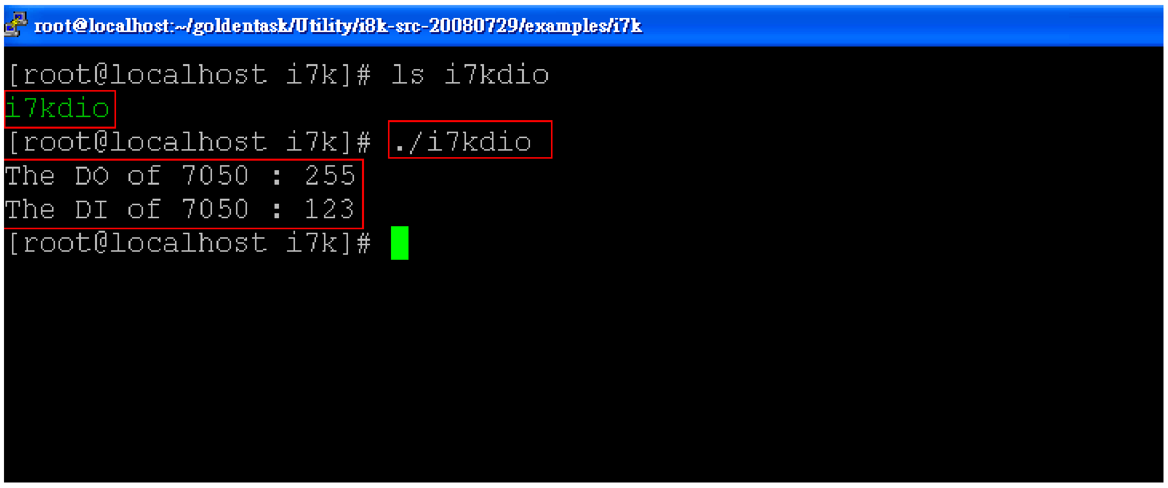
User could use WinSCP software or linux "scp" command to transmit **"LinPAC-SDK.tar.gz"** from Windows or Linux PC to LX-8X31, or you can use command "wget http://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x8x/atom/sdk/linpac-8X81_sdk.tar.gz" to download LinPAC-SDK.tar.gz in LX-8X31.

STEP 2：(To connect to LX-8X31 to execute i7kdio)

User could use putty software to connect to LX-8X31. User could refer to the "4.LX-8X31 System Settings" to know how to use Putty to connect to LX-8X31.

STEP 3：(To execute i7kdio in the “i8k-LinPAC/examples/lx-8x31/i7k)

After users connect to LX-8X31 , user could find the demo “i7kdio” in the directory **“i8k-LinPAC/examples/lx-8x31/i7k**. The result of execution refers to Fig. 7-1.



```
root@localhost:~/goldentask/Utility/i8k-src-20080729/examples/i7k
[root@localhost i7k]# ls i7kdio
i7kdio
[root@localhost i7k]# ./i7kdio
The DO of 7050 : 255
The DI of 7050 : 123
[root@localhost i7k]#
```

Fig. 7-1

“ **The DO of I-7050：255** (=2^**8**-1)” means DO channel 0 ~ 7 will output and “ **The DI of I-7050：123** (=127-2^**2**)” means there is input in DI channel 2.

7.2 I-7k Modules AIO Control Demo

This demo – **i7kaio.c** will illustrate how to control the AI/AO with the I-7017 (8 AI channels) and I-7021 modules (1 AO channel). The address for the I-7021 and I-7017 modules are in the RS-485 network where 05 and 03 are separate and the baudrate is 9600.

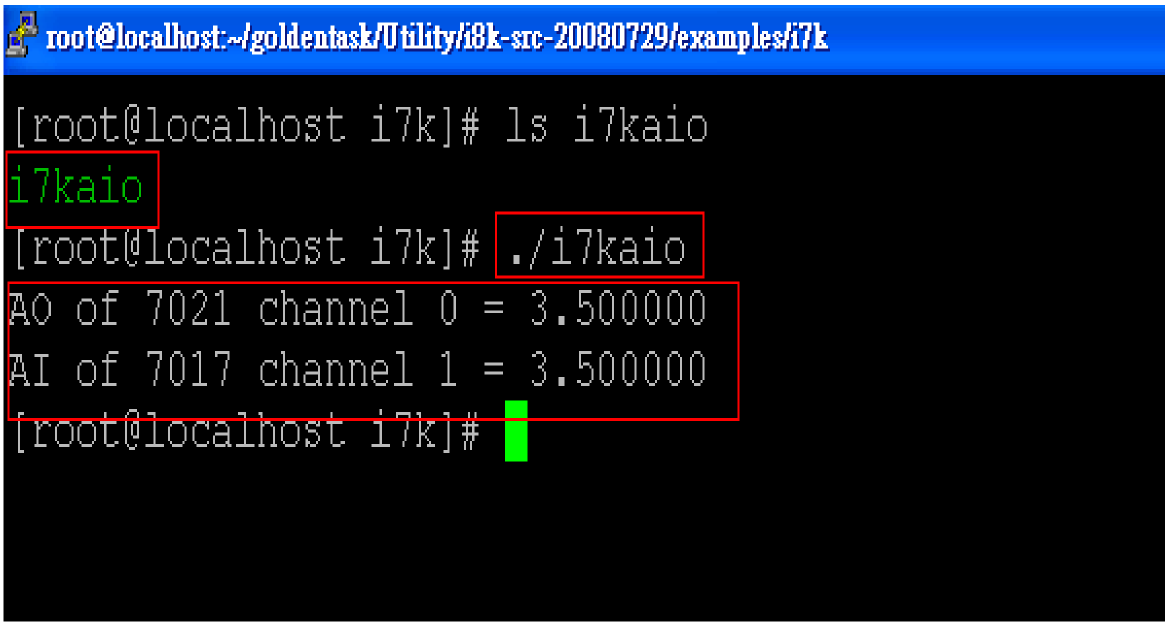
The result of this demo allows the I-7021 module’s AO channel to output 3.5V and the I-7017‘s AI channel 2 to input. The source code of this demo program is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
```

```
WORD wBuf[12];
float fBuf[12];

/* --------------------------------------------------------------- */
int main()
{
      int i,j, wRetVal;
      DWORD temp;

      wRetVal = Open_Com(ttyS1, 9600, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed!\n");
            return (-1);
      }

      //--- Analog output ----   ****   7021 -- AO   ****
      i = 0;
      wBuf[0] = ttyS1;            // COM Port
      wBuf[1] = 0x05;             // Address
      wBuf[2] = 0x7021;           // ID
      wBuf[3] = 0;                // CheckSum disable
      wBuf[4] = 100;              // TimeOut , 100 msecond
      //wBuf[5] = i;              // Not used if module ID is 7016/7021
                                  // Channel No.(0 to 1) if module ID is 7022
                                  // Channel No.(0 to 3) if module ID is 7024
      wBuf[6] = 0;                // string debug
      fBuf[0] = 3.5;               // Analog Value

      wRetVal = AnalogOut(wBuf, fBuf, szSend, szReceive);
      if (wRetVal)
            printf("AO of 7021 Error !, Error Code=%d\n", wRetVal);
      else
            printf("AO of 7021 channel %d = %f \n",i,fBuf[0]);

      //--- Analog Input ----   ****   7017 -- AI   ****
      j = 1;
      wBuf[0] = ttyS1;            // COM Port
      wBuf[1] = 0x03;             // Address
```

```
        wBuf[2] = 0x7017;           // ID
        wBuf[3] = 0;                // CheckSum disable
        wBuf[4] = 100;              // TimeOut , 100 msecond
        wBuf[5] = j;                // Channel of AI
        wBuf[6] = 0;                // string debug

        wRetVal = AnalogIn(wBuf, fBuf, szSend, szReceive);
        if (wRetVal)
            printf("AI of 7017 Error !, Error Code=%d\n", wRetVal);
        else
            printf("AI of 7017 channel %d = %f \n",j,fBuf[0]);

        Close_Com(ttyS1);

        return 0;
}
```

All the steps from programming to execution are the same as those in the section 7.1. The result of execution refers to Fig. 7-2.



