Software Guide

ICP DAS LP-9x21 SDK

Implement industry control with Linux Technique

(Version 1.3)

Warranty

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

Warning

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

Copyright

Trademarks

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

License

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

Contents

1. Introduction

LP-9x21 is the new generation Linux-based PAC (Programmable Automation Controller) from ICP DAS and is equipped with a Cortex-A8 CPU (1.0 GHz) running a Linux kernel 3.x operation system, multiple communication interfaces (VGA, USB, Ethernet and RS-232/485) and 2/4/8 slots for high performance parallel I/O modules (high profi le I-9K series) and serial I/O modules (high profile I-97K series).

Main advantage of the LP-9x21 is its high quality control system, including its stably properties, open source and the standard LinPAC SDK for Windows and Linux using the GNU C language, JAVA and GUI software. The main purpose of LP-9x21 is to allow the numerous enthusiastic Linux users to control their own embedded system easily within the Linux environment.

ICP DAS also provides a library file, libi8k.a, which includes all the functions from I-7000/9000/97000 series modules used in the LP-9x21 Embedded Controller. The library is specifically designed for I-7000/9000/97000 series modules based on the Linux platform for use with the LP-9x21 controller. Custom applications can easily be develop for the LP-9x21 using either C or Java, and.NET applications will also be supported in the future. The various functions contained in the library are divided into sub-group functions for ease of use within the different applications.

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

Please note:

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

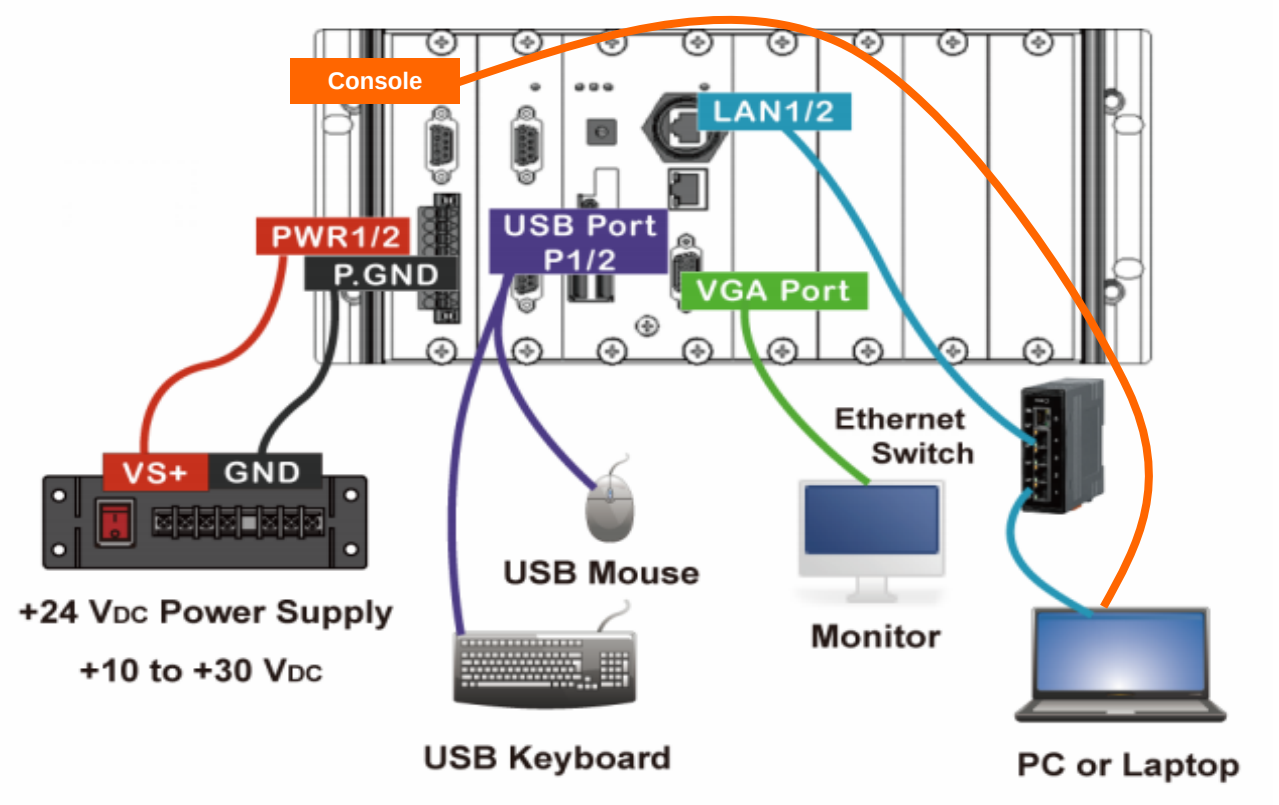


Fig. 1-1

The LP-9x21 controller contains built in HTTP, FTP, Telnet, SSH, and SFTP Servers, meaning that file transfer or remote control is much more convenient with the LP-9x21. For network communication, **wireless**, **Bluetooth** transfer protocols and **Modem**, **GPRS, ADSL, Firewall** functions are also supported.

The architecture of the LP-9x21 hardware is illustrated in the figure below.

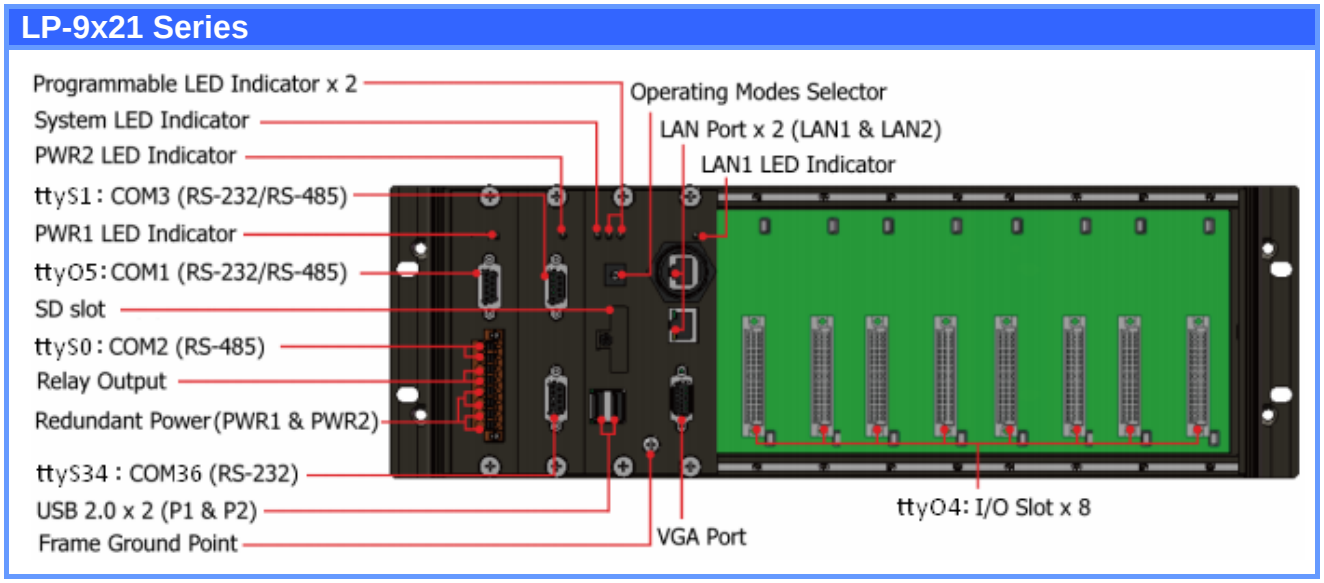


Fig. 1-2

1.1 Installing the RJ-45 waterproof connector assembly

The LP-9000 series is equipped with an RJ-45 waterproof connector to withstand contaminant in dusty environment.

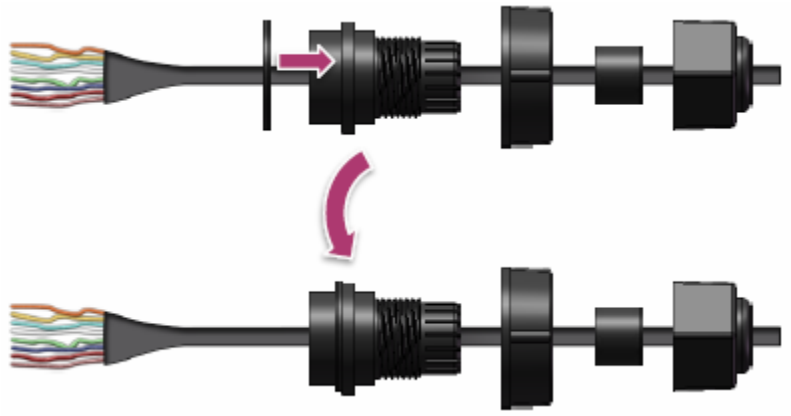
The RJ-45 waterproof connector is optional for use with LAN1 port. If you do not need the RJ-45 waterproof connector, you can remove the cap and just plug in a regular Ethernet cable.

If you want to use the RJ-45 waterproof connector for protecting the connection, follow the instructions below.

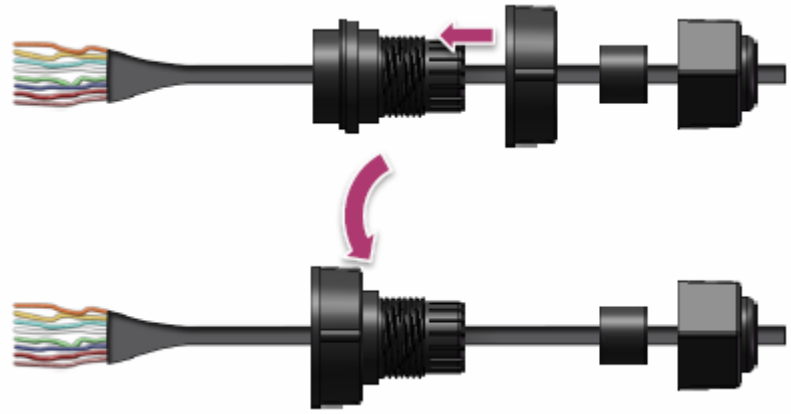
Step 1: Remove the RJ-45 connector from the RJ-45 cable

Step 2: Feed the end of the RJ-45 cable through the (A) sealing nut, (B) rubber sealing insert, (C) clamping ring, (D) panel gasket and (E) cable gland base

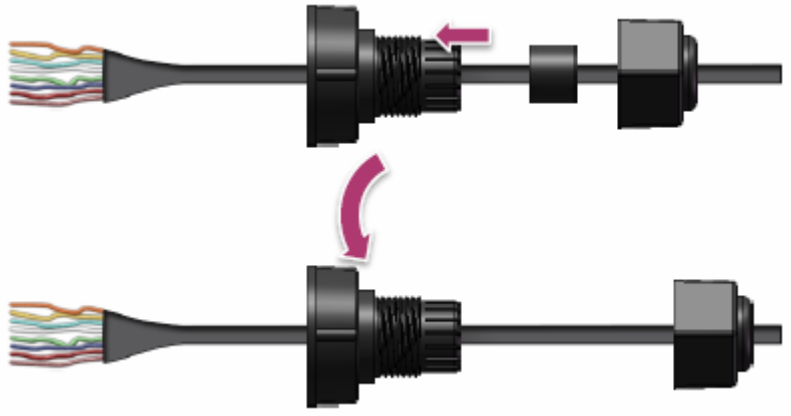
Step 3: Wrap the (E) clamping ring around the (D) rubber sealing insert



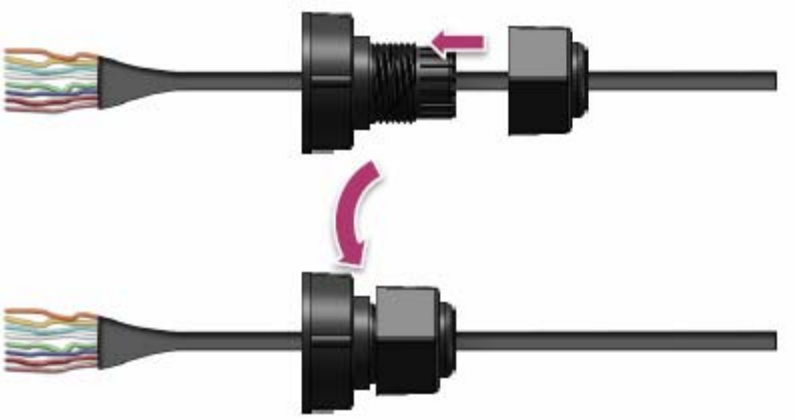
Step 4: Wrap the (C) cable gland base around the (D) clamping ring



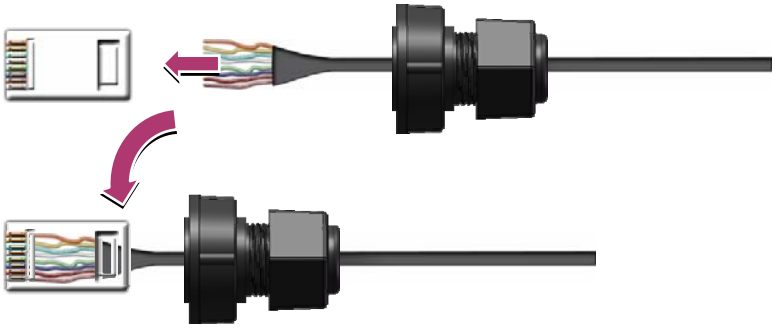
Step 5: Insert the (B) rubber sealing insert into the (D) clamping ring



Step 6: Push the (E) sealing nut forward and Hand-tighten it to seal the assembly

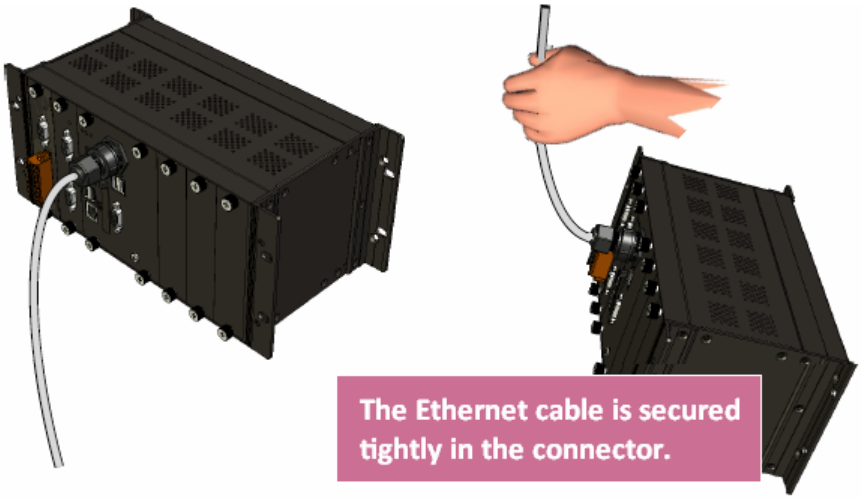


Step 7: Insert the RJ-45 cable into the RJ-45 connector



Step 8: Push the RJ-45 waterproof connector assembly forward

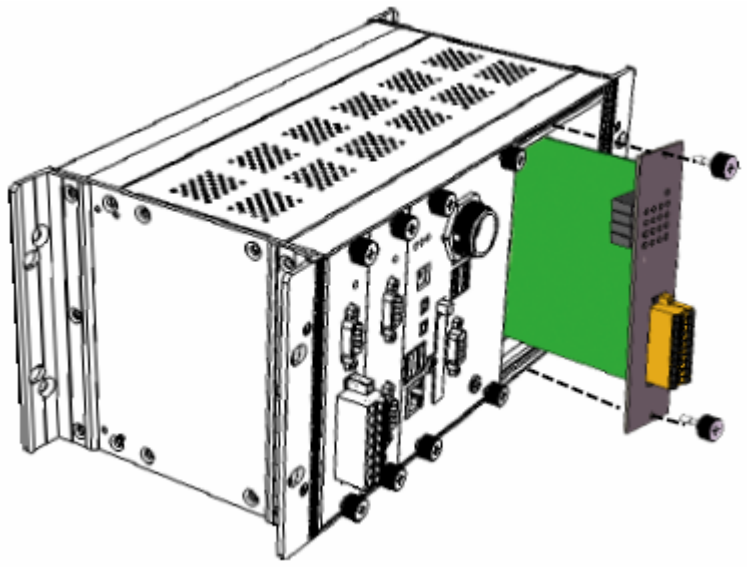
Step 9: Insert the Ethernet cable and screw the RJ-45 waterproof into the receptacle



1.2 Inserting the I/O Modules

LP-9000 has 2/4/8 I/O expansion slots to support I-9K and I-97K series I/O modules. Before choosing the right I/O modules, you first need to know the I/O expansion capacities in order to choose the best expansion module for achieving maximal efficiency. For more information about the I/O expansion modules that are compatible with the LP-9000, please refer to: http://www.icpdas.com/root/product/solutions/remote_io/i-9k_i-97k/i-9k_i-97k_selection.html

Step 1: Insert the I/O module

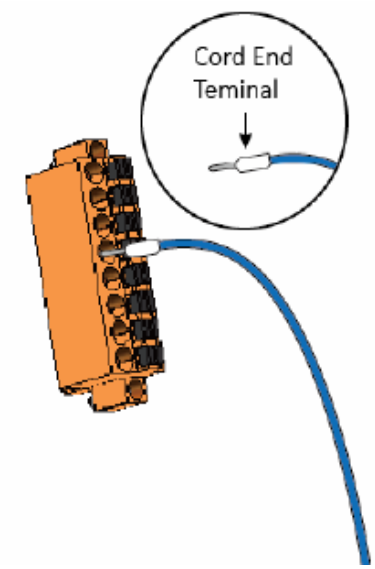


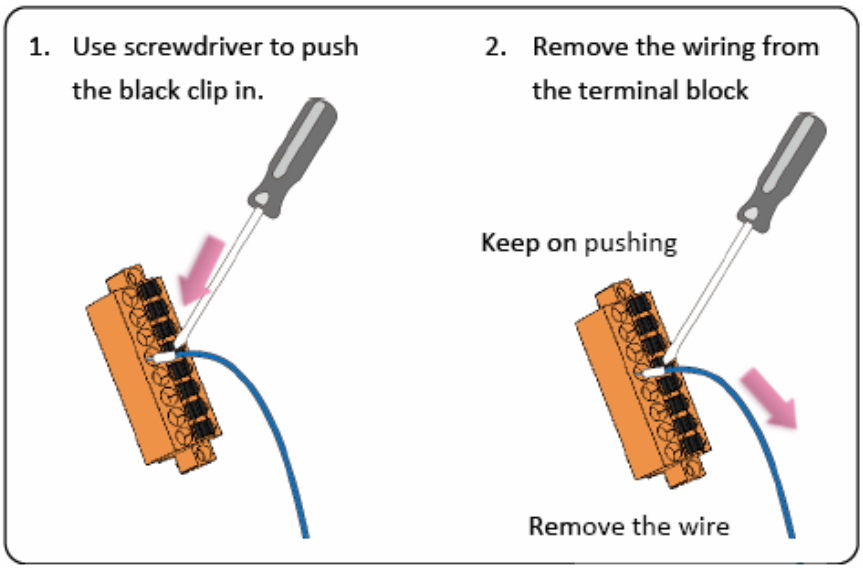
Tips & Warnings

If you do not expand the I/O module full, please keep the top case of the unused slot to protect the backplane from dirt, dust and damage from foreign objects.

Step 2: Wiring connection

The metal part of the cord end terminal on the wire can be direct wired to the terminal of LP-9000.





2. Installation of the LinPAC AM335x SDK

The “LinPAC AM335x SDK” is a development toolkit provided by ICP DAS, which can be used to easily develop custom applications for the LP-52xx/ 8x21/ 9x21 embedded controller platform. The toolkit consists of the following items:

- ❑ LinPAC AM335x SDK (Linaro GCC toolchain, Libraries, header, examples files, etc.)
- ❑ Code::Blocks project file (Windows platform only)
- ❑ Basic Linux commands (Windows platform only)

The topic provides LinPAC AM335x SDK installation instructions for the following platforms:

- ❑ Linux
  - ◆ Download/Install LinPAC AM335x SDK on Linux
- ❑ Windows
  - ◆ Download/Install LinPAC AM335x SDK on Windows
  - ◆ Integrating LinPAC AM335x SDK with Code::Blocks IDE

NOTE:

1. The latest Linux AM335x SDK is integrate AM335x series (LP-52xx/8x2x/9x2x) SDK.
2. The names of all the I/O module's API functions must begin with the prefix "I8K".
3. The I-8K and I-9K I/O modules using the same API function and examples.
4. More detailed information, user can refer to readme.txt file here:
   C:\cygwin\LinPAC_AM335x_SDK\examples\readme.txt file, or
   root@LinuxPC-ICPDAS:/icpdas/linpac_am335x_sdk/i8k/examples/readme.txt.

2.1 Quick Installation of the LinPAC AM335x SDK

2.1.1 Download/Install SDK on Linux

1. To create a “**icpdas**” folder in root directory, maybe you need to change the root user by ‘sudo’ or ‘su’ command (refer to Fig.2-1).

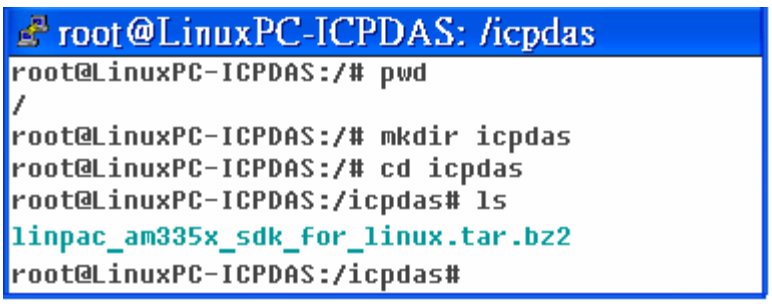


Fig. 2-1

2. Insert the installation CD into your CD-ROM driver (refer to Fig.2-2 and 2-3). Locate the "**linpac_am335x_sdk_for_linux.tar.bz2**" file in the \napdos\LP-9x21\SDK\ folder (or visit the ICP DAS website to download the latest version: http://ftp.icpdas.com.tw/pub/cd/linpac/napdos/lp-9x21/sdk/).