```
root@localhost:~/goldentask/Utility/i8k-src-20080729/examples/i7k
[root@localhost i7k]# ls i7kaio
i7kaio
[root@localhost i7k]# ./i7kaio
AO of 7021 channel 0 = 3.500000
AI of 7017 channel 1 = 3.500000
[root@localhost i7k]#
```

Fig. 7-2

7.3 I-87k Modules DIO Control Demo

When using I-87k modules for I/O control of the LX-8X31, the program will be a little different, according to the location of I-87k modules. There are three conditions for the location of the I-87k modules:

(1) When I-87k DIO modules are **in the LX-8X31 slots**, the two functions “ Open_Slot ” and “ ChangeToSlot ”, must be added before using other functions for the I-87k modules and the function of “Close_Slot() “ also needs to be added to the end of the program. Please refer to demo in section 7.3.1.

(2) When I-87K DIO modules are **in the I-87k I/O expansion unit slots**, then please refer to the demo in section 7.3.2.

(3) When the I-87k DIO modules are **in the I-8000 controller slots**, then the I-87k modules will be regarded as I-8k modules and so please refer to I/O control of I-8k modules in section 7.5.2

7.3.1 I-87k Modules in slots of LX-8X31

This demo – **i87kdio.c** will illustrate how to control the DI/DO with the I-87054 module (8 DO channels and 8 DI channels). The I-87054 module is in slot 3 of the LX-8X31. The address and baudrate in the LX-8X31 are constant and they are 00 and 115200 respectively. The result of this demo lets DO channel 0 ~ 7 of I-87054 output and DI channel 1 of I-87054 input. The source code of this demo program is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
DWORD dwBuf[12];
float fBuf[12];

/* ---------------------------------------------------------------- */
int main()
{
```

```
          int i, wRetVal;
          DWORD temp;

          //Check Open_Slot
          wRetVal = Open_Slot(0);
          if (wRetVal > 0) {
               printf("open Slot failed!\n");
               return (-1);
          }

          //Check Open_Com1
          wRetVal = Open_Com(ttySA4, 115200, Data8Bit, NonParity,
OneStopBit);
          if (wRetVal > 0) {
               printf("open port failed!\n");
               return (-1);
          }

          //Choose Slot3
          ChangeToSlot(3);
          //--- digital output ----   **(DigitalOut_87k()**)
          dwBuf[0] = ttySA4                // COM Port
          dwBuf[1] = 00;                   // Address
          dwBuf[2] = 0x87054;              // ID
          dwBuf[3] = 0;                    // CheckSum disable
          dwBuf[4] = 100;                  // TimeOut , 100 msecond
          dwBuf[5] = 0xff;                 // digital output
          dwBuf[6] = 0;                    // string debug
        wRetVal = DigitalOut_87k(dwBuf, fBuf, szSend, szReceive); // DO Output
        printf("DO Value= %u", dwBuf[5]);

        //--- digital Input ----    **(DigitalIn_87k()**)
          dwBuf[0] = ttySA4;               // COM Port
          dwBuf[1] = 00;                   // Address
          dwBuf[2] = 0x87054;              // ID
          dwBuf[3] = 0;                    // CheckSum disable
          dwBuf[4] = 100;                  // TimeOut , 100 msecond
          dwBuf[6] = 0;                       // string debug
          getch();
          DigitalIn_87k(dwBuf, fBuf, szSend, szReceive);     // DI Input
         printf("DI= %u",dwBuf[5])
          //--- digital output ----      ** Close DO **
          dwBuf[0] = ttySA4;                  // COM Port
          dwBuf[1] = 00;                      // Address
          dwBuf[2] = 0x87054;                 // ID
          dwBuf[3] = 0;                       // CheckSum disable
          dwBuf[4] = 100;                     // TimeOut , 100 msecond
          dwBuf[5] = 0x00;                    // digital output
          dwBuf[6] = 0;                       // string debug
          getch();                           // push any key to continue
         wRetVal = DigitalOut_87k(dwBuf, fBuf, szSend, szReceive);

          Close_Com(ttySA4);
          Close_SlotAll();
          return 0;
      }
```

7.3.2 I-87k Modules in slots of I-87k I/O expansion unit

If the I-87k modules are in the slots of the I-87k I/O expansion unit, the above program needs to be modified in three parts：

(1) The functions of **Open_Slot()** , **ChangeToSlot(), Close_SlotAll()** will be deleted.

(2) The **address** and **baudrate** of I-87k modules in the network of RS-485 need to be set by DCON Utility

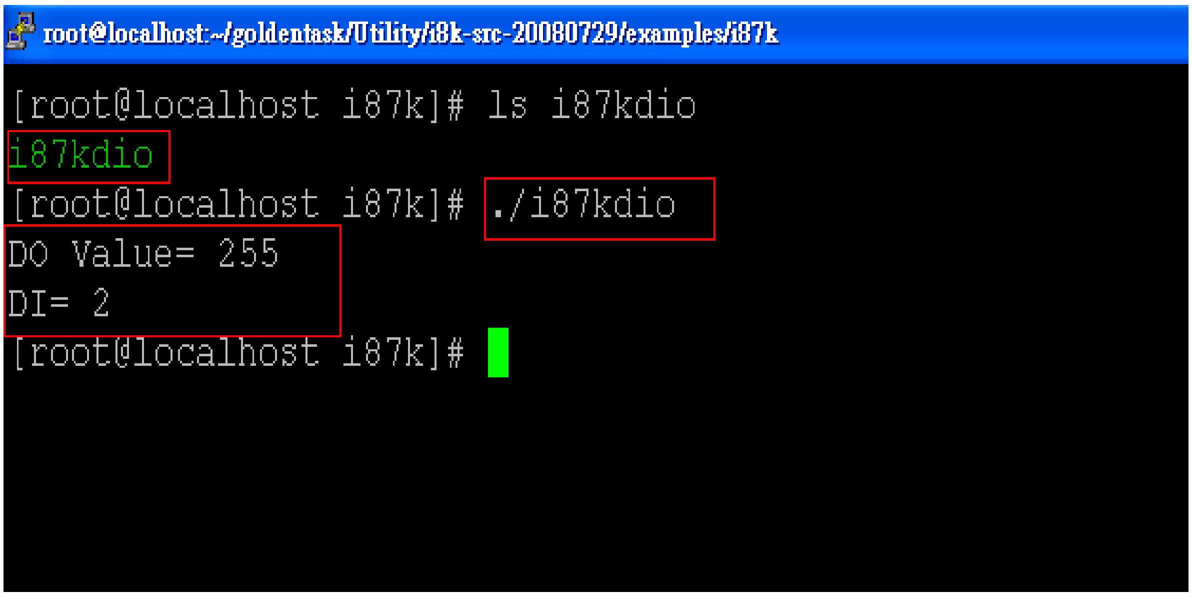
(3) **Open ttySA4**(internal serial port of LX-8X31) will be modified to **open ttyS1** (RS-485 port of LX-8X31)

The address and baudrate of the I-87054 in the RS-485 network are set to be 06 and 9600 separately by the DCON Utility. The source code of this demo program – **i87kdio_87k.c** is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
DWORD dwBuf[12];
float fBuf[12];

/* --------------------------------------------------------------- */
int main()
{
      int i, wRetVal;
      DWORD temp;
      //Check Open_Com3
      wRetVal = Open_Com(ttyS1, 9600, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed!\n");
            return (-1);
      }
      //--- digital output ----    **(DigitalOut_87k()**)
      dwBuf[0] = ttyS1;                   // COM Port
      dwBuf[1] = 06;                // Address
      dwBuf[2] = 0x87054;           // ID
      dwBuf[3] = 0;                 // CheckSum disable
      dwBuf[4] = 100;               // TimeOut , 100 msecond
      dwBuf[5] = 0xff;              // digital output
      dwBuf[6] = 0;                 // string debug
   wRetVal = DigitalOut_87k(dwBuf, fBuf, szSend, szReceive);   // DO Output
   printf("DO Value= %u", dwBuf[5]);

   //--- digital Input ----     **(DigitalIn_87k()**)
      dwBuf[0] = ttyS1;                   // COM Port
      dwBuf[1] = 06;                // Address
      dwBuf[2] = 0x87054;           // ID
      dwBuf[3] = 0;                 // CheckSum disable
      dwBuf[4] = 100;               // TimeOut , 100 msecond
      dwBuf[6] = 0;                 // string debug
      getch();
```

```
        DigitalIn_87k(dwBuf, fBuf, szSend, szReceive);     // DI Input
    printf("DI= %u",dwBuf[5]);
        //--- digital output ----     ** Close DO **
        dwBuf[0] = ttyS1;                       // COM Port
        dwBuf[1] = 06;                  // Address
        dwBuf[2] = 0x87054;             // ID
        dwBuf[3] = 0;                   // CheckSum disable
        dwBuf[4] = 100;                 // TimeOut , 100 msecond
        dwBuf[5] = 0x00;                // digital output
        dwBuf[6] = 0;                   // string debug
        getch();                        // push any key to continue
    wRetVal = DigitalOut_87k(dwBuf, fBuf, szSend, szReceive);

    Close_Com(ttyS1);
return 0;
}
```

All the steps from programming to execution are the same as those in the section 7.1. The result of execution refers to Fig. 7-3.


root@localhost:~/goldentask/Utility/i8k-src-20080729/examples/i87k

```
[root@localhost i87k]# ls i87kdio
i87kdio
[root@localhost i87k]# ./i87kdio
DO Value= 255
DI= 2
[root@localhost i87k]#
```

Fig. 7-3

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

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

7.4 I-87k Modules AIO Control Demo

When using I-87k modules for I/O control of the LX-8X31, according to the location of the I-87k modules, the program will be a little different. There are three conditions for the location of the I-87k modules：

(1) When the I-87k AIO modules are **in the LX-8X31 slots**, the two functions " Open_Slot " and " ChangeToSlot " must be added before using the other functions of the I-87k modules and the function " Close_Slot() " also needs to be added to the end of the program. Please refer to the demo in section 7.4.1.

(2) When I-87K AIO modules are **in the I-87k I/O expansion unit slots**, please refer to the demo in section 7.4.2.

(3) When the I-87k AIO modules are **in the I-8000 controller slots**, the I-87k modules will be regarded as I-8k modules and so please refer to I/O control of I-8k modules in section 7.4.3.

7.4.1 I-87k Modules in slots of LX-8X31

This demo – **i87kaio.c** will illustrate how to control the AI/AO with the I-87022 module (2 AO channels) and the I-87017 module (8 AI channels).The I-87022 and I-87017 modules are plugged into slot 2 and slot 3 of the LX-8X31 separately. The address and baudrate in the LX-8X31 are constant and they are 00 and 115200 separately. The result of this demo lets AO channel 0 of I-87022 output 2.5V and AI channel 1 of I-87017 input. The source code of this demo program is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
DWORD wBuf[12];
DWORD   wBuf7[12];
```

```
float fBuf[12];
/* --------------------------------------------------------------- */
int main()
{
      int i,j, wRetVal;
      DWORD temp;

    //Check Open_Slot
      wRetVal = Open_Slot(0);
      if (wRetVal > 0) {
            printf("open Slot failed!\n");
            return (-1);
      }

      //Check Open_Com1
      wRetVal = Open_Com(ttySA4, 115200, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed!\n");
            return (-1);
      }

      ChangeToSlot(2);
      //--- Analog output ----    ****     87022 -- AO    ****
      i=0;
      wBuf[0] = ttySA4;             // COM Port
      wBuf[1] = 0x00;               // Address
      wBuf[2] = 0x87022;            // ID
      wBuf[3] = 0;                  // CheckSum disable
      wBuf[4] = 100;                // TimeOut , 100 msecond
      wBuf[5] = i;                  // Channel Number of AO
      wBuf[6] = 0;                  // string debug
      fBuf[0] = 2.5;                 // AO Value

      wRetVal = AnalogOut_87k(wBuf, fBuf, szSend, szReceive);
      if (wRetVal)
            printf("AO of 87022 Error !, Error Code=%d\n", wRetVal);
      else
            printf("AO of 87022 channel %d = %f \n",i,fBuf[0]);
      ChangeToSlot(3);
      //--- Analog Input ----    ****     87017 -- AI    ****
      j=1;
      wBuf7[0] = ttySA4;          // COM Port
      wBuf7[1] = 0x00;            // Address
      wBuf7[2] = 0x87017;         // ID
      wBuf7[3] = 0;               // CheckSum disable
      wBuf7[4] = 100;             // TimeOut , 100 msecond
      wBuf7[5] = j;               //Channel Number of AI
      wBuf7[6] = 0;               // string debug

      wRetVal = AnalogIn_87k(wBuf7, fBuf, szSend, szReceive);
      if (wRetVal)
            printf("AI of 87017 Error !, Error Code=%d\n", wRetVal);
      else
            printf("AI of 87017 channel %d = %f \n",j,fBuf[0]);

      Close_Com(ttySA4);
      Close_SlotAll();
      return 0;
}
```

7.4.2 I-87k Modules in slots of I-87k I/O expansion unit

If the I-87k modules are in slots of I-87k I/O expansion unit, the above program needs to be modified in three parts：

(1) The functions of **Open_Slot()** , **ChangeToSlot(), Close_SlotAll()** will be deleted.

(2) The **address** and **baudrate** of I-87k modules in the network of RS-485 need to be set by DCON Utility

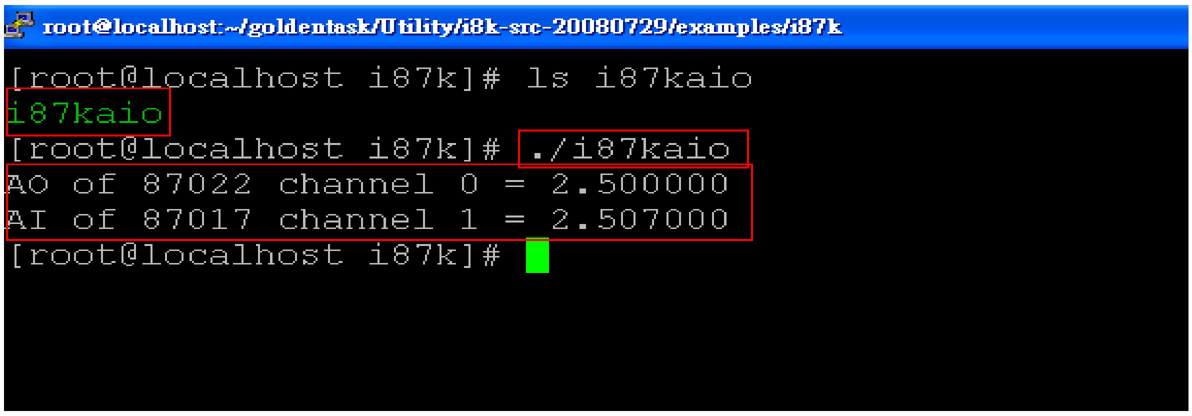
(3) **Open ttySA4** (internal serial port of LX-8X31) will be modified to **open ttyS1** (RS-485 port of LX-8X31)

The addresses I-87022 and I-87017 are in the RS-485 network and are set to be 01 and 02 separately and the baudrate is 9600 by DCON Utility. The source code of this demo program – **i87kaio_87k.c** is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
DWORD wBuf[12];
DWORD   wBuf7[12];
float fBuf[12];
/* --------------------------------------------------------------- */
int main()
{
      int i,j, wRetVal;
      DWORD temp;
      //Check Open_Com3
      wRetVal = Open_Com(ttyS1, 9600, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed!\n");
            return (-1);
      }

      //--- Analog output ----    ****    87022 -- AO    ****
      i=0;
      wBuf[0] = ttyS1;                  // COM Port
      wBuf[1] = 0x01;              // Address
      wBuf[2] = 0x87022;           // ID
      wBuf[3] = 0;                 // CheckSum disable
      wBuf[4] = 100;               // TimeOut , 100 msecond
      wBuf[5] = i;                 // Channel Number of AO
      wBuf[6] = 0;                 // string debug
      fBuf[0] = 2.5;                // AO Value

      wRetVal = AnalogOut_87k(wBuf, fBuf, szSend, szReceive);
            if (wRetVal)
                  printf("AO of 87022 Error !, Error Code=%d\n", wRetVal);
            else
                  printf("AO of 87022 channel %d = %f \n",i,fBuf[0]);
      //--- Analog Input ----    ****    87017 -- AI    ****
```

```
        j=1;
        wBuf7[0] = ttyS1;              // COM Port
        wBuf7[1] = 0x02;               // Address
        wBuf7[2] = 0x87017;            // ID
        wBuf7[3] = 0;                  // CheckSum disable
        wBuf7[4] = 100;                // TimeOut , 100 msecond
        wBuf7[5] = j;                  //Channel Number of AI
        wBuf7[6] = 0;                  // string debug

        wRetVal = AnalogIn_87k(wBuf7, fBuf, szSend, szReceive);
                if (wRetVal)
                        printf("AI of 87017 Error !, Error Code=%d\n", wRetVal);
                else
                        printf("AI of 87017 channel %d = %f \n",j,fBuf[0]);

        Close_Com(ttyS1);
                return 0;
}
```

All the steps from programming to execution are the same as those in the section 7.1. The result of execution refers to Fig. 7- 4.