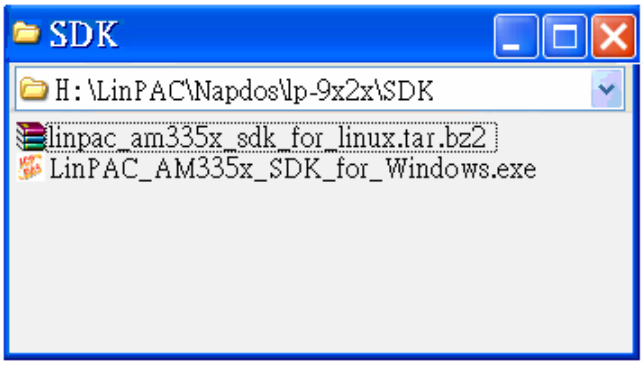


Fig. 2-2

Fig. 2-3

3. Try the following command to decompress file (refer to Fig.2-4).

   # tar **jxvf** **linpac_am335x_sdk_for_linux.tar.bz2**

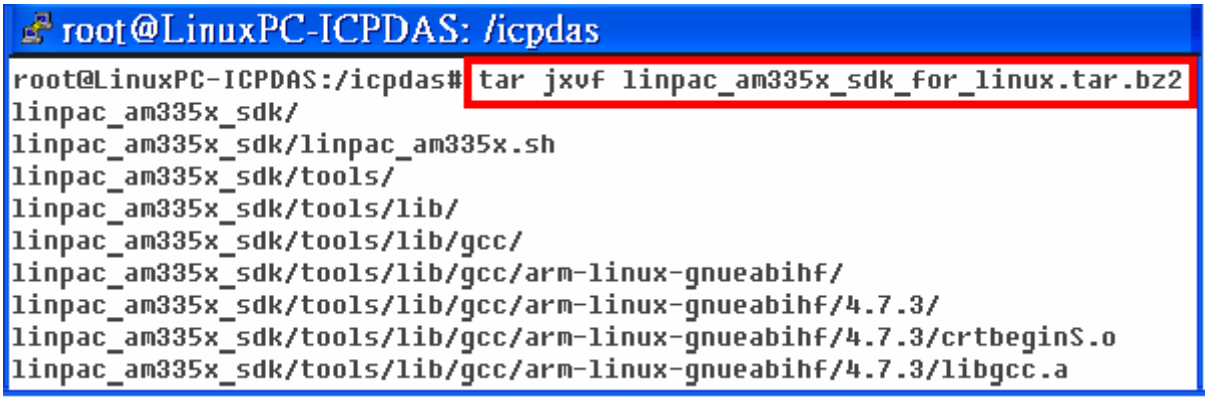


Fig. 2-4

4. Before compile the program, you need to set LinPAC AM335x SDK path in environment variables: using the provided environment variable script, which is called **linpac_am335x.sh** (refer to Fig.2-5).

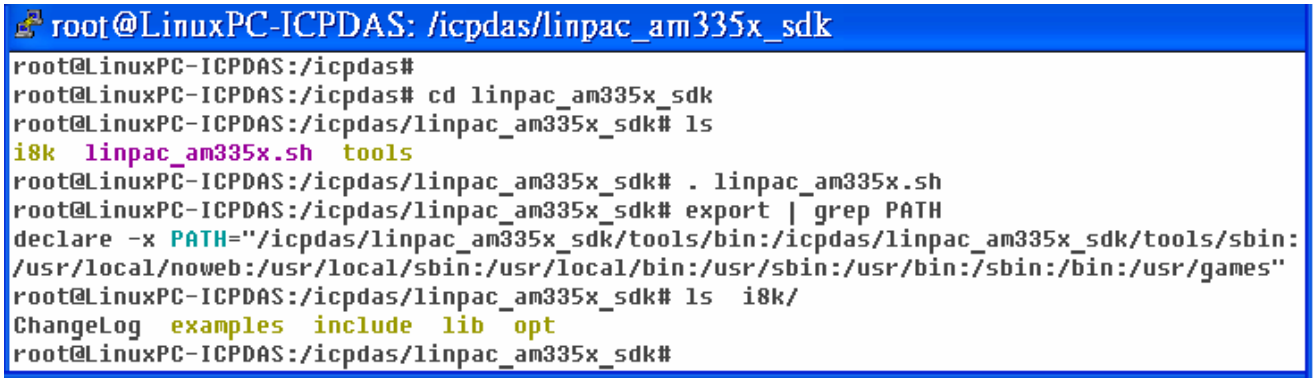


Fig. 2-5

5. Type '**make**' on the command line it will execute the compile command according to the Makefile (refer to Fig.2-6).

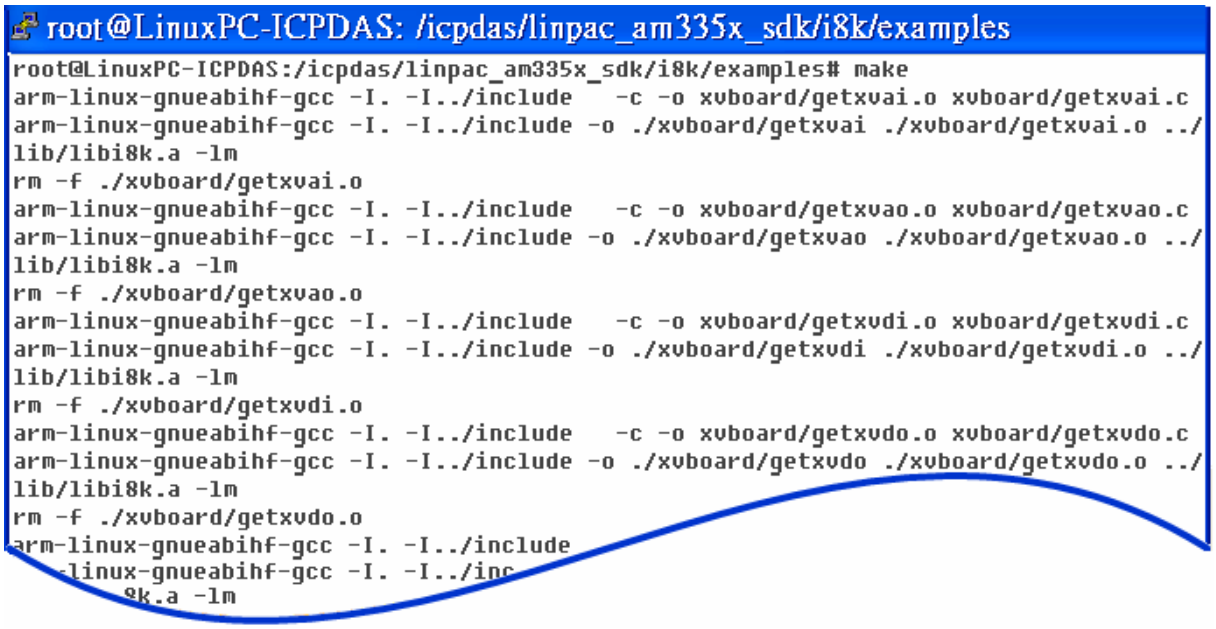


Fig. 2-6

2.1.2 Download/Install SDK on Windows

The **LinPAC_AM335x_SDK_for_Windows.exe** file provides compilers, library, header, examples, and IDE workspace file (for Code::Blocks project).

1. Insert the installation CD into your CD-ROM driver.
2. Open the \napdos\LP-9x21\SDK\ folder and double-click the icon for the "LinPAC_AM335x_SDK_for_Windows.exe" file, when the Setup Wizard is displayed, click the "Next>" button to continue, refer to Fig. 2-7 and Fig.2-8.

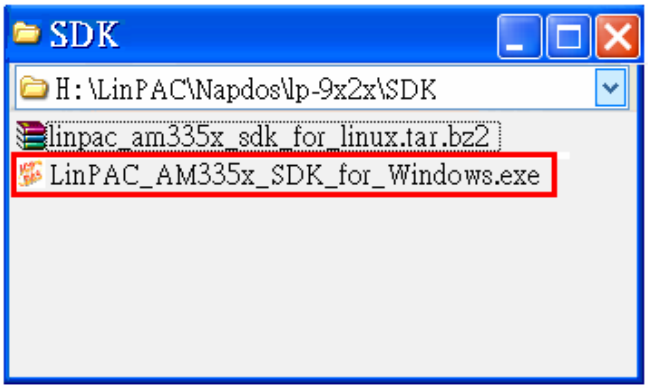


Fig. 2-7

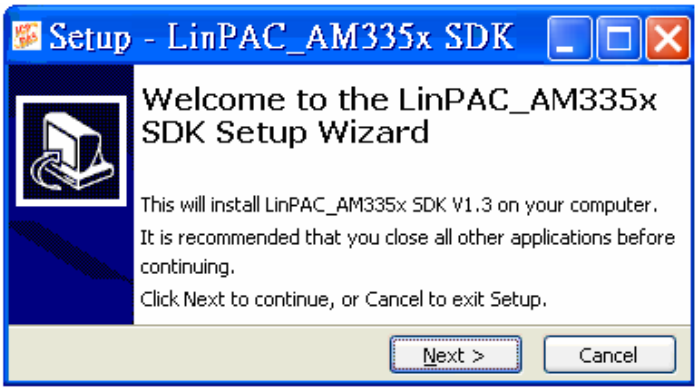


Fig. 2-8

3. Click the “I accept the agreement” option and then click the “Next” button, refer to Fig. 2-9.

4. Select Start Menu Folder option and then click the “Next” button, refer to Fig. 2-10.

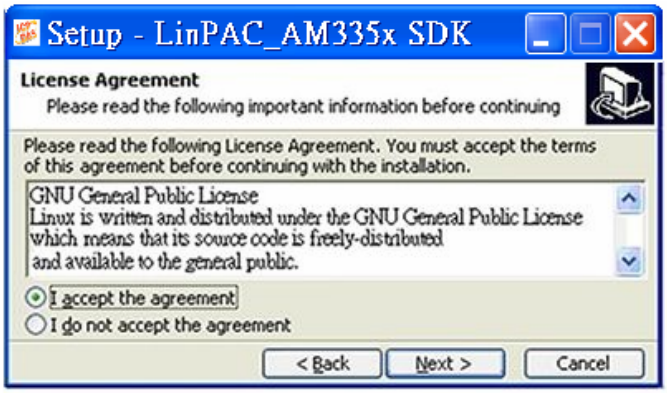


Fig. 2-19

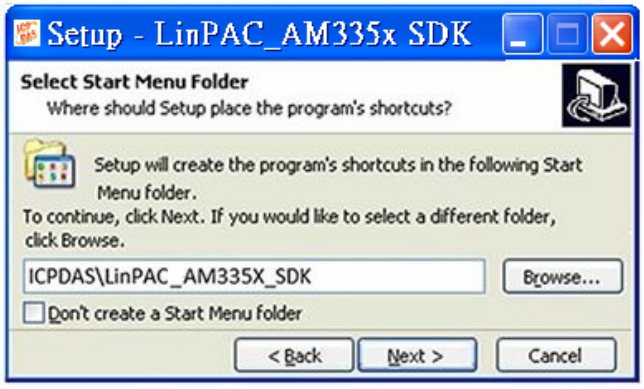


Fig. 2-10

5. The LinPAC AM335x SDK files will be extracted and installed and a progress bar will be displayed to indicate the status, refer to Fig 2-11.

6. Once the software has been successfully installed, click the “Finish” button to complete the development toolkit installation, refer to Fig. 2-12.

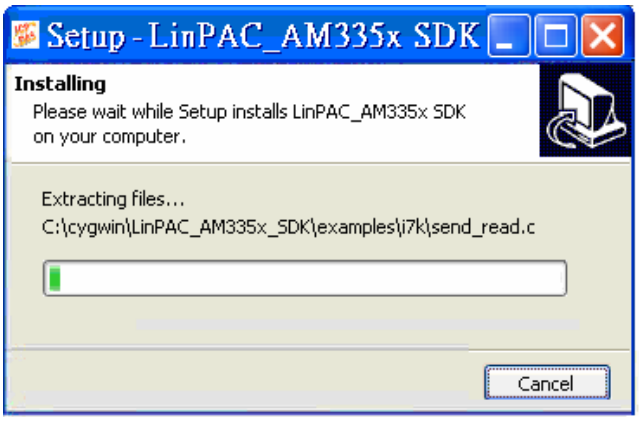


Fig. 2-11

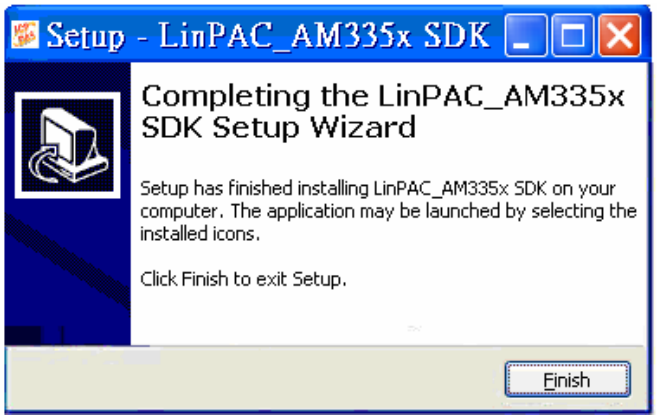


Fig. 2-12

7. Open the LinPAC AM335x SDK installation directory, the default data directory location is “C:\cygwin\”, user can see the contents of folder. Refer to Fig 2-13 and Fig 2-14.

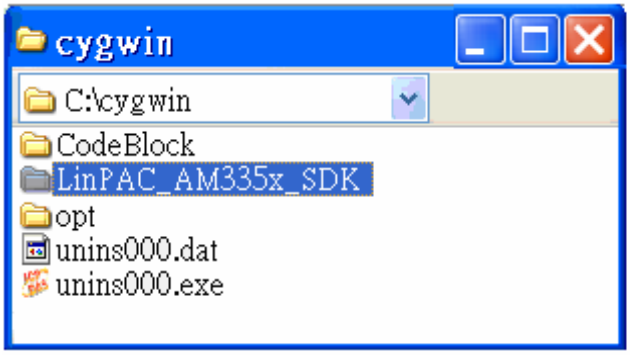


Fig. 2-13

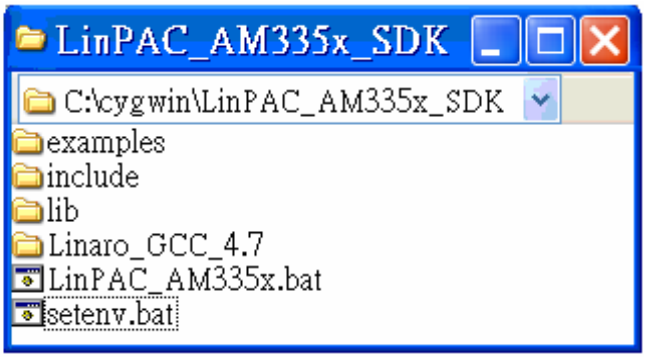


Fig. 2-14

8. From the desktop, double-click the shortcut icon for the “**LinPAC AM335x Build Environment**” or click the “Start” > ”Programs” > ”ICPDAS” > ”LP-9x21 SDK” > ” LinPAC AM335x Build Environment”.
   A Command Prompt window will then be displayed that allows applications for the LinPAC AM335x to be compiled. Refer to Fig. 2-15 and Fig 2-16.

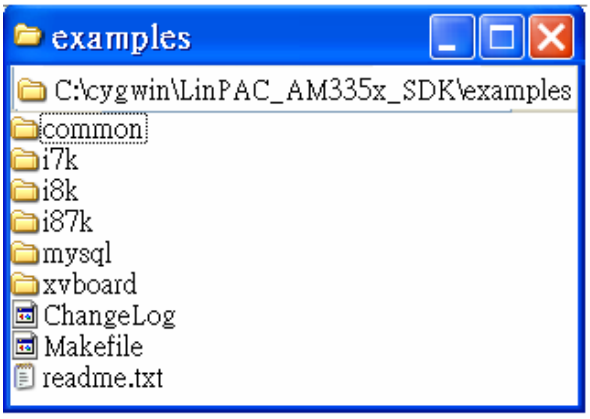


Fig. 2-15

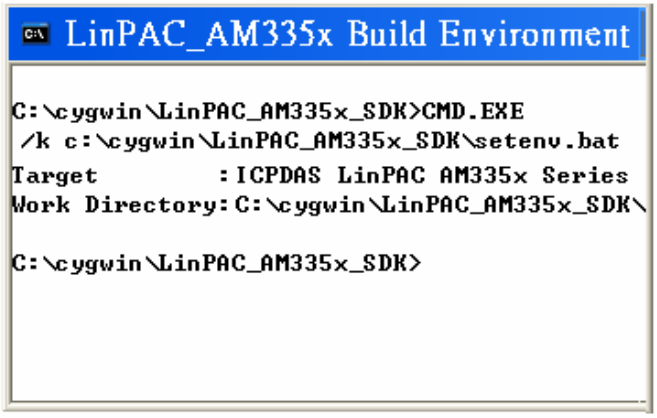


Fig. 2-16

9. Type “**make**”. A Command Prompt window will then be displayed that allows applications for the LP-9x21 to be compiled. Refer to Fig. 2-17.

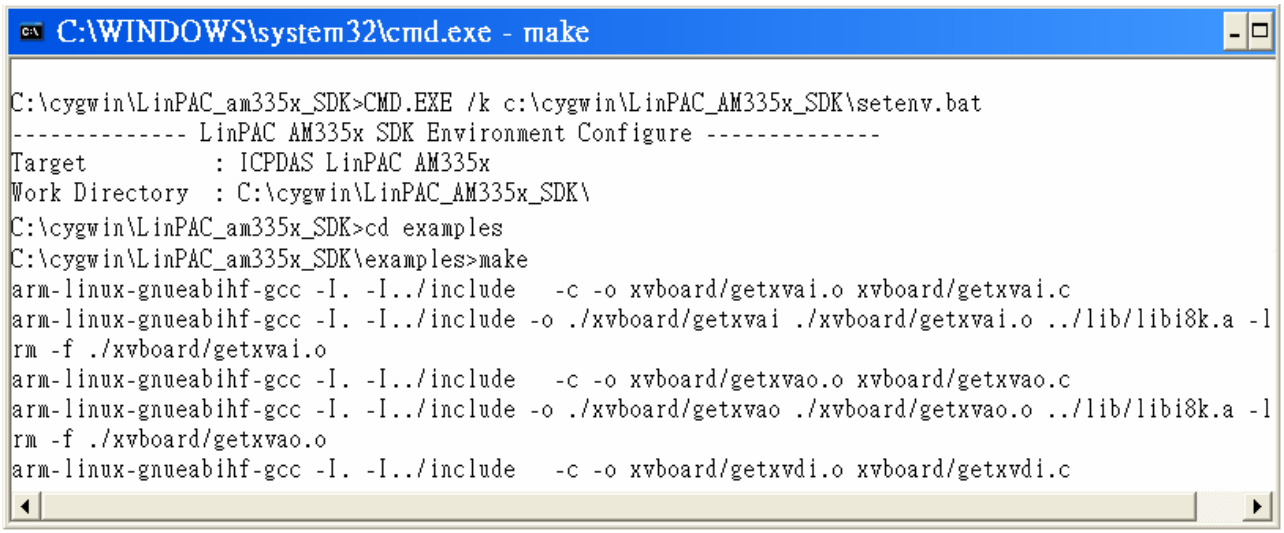


Fig. 2-17

2.1.3 Integrating SDK with Code::Blocks IDE

This tutorial gives you easy-to-follow instructions, with screenshots, for setting up a compiler (the Linaro GCC compiler), a tool that will let you turn the code that you write into programs, and Code::Blocks IDE, a free development environment. This tutorial explains how to integrate LinPCA AM335x SDK with Code::Blocks IDE on Windows platform.

Step 1: Download Code::Blocks IDE

- ❑ Go to this website: http://www.codeblocks.org/downloads/binaries
- ❑ Go to the Windows 2000/ XP / Vista / 7 section, and download Windows version.