```
root@localhost:~/goldentask/Utility/i8k-src-20080729/examples/i87k
[root@localhost i87k]# ls i87kaio
i87kaio
[root@localhost i87k]# ./i87kaio
AO of 87022 channel 0 = 2.500000
AI of 87017 channel 1 = 2.507000
[root@localhost i87k]#
```

Fig. 7-4

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

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

7.5 I-8k Modules DIO Control Demo

I8000.c of Libi8k.a is the source file for i8k modules in slots of I-8000 controller. **Slot.c** of Libi8k.a is the source file for i8k modules in slots of LX-8X31. Therefore the functions for i8k modules in slots of LX-8X31 and in slots of I-8000 controller are different completely. There are two conditions for the location of the I-8k modules：

(1) When I-8K DIO modules are **in the LX-8X31** , then please refer to the demo in section 7.5.1.

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

7.5.1 I-8k Modules in slots of LX-8X31

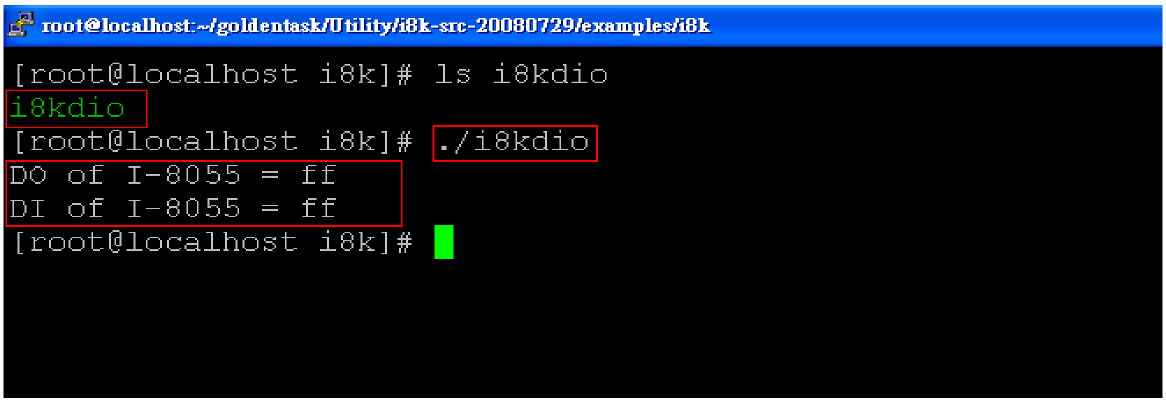
In this section, this demo program – **i8kdio.c** will introduce how to control the DI/DO with the I-8055 (8 DO channels and 8 DI channels) module and it is plugged into slot 3 of the LX-8X31.

The address and baudrate in the LX-8X31 are constant and they are 00 and 115200 separately. The result of this demo lets DO channel 0 ~7 of I-8055 output and DI channel 0 of I-8055 input. The source code of this demo program is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
DWORD dwBuf[12];
float fBuf[12];
/* --------------------------------------------------------------- */
int main()
{
    int i,j, wRetVal;
    WORD DOval,temp;
```

```
        wRetVal = Open_Slot(3);
        if (wRetVal > 0) {
                printf("open Slot failed!\n");
                return (-1);
        }

        //I-8055_DO
        DO_8(3,255);
        printf("DO of I-8055 = 0x%x \n", 255);


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

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

All the steps from programming to execution are the same as those in the section 7.1. The result of execution refers to Fig. 7-5.



```
root@localhost:~/goldentask/Utility/i8k-src-20080729/examples/i8k
[root@localhost i8k]# ls i8kdio
i8kdio
[root@localhost i8k]# ./i8kdio
DO of I-8055 = ff
DI of I-8055 = ff
[root@localhost i8k]#
```

Fig. 7-5

7.5.2 I-8k Modules in slots of I-8000 Controller

In this section, this demo program – **i8kdio_8k.c** will illustrate how to control the DI/DO with the I-8055 (8 DO channels and 8 DI channels) module. Please follow the below steps to configure the hardware：

(1) Put the I-8055 module in slot 0 of I-8000 controller.

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

firmware.

(3) Connect the **ttyS0** of LX-8X31 to the ttySA4 of I-8000 controller with the RS-232 cable.

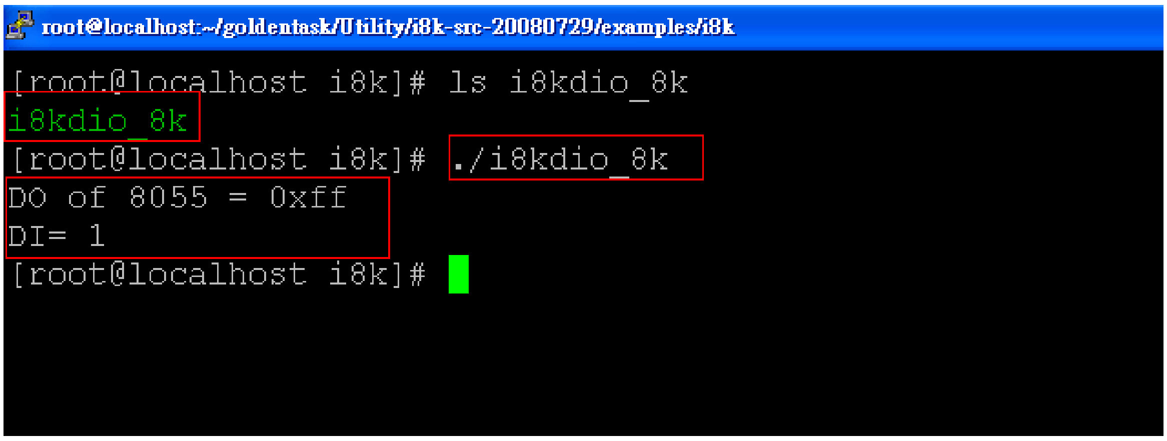
The address of I-8000 controller is 01 and the baudrate is 115200 that can be modified by DCON Utility. The result of this demo lets DO channel 0 ~7 of I-8055 output and DI channel 0 of I-8055 input. The source code of this demo program is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"

char szSend[80], szReceive[80];
DWORD dwBuf[12];
float fBuf[12];

/* -------------------------------------------------------------------- */
int main()
{
      int i, wRetVal;
      DWORD temp;

      //Check Open_Com3
      wRetVal = Open_Com(ttyS0, 115200, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed!\n");
            return (-1);
      }

      //--- digital output ----    **(DigitalOut_8K()**)
      dwBuf[0] = ttyS0;              // COM Port
      dwBuf[1] = 01;                 // Address
      dwBuf[2] = 0x8055;             // ID
      dwBuf[3] = 0;                  // CheckSum disable
      dwBuf[4] = 100;                // TimeOut , 100 msecond
      dwBuf[5] = 0xff;               // digital output
      dwBuf[6] = 0;                  // string debug
```

```
        dwBuf[7] = 1;                    // slot number

     wRetVal = DigitalOut_8K(dwBuf, fBuf, szSend, szReceive);
     if (wRetVal)
                    printf("DO of 8055 Error !, Error Code=%d\n", wRetVal);
        else
                    printf("DO of 8055 = 0x%x" ,dwBuf[5]);

     //--- digital Input ----     **(DigitalIn_8K()**)
        dwBuf[0] = ttyS0;                  // COM Port
        dwBuf[1] = 01;                     // Address
        dwBuf[2] = 0x8055;                 // ID
        dwBuf[3] = 0;                      // CheckSum disable
        dwBuf[4] = 100;                    // TimeOut , 100 msecond
        dwBuf[6] = 0;                      // string debug
        dwBuf[7] = 1;                      // slot number

        getch();
        DigitalIn_8K(dwBuf, fBuf, szSend, szReceive);
        printf("DI = %u",dwBuf[5]);

     //--- digital output ----     ** Close DO **
        dwBuf[0] = ttyS0;                 // COM Port
        dwBuf[1] = 01;                    // Address
        dwBuf[2] = 0x8055;                // ID
        dwBuf[3] = 0;                     // CheckSum disable
        dwBuf[4] = 100;                   // TimeOut , 100 msecond
        dwBuf[5] = 0x00;                  // digital output
        dwBuf[6] = 0;                     // string debug
        dwBuf[7] = 1;                     // slot number
        getch();                          // push any key to continue
        wRetVal = DigitalOut_8K(dwBuf, fBuf, szSend, szReceive);

        Close_Com(ttyS0);
        return 0;
  }
```

All the steps from programming to execution are the same as those in the section

7.1. The result of execution refers to Fig. 7-6.



```
root@localhost:~/goldentask/Utility/i8k-src-20080729/examples/i8k
[root@localhost i8k]# ls i8kdio_8k
i8kdio_8k
[root@localhost i8k]# ./i8kdio_8k
DO of 8055 = 0xff
DI= 1
[root@localhost i8k]#
```

Fig. 7-6

7.6 I-8k Modules AIO Control Demo

I8000.c of Libi8k.a is the source file for i8k modules in slots of I-8000 controller. **Slot.c** of Libi8k-3600.a is the source file for i8k modules in slots of the LX-8X31. Therefore the functions for the i8k modules in LX-8X31 slots and in the I-8000 controller slots are completely different. There are two conditions for the location of the I-8k modules：

(1) When I-8K AIO modules are **in the LX-8X31**, then please refer to the demo in section 7.6.1.