Step 2: Install Code::Block IDE

- ❑ The default install location is the C:\Program Files\CodeBlocks folder.
- ❑ A complete manual for Code::Blocks is available here: http://www.codeblocks.org/user-manual

Step 3: Running in Code::Block IDE

- ❑ All files and settings that are included in a LinPCA_AM335x workspace file.
- ❑ Open the **C:\cygwin\CodeBlock** folder, and double click the " **LinPAC_AM335x_SDK** " as below (refer to Fig. 2-18):

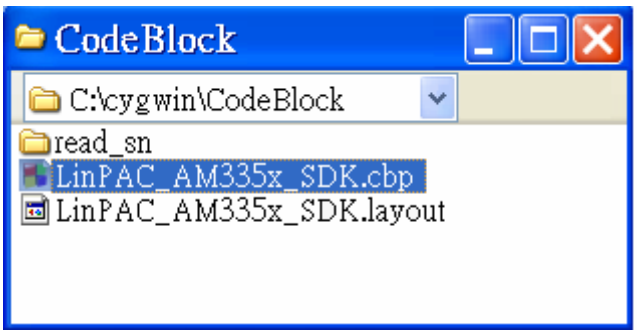


Fig. 2-18

- ❑ Following window will come up (refer to Fig. 2-19):

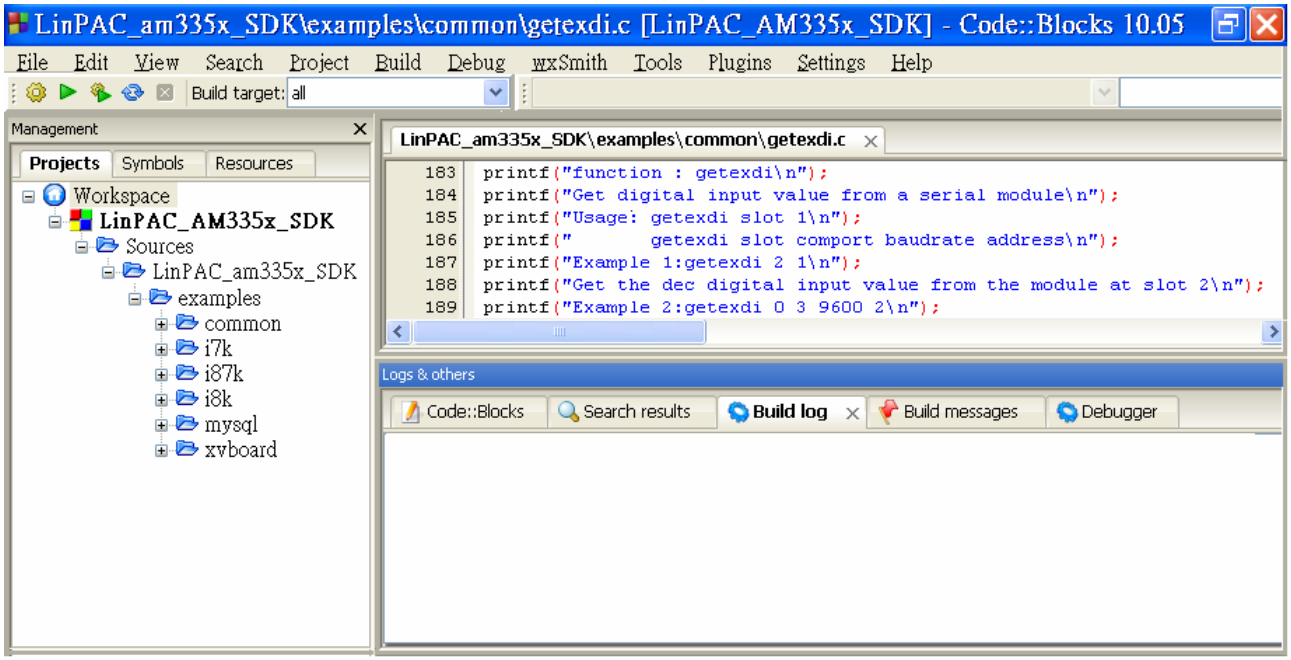


Fig. 2-19

- Check Compiler settings for Linaro GCC cross compiler : Click "Settings" > "Compiler" > "Toolchain executables tab" :

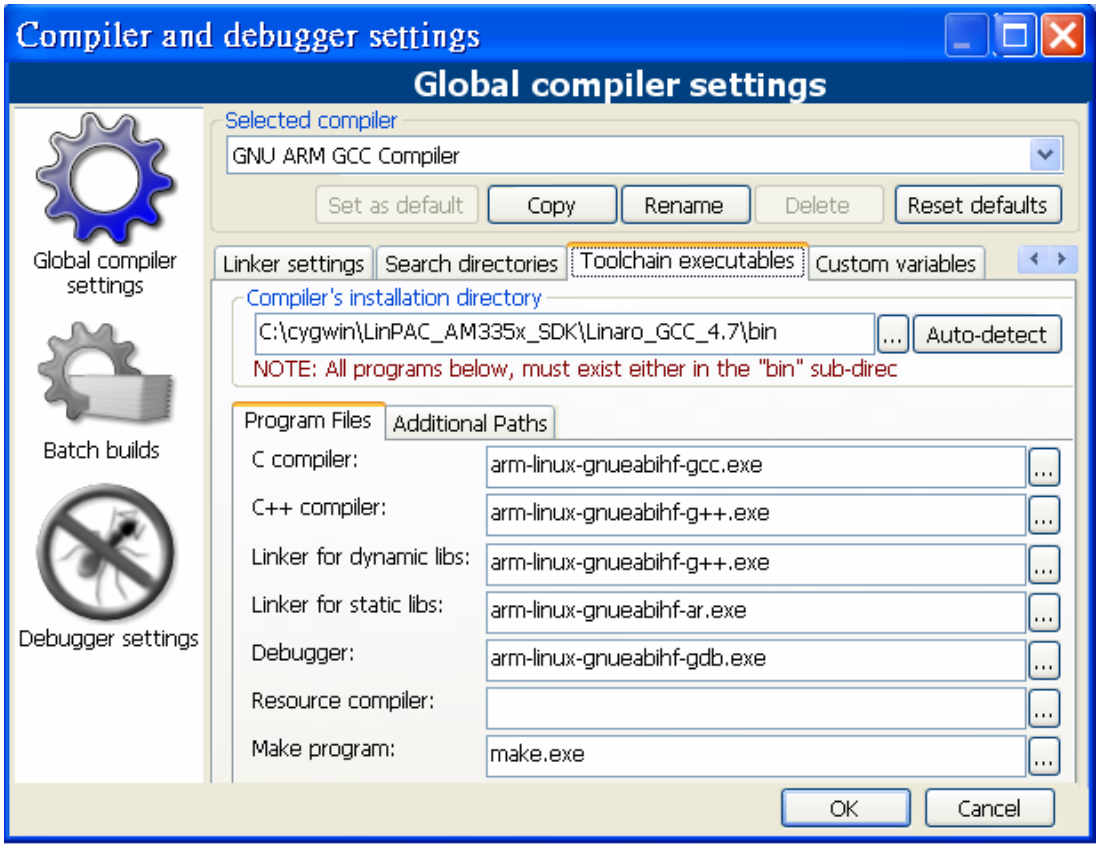


Fig. 2-20

- Click Build options, and it will compile the LinPAC_AM335x project completely.

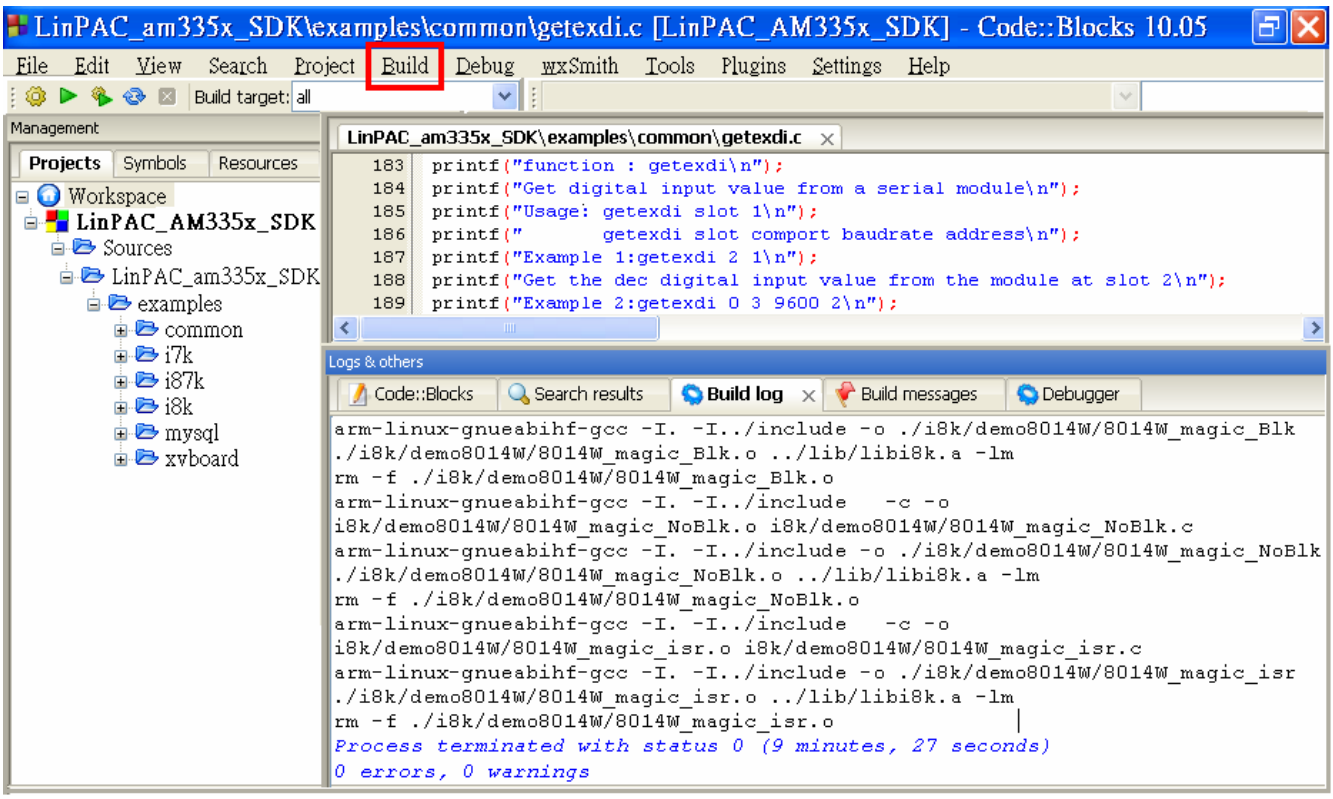


Fig. 2-21

【Note】

If you observer some characters may not display properly in cmd.exe, change the code page for the console only, do the following:

- ❑ Double-click the shortcut icon for the "**LinPAC_AM335x Build Environment**" (Refer to Fig. 2-22).

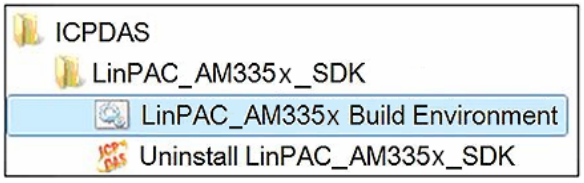


Fig. 2-22

- ❑ Type command: **chcp 65001** (Refer to Fig. 2-23 and Fig 2-24).

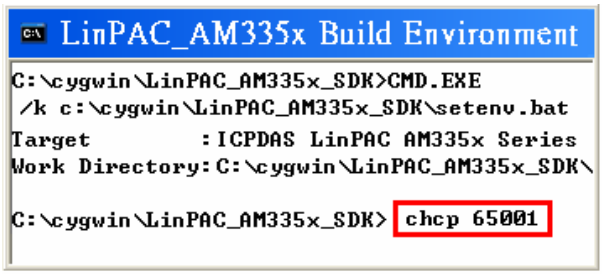


Fig. 2-23

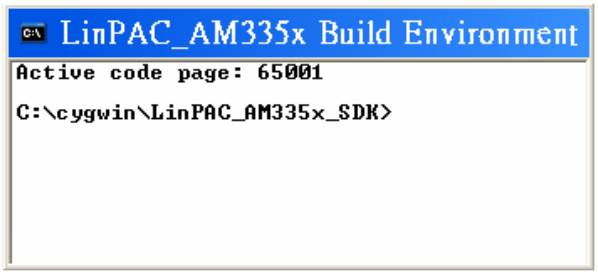


Fig. 2-24

2.2 Introduction of the LinPAC AM335x SDK

This section will discuss some of the techniques that are adopted in the LinPAC_AM335x, including detailed explanations that describe how to easily use the LinPAC_AM335x SDK. The LinPAC_AM335x SDK is based on Cygwin and is also a Linux-like environment for Microsoft Windows systems, and provides a powerful GCC cross-compiler and an IDE (Integrated Development Environment) that enables LP-9x21 applications to be quickly developed. Therefore, once an application has been created, the LinPAC_AM335x SDK can be used to compile it into an executable file that can be run on the LinPAC AM335x series embedded controller.

2.2.1 Introduction to Cygwin

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

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

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

2.2.2 Introduction to Cross-Compilation

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

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

2.2.3 Download the LinPAC AM335x SDK

- **For Windows systems : (**Extract the .exe file into to the **C:\ driver**.**)**
  Download the **linpac_am335x_sdk_for_windows.exe** file from:

ftp://ftp.icpdas.com.tw/pub/cd/linpac/napdos/lp-9x2x/sdk/linpac_am335x_sdk_for_windows.exe

- **For Linux systems : (**Extract the .bz2 file into to the **root (/) directory**.**)**
  Download the **linpac_am335x_sdk_for_linux.tar.bz2** file from:

ftp://ftp.icpdas.com.tw/pub/cd/linpac/napdos/lp-9x2x/sdk/linpac_am335x_sdk_for_linux.tar.bz2

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

3. The Architecture of LIBI8K.A in the LP-9x21

The library file **libi8k.a** is designed for I-7000/9000/87000 applications running on the LP-9x21 Embedded Controller based on the Linux operating system, and can be applied when developing custom applications **using the GNU C language**. ICP DAS provides a wide variety of demo programs that can be used to easily understand how to implement the functions and ensure that custom projects and applications can be quickly developed.

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

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

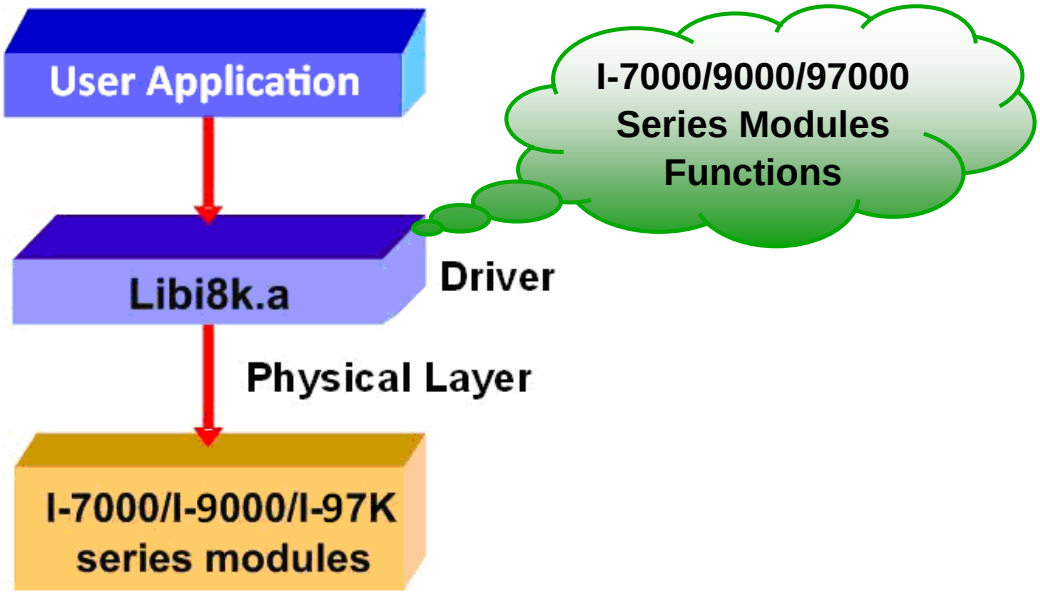


Fig. 3-1

Functions for the LP-9x21 Embedded Controller are divided into sub-groups for ease of use within the different applications:

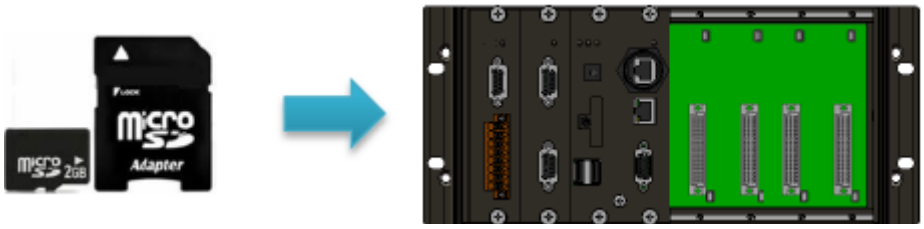
1. System Information Functions
2. Digital Input/Output Functions
3. Analog Input Functions
4. Analog Output Functions
5. 3-axis Encoder Functions
6. 2-axis Stepper/Servo Functions

The functions contained in the libi8k.a library are specifically designed for the LP-9x21 controller, and those functions needed for specific applications can easily be determined from the descriptions provided in chapter 6 and from the demo programs described in chapter 7.

4. LP-9x21 System Settings

The following is a guide to easily configuration the LP-9x21.

4.1 Using a microSD Card



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

(1) Type "**cat /proc/diskstats**" to find the device name of microSD card.

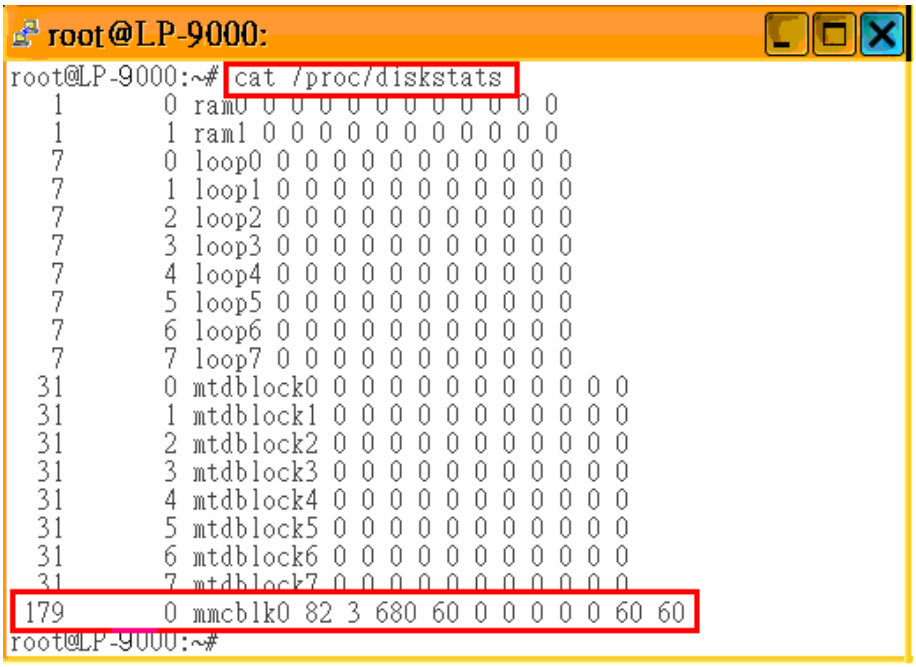


Fig 4-1

(2) Type "**mkdir /mnt/hda**" to create a directory named "hda" (Refer to Fig 4-2).

(3) Files contained on a mounted microSD card can be accessed from the **/mnt/hda** directory (Refer to Fig 4-2).

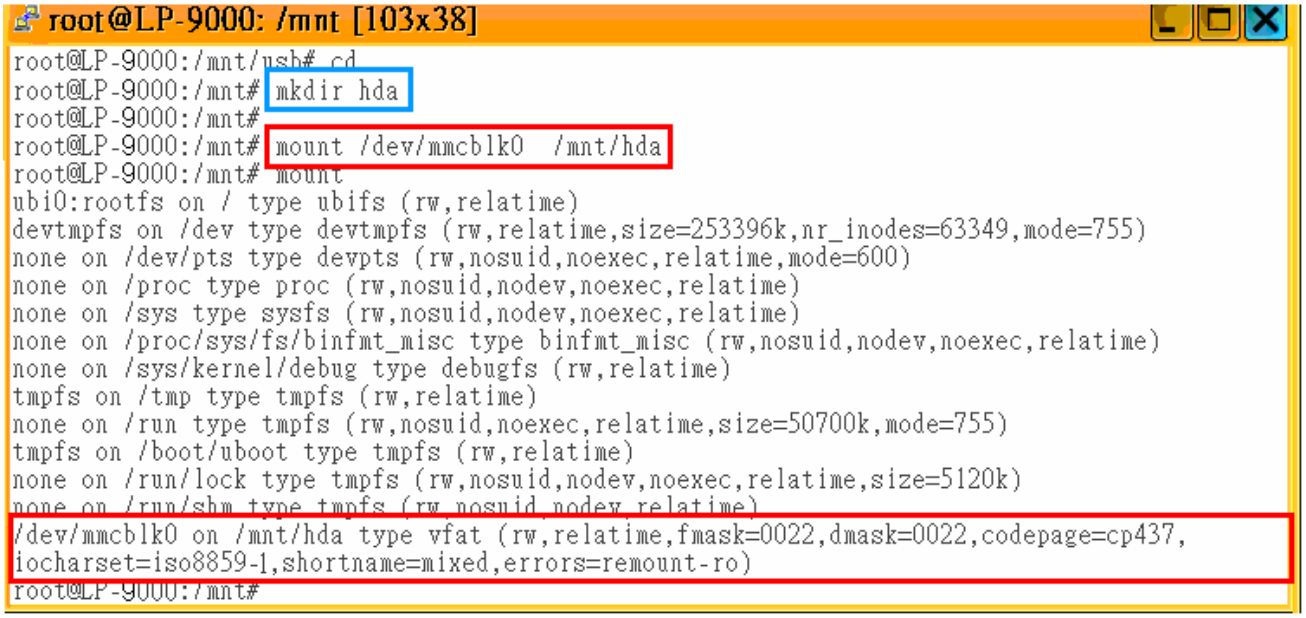


Fig 4-2

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

1. Unmount the microSD card before removing it.
2. Do not power off or reboot the LP-9x21 while data is being written to or read from the microSD card.
3. The microSD card must be formatted with the VFAT/EXT2/EXT3 file system.

4.1.1 Mounting a microSD Card

To use a microSD card, insert the microSD card into the socket on the LP-9x21, and it will be automatically mounted when the LP-9x21 is booted. The files of SD card can then be access from the **/mnt/hda** directory.

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

4.1.2 Unmounting the microSD Card

Before removing the microSD card from the LP-9x21, unmount the card by entering the following steps:

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

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

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

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

4.1.3 Scanning and repairing a microSD Card

After the LP-9x21 is booted, the microSD card will be named "**/dev/mmcblk0p1**". It is recommended that the microSD card is unmounted first before attempting to perform a scan or repair.

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

4.2 Using a USB Storage Device

USB storage devices are not automatically mounted to the LP-9x21, so it must be manually mounted before attempting to access the USB storage device.

4.2.1 Mounting a USB Storage Device

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

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

(2) Type "**cat /proc/diskstats | grep sda ***" to find the device node of usb.

(3) Type "**mount /dev/sda1 /mnt/usb**" to mount the USB storage device to the usb directory and then type "**ls /mnt/usb**" to view the contents of the USB storage device. (Refer to Fig 4-3)

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

Fig 4-3

4.2.2 Unmounting the USB Storage Device

Before removing the USB storage device from the LP-9x21, the device must be unmounted to prevent any damage to the device. To unmount the device, type the "**umount /mnt/usb**" command and then remove the USB storage device.

4.3 WDT

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

(1) Type the "**devmem2 0x01000000 b 0x81**" to set and enable WDT(Refer to Fig 4-4).

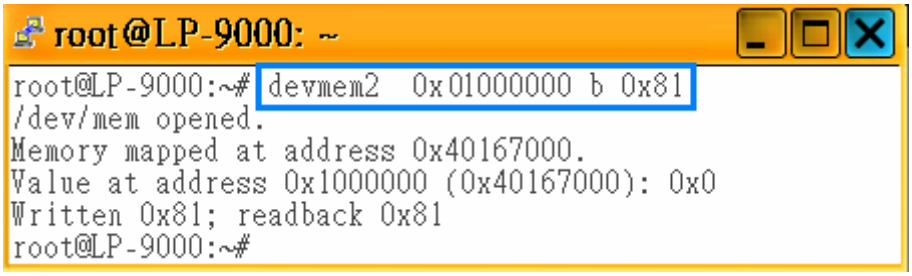


Fig 4-4

(2) Type the "**devmem2 0x01000000 b**" to read WDT setting. (Refer to Fig 4-5)

(3) Type the "**devmem2 0x0100f000 b 0x01**" to reboot WDT immediately (Refer to Fig 4-5).

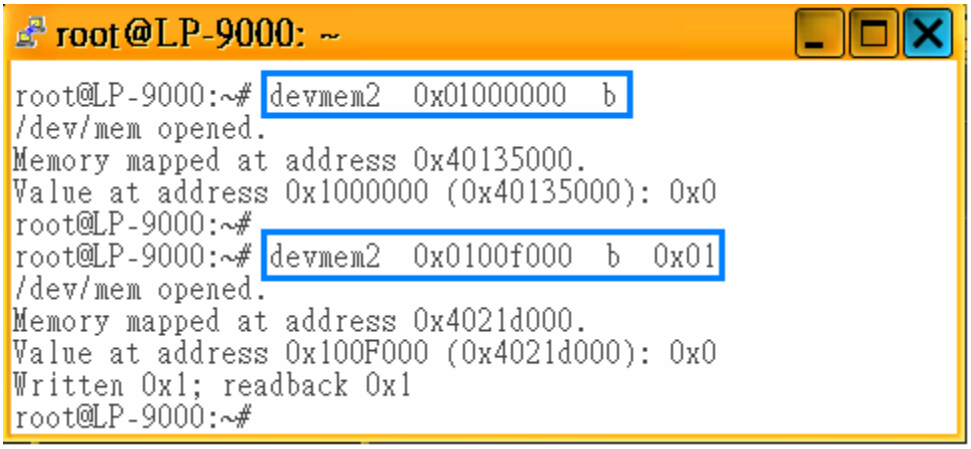


Fig 4-5

(4) Type the "**devmem2 0x0100f000 b**" to refresh WDT (Refer to Fig 4-6).

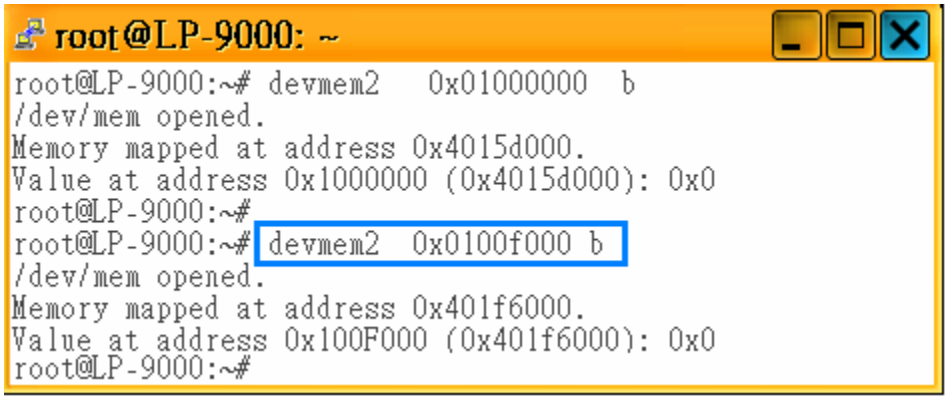


Fig 4-6

4.4 Execute Demo at Boot Time

User can refer to below steps to auto-execute demo “helloworld” at boot time in LP-8x21/9x21.

1. Copy SDK demo “i8k/examples/common/helloworld” to “/usr/sbin”

2. Create script file in “/etc/init.d”

User can use “vi” command to create the script file in “/etc/init.d” and add below script language to the file.

```
root@ LP-9000:~# vi /etc/init.d/hello
```

```
#!/bin/sh

### BEGIN INIT INFO
# Provides: ICP DAS
# Required-Start:
# Required-Stop:
# Should-Start:
# Should-Stop:
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Start and stop hello
# Description: hello
### END INIT INFO

helloworld > /tmp/test.log
```


3. Use “update-rc.d” command to add the script “hello” automatically.

```
root@ LP-9000:~# chmod +x /etc/init.d/hello
root@ LP-9000:~# update-rc.d hello defaults
```

4. After setting the file, the LP-8x21/9x21 will execute binary “helloworld” at boot time

4.5 LED Indicators

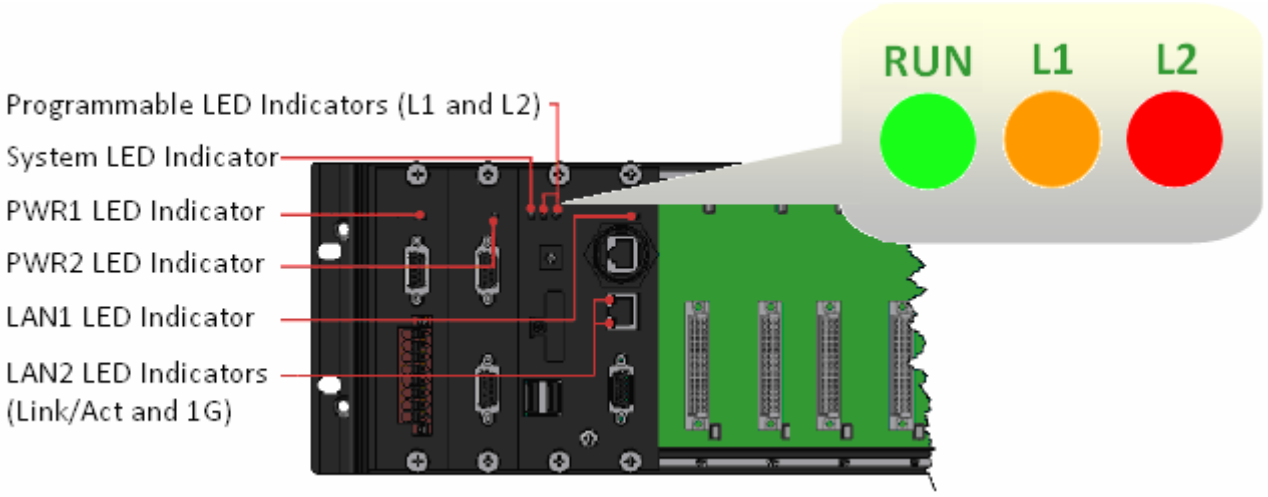


Fig 4-7

The process can be divided into four steps, user can refer to the demo code provides in LP-9x2x SDK for more details, which are described below:

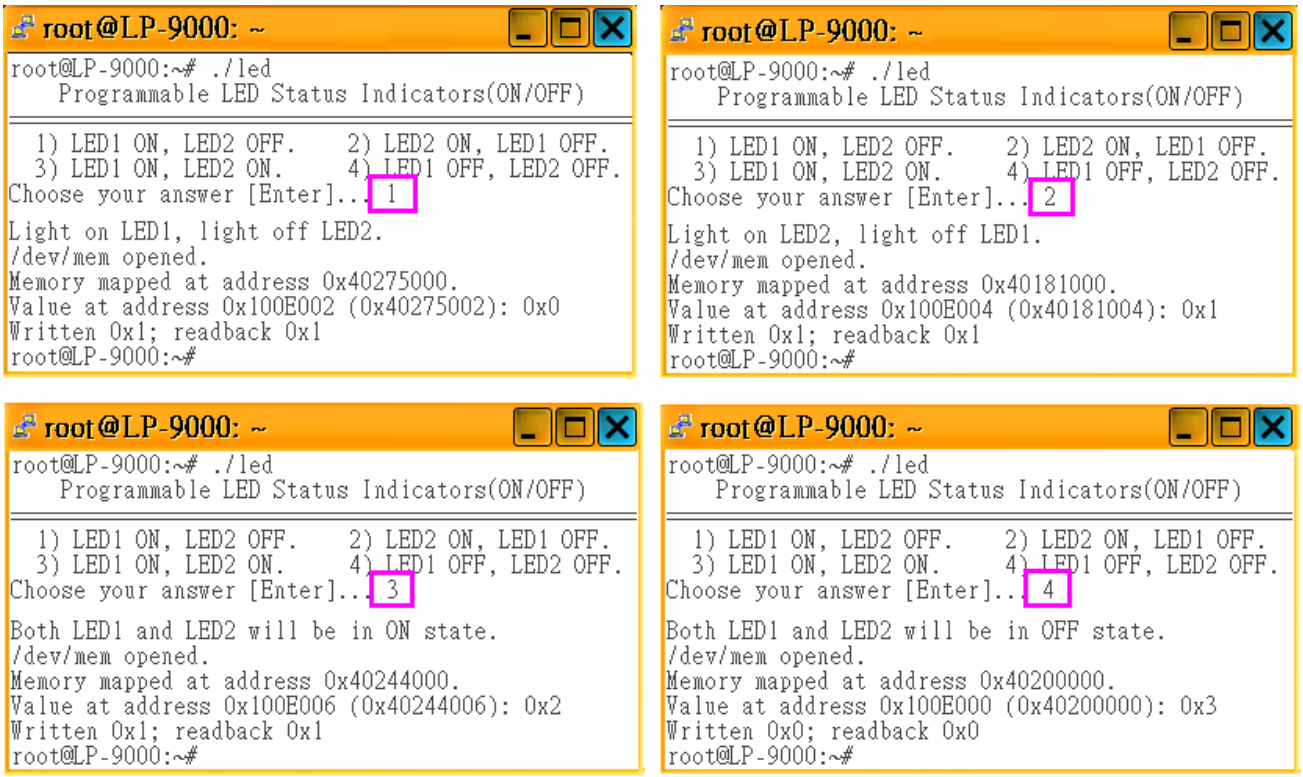


Fig 4-8

5. Instructions for the LP-9x21

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

5.1 Basic Linux Instructions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.1.10 chmod: changes the access permissions for a file

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

For example: chmod 754 test.exe

7 5 4 －＞ 111(read, write, execute)

101(read, write, execute)

100(read, write, execute)

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

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

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

5.1.11 uname: displays the Linux version information

5.1.12 ps: displays a list of the currently active procedures

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

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

！ ：temporarily exits the FTP

exit：back to the ftp

bin：transfers files in “binary” mode

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

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

bye：exits FTP

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

Syntax: telnet IPAddress

For example: telnet 192.168.0.200 (will allow remote control of the LP-9x21)

5.1.15 date: prints or sets the system date and time

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

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

5.1.17 netstat: displays the current state of the network

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

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

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

Syntax: ping IPAddress

For example：ping 192.168.0.1

5.1.20 clear: clears the screen

5.1.21 passwd: used to change the password

5.1.22 reboot: reboots the LinPAC (or use ‘shutdown -r now’)

5.2 General GCC Instructions

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

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

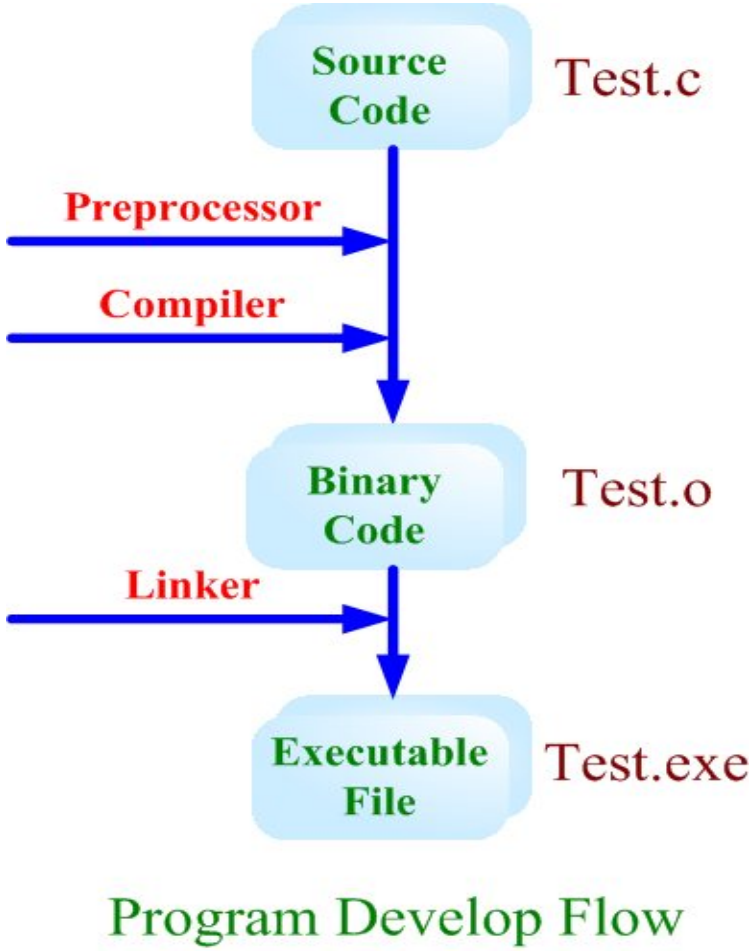


Fig. 5-1

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

5.2.1 Compile without linking to the LP-9x21 library

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

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

Parameter:

-o target: assigns the name of the output file

source.c: the C source code file

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

Output File：helloworld.exe

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

Command: arm-linux-gnueabihf-gcc –c source.c

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

Parameter:

-o target: assigns the name of the output file

source.c: the C source code file

object.o: the object file

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

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

Output File: main.exe

5.2.2 Compile by linking to the LP-9x21 library (libi8k.a)

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

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

Parameters:

-lincludeDir: the path to the include files

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

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

-o target: assigns the name of the output file

source.c: the C source code file

library: the path of library

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

Output File: test.o

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

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

Parameter:

-lincludeDir: the path of include files

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

-o target: assigns the name of the output file

source.o: the object file

library: the path to the library

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

Output File: test.exe

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

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

Parameter:

-lincludeDir: the path of include files

-lm: include math library (libm.a)

-o target: assign the name of output file

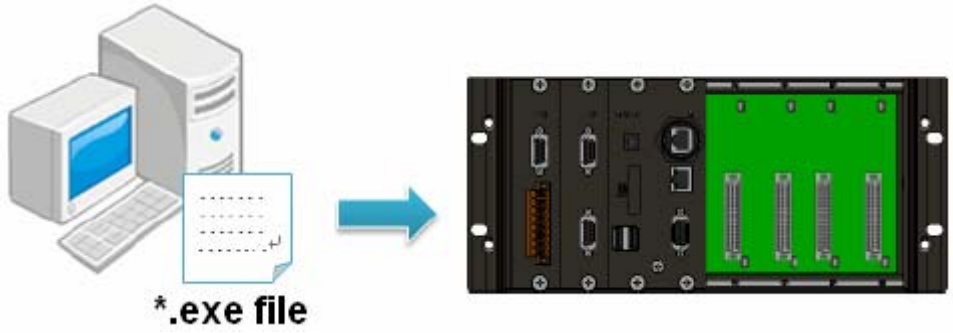
source.c: source code of C

library: the path of library

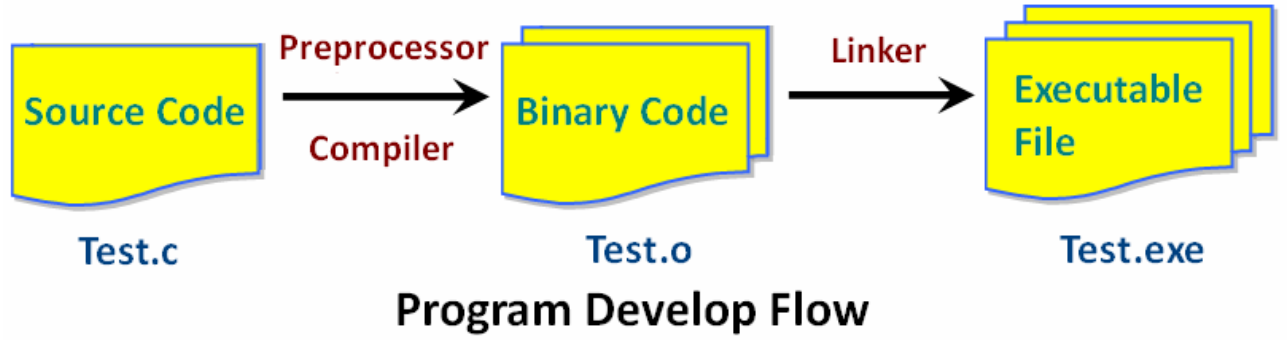
Example: arm-linux-gnueabihf-gcc –I. –I../include –lm –o test.exe test.c ../lib/libi8k.a

Output File: test.exe

5.3 A Simple Example – Helloworld.c



This section describes how to 1) compile the helloworld.c file to the helloworld.exe executable file, 2) transfer the executable file (helloworld.exe) to the LP-9x21 using FTP, and 3) execute this file via the Telnet Server on the LP-9x21.



This process can be accomplished using a single PC without requiring an additional monitor for the LP-9x21. No ICP DAS modules are used in this example. However to use ICP DAS modules to control the system, refer to the demo described in the chapter 7.

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

STEP 1: Compile helloworld.c to helloworld.exe

(1) Open the LinPAC_AM335x SDK (refer to step 8 in section 2.1) and then type "**cd examples/common**" to change the path to **C:/cygwin/LinCon8k/examples/common**.

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

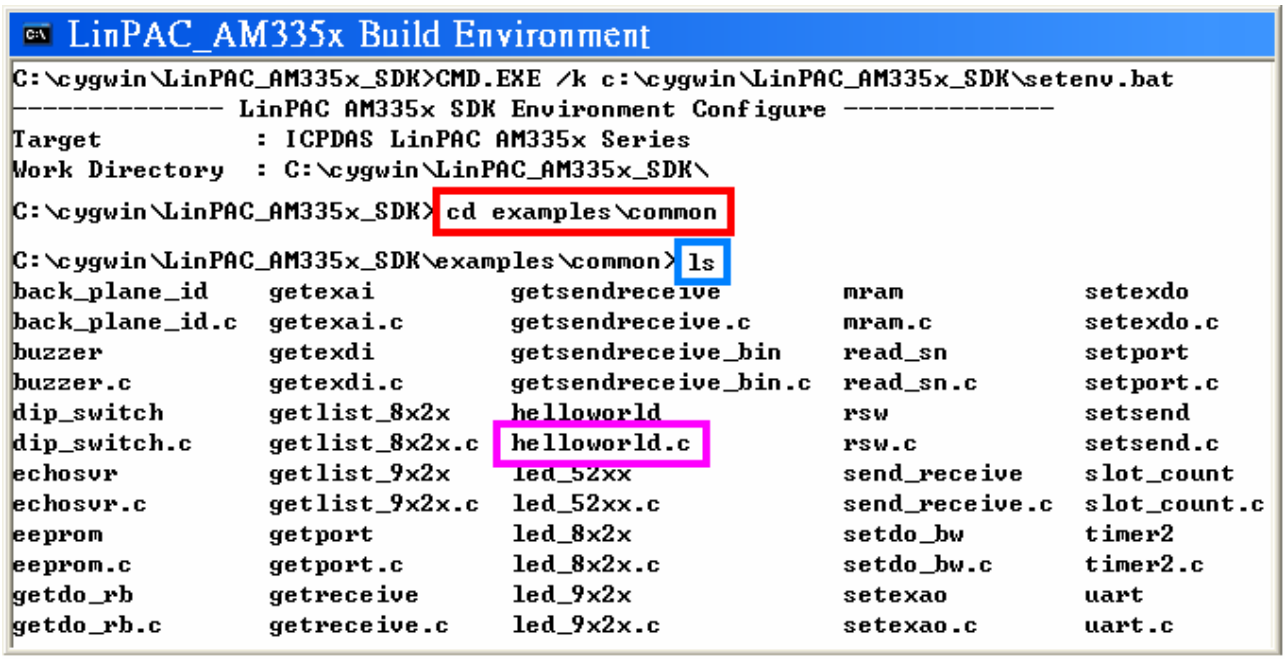


Fig. 5-2

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

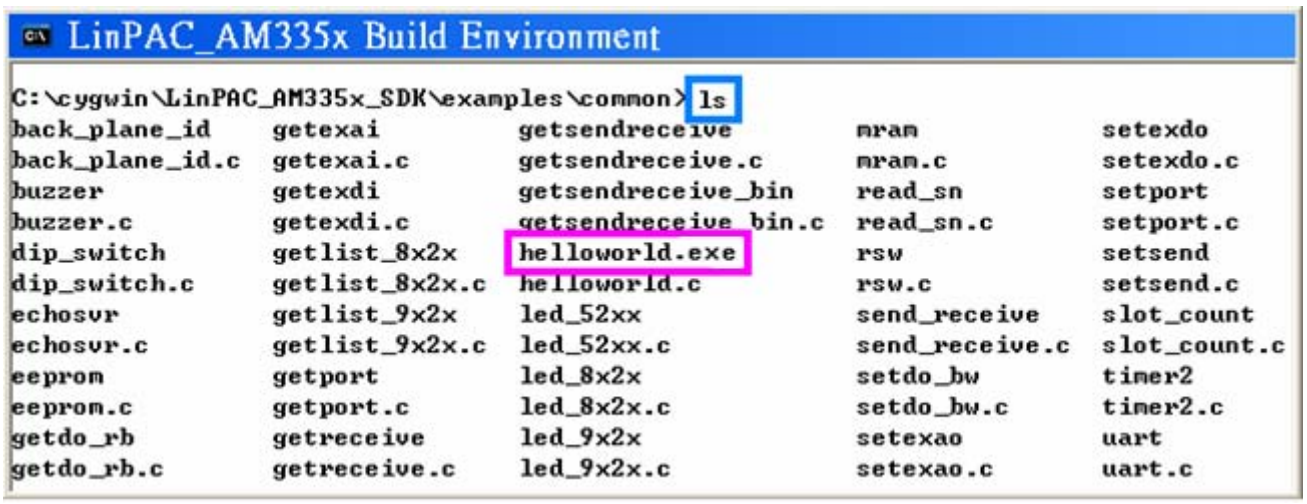


Fig. 5-3

STEP 2: Transfer helloworld.exe to the LP-9x21

Two methods can be used to transfer files to the LP-9x21:

＜Method one＞ Using the "DOS Command Prompt"

(1) Open a "DOS Command Prompt" and type the ftp IP Address of the LP-9x21, for example: **ftp 192.168.0.200** to establish a connection to the FTP Server on the LP-9x21. When prompted, type the **User_Name (default value is "root")** and **Password (default value is "icpdas")** to establish a connection to the LP-9x21.