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

7.6.1 I-8k Modules in slots of LX-8X31

In this section, this demo program – **i8kaio.c** will illustrate how to control the AI/AO with the I-8024 (4 AO channels) and I-8017 (8 AI channels) module and they are in slot 1 and slot 2 of the LX-8X31 separately.

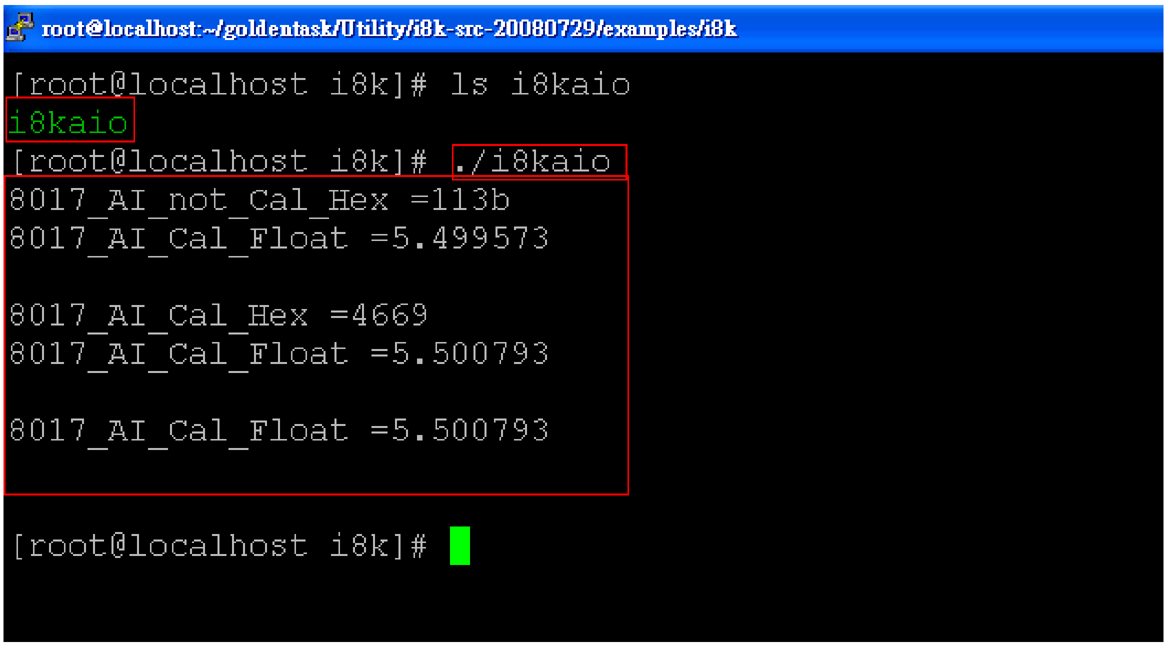
The address and baudrate in the LX-8X31 are constant and they are 00 and 115200 separately. The result of this demo lets AO voltage channel 0 of I-8024 output 5.5V and AI channel 2 of I-8017H input. The source code of this demo program is as follows：

```
#include<stdio.h>
#include<stdlib.h>
```

```
#include "msw.h"

char szSend[80], szReceive[80];
DWORD dwBuf[12];
float fBuf[12];

/* --------------------------------------------------------------- */
int main()
{
	int i, wRetVal,j;
	float fAi;
	int hAi, chAi, Succ;
	int Arr_hAi[5];
	float Arr_fAi[5];

	//I-8024
	wRetVal = Open_Slot(1);
	if (wRetVal > 0) {
		printf("open Slot failed!\n");
		return (-1);
	}

	//I8024 Initial
	I8024_Initial(1);

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

	/***************************************/
	//I-8017H
	wRetVal = Open_Slot(2);
	if (wRetVal > 0) {
		printf("open Slot failed!\n");
		return (-1);
	}

	//I8017H Initial
```

```
        I8017_Init(2);
        //I8017H _Channel Setup
        I8017_SetChannelGainMode(2,2,0,0);

        // First Method：Get AI Value
        hAi = I8017_GetCurAdChannel_Hex(2);     //Get Not-calibrated AI Hex Value
        printf("8017_AI_not_Cal_Hex =%x\n",hAi);
        fAi = HEX_TO_FLOAT_Cal(hAi,2,0);         //Not-calibrated AI Hex Value modify
to calibrated AI Float Value
        printf("8017_AI_Cal_Float =%f\n\n",fAi);

        // Second Method：Get AI Value
        hAi = I8017_GetCurAdChannel_Hex_Cal(2);   //Get Calibrated AI Hex Value
        printf("8017_AI_Cal_Hex =%x\n",hAi);
        fAi = CalHex_TO_FLOAT(hAi,0);                //Calibrated AI Hex Value modify
to Calibrated AI Float Value
        printf("8017_AI_Cal_Float =%f\n\n",fAi);

        // Third Method：Get AI Value
        fAi = I8017_GetCurAdChannel_Float_Cal(2);   //Get Calibrated AI Float Value
        printf("8017_AI_Cal_Float =%f\n\n\n",fAi);

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

All the steps from programming to execution are the same as those in the section 7.1. The result of execution refers to Fig. 7-7.



```
root@localhost:~/goldentask/Utility/i8k-src-20080729/examples/i8k
[root@localhost i8k]# ls i8kaio
i8kaio
[root@localhost i8k]# ./i8kaio
8017_AI_not_Cal_Hex =113b
8017_AI_Cal_Float =5.499573

8017_AI_Cal_Hex =4669
8017_AI_Cal_Float =5.500793

8017_AI_Cal_Float =5.500793


[root@localhost i8k]# 
```

Fig. 7-7

7.6.2 I-8k Modules in slots of I-8000 Controller

In this section, this demo program – **i8kaio_8k.c** will introduce how to control the AI/AO with the I-8024 (4 AO channels) and I-8017 (8 AI channels) module and they are plugged into slot 0 and slot 1 of the I-8000 controller separately. Please follow the below steps to configure the hardware：

(1) Put the I-8024 and I-8017 modules in slot 0 and slot 1 of I-8000 controller.

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

(3) Connect **ttyS0** of LX-8X31 to ttySA4 of I-8000 controller with RS-232 cable.

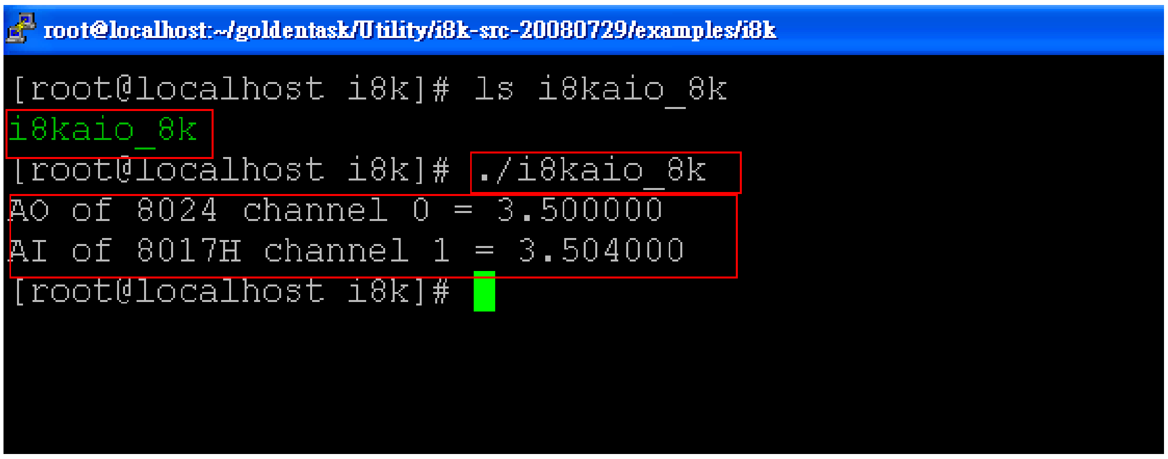
The address and baudrate of I-8000 controller are 01 and 115200 that can be modified by DCON Utility. The result of this demo lets AO voltage channel 0 of 8024 output 3.5V and AI channel 2 of 8017H input. The source code of this demo program is as follows：

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"
```

```
char szSend[80], szReceive[80];
DWORD wBuf[12];
float fBuf[12];

/* --------------------------------------------------------------- */
int main()
{
    int i,j, wRetVal;
    DWORD temp;

    wRetVal = Open_Com(ttyS0, 115200, Data8Bit, NonParity, OneStopBit);
    if (wRetVal > 0) {
        printf("open port failed!\n");
        return (-1);
    }

    //--- Analog output ----   ****    8024 -- AO   ****
    i = 0;
    wBuf[0] = ttyS0;            // COM Port
    wBuf[1] = 0x01;             // Address
    wBuf[2] = 0x8024;           // ID
    wBuf[3] = 0;                // CheckSum disable
    wBuf[4] = 100;              // TimeOut , 100 msecond
    wBuf[5] = i;                // Channel No. of AO
    wBuf[6] = 0;                // string debug
    wBuf[7] = 0;                // Slot Number
    fBuf[0] = 3.5;

    wRetVal = AnalogOut_8K(wBuf, fBuf, szSend, szReceive);
    if (wRetVal)
        printf("AO of 8024 Error !, Error Code=%d\n", wRetVal);
    else
        printf("AO of 8024 channel %d = %f \n",i,fBuf[0]);

    //--- Analog Input ----   ****    8017H -- AI   ****
    j = 2;
    wBuf[0] = ttyS0;            // COM Port
    wBuf[1] = 0x01;             // Address
```

```
        wBuf[2] = 0x8017;           // ID
        wBuf[3] = 0;                // CheckSum disable
        wBuf[4] = 100;              // TimeOut , 100 msecond
        wBuf[5] = j;                // Channel of AI
        wBuf[6] = 0;                // string debug
        wBuf[7] = 1;                // Slot Number

        wRetVal = AnalogIn_8K(wBuf, fBuf, szSend, szReceive);
        if (wRetVal)
              printf("AI of 8017H Error !, Error Code=%d\n", wRetVal);
        else
              printf("AI of 8017H channel %d = %f \n",j,fBuf[0]);

        Close_Com(ttyS0);
        return 0;
}
```