(2) Before transferring the files to the LP-9x21, type the "**bin**" command to ensure that the file is transferred to the LP-9x21 in **binary mode**. (refer to Fig. 5-4)

(3) Type the command "**put C:/cygwin/LinCon8k/examples/common/helloworld.exe helloworld.exe**" to transfer the helloworld.exe file to the LP-9x21. Once the message "**Transfer complete**" is displayed, then transfer process has been completed. To disconnect from the LP-9x21, type the "**bye**" command to return to the PC console (refer to Fig. 5-4).

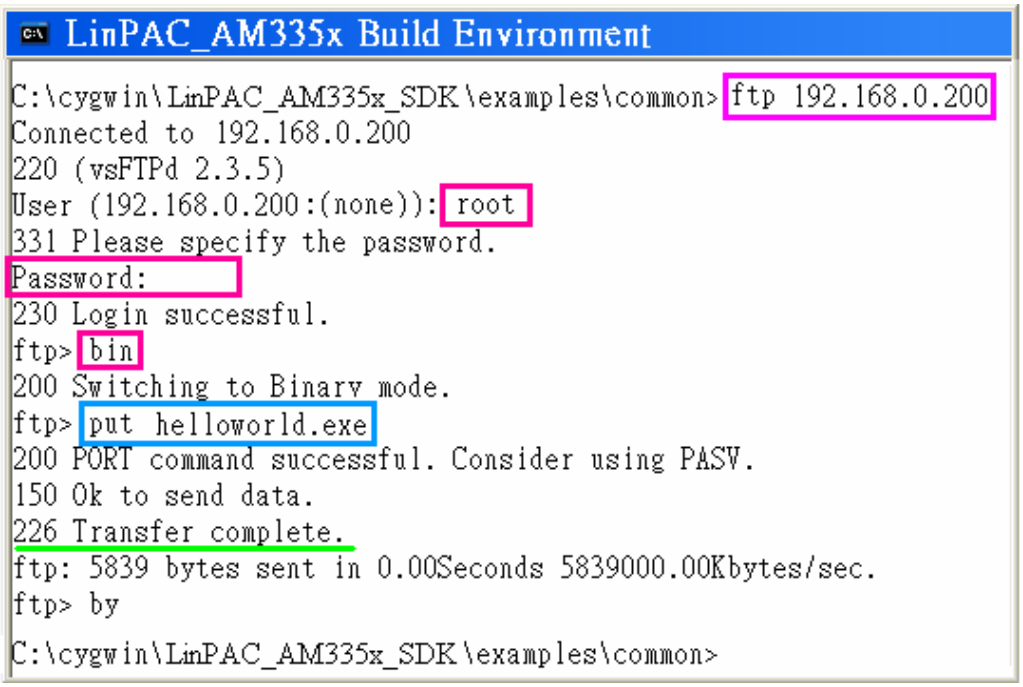


Fig.5-4

＜Method two＞ Using an FTP Client:

(1) Open the FTP Software (for example, FileZilla - The free FTP solution for both client and server, https://filezilla-project.org/) and add an FTP Host to the LP-9x21. Type the **User_Name (default value is "root")** and **Password (default value is "icpdas")**. Then click the **"Connect"** button to establish a connection to the ftp server on the LP-9x21. (refer to Fig. 5-5).

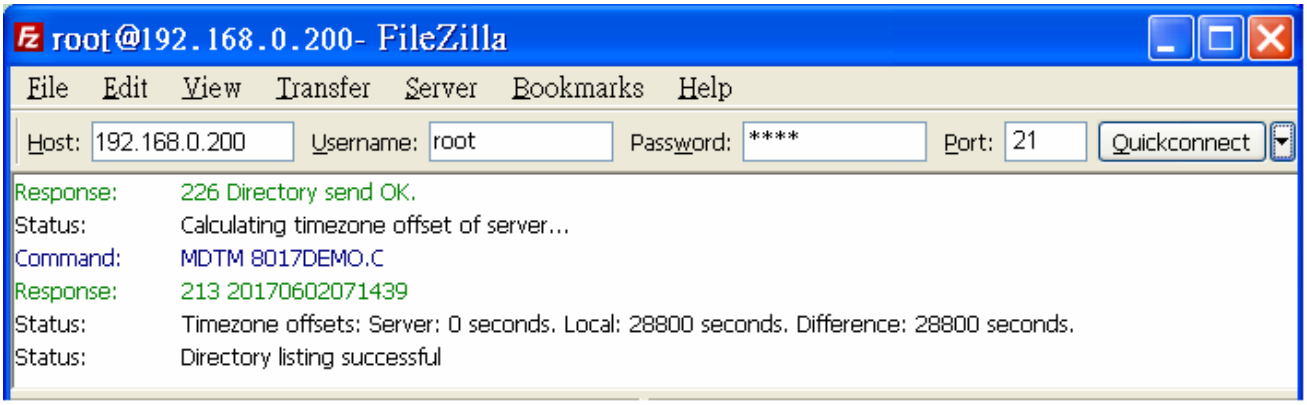


Fig.5-5

(2) Upload the "**Helloworld.exe**" file to the LP-9x21. (refer to Fig. 5-6).

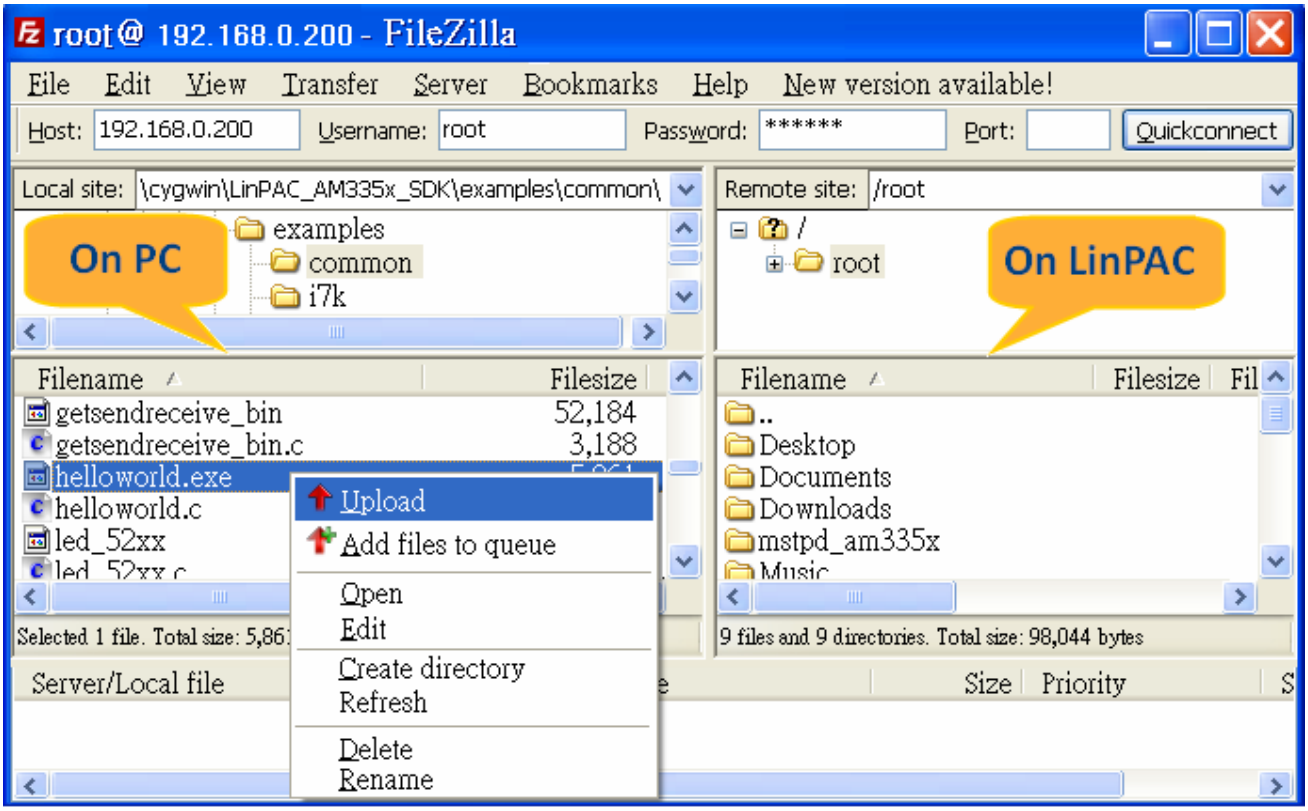


Fig. 5-6

(3) Click the helloworld.exe file in the LP-9x21 to select it and then right click the file icon and click the **"Permissions"** option. In the Properties dialog box, type 777 into the Numeric textbox, and then click the OK button. Refer to Fig. 5-7 and Fig. 5-8 for more details.

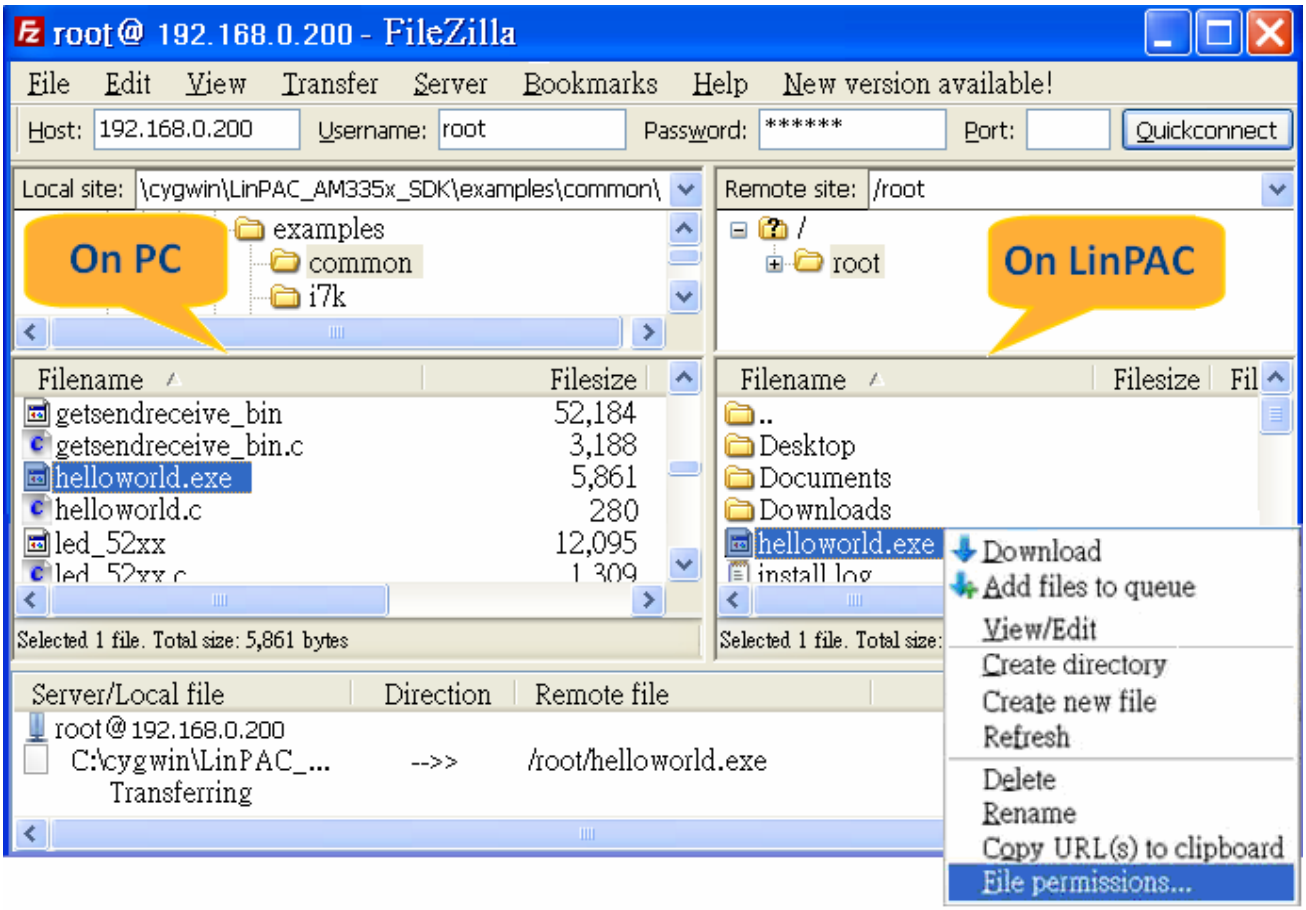


Fig.5-8

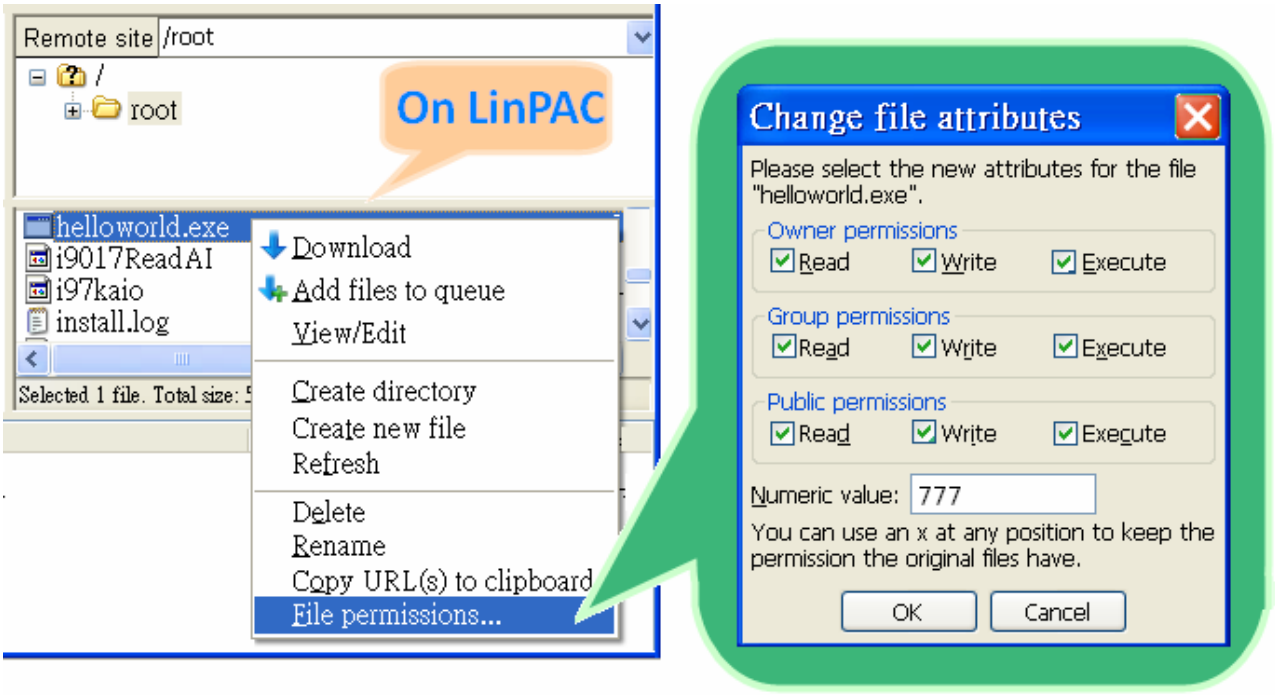


Fig.5-8

STEP 3: Use telnet to access the LP-9x21 and execute the program

(1) Open a “DOS Command Prompt” or a “Putty Prompt” which is an SSH and telnet client. Download PuTTY tool: http://www.putty.org/

(2) Type the IPAddress of the LP-9x21, for example: **telnet 192.168.0.200**, to establish a connection to the Ttelnet Server on the LP-9x21. When prompted, type the **User_Name** and **Password** to establish a connection to the LP-9x21. If the **“#”** prompt character is displayed, it signifies that a connection to the telnet server on the LP-9x21 has been successfully established.

(3) Type the “**ls -l**“ command to list all the files in the /root directory and verify that the helloworld.exe file is present. Type the “**chmod 777 helloworld.exe**” command to change the permissions for the helloworld.exe file. Type the “**ls -l**“ command again to list all the files in the /root directory and verify the permissions assigned to the “helloworld.exe” file. This means that the file is executable. Execute the “**./helloworld.exe**“ file by typing and the message “ICPDAS hello world!” will be displayed.

The **compile**, **transfer** and execution processes are now complete. (refer to Fig. 5-9)

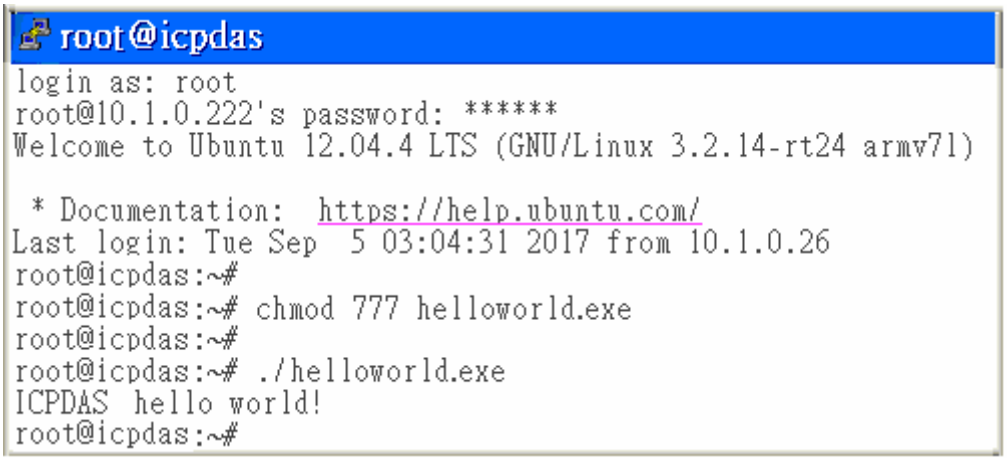


Fig.5-9

5.4 i-Talk Utility

The **i-Talk utility** provides **fifteen instructions** that make it convenient for users to access the modules and the hardware in the LP-9x21 and can be found in the path – **/usr/sbin/iTalk**. An overview of the i-Talk utility functions is given below: (Refer to Fig. 5-12)

Instruction	Description
getlist	Lists the names of all modules inserted in the LinPAC-9000
setdo	Sets the Digital Output value for I-9K modules
setao	Sets the Analog Output value for I-9K modules
getdi	Reads the Digital Input value for I-9K modules
getai	Reads the Analog Input value for I-9K modules
setexdo	Sets the Digital Output value for I-7K/97K modules
setexao	Sets the Analog Output value for I-7K/97K modules
getexdi	Reads the Digital Input value for I-7K/97K modules
getexai	Reads the Analog Input value for I-7K/97K modules
setport	Sets the Port offset value for the module
getport	Reads the Port offset value for the module
getsend	Sends a string from the LinPAC COM port
getreceive	Receives a string from the LinPAC COM port
getsendreceive	Sends/Receives a string from the LinPAC COM port

Fig. 5-12

The Fig. 5-13 provides details of the demo programs that illustrate how to use the I-talk utility. The **I-9024** (AO Module), **I-9017** (AI Module) and the **I-9055W** (DIO Module) are used in the examples shown, and they are inserted in slots 1, 2 and 3 of the LinPAC respectively. Typing the name of the instruction will display usage details for the instruction.