All the steps from programming to execution are the same as those in the section 7.1. The result of execution refers to Fig. 7-8

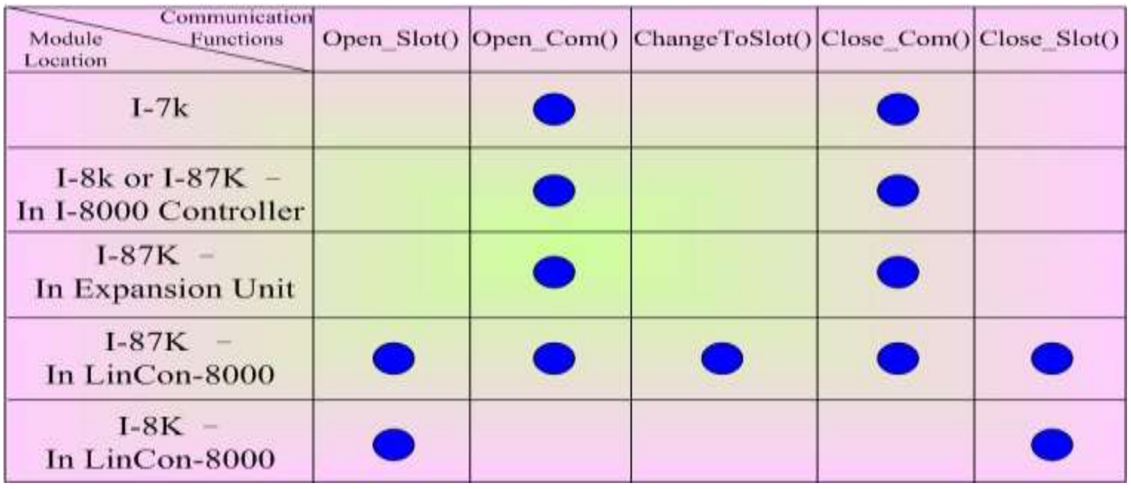


```
root@localhost:~/goldentask/Utility/i8k-src-20080729/examples/i8k
[root@localhost i8k]# ls i8kaio_8k
i8kaio_8k
[root@localhost i8k]# ./i8kaio_8k
AO of 8024 channel 0 = 3.500000
AI of 8017H channel 1 = 3.504000
[root@localhost i8k]#
```

Fig. 7-8

7.7 Conclusion of Module Control Demo

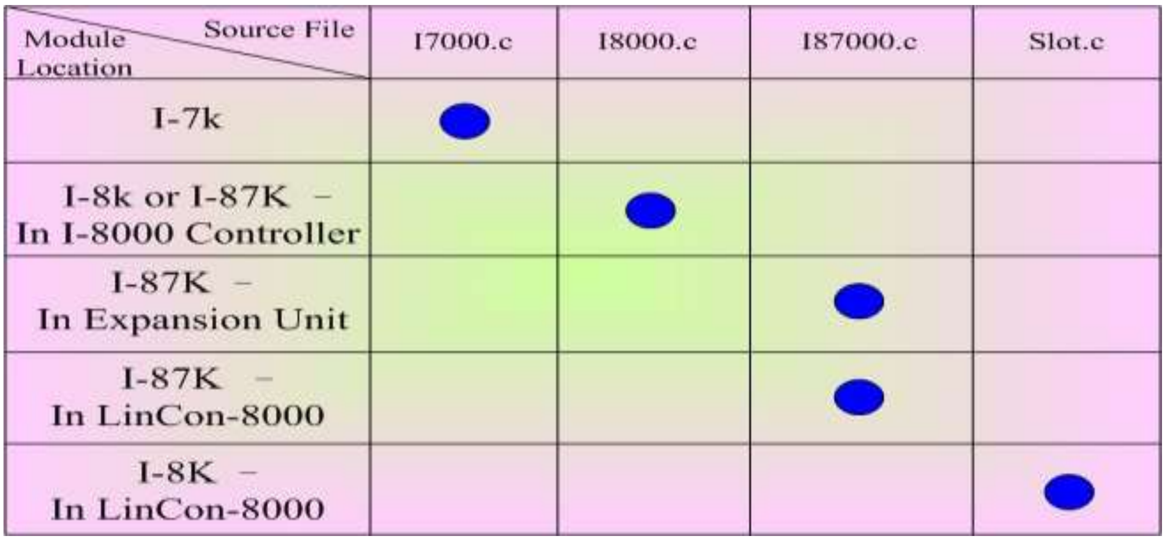
Fig. 7-9 is the table of communication functions for the I-7000/I-8000/I-87000 modules in different locations. When using the ICP DAS modules in the LX-8X31, this table will be helpful to let users understand which functions of communication should be used.



Communication Functions Module Location	Open_Slot()	Open_Com()	ChangeToSlot()	Close_Com()	Close_Slot()
I-7k		●		●	
I-8k or I-87K – In I-8000 Controller		●		●	
I-87K – In Expansion Unit		●		●	
I-87K – In LinCon-8000	●	●	●	●	●
I-8K – In LinCon-8000	●				●

Fig 7-9

Fig. 7-10 is the table of source files for the I-7000/I-8000/I-87000 modules in different locations. When using ICP DAS modules in the LX-8X31, this table will be helpful to let users understand which source files of the libi8k-3600.a should be called.



Source File Module Location	I7000.c	I8000.c	I87000.c	Slot.c
I-7k	●			
I-8k or I-87K – In I-8000 Controller		●		
I-87K – In Expansion Unit			●	
I-87K – In LinCon-8000			●	
I-8K – In LinCon-8000				●

Fig. 7-10

8. LX-8X31 Library Reference in C Language

In this chapter, all the functions of **libi8k-3600** will be listed to allow users to able to look them up quickly.

8.1 List Of System Information Functions

```
void ChangeToSlot(char slot)
int Open_Slot(int slot)
void Close_Slot(int slot)
int Open_Slot(void)
void Close_SlotAll(void)
int GetModuleType(char slot)
int GetNameOfModule(char slot)
```

8.2 List Of Digital Input/Output Functions

```
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)
```

8.3 List Of Watch Dog Timer Functions

void EnableWDT(unsigned int msecond)

void DisableWDT(void)

unsigned int WatchDogSWEven(void)

void ClearWDTSWEven(unsigned int rcsr)

8.4 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)

8.5 List Of Analog Input Functions

int I8017_Init(int slot)

void I8017_SetLed(int slot, unsigned int led)

void I8017_SetChannelGainMode (int slot, int ch, int gain, int mode)

int I8017_GetCurAdChannel_Hex (int slot)

int I8017_AD_POLLING(int slot, int ch, int gain, unsigned int datacount, int *DataPtr)

float HEX_TO_FLOAT_Cal(int HexValue, int slot, int gain)

void ARRAY_HEX_TO_FLOAT_Cal(int *HexValue, float *FloatValue, int slot, int gain,int len)

int I8017_Hex_Cal(int data)

int I8017_Hex_Cal_Slot_Gain(int slot, int gain, int data)

float CalHex_TO_FLOAT(int HexValue,int gain)

void ARRAY_CalHex_TO_FLOAT(int *HexValue, float *FloatValue, int gain, int len)

int I8017_GetCurAdChannel_Hex_Cal(int slot)

int I8017_AD_POLLING_Cal(int slot, int ch, int gain, unsigned int datacount,

```
                                  int *DataPtr)
int I8017_GetCurAdChannel_Float_Cal(int slot)
```