Instruction	Example
getlist	*getlist* Lists the names of all modules inserted in the LinPAC-9000
setdo	*setdo {slot} {data}* ***setdo 3 3*** Sets channels 1 and 2 on the I-8055W module to ON
setao	*setao {slot} {channel} {data}* ***setao 1 0 2.2*** Sets the Analog Output value for channel 0 on the I-9024 module to 2.2 V.
getdi	*getdi {slot} {type}* ***getdi 3 8*** Reads the 8-bit Digital Input value from the I-9055W module
getai	*getai {slot} {channel} {gain} {mode}* ***getai 2 0 0 0*** Reads the Analog Input value from the I-9017 module
setexdo	*setexdo {slot} {1} {data}* **setexdo 2 1 3** Sets the digital output value to the I-97K module at slot 2 *setexdo {slot} {comport} {data} {baudrate} {address}* **setexdo 0 3 55 9600 2** Sets the dec digital output value to the I-7K module at COM3 port
setexao	*setexao {slot} {1} {data} {channel}* **setexao 3 1 6.7 5** Sets channel 5 analog value 6.7 to the I-97K module at slot 3 *setexao {slot} {comport} {data} {channel} {baudrate} {address}* **setexao 0 3 6.7 5 9600 1** Sets channel 5 analog value 6.7 to the I-7K module at COM3 port
getexdi	*getexdi {slot} {1}* **getexdi 2 1** Get the dec digital input value from the I-97Kmodule at slot 2 *getexdi {slot} {comport} {baudrate} {address}* **getexdi 0 3 9600 2** Get the dec digital input value from the I-7K module at COM3 port
getexai	*getexai {slot} {1} {channel}* **getexai 2 1 5** Get channel 5 analog value from the I-97K module at slot 2 *getexai {slot} {comport} { channel } {baudrate} {address}* **getexai 0 3 5 115200 2** Get channel 5 analog value from the I-7K module at COM3 port

Fig. 5-13

6. LIBI8K.A

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

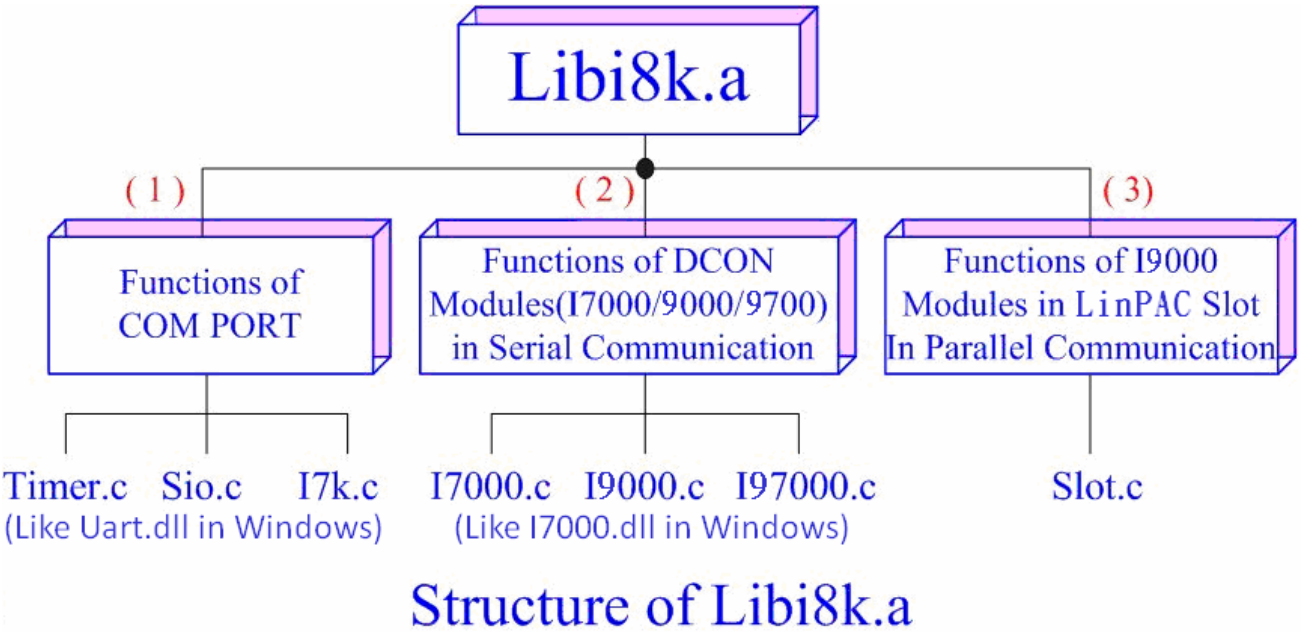


Fig. 6-1

The functions node (1) and (2) in the diagram for the Libi8k.a library are the same as those of the DCON.DLL Driver (including Uart.dll and I7000.dll) as used in the DCON modules (I-7000/ I-9000/ I-97000 for serial communication). For more details, refer to the DCON.DLL Driver manual which includes adescription of how to use the functions on DCON modules. The DCON.DLL Driver has now been included in the Libi8k.a library.

Functions (3) in the diagram contains the most important functions, as they are especially designed for I-9000 modules inserted in the LP-9x21 slots. They are different from functions (1) and (2) because the communication method for I-9000 modules inserted in the LP-9x21 is based on parallel mode rather than serial mode. Accordingly, the I9000.c library has been completely rewritten especially for I-9000 modules inserted in LP-9x21 slots and has been rename as slot.c. The following is an introduction to the funcitions for slot.c, which can be arranged into four main categories:

1. System Information Functions
2. Digital Input/Output Functions
3. Analog Output Functions
4. Analog Input Functions

Note that when using a development tool to create develop applications, the **msw.h** file must be included in the header of the source program, and the **Libi8k.a** library file must also be linked. To control ICP DAS remote I/O modules such as the I-7K, I-9K and I-97K series modules **via the COM2 or COM3 or COM36 ports on the LP-9x21**, the functions are the same as those included in the DCON DLL. To control **I-9K series modules** that are inserted in the slots of the LP-9x21, the functions are different and they are described in more detail below:

6.1 System Information Functions

■ Open_Slot

Description:

This function is used to open and initialize a specific slot on the LP-9x21, and will be used by I-9K or I-97K modules inserted in the LP-9x21. For example, to send or receive data from a specific slot, this function must be called first before any other functions can be used.

Syntax:

[C]

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

Parameter:

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

Return Values:

0: The slot was successfully initialized.

Other: The initialization failed.

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

Example:

```
Int slot=1;
Open_Slot(slot);
// The first slot in the LP-9x21 will be open and initiated.
```

Remark:

■ Close_Slot

Description:

After using the of Open_Slot() function to open and initialize a specific slot on the LP-9x21, the Close_Slot() function must also be used to close the slot. This function will be used by I-9K or I-97K series modules inserted in the LP-9x21. For example, the Close_Slot() function should be called after sending or receiving data from the specified slot.

Syntax:

[C]

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

Parameter:

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

Return Value:

None

Examples:

```
int slot=1;
Close_Slot(slot);
// The first slot in the LP-9x21 will be closed.
```

Remark:

■ Open_SlotAll

Description:

This function is used to open and initialize **all slots** on the LP-9x21. For example, to send or receive data from multiple slots, this function can be used to simplify the program, and other functions can be used.

Syntax:

[C]

```
int Open_SlotAll(void)
```

Parameter:

None

Return Value:

0: The slot was successfully initialized.

Other: The initialization failed.

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

Examples:

```
Open_SlotAll();
// All slots in the LP-9x21 will be open and initiated.
```

Remark:

■ Close_SlotAll

Description:

If you the Open_SlotAll() function was used to open and initialize all the slots on the LP-9x21, the Close_SlotAll() function can be used to quickly close them simultaneously. For example, the Close_SlotAll() function can be called after sending or receiving data from multiple slots to close all the slots at the same time.

Syntax:

[C]

```
void Close_SlotAll(void)
```

Parameter:

None

Return Value:

None

Examples:

```
Close_Slot();
// All slots in the LP-9x21 will be closed.
```

Remark:

■ ChangeToSlot

Description:

This function is used to assign serial control to the specified slots for to allow control of the I-97K series. The serial bus on the backplane of the LP-9x21 is used for mapping through to COM1. For example, to send or receive data from a specified slot, this function should be called first, and then other series functions can be used.

Syntax:

[C]

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

Parameter:

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

Return Value:

None

Examples:

```
char slot=2;
ChangeToSlot (slot);
// The first slot on the LP-9x21 is specified as the COM1 port.
```

Remark:

■ Open_Com

Description:

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

Syntax:

[C]

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

Parameter:

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

baudrate: [Input] 1200/2400/4800/9600/19200/38400/57600/115200

cDate : [Input] Data5Bit, Data6Bit, Dat7Bit, Data8Bit

cParity : [Input] NonParity, OddParity, EvenParity

cStop : [Input] OneStopBit, TwoStopBit

Return Values:

0: The com port was successfully initialized.

Other: The initialization failed.

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

Examples:

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

Remark:

■ Close_Com

Description:

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

Syntax:

[C]

```
BOOL Close_Com(char port)
```

Parameter:

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

Return Value:

None

Example:

Close_Com (COM3);

Remark:

■ Send_Receive_Cmd

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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



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

Remark:

■ Send_Cmd

Description:

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

Syntax:

[C]

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

Parameter:

port : [Input] 1=COM1, 2=COM2, 3=COM3..., 255=COM255.

szCmd : [Input] Sending command string

wTimeOut : [Input] Communicating timeout setting, the unit=1ms

wChkSum : [Input] 0=Disable, 1=Enable

Return Value:

None

Examples:

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

Remark:

■ Receive_Cmd

Description:

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

Syntax:

[C]

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

Parameter:

port : [Input] 1=COM1, 2=COM2, 3=COM3..., 255=COM255.

szResult : [Output] Sending command string

wTimeOut : [Input] Communicating timeout setting, the unit=1ms

wChkSum : [Input] 0=Disable, 1=Enable

Return Value:

None

Examples:

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

Remark:

■ Send_Binary

Description:

Send out the command string by fix length, which is controlled by the parameter “iLen”. The difference between this function and Send_cmd is that Send_Binary terminates the sending process by the string length “iLen” instead of the character "CR"(Carry return). Therefore, this function can send out command string with or without null character under the consideration of the command length. Besides, because of this function without any error checking mechanism (Checksum, CRC, LRC... etc.), users have to add the error checking information to the raw data by themselves if communication checking system is required. Note that this function is usually applied to communicate with the other device, but not for ICP DAS DCON (I-7000/9000/97K) series modules.

Syntax:

[C]

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

Parameter:

port : [Input] 1=COM1, 2=COM2, 3=COM3..., 255=COM255.

szCmd : [Input] Sending command string

iLen : [Input] The length of command string.

Return Value:

None

Examples:

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

Remark:

■ Receive_Binary

Description:

This function is applied to receive the fix length response. The length of the receiving response is controlled by the parameter “iLen”. The difference between this function and Receive_cmd is that Receive_Binary terminates the receiving process by the string length “iLen” instead of the character "CR"(Carry return). Therefore, this function can be used to receive the response string data with or without null character under the consideration of receiving length. Besides, because of this function without any error checking mechanism (checksum, CRC, LRC... etc.), users have to remove from the error checking information from the raw data by themselves if communication checking system is used. Note that this function is usually applied to communicate with the other device, but not for ICP DAS DCON (I-7000/9000/97K) series modules.

Syntax:

[C]

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

Parameter:

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

Return Value:

None

Examples:

```
int m_length=10;
char m_port=3;
char m_szSend[40];
char m_szReceive[40];
DWORD m_baudrate=115200;
WORD m_wt;
WORD m_timeout=10;
```



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

Remark:

■ sio_open

Description:

This function is used to open and initiate a specified serial port in the LP-9x21. The n-port modules in the LP-9x21 will use this function. For example, if you want to send or receive data from a specified serial port, this function must be called first. Then the other functions can be used later.

Syntax:

[C]

```
int sio_open(const char *port, speed_t baudrate, tcflag_t data, tcflag_t parity,
             tcflag_t stop)
```

Parameter:

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

Return Value:

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

ERR_PORT_OPEN is for Failure.

Examples:

```
#define   COM_M1   "/dev/ttyS2"          // Defined the first port of I-9144 in slot 1
char fd[5];
fd[0]=sio_open(COM_M1, B9600, DATA_BITS_8, NO_PARITY,ONE_STOP_BIT);
if (fd[0] == ERR_PORT_OPEN) {
      printf("open port_m failed!\n");
      return (-1);
}
// The I-9114 is inserted in slot 1 and the first port will be open and initiated.
```

Remark:

This function can be applied on modules: I-9114W and I-9144.

■ sio_close

Description:

If you have used the function of sio_open() to open the specified serial port in the LP-9x21, you need to use the sio_close() function to close the specified serial port in the LP-9x21. For example, once you have finished sending or receiving data from a specified serial port, this function would then need to be called.

Syntax:

[C]

```
int sio_close(int port)
```

Parameter:

port : [Input] device name: /dev/ttyS2, /dev/ttyS3…/dev/ttyS34

Return Value:

None

Examples:

```
#define   COM_M2   "/dev/ttyS3"      // Defined the second port of I-9144 in slot 1
char fd[5];
fd[0]=sio_open(COM_M2, B9600, DATA_BITS_8, NO_PARITY,ONE_STOP_BIT);
sio_close (fd[0]);
// The second port of I-9144 in slot 1 will be closed.
```

Remark:

This function can be applied on modules: I-9114 and I-9144.

■ GetModuleType

Description:

This function is used to retrieve which type of 9000 series I/O module is inserted in a specific I/O slot in the LP-9x21. This function performs a supporting task in the collection of information related to the system's hardware configurations.

Syntax:

[C]

```
int GetModuleType(int slot)
```

Parameter:

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

Return Value:

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

Type	Value
_PARALLEL	0x80
_AI	0xA0
_AO	0xA1
_DI8	0xB0
_DI16	0xB1
_DI32	0xB2
_DO6	0xC0
_DO8	0xC1
_DO16	0xC2
_DO32	0xC3
_DI4DO4	0xD0
_DI8DO8	0xD1
_DI16DO16	0xD2
_MOTION	0xE2
_CAN	0XF0

Examples:

```
int slot=1;
int moduleType;
Open_Slot(slot);
printf("GetModuleType= 0x%X \n", GetModuleType(slot));
Close_Slot(slot);
// The I-9041 card is inserted in slot 1 of LP-9x21 and has a return Value : 0xC3
```

Remark:

■ GetNameOfModule_9K

Description:

This function is used to retrieve the name of an 9000 series I/O module, which is plugged into a specific I/O slot in the LP-9x21. This function supports the collection of system hardware configurations.

Syntax:

[C]

```
int GetNameOfModule_9K(int slot)
```

Parameter:

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

Return Value:

I/O module ID. For Example, the I-9017 will return 9017.

Examples:

```
int slot=1;
int moduleID;
Open_Slot(slot);
moduleID= GetNameOfModule_9K(slot);
Close_Slot(slot);
// The I-9017 module is inserted in slot 1 of LP-9x21.
// Returned Value: moduleName=” 9017 ”
```

Remark:

■ Read_SN

Description:

This function is used to retrieves the hardware serial identification number on the LP-9x21 main controller. This function supports the control of hardware versions by reading the serial ID chip

Syntax:

[C]

```
void read_sn(char sn[])
```

Parameter:

sn : [Output] Receive the serial ID number.

Return Value:

None

Examples:

```
int rs = 0;
char sn[16];
rs = read_sn(sn);
if(rs)
    printf("read sn fail!\n");
else
    printf("LP-9x21 SN : %s\n", sn);
```

Remark:

■ GetBackPlaneID

Description:

This function is used to retrieve the back plane ID number in the LP-9x21.

Syntax:

[C]

```
int GetBackPlaneID()
```

Parameter:

None

Return Value:

Backplane ID number.

Examples:

```
int id;
id=GetBackPlaneID();
printf("GetBackPlanel =%d \n", id);
// Get the LP-9x21 backplane id . Returned Value: GetBackPlanel = 13
```

Remark:

■ GetSlotCount

Description:

This function is used to retrieve the number of slot in the LP-9x21.

Syntax:

[C]

```
int GetSlotCount()
```

Parameter:

None

Return Value:

Number of slot.

Examples:

```
int number;
number= GetSlotCount();
printf("GetSlotCount =%d \n", number);
// Get the LinPAC-9821 slot count.
// Returned Value: GetSlotCount = 8
```

Remark:

■ GetDIPswitch

Description:

This function is used to retrieve the DIP switch value in the LP-9x21.

Syntax:

[C]

```
int GetDIPswitch()
```

Parameter:

None

Return Value:

DIP switch value.

Examples:

```
int value;
value= GetDIPswitch();
printf("GetDIPswitch =%d \n", value);
// Get the LP-9x21 DIP switch value.
// Returned Value: GetDIPswitch = 128
```

Remark:

This function can be applied on PAC: LP-9421, LP-9821.

■ rotary_switch_read

Description:

This function is used to retrieve the rotary ID number in the LP-9x21.

Syntax:

[C]

```
int rotary_switch_read (&value)
```

Parameter:

value: [Output] Rotary switch ID number.

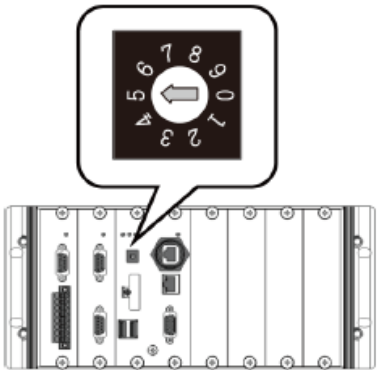
Return Value:

0: The slot was successfully initialized.

Other: The initialization failed.

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

Examples:



```
int    result=0 ;
unsigned char value=0;
rotary_switch_read (&value);
if(result)
{
      printf("rsw(%d) : rotary switch read error\n",result);
      return FAILURE;
}
else
{
      printf("%d\n", value);        //Get the LP-9x21 rotary id
}
```

If user turn the rotary switch to the 1 position, would get the returned value: 1.

Remark:

Rsw	0	1	2	3	4	5	6	7	8	9
ID	0	1	2	3	4	5	6	7	8	9

■ GetSDKversion

Description:

This function is used to retrieve the version of LP-9x21 SDK.

Syntax:

[C]

```
float GetSDKversion(void)
```

Parameter:

None

Return Value:

Version number.

Examples:

```
printf(" GetSDKversion = %4.2f \n ", GetSDKversion());
// Get the LP-9x21 SDK version number.
// Returned Value: GetSDKversion = 1.
```

Remark:

6.2 Digital Input/Output Functions

6.2.1 For I-9000 modules via parallel port

■ DO_8

Description:

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

Syntax:

[C]

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Examples:

```
int slot=1;
unsigned char data=3;
DO_8(slot, data);
// The I-9064 is inserted in slot 1 of LP-9x21 and can turn on channel 0 and 1.
```

Remark:

■ DO_16

Description:

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

Syntax:

[C]

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Examples:

```
int slot=1;
unsigned int data=3;
DO_16(slot, data);
 // The I-9057 is inserted in slot 1 of LP-9x21 and can turn on channel 0 and 1.
```

Remark:

■ DO_32

Description:

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

Syntax:

[C]

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Examples:

```
int slot=1;
unsigned int data=3;
DO_32(slot, data);
// The I-9041 is inserted in slot 1 of LP-9x21 and can turn on channel 0 and 1.
```

Remark:

This function can be applied on module: I-9041.

■ DI_8

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

Input data

Examples:

```
int slot=1;
unsigned char data;
data=DI_8(slot);
// The I-9058 is inserted in slot 1 of LP-9x21 and has inputs in channel 0 and 1.
// Returned value: data=0xfC
```

Remark:

There are two kind of Input type:

Input Type	On State	Off State	Modules
1 (Dry contact)	LED On, Readback as **1**	LED Off, Readback as **0**	I-9058
2 (Wet contact)	LED On, Readback as **0**	LED Off, Readback as **1**	I-9048, I-9052

■ DI_16

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

Input data

Examples:

```
int slot=1;
unsigned int data;
data=DI_16(slot);
// The I-9053 is inserted in slot 1 of LP-9x21 and has inputs in channel 0 and 1.
// Returned value: data=0xfffC
```

Remark:

There are two kind of Input type:

Input Type	On State	Off State	Modules
1 (Dry contact)	LED On, Readback as **1**	LED Off, Readback as **0**	I-9046
2 (Wet contact)	LED On, Readback as **0**	LED Off, Readback as **1**	I-9051, I-9053

■ DI_32

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

Input data

Examples:

```
int slot=1;
unsigned long data;
data=DI_32(slot);
// The I-9040 is inserted in slot 1 of LP-9x21 and has inputs in channels 0 and 1.
// Returned value: data=0xfffffffC
```

Remark:

Input Type	On State	Off State	Modules
1 (Wet contact)	LED On, Readback as **0**	LED Off, Readback as **1**	I-9040W

■ DIO_DO_8

Description:

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

Syntax:

[C]

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Examples:

```
int slot=1;
unsigned char data=3;
DIO_DO_8(slot, data);
// The I-9054 is inserted in slot 1 of LP-9x21 and can turn on channels 0 and 1.
// It not only outputs a value, but also shows 16LEDs.
```

Remark:

■ DIO_DO_16

Description:

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

Syntax:

[C]

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Examples:

```
int slot=1;
unsigned int data=3;
DIO_DO_16(slot, data);
// The I-9042 is inserted in slot 1 of LP-9x21 and can turn on the channels 0 and 1.
// It not only outputs a value, but also shows 32LEDs.
```

Remark:

■ DIO_DI_8

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

Input data

Examples:

```
int slot=1;
unsigned char data;
data=DIO_DI_8(slot);
// The I-9054 is inserted in slot 1 of LP-9x21 and has inputs in channel 0 and 1.
// Returned value: data=0xfC
```

Remark:

■ DIO_DI_16

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

Input data

Examples:

```
int slot=1;
unsigned char data;
data=DIO_DI_16(slot);
// The I-9042 is inserted in slot 1 of LP-9x21 and has inputs in channel 0 and 1.
// Returned value: data=0xfffC
```

Remark:

This function can be applied in modules: I-9042.

■ DO_8_RB、DO_16_RB、DO_32_RB

DIO_DO_8_RB、DIO_DO_16_RB

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

all DO channel status

Examples:

```
int slot=1;
Open_Slot(slot);
printf("%u",DO_32_RB(slot));
Close_Slot(slot);
// The I-9041 module is inserted in slot 1 of LP-9x21 and return all DO channel status.
```

Remark:

■ DO_8_BW、DO_16_ BW、DO_32_ BW

DIO_DO_8_ BW、DIO_DO_16_ BW

Description:

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

Syntax:

[C]

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

Parameter:

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

bit : [Input] channel of module.

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

Return Value:

None

Examples:

```
int slot=1, bit=0, data=1;
Open_Slot(slot);
DO_32_BW(slot, bit, data);
Close_Slot(slot);
// The I-9041 module is inserted in slot 1 of LP-9x21 and just turn on channel 0 of
   I-9041W.
```

Remark:

■ DI_8_BW、DI_16_ BW、DI_32_ BW

Description:

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

Syntax:

[C]

```
int DI_8_BW (int slot, int bit)
int DI_16_BW (int slot, int bit)
int DI_32_BW (int slot, int bit)
```

Parameter:

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

bit : [Input] channel of module.

Return Value:

None

Examples:

```
int slot=1, bit=0;
Open_Slot(slot);
printf("DI channel %d = %d\n", bit, DI_32_BW(slot, bit));
Close_Slot(slot);
// The I-9040 module is inserted in slot 1 of LP-9x21 and return channel 0 status (0: ON；1: OFF).
```

Remark:

■ UDIO_WriteConfig_16

Description:

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

Syntax:

[C]

```
unsigned short UDIO_WriteConfig_16(int slot, unsigned short config)
```

Parameter:

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

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

Return Value:

None

Examples:

```
int slot=1;
unsigned short config=0xffff;
UDIO_WriteConfig_16(slot, config);
// The I-9050 is inserted in slot 1 of LP-9x21.
// WriteConfig: 0xffff (ch 0 to ch15 is DO mode)
```

Remark:

This function can be applied on modules: I-9050.

■ UDIO_ReadConfig_16

Description:

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

Syntax:

[C]

```
unsigned short UDIO_ReadConfig_16(int slot)
```

Parameter:

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

Return Value:

None

Examples:

```
int slot=1;
unsigned int ret;
unsigned short config=0x0000;
UDIO_WriteConfig_16(slot, config);
ret=UDIO_ReadConfig_16(slot);
printf("Read the I/O Type is : 0x%04lx \n\r",ret);
// The I-9050 is inserted in slot 1 of LP-9x21.
// WriteConfig: 0x0000 (ch 0 to ch15 is DI mode)
// Read the I/O Type is: 0x0000
```

Remark:

This function can be applied on modules: I-9050.

■ UDIO_DO16

Description:

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

Syntax:

[C]

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

Parameter:

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

config : [Input] output data.

Return Value:

None

Examples:

```
int slot=1;
unsigned int data;
unsigned short config =0x00ff;
UDIO_WriteConfig_16(slot, config);
scanf("%d:",&data);
UDIO_DO16(slot, data);
printf("DO(Ch0 to Ch7) of I-8050 in Slot %d = 0x%x\n\r",slot, data);
// The I-9050 is inserted in slot 1 of LP-9x21
// WriteConfig: 0x00ff (ch0 to ch7 is DO mode and ch8 to ch15 is DI mode)
// Input DO value: 255
// DO(Ch0 to Ch7) of I-8050 in Slot 1 = 0xff
```

Remark:

This function can be applied on modules: I-9050.

■ UDIO_DI16

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

None

Examples:

```
int slot=1;
unsigned int data;
unsigned short config =0xff00;
UDIO_WriteConfig_16(slot, config);
data=UDIO_DI16(slot);
printf("DI(Ch0 to Ch7) of I-8055 in Slot %d = 0x%x\n\r",slot, data);
scanf("%d:",&data);
UDIO_DO16(slot, data);
printf("DO(Ch8 to Ch15) of I-8050 in Slot %d = 0x%x\n\r",slot, data);
// The I-9050 is inserted in slot 1 of LP-9x21.
// WriteConfig: 0xff00 (ch0 to ch7 is DI mode and ch8 to ch15 is DO mode)
// DI(Ch0 to Ch7) of I-8050 in Slot 1 = 0xfbff
// Input DO value: 255
// DO(Ch8 to Ch15) of I-9050 in Slot 1 = 0xff
```

Remark:

This function can be applied on modules: I-9050.

6.2.2 For I-7000/I-9000/I-97000 modules via serial port

6.2.2.1 I-7000 series modules

■ DigitalOut

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=3;
```



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

Remark:

■ DigitalBitOut

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

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



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

Remark:

■ DigitalOutReadBack

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

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



```
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7050;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[6] = 0;
DigitalOutReadBack (wBuf, fBuf, szSend, szReceive);
DO=wBuf[5];
Close_Com(COM3);
```

Remark:

■ DigitalOut_7016

Description:

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

Syntax:

[C]

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

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7016
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
wBuf[5] :	[Input] 2-bit digital output data in decimal format
wBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7] :	[Input] 0 : Bit0, Bit1 output 1 : Bit2, Bit3 output
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-7000 series modules.
szReceive :	[Output] Result string receiving from I-7000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=3;
```



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

Remark:

■ DigitalIn

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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



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

Remark:

■ DigitalInLatch

Description:

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

Syntax:

[C]

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

Parameter:

wBuf: WORD Input/Output argument talbe
wBuf[0] : [Input] COM port number, from 1 to 255
wBuf[1] : [Input] Module address, form 0x00 to 0xFF
wBuf[2] : [Input] Module ID, 0x7041/44/50/52/53/55/58/60/63/65/66
wBuf[3] : [Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] : [Input] Timeout setting , normal=100 milliseconds
wBuf[5] : [Input] 0: low Latch mode ; 1:high Latch mode
wBuf[6] : [Input] 0 → no save to szSend &szReceive
1 → Save to szSend &szReceive

wBuf[7] : [Output] Latch value

fBuf : Not used.

szSend : [Input] Command string to be sent to I-7000 series modules.

szReceive : [Output] Result string receiving from I-7000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

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

Remark:

■ ClearDigitalInLatch

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

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



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

Remark:

■ DigitalInCounterRead

Description:

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

Syntax:

[C]

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

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0] :	[Input] COM port number, from 1 to 255
wBuf[1] :	[Input] Module address, form 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7041/44/50/51/52/53/55/58/60/63/65
wBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] :	[Input] Timeout setting , normal=100 milliseconds

wBuf[5] : [Input] The digital input Channel No.

wBuf[6] : [Input] 0 → no save to szSend &szReceive

1 → Save to szSend &szReceive

wBuf[7] : [Output] Counter value of the digital input channel No.

fBuf : Not used.

szSend : [Input] Command string to be sent to I-7000 series modules.

szReceive : [Output] Result string receiving from I-7000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

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

Remark:

■ ClearDigitalInCounter

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

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



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

Remark:

■ ReadEventCounter

Description:

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

Syntax:

[C]

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

Parameter:

wBuf: WORD Input/Output argument talbe
wBuf[0] : [Input] COM port number, from 1 to 255
wBuf[1] : [Input] Module address, form 0x00 to 0xFF
wBuf[2] : [Input] Module ID, 0x7011/12/14/16
wBuf[3] : [Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] : [Input] Timeout setting , normal=100 milliseconds
wBuf[5] : Not used

wBuf[6] : [Input] 0 → no save to szSend &szReceive
1 → Save to szSend &szReceive
wBuf[7] : [Output] The value of event counter
fBuf : Not used.
szSend : [Input] Command string to be sent to I-7000 series modules.
szReceive : [Output] Result string receiving from I-7000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

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

Remark:

■ ClearEventCounter

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

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



```
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7012;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[6] = 0;
ClearEventCounter (wBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

6.2.2.2 I-9000 series modules

■ DigitalOut_8K

Description:

This function is used to set the digital output value of digital output module for I-9000 series modules.

Syntax:

[C]

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

Parameter:

dwBuf: WORD Input/Output argument talbe
dwBuf[0] : [Input] COM port number, from 1 to 255
dwBuf[1] : [Input] Module address, form 0x00 to 0xFF
dwBuf[2] : [Input] Module ID, 0x9041/42/54/55/56/57/60/63/64/65/66/67/68/77
dwBuf[3] : [Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] : [Input] Timeout setting , normal=100 milliseconds
dwBuf[5] : [Input] 16-bit digital output data
dwBuf[6] : [Input] 0 → no save to szSend &szReceive
1 → Save to szSend &szReceive
dwBuf[7] : [Input] Slot number; the I/O module installed in I-9000 main unit.
fBuf : Not used.
szSend : [Input] Command string to be sent to I-9000 series modules.
szReceive : [Output] Result string receiving from I-9000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
```



```
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9041;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 10;                    // digital output
dwBuf[6] = 0;
dwBuf[7] = m_slot;
DigitalOut_8K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalBitOut_8K

Description:

This function is used to set the digital value of the digital output channel No. of I-9000 series modules. Theoutput value is “0” or “1”.

Syntax:

[C]

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

Parameter:

dwBuf: DWORD Input/Output argument talbe
dwBuf[0] : [Input] COM port number, from 1 to 255
dwBuf[1] : [Input] Module address, form 0x00 to 0xFF
dwBuf[2] : [Input] Module ID, 0x9041/42/54/55/56/57/60/63/64/65/66/67/68/77
dwBuf[3] : [Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] : [Input] Timeout setting , normal=100 milliseconds
dwBuf[5] : [Input] 16-bit digital output data
dwBuf[6] : [Input] 0 → no save to szSend &szReceive

1 → Save to szSend &szReceive

dwBuf[7] : [Input] Slot number; the I/O module installed in I-9000 main unit.

dwBuf[8] : [Input] The output channel No.

fBuf : Not used.

szSend : [Input] Command string to be sent to I-9000 series modules.

szReceive : [Output] Result string receiving from I-9000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9041;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 10;                    // digital output
dwBuf[6] = 0;
dwBuf[7] = m_slot;
dwBuf[8] = 3;
DigitalBitOut_8K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalIn_8K

Description:

This function is used to obtain the digital input value from I-9000 series modules.

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x9040/42/51/52/54/55/58/63/77
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Output] 16-bit digital output data
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-9000 main unit.
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-9000 series modules.
szReceive :	[Output] Result string receiving from I-9000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

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



```
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9040;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 10;                    // digital output
dwBuf[6] = 0;
dwBuf[7] = m_slot;
DigitalIn_8K(dwBuf, fBuf, szSend, szReceive);
DI=dwBuf[5];
Close_Com(COM3);
```

Remark:

■ DigitalInCounterRead_8K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf: DWORD Input/Output argument talbe
dwBuf[0] : [Input] COM port number, from 1 to 255
dwBuf[1] : [Input] Module address, form 0x00 to 0xFF
dwBuf[2] : [Input] Module ID, 0x9040/51/52/53/54/55/58/63
dwBuf[3] : [Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] : [Input] Timeout setting , normal=100 milliseconds
dwBuf[5] : [Input] Channel No.

dwBuf[6] : [Input] 0 → no save to szSend &szReceive
1 → Save to szSend &szReceive
dwBuf[7] : [Input] Slot number; the I/O module installed in I-9000 main unit.
dwBuf[8] : [Output] DigitalIn counter value
fBuf : Not used.
szSend : [Input] Command string to be sent to I-9000 series modules.
szReceive : [Output] Result string receiving from I-9000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD DI_counter;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9040;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 0;
dwBuf[6] = 0;
dwBuf[7] = m_slot;
DigitalInCounterRead_8K(dwBuf, fBuf, szSend, szReceive);
DI_counter=dwBuf[8];
Close_Com(COM3);
```

Remark:

■ ClearDigitalInCounter_8K

Description:

This function is used to clear the counter value of the digital input channel No. of I-9000 series modules.

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x9040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Input] Channel No.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-9000 main unit.
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-9000 series modules.
szReceive :	[Output] Result string receiving from I-9000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

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



```
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9040;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 0;
dwBuf[6] = 0;
dwBuf[7] = m_slot;
ClearDigitalInCounter_8K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalInLatch_8K

Description:

This function is used to obtain the digital input latch value of the high or low latch mode of I-9000 series modules.

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x9040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds

dwBuf[5] : [Input] 0→ select to latch low
1→ select to latch high
dwBuf[6] : [Input] 0 → no save to szSend &szReceive
1 → Save to szSend &szReceive
dwBuf[7] : [Input] Slot number; the I/O module installed in I-9000 main unit.
dwBuf[8] : [Output] Latched data
fBuf : Not used.
szSend : [Input] Command string to be sent to I-9000 series modules.
szReceive : [Output] Result string receiving from I-9000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD DI_latch;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9040;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 0;
dwBuf[6] = 0;
dwBuf[7] = m_slot;
DigitalInLatch_8K(dwBuf, fBuf, szSend, szReceive);
DI_latch=dwBuf[8];
Close_Com(COM3);
```

Remark:

■ ClearDigitalInLatch_8K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x9040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	Not used.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-9000 main unit.
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-9000 series modules.
szReceive :	[Output] Result string receiving from I-9000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
```



```
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9040;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 0;
dwBuf[6] = 0;
dwBuf[7] = m_slot;
ClearDigitalInLatch_8K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

6.2.2.3 I-97000 series modules

■ DigitalOut_97K

Description:

This function is used to set the digital output value of the digital output module for I-97000 series modules.

Syntax:

[C]

```
WORD DigitalOut_97K(DWORD dwBuf[], float fBuf[], char szSend[],char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x97041/54/55/57/58/60/63/64/66/68
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Input]16-bit digital output data.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-97000 series modules.
szReceive :	[Output] Result string receiving from I-97000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

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



```
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x87054;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 3;
dwBuf[6] = 0;
DigitalOut_97K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalOutReadBack_97K

Description:

This function is used to read back the digital output value of the digital output module for I-97000 series modules.

Syntax:

[C]

```
WORD DigitalOutReadBack_97K(DWORD dwBuf[], float fBuf[], char szSend[],
                            char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x97041/54/55/57/58/60/63/64/66/68
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds

dwBuf[5] : [Output]16-bit digital output data.
dwBuf[6] : [Input] 0 → no save to szSend &szReceive
1 → Save to szSend &szReceive
fBuf : Not used.
szSend : [Input] Command string to be sent to I-97000 series modules.
szReceive : [Output] Result string receiving from I-97000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD DO;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97054;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[6] = 0;
DigitalOutReadBack_97K (dwBuf, fBuf, szSend, szReceive);
DO=dwBuf[5] ;
Close_Com(COM3);
```

Remark:

■ DigitalBitOut_97K

Description:

This function is used to set the digital output value of the specific digital output channel No. of the digital output module for I-97000 series modules. The output value is only for “0” or “1”.

Syntax:

[C]

```
WORD DigitalBitOut_97K(DWORD dwBuf[], float fBuf[], char szSend[],
                       char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x97041/54/55/57/58/60/63/64/66/68
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Input]1-bit digital output data.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] The digital output channel No.
dwBuf[8] :	[Input] Data to output(0 or 1)
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-97000 series modules.
szReceive :	[Output] Result string receiving from I-97000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD dwBuf[12];
DWORD m_port=3;
```



```
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97054;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 0;
dwBuf[7] = 1;
dwBuf[8] = 1;
DigitalBitOut_97K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalIn_97K

Description:

This function is used to obtain the digital input value from I-97000 series modules.

Syntax:

[C]

```
WORD DigitalIn_97K(DWORD dwBuf[], float fBuf[],char szSend[], char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x97040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Output]16-bit digitalintput data.

dwBuf[6] : [Input] 0 → no save to szSend &szReceive
1 → Save to szSend &szReceive
fBuf : Not used.
szSend : [Input] Command string to be sent to I-97000 series modules.
szReceive : [Output] Result string receiving from I-97000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD DI;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97054;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[6] = 0;
DigitalIn_97K(dwBuf, fBuf, szSend, szReceive);
DI=dwBuf[5];
Close_Com(COM3);
```

Remark:

■ DigitalInLatch_97K

Description:

This function is used to obtain the digital input latch value of the high or low latch mode of I-97000 series modules.

Syntax:

[C]

WORD DigitalInLatch_97K(DWORD dwBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x97040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Input] 0:low latch mode, 1:high latch mode
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Output] Latch value
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-97000 series modules.
szReceive :	[Output] Result string receiving from I-97000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD DI_latch;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
```



```
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97051;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 0;
DigitalInLatch_97K(dwBuf, fBuf, szSend, szReceive);
DI_latch=dwBuf[7];
Close_Com(COM3);
```

Remark:

■ ClearDigitalInLatch_97K

Description:

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

Syntax:

[C]

```
WORD ClearDigitalInLatch_97K(DWORD dwBuf[], float fBuf[], char szSend[],
                             char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x97040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable

dwBuf[4] : [Input] Timeout setting , normal=100 milliseconds
dwBuf[5] : Not used
dwBuf[6] : [Input] 0 → no save to szSend &szReceive
1 → Save to szSend &szReceive
fBuf : Not used.
szSend : [Input] Command string to be sent to I-97000 series modules.
szReceive : [Output] Result string receiving from I-97000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97051;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[6] = 0;
ClearDigitalInLatch_97K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ DigitalInCounterRead_97K

Description:

This function is used to obtain the counter value of the digital input channel No. of I-97000 series modules.

Syntax:

[C]

WORD DigitalInCounterRead_97K(DWORD dwBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x97040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Input] The digital input channel No.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Output] Counter value of the digital input channel No.
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-97000 series modules.
szReceive :	[Output] Result string receiving from I-97000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD DI_counter;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
```



```
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97051;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 0;
DigitalInCounterRead_97K (dwBuf, fBuf, szSend, szReceive);
DI_counter=dwBuf[7];
Close_Com(COM3);
```

Remark:

■ ClearDigitalInCounter_97K

Description:

This function is used to clear the counter value of the digital input channel No. of I-97000 series modules.

Syntax:

[C]

```
WORD ClearDigitalInCounter_97K(DWORD dwBuf[], float fBuf[], char szSend[],
                               char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x97040/51/52/53/54/55/58/63
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable

dwBuf[4] : [Input] Timeout setting , normal=100 milliseconds
dwBuf[5] : [Input] The digital input channel No.
dwBuf[6] : [Input] 0 → no save to szSend &szReceive
1 → Save to szSend &szReceive
fBuf : Not used.
szSend : [Input] Command string to be sent to I-97000 series modules.
szReceive : [Output] Result string receiving from I-97000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_slot=1;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97051;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 0;
ClearDigitalInCounter_97K (dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

6.3 Analog Input Functions

6.3.1 For I-9000 modules via parallel port

■ i9017_GetFirmwareVersion

Description:

This function is used to get the lattice version of I-9017 at specific slot.

Syntax:

[C]

```
short i9017_GetFirmwareVersion(int slot , short* version)
```

Parameter:

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

*version : [Output] version

Return Value:

> 0 : Version no.

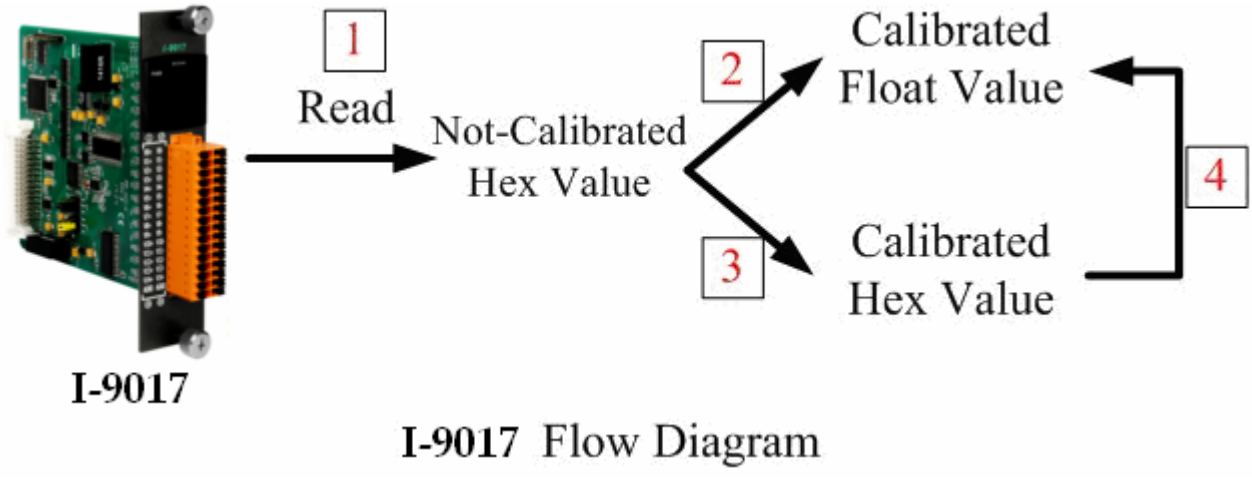
<=0 :error

Example:

```
int slot=1;
short version=0;
i9017_ GetFirmwareVersion(slot, &version);
printf("I-9017 at Slot%d, firmware version= %d",slot, &version);
// The I-9017 card is inserted in slot 1 of LP-9x21 and initializes the module.
```

Remark:

This function can be applied on module: I-9017.



I-9017 Flow Diagram

Fig.6-2

■ i9017_Init

Description:

This function is used to initialize the I-9017 modules (Analog input module) into the specified slot. Users must execute this function before trying to use other functions within the I-9017 modules.

Syntax:

[C]