8.6 List Of Analog Output Functions

```
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)
```

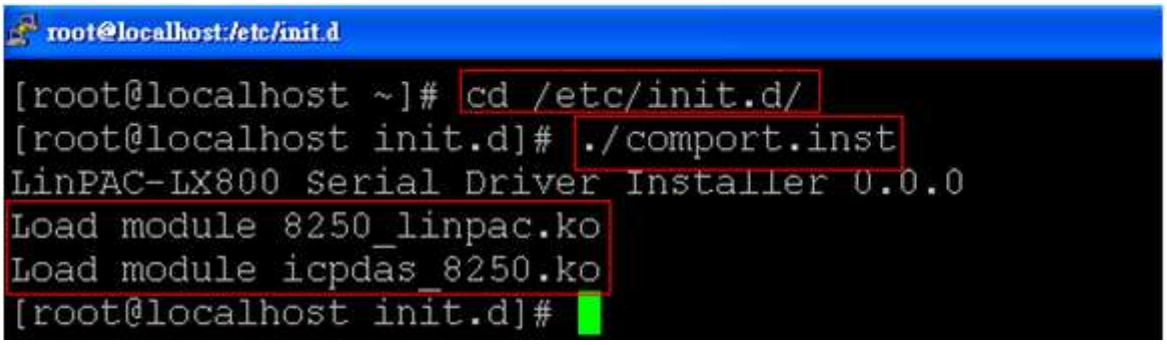
9. Additional Support

In this chapter, ICP DAS provides extra module supported and instructions to enhance LX-8X31 functionality and affinity.

9.1 N-Port Serial Port and Modules Driver Install

The comport linux driver can be used in LX-8X31 to support serial modules (e.g. i-8114w). For LX-8X31 Linux OS, the recommended in installation and uninstall steps as follows:

(1) To type **“cd /etc/init.d/”** and type **“./comport.inst”** to install serial module driver automatically (refer to Fig 9-1).

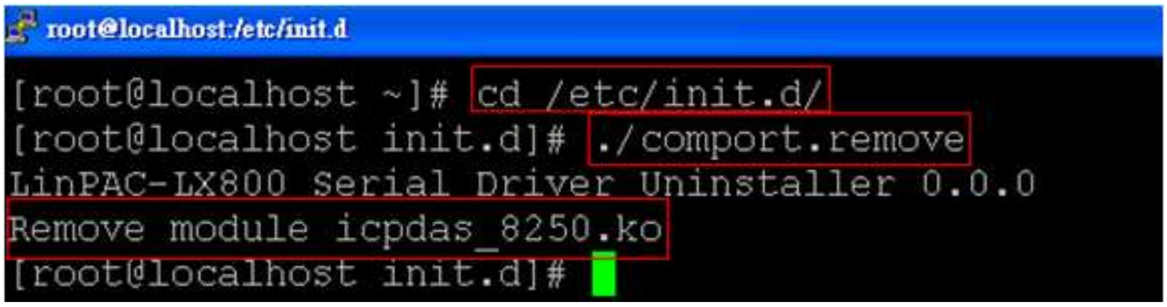


```
root@localhost:/etc/init.d
[root@localhost ~]# cd /etc/init.d/
[root@localhost init.d]# ./comport.inst
LinPAC-LX800 Serial Driver Installer 0.0.0
Load module 8250_linpac.ko
Load module icpdas_8250.ko
[root@localhost init.d]#
```

Fig 9-1

(2) To type **“dmesg”** to check the status of installing comport driver.

(3) If user want to remove the comport driver, user could type **“cd /etc/init.d/”** and type **“./comport.remove”** to remove comport driver automatically (refer to 9-2).



```
root@localhost:/etc/init.d
[root@localhost ~]# cd /etc/init.d/
[root@localhost init.d]# ./comport.remove
LinPAC-LX800 Serial Driver Uninstaller 0.0.0
Remove module icpdas_8250.ko
[root@localhost init.d]#
```

Fig 9-2

9.2 N-Port Module (i-8114w or other serial modules) Support

I**-8114w** module provide **four serial ports** respectively. User can install it into the LX-8X31 slots. In this way, users can use more serial ports in the
LX-8X31 and the expanded maximum number of serial port in the LX-8X31 will be twenty-eight. The LX-8X31 is a multi-tasking unit, therefore users can control all the serial ports simultaneously.
Fig.9-3 is the serial port number corresponding to the **device name** in the LX-8X31.

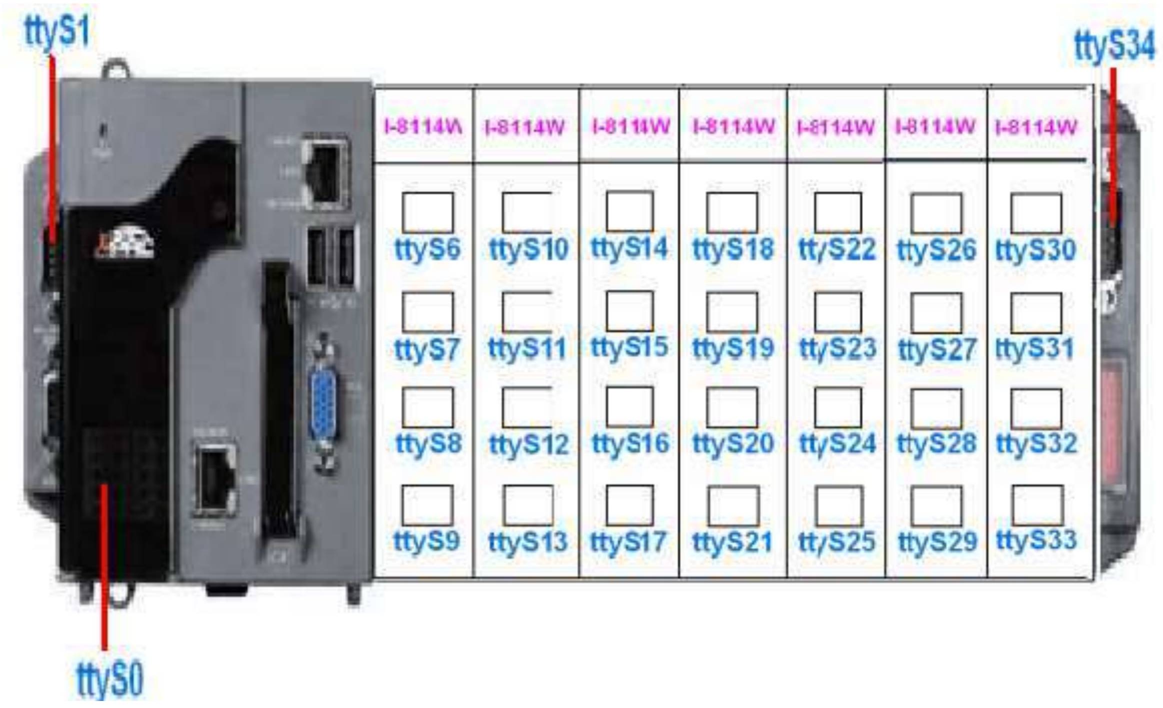


Fig 9-3

9.4.1 How to boot LX-8X31 without loading X-window

LX-8X31 can boot without loading X-window by the steps as follows :

(1) To type “ **vim /etc/default** /grub“ to modify grub file.

(2) Set line “GRUB_CMDLINE_LINUX_DEFAULT“ from
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash" to
GRUB_CMDLINE_LINUX_DEFAULT="text"

(3) Use command “update-grub” to update grub

(4) reboot your LX-8X31

9.4.2 Enabling X-window load at boot time

LX-8X31 can boot and loading X-window at the same time by the steps as follows:

(1) To type " **vim /etc/default** /grub" to modify grub file.

(2) Set line "GRUB_CMDLINE_LINUX_DEFAULT" from
GRUB_CMDLINE_LINUX_DEFAULT="text" to
GRUB_CMDLINE_LINUX_DEFAULT=" quiet splash "

(3) Use command "update-grub" to update grub

(4) reboot your LX-8X31

9.5 Application Support

There are many applications in the LX-8X31.

(1)Provide Web Browser

Users can see the Web Page by using the Web Browser built in the LX-8X31. Just click the icon "firefox" (refer to Fig 9-4) to open the web browser.

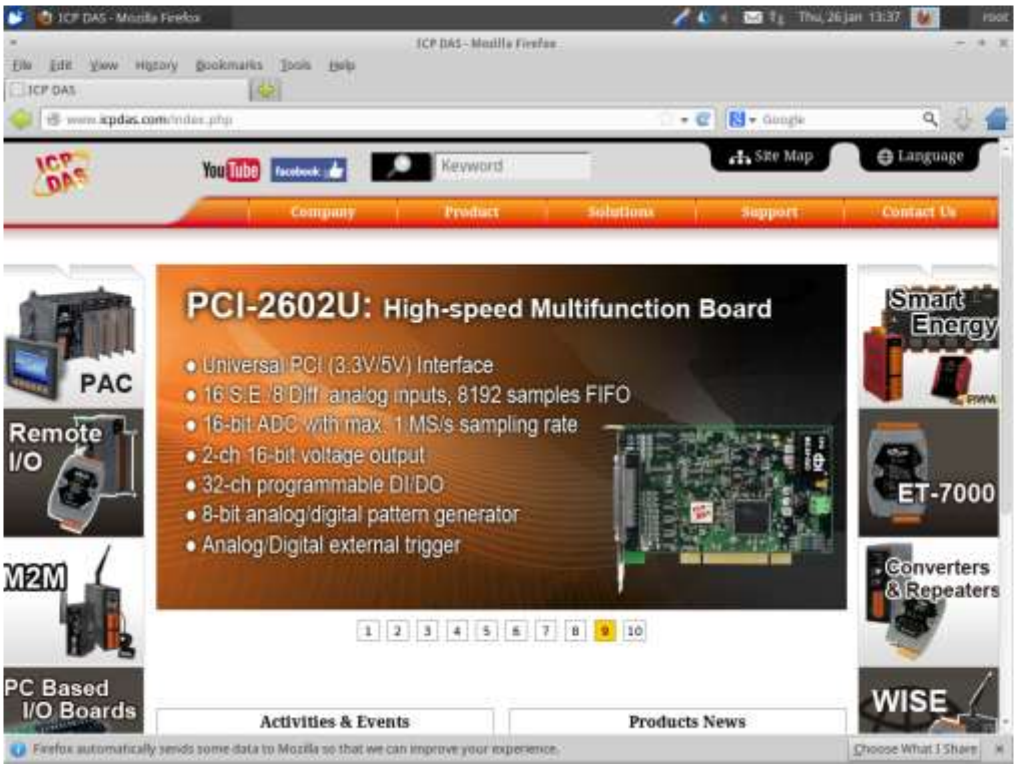


Fig 9-4

(2)Using “apt” to Install Application

“apt” is a software installation tool for Debian linux and Ubuntu linux. It is a complete software management system. The “apt” is designed to use over network/internet. If you want to install more application, you could use “apt-get install” to install automatically.

9.6 Other Optional Function

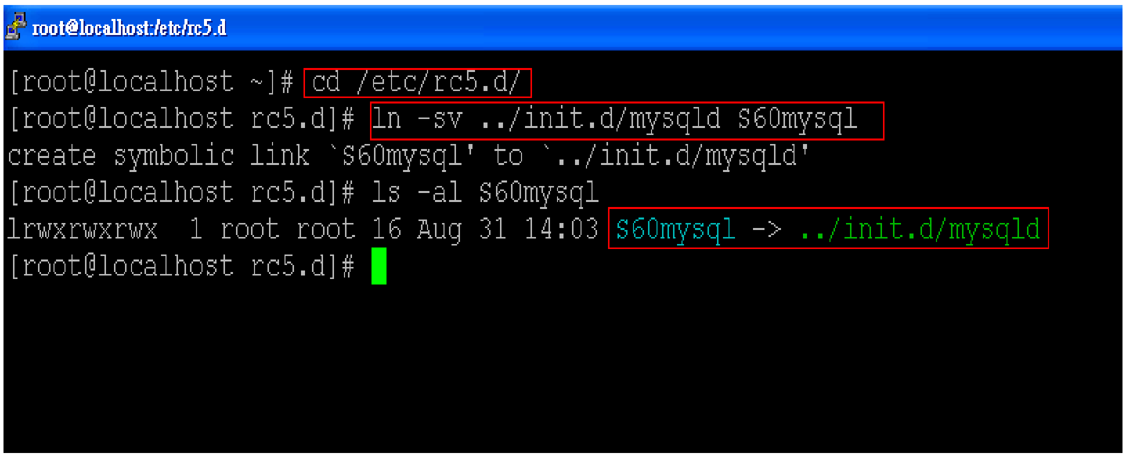
These optional functions are listed below all supported in the LX-8X31.

(1) Support MySQL

MySQL is a small database server and it is “ Relational DataBase Management System (RDBMS) “. By using MySQL, users can add or delete data easily and it is open source and supports many platforms, like UNIX、Linux or Windows operating system. If users want to startup MySQL server at boot time (default system wouldn’t startup mysql server at boot time), user could follow under steps:

(1) Type “ **cd /etc/rc5.d** “ to into default run level(please refer to Fig 9-5).

(2) Type ” **ln -sv ../init.d/mysqld S60mysql** “ to make a symbolic link into the script file and it will be executed automatically at boot time(please refer to Fig 9-5).



```
[root@localhost ~]# cd /etc/rc5.d/
[root@localhost rc5.d]# ln -sv ../init.d/mysqld S60mysql
create symbolic link `S60mysql' to `../init.d/mysqld'
[root@localhost rc5.d]# ls -al S60mysql
lrwxrwxrwx  1 root root 16 Aug 31 14:03 S60mysql -> ../init.d/mysqld
[root@localhost rc5.d]#
```

Fig 9-5

(2) Support PHP

PHP is a kind of “ open source script language “ and used to design active web page. When PHP combined with MySQL are cross-platform. It means that users can develop in Windows and serve on a Linux platform. (Refer to Fig 9-6)

PHP has been built in the LX-8X31 Kernel so users just boot up LX-8X31 and can use PHP directly in the LX-8X31.

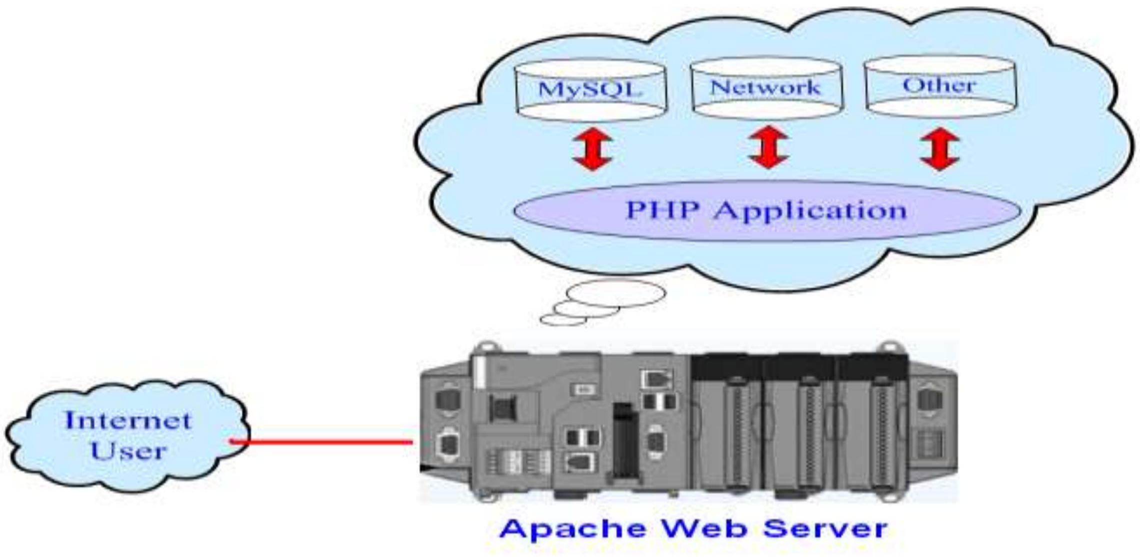


Fig 9-6

(3) Support Perl

Perl（Practical Extraction and Report Language）is also an “ open source script language “ and has been built in the LX-8X31 Kernel so users just boot up LX-8X31 and can use Perl directly in the LX-8X31.

(4) Support GCC

GCC（GNU Compiler Collection）is a compiler system produced by the GNU Project supporting various programming languages. The GCC has been built in the LX-8X31 system so users can use GCC to compiler the software directly in the LX-8X31.

Appendix A. Service Information

This appendix will show how to contact ICP DAS when you have problems in the LX-8X31 or other products.

Internet Service :

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

1. ICP DAS Web Site : http://www.icpdas.com/

2. Software Download :http://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x8x/atom

3. E-mail for Technical Support : service@icpdas.com

Manual Revision:

Manual Edition	Revision Date	Revision Details
V 0.1	2017. 1.26	1. The first version of user manual.