```
int i9017_Init(int slot)
```

Parameter:

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

Return Value:

The version of library

Example:

```
int slot=1,ver;
ver=i9017_Init(slot);
// The I-9017 is inserted in slot 1 of LP-9x21 and initializes the module.
```

Remark:

This function can be applied on module: I-9017.

■ i9017_SetLed

Description:

Turns the I-9017 modules LED's on/off. They can be used to act as an alarm.

Syntax:

[C]

```
void i9017_SetLed(int slot, unsigned short iLedValue)
```

Parameter:

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

iLedValue: [Input] range from 0 to 0xffff

Return Value:

None

Example:

```
int slot=1;
unsigned short iLedValue led=0x0001;
i9017_Init(int slot);
i9017_SetLed (slot, iLedValue);
// There will be a LED light on channel 0 of the I-9017 card which is inserted in slot 1 on the LP-9x21.
```

Remark:

This function can be applied on module: I-9017.

■ i9017_GetSingleEndJumper

Description:

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

Syntax:

[C]

```
short i9017_GetSingleEndJumper (int slot)
```

Parameter:

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

Return Value:

1: Single-End mode

0: Differential mode.

Example:

```
int slot=1;
i9017_Init(int slot);
printf("mode=%d", i9017_GetSingleEndJumper(slot));
```

Remark:

This function can be applied on module: I-9017.

■ I9017_Set_ChannelGainMode

Description:

This function is used to configure the range and mode of the analog input channel for the I-9017 modules in the specified slot before using the ADC (analog to digital converter).

Syntax:

[C]

```
void i9017_Set_ChannelGainMode (int slot, int channel, int gain, int mode)
```

Parameter:

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

ch : [Input] Single-end mode→ Range 0 to 15

Differentail mode→ Range 0 to 7

gain : [Input] input range:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

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

Return Value:

None

Example:

```
int slot=1,ch=0,gain=0;
i9017_Init(slot);
i9017_ Set_ChannelGainMode (slot, ch, gain,0);
// The I-9017 card is inserted in slot 1 of LP-9x21, and the range of the data // value from channel 0 for I-9017 will be -10 to +10 V.
```

Remark:

This function can be applied on module: I-9017.

■ i9017_ ReadAI

Description:

This function is used to read a floating point input (calibrated) from one specified channel.

Syntax:

[C]

```
short i9017_ ReadAI(int slot, int iChannel, int iGain, float* fValue)
```

Parameter:

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

ch : [Input] Single-end mode→ Range 0 to 15

Differentail mode→ Range 0 to 7

gain : [Input] input range:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

*fValue : [Output] the floating-point data

Return Value:

0 = No Error

Example:

```
int slot=1,ch=0,gain=0;    // "+/-10V"
float fVal=0.0;
for(ch=0;ch<8;ch++)
{
    i9017_ReadAI( slot, ch, gain, & fVal);
    printf("\n[%02d]= [ %05.4f ]",ch, gain, fVal);
}
```


■ i9017_ReadAIHex

Description:

This function is used to read a hexadecimal input (calibrated) from one specified channel.

Syntax:

[C]

```
short i9017_ReadAIHex (int slot, int iChannel,int iGain, short* iValue)
```

Parameter:

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

ch : [Input] Single-end mode→ Range 0 to 15

Differentail mode→ Range 0 to 7

gain : [Input] input range:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

*iValue : [Output] the floating-point data

Return Value:

0 : indicates success.

Not 0 : indicates failure.

Examples:

```
int slot=1, ch=0, gain=0;
short hValue= 0.0;
for(ch=0;ch<8;ch++)
{
    i9017_ReadAIHex(slot, ch, gain, & hVal);
    printf("\n[%02d]= [ %04X ]",ch, gain, hVal);
}
```

Remark:

■ i9017_ ReadGainOffset_Info

Description:

This function is used to obtain the gain and offset values for each input type.

Syntax:

[C]

short i9017_ReadGainOffset_Info(int slot,int Gain,unsigned short* GainValue, short* OffsetValue)

Parameter:

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

gain : [Input] input range:

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

*GainValue : [Output] gain value for the input range

*offsetValue :[Output] offset value for the input range

Return Value:

0 : indicates success.

Not 0 : indicates failure.

Examples:

```
unsigned short gVal=0;
short oVal=0;
i9017_ReadGainOffset(slot, gain, &gVal, &oVal);
printf("\nThe Gain and Offset for Calibration is Gain=%u; Offset=%d", gVal, oVal);
```

Remark:

This function can be applied on module: I-9017.

6.3.2 For I-7000/I-9000/I-97000 modules via serial port

6.3.2.1 I-7000 series modules

■ AnalogIn

Description:

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

Syntax:

[C]

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

Parameter:

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

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

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



```
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7016;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[5] = 0;
wBuf[6] = 1;
AnalogIn (wBuf, fBuf, szSend, szReceive);     // szSend="#02" , szReceive=">+001.9"
AI = fBuf[0];                                 // AI = 1.9
Close_Com(COM3);
```

Remark:

■ AnalogInHex

Description:

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

Syntax:

[C]

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

Parameter:

wBuf: WORD Input/Output argument talbe

wBuf[0] : [Input] COM port number, from 1 to 255

wBuf[1] : [Input] Module address, form 0x00 to 0xFF

wBuf[2] : [Input] Module ID, 0x7005/11/12/13/14/15/16/17/18/19/33

wBuf[3] : [Input] 0= Checksum disable; 1= Checksum enable

wBuf[4] : [Input] Timeout setting , normal=100 milliseconds

wBuf[5] : [Input] Channel number for multi-channel

wBuf[6] : [Input] 0 → no save to szSend & szReceive

1 → Save to szSend & szReceive

wBuf[7] : [Ouput] The analog input value in “Hexadecimal “ format

fBuf : Not used.

szSend : [Input] Command string to be sent to I-7000 series modules.

szReceive : [Output] Result string receiving from I-7000 series modules .

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Example:

```
float AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=3;
```



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

Remark:

■ AnalogInFsr

Description:

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

Syntax:

[C]

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

Parameter:

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

wBuf[6] : [Input] 0 → no save to szSend & szReceive
1 → Save to szSend &szReceive
fBuf : Float Input/Output argument table.
fBuf[0] : [Output] Analog input value return
szSend : [Input] Command string to be sent to I-7000 series modules.
szReceive : [Output] Result string receiving from I-7000 series modules .
Note : Users have to use DCON utility to set up the analog input configuration of the module in hex format.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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

Remark:

■ AnalogInAll

Description:

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

Syntax:

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

Parameter:

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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

Remark:

■ ThermocoupleOpen_7011

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

```
WORD state;
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
```



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

Remark:

■ SetLedDisplay

Description:

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

Syntax:

[C]

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

Parameter:

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

wBuf[5] : [Input] Set display channel

wBuf[6] : [Input] 0 → no save to szSend & szReceive

1 → Save to szSend & szReceive

fBuf : Not used.

szSend : [Input] Command string to be sent to I-7000 series modules.

szReceive : [Output] Result string receiving from I-7000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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

Remark:

■ GetLedDisplay

Description:

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

Syntax:

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

Parameter:

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

Return Value:

0 is for Success
Not 0 is for Failure

Examples:

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



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

Remark:

6.3.2.2 I-9000 series modules

■ AnalogIn_8K

Description:

This function is used to obtain input value form I-9000 analog input series modules.

Syntax:

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

Parameter:

dwBuf: DWORD Input/Output argument talbe
dwBuf[0] : [Input] COM port number, from 1 to 255
dwBuf[1] : [Input] Module address, form 0x00 to 0xFF
dwBuf[2] : [Input] Module ID, 0x9017
dwBuf[3] : [Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] : [Input] Timeout setting , normal=100 milliseconds
dwBuf[5] : [Input] Channel number of analog input module
dwBuf[6] : [Input] 0 → no save to szSend & szReceive
1 → Save to szSend & szReceive
dwBuf[7] : [Input] Slot number.
fBuf : Float Input/Ouput argument table.
fBuf[0] : [Output] Analog input value
szSend : [Input] Command string to be sent to I-9000 series modules.
szReceive : [Output] Result string receiving from I-9000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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



```
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
AnalogIn_8K (dwBuf, fBuf, szSend, szReceive);
AI = fBuf[0];
Close_Com(COM3);
```

Remark:

■ AnalogInHex_8K

Description:

This function is used to obtain input value in “Hexadecimal” form I-9000 analog input series modules.

Syntax:

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

Parameter:

dwBuf: DWORD Input/Output argument talbe
dwBuf[0] : [Input] COM port number, from 1 to 255
dwBuf[1] : [Input] Module address, form 0x00 to 0xFF
dwBuf[2] : [Input] Module ID, 0x9017
dwBuf[3] : [Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] : [Input] Timeout setting , normal=100 milliseconds

dwBuf[5] : [Input] Channel number of analog input module
dwBuf[6] : [Input] 0 → no save to szSend & szReceive
1 → Save to szSend &szReceive
dwBuf[7] : [Input] Slot number.
dwBuf[8] : [Output] The analog input value in Hex format.
fBuf : Not used.
szSend : [Input] Command string to be sent to I-9000 series modules.
szReceive : [Output] Result string receiving from I-9000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

```
DWORD AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
AnalogInHex_8K (dwBuf, fBuf, szSend, szReceive);
AI = dwBuf[8];
Close_Com(COM3);
```

Remark:

■ AnalogInFsr_8K

Description:

This function is used to obtain input value in “FSR” form I-9000 analog input series modules. The “FSR” means “Percent” format.

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x9017
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Input] Channel number of analog input module
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number.
fBuf :	Float input/Output argument table.
fBuf[0] :	[Output] The analog input value.
szSend :	[Input] Command string to be sent to I-9000 series modules.
szReceive :	[Output] Result string receiving from I-9000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

```
float AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
```



```
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
AnalogInFsr_8K (dwBuf, fBuf, szSend, szReceive);
AI = fBuf[0];
Close_Com(COM3);
```

Remark:

■ AnalogInAll_8K

Description:

This function is used to obtain input value of all channels form I-9000 analog input series modules.

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x9017
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number.
fBuf :	Float input/Output argument table.
fBuf[0] :	[Output] Analog input value of channel 0.
fBuf[1] :	[Output] Analog input value of channel 1.
fBuf[2] :	[Output] Analog input value of channel 2.
fBuf[3] :	[Output] Analog input value of channel 3.
fBuf[4] :	[Output] Analog input value of channel 4.
fBuf[5] :	[Output] Analog input value of channel 5.
fBuf[6] :	[Output] Analog input value of channel 6.
fBuf[7] :	[Output] Analog input value of channel 7.
szSend :	[Input] Command string to be sent to I-9000 series modules.
szReceive :	[Output] Result string receiving from I-9000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

```
float AI[12];
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[6] = 1;
dwBuf[7] = 1;
AnalogInAll_8K (dwBuf, fBuf, szSend, szReceive);
AI[0] = fBuf[0];
AI[1] = fBuf[1];
AI[2] = fBuf[2];
AI[3] = fBuf[3];
AI[4] = fBuf[4];
AI[5] = fBuf[5];
AI[6] = fBuf[6];
AI[7] = fBuf[7];
Close_Com(COM3);
```

Remark:

6.3.2.3 I-97000 series modules

■ AnalogIn_97K

Description:

This function is used to obtain input value form I-97000 series analog input modules.

Syntax:

[C]

WORD AnalogIn_97K(DWORD dwBuf[], float fBuf[],char szSend[],char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x97013/15/16/17/18/19
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Input] Channel number for multi-channel
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Output] The analog input value return
szSend :	[Input] Command string to be sent to I-97000 series modules.
szReceive :	[Output] Result string receiving from I-97000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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



```
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
AnalogIn_97K(dwBuf, fBuf, szSend, szReceive);
AI = fBuf[0];
Close_Com(COM3);
```

Remark:

■ AnalogInHex_97K

Description:

This function is used to obtain input value in “Hexadecimal” form I-97000 series analog input modules.

Syntax:

[C]

```
WORD AnalogInHex_97K(DWORD dwBuf[], float fBuf[],char szSend[],
                     char szReceive[])
```

Parameter:

dwBuf: DWORD Input/Output argument talbe
dwBuf[0] : [Input] COM port number, from 1 to 255
dwBuf[1] : [Input] Module address, form 0x00 to 0xFF
dwBuf[2] : [Input] Module ID, 0x97013/15/16/17/18/19
dwBuf[3] : [Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] : [Input] Timeout setting , normal=100 milliseconds
dwBuf[5] : [Input] Channel number for multi-channel

dwBuf[6] : [Input] 0 → no save to szSend & szReceive
1 → Save to szSend & szReceive
dwBuf[7] : [Output] The analog input value in “Hex” format.
fBuf : Not used.
szSend : [Input] Command string to be sent to I-97000 series modules.
szReceive : [Output] Result string receiving from I-97000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

```
DWORD AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
AnalogInHex_97K(dwBuf, fBuf, szSend, szReceive);
AI = dwBuf[8];
Close_Com(COM3);
```

Remark:

■ AnalogInFsr_97K

Description:

This function is used to obtain input value in “FSR” form I-97000 series analog input modules.

Syntax:

[C]

```
WORD AnalogInFsr_97K(DWORD dwBuf[], float fBuf[],char szSend[],
                     char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x97013/15/16/17/18/19
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Input] Channel number for multi-channel
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Output] The analog input value
szSend :	[Input] Command string to be sent to I-97000 series modules.
szReceive :	[Output] Result string receiving from I-97000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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



```
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
AnalogInHex_97K(dwBuf, fBuf, szSend, szReceive);
AI = fBuf[0];
Close_Com(COM3);
```

Remark:

■ AnalogInAll_97K

Description:

This function is used to obtain input value of all channels form I-97000 series analog input modules.

Syntax:

[C]

```
WORD AnalogInAll_97K(DWORD dwBuf[], float fBuf[],char szSend[],
                     char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x97013/15/16/17/18/19
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Output] Analog input value of channel 0
fBuf[1] :	[Output] Analog input value of channel 1
fBuf[2] :	[Output] Analog input value of channel 2
fBuf[3] :	[Output] Analog input value of channel 3
fBuf[4] :	[Output] Analog input value of channel 4
fBuf[5] :	[Output] Analog input value of channel 5
fBuf[6] :	[Output] Analog input value of channel 6
fBuf[7] :	[Output] Analog input value of channel 7
szSend :	[Input] Command string to be sent to I-97000 series modules.
szReceive :	[Output] Result string receiving from I-97000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

```
float AI[12];
DWORD AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97017;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[6] = 1;
AnalogInAll_97K(dwBuf, fBuf, szSend, szReceive);
AI[0] = fBuf[0];
AI[1] = fBuf[1];
AI[2] = fBuf[2];
AI[3] = fBuf[3];
AI[4] = fBuf[4];
AI[5] = fBuf[5];
AI[6] = fBuf[6];
AI[7] = fBuf[7];
Close_Com(COM3);
```

Remark:

6.4 Analog Output Functions

6.4.1 For I-9000 modules via parallel port

■ i9024_Initial

Description:

This function is used to initialize the I-9024 module in the specified slot. You must implement this function before you try to use the other I-9024 functions.

Syntax:

[C]

```
short i9024_Initial(int slot)
```

Parameter:

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

Return Value:

None

Examples:

```
int slot=1;
i9024_Initial(slot);
// The I-9024 is inserted in slot 1 of LP-9x21 and initializes the I-9024 module.
```

Remark:

This function can be applied on module: I-9024.

■ i9024_VoltageOut

Description:

This function is used to send the voltage float value to the I-9024 module with the specified channel and slot in the LP-9x21 system.

Syntax:

[C]

```
void i9024_VoltageOut(int slot, int ch, float data)
```

Parameter:

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

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

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

Return Value:

None

Examples:

```
int slot=1, ch=0;
float data=3.0f;
i9024_Initial(slot);
i9024_VoltageOut(slot, ch, data);
//The I-9024 module output the 3.0V voltage from the channel 0.
```

Remark:

This function can be applied on module: I-9024.

■ i9024_CurrentOut

Description:

This function is used to initialize the I-9024 module in the specified slot for current output. Users must call this function before trying to use the other I-9024 functions for current output.

Syntax:

[C]

```
void i9024_CurrentOut(int slot, int ch, float cdata)
```

Parameter:

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

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

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

Return Value:

None

Examples:

```
int slot=1, ch=0;
float cdata=10.0f;
i9024_Initial(slot);
i9024_CurrentOut(slot, ch, data);
// Output the 10.0mA current from the channel 0 of I-9024 module.
```

Remark:

This function can be applied on module: I-9024.

■ i9024_VoltageHexOut

Description:

This function is used to send the voltage value in the Hex format to the specified channel in the I-9024 module, which is inserted in the slot in the LP-9x21 system.

Syntax:

[C]

```
void i9024_VoltageHexOut(int slot, int ch, int hdata)
```

Parameter:

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

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

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

Return Value:

None

Examples:

```
int slot=1, ch=0; data=0x3000;
i9024_Initial(slot);
i9024_VoltageHexOut(slot, ch, data);
// The I-9024 module output the 5.0V voltage from the channel 0.
```

Remark:

This function can be applied on module: I-9024.

■ i9024_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-9024, which is plugged into the slot in the LP-9x21 system.

Syntax:

[C]

```
void i9024_CurrentHexOut(int slot, int ch, int hdata)
```

Parameter:

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

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

hdata : [Input] Output data with hexadecimal

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

Return Value:

None

Examples:

```
int slot=1, ch=0; data=0x2000;
i9024_Initial(slot);
i9024_CurrentHexOut(slot, ch, data);
// Output the 10.0mA current from the channel 0 of I-9024 module.
```

Remark:

This function can be applied on module: I-9024.

6.4.2 For I-7000/I-9000/I-97000 modules via serial port

6.4.2.1 I-7000 series modules

■ AnalogOut

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=3;
```



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

Remark:

■ AnalogOutReadBack

Description:

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

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

Syntax:

[C]

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

Parameter:

wBuf: WORD Input/Output argument talbe

wBuf[0] : [Input] COM port number, from 1 to 255

wBuf[1] : [Input] Module address, form 0x00 to 0xFF

wBuf[2] : [Input] Module ID, 0x7016/21/22/24

wBuf[3] : [Input] 0= Checksum disable; 1= Checksum enable

wBuf[4] : [Input] Timeout setting , normal=100 milliseconds

wBuf[5] : [Input] 0 : command $AA6 read back
1 : command $AA8 read back

Note 1) When the module is I-7016: Don't care.
2) When the module is I-7021/22, analog output of current path read back ($AA8)
3) When the module is I-7024, the updating value in a specific Slew rate ($AA8)
(For more information, please refer to I-7021/22/24 manual)

wBuf[6] : [Input] 0 → no save to szSend &szReceive
1 → Save to szSend &szReceive

wBuf[7] : [Input] The analog output channel No. (0 to 3) of module I-7024
No used for single analog output module

fBuf : Float Input/Ouput argument table.

fBuf[0] : [Output] Analog output read back value

szSend : [Input] Command string to be sent to I-7000 series modules.

szReceive : [Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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

Remark:

■ AnalogOutHex

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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



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

Remark:

■ AnalogOutFsr

Description:

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

Syntax:

[C]

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

Parameter:

wBuf: WORD Input/Output argument talbe
wBuf[0] : [Input] COM port number, from 1 to 255
wBuf[1] : [Input] Module address, form 0x00 to 0xFF
wBuf[2] : [Input] Module ID, 0x7021/21P/22
wBuf[3] : [Input] 0= Checksum disable; 1= Checksum enable
wBuf[4] : [Input] Timeout setting , normal=100 milliseconds
wBuf[5] : [Input] The analog output channel number

(No used for single analog output module)

wBuf[6] : [Input] 0 → no save to szSend &szReceive

1 → Save to szSend &szReceive

fBuf : Float Input/Output argument table.

FBuf[0] : [Input] Analog output value in % of Span data format.

szSend : [Input] Command string to be sent to I-7000 series modules.

szReceive : [Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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

Remark:

■ AnalogOutReadBackHex

Description:

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

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

Syntax:

[C]

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

Parameter:

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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

Remark:

■ AnalogOutReadBackFsr

Description:

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

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

Syntax:

[C]

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

Parameter:

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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

Remark:

6.4.2.2 I-9000 series modules

■ AnalogOut_9K

Description:

This function is used to obtain analog value of analog output module for I-9000 series modules.

Syntax:

[C]

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

Parameter:

dwBuf: DWORD Input/Output argument talbe
dwBuf[0] : [Input] COM port number, from 1 to 255
dwBuf[1] : [Input] Module address, form 0x00 to 0xFF
dwBuf[2] : [Input] Module ID, 0x9024
dwBuf[3] : [Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] : [Input] Timeout setting , normal=100 milliseconds
dwBuf[5] : [Input] The defined analog output channel No.
dwBuf[6] : [Input] 0 → no save to szSend &szReceive
1 → Save to szSend &szReceive
dwBuf[7] : [Input] Slot number
fBuf : Float Input/Ouput argument table.
fBuf[0] : [Input] Analog output value
szSend : [Input] Command string to be sent to I-9000 series modules.
szReceive : [Output] Result string receiving from I-9000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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



```
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
fBuf[0] = 2.55
AnalogOut_8K (dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ AnalogOutReadBack_9K

Description:

This function is used to read back the analog value of analog output module for I-9000 series modules.

Syntax:

[C]

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

Parameter:

dwBuf: DWORD Input/Output argument talbe
dwBuf[0] : [Input] COM port number, from 1 to 255
dwBuf[1] : [Input] Module address, form 0x00 to 0xFF
dwBuf[2] : [Input] Module ID, 0x9024
dwBuf[3] : [Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] : [Input] Timeout setting , normal=100 milliseconds
dwBuf[5] : [Input] The defined analog output channel No.

dwBuf[6] : [Input] 0 → no save to szSend &szReceive
1 → Save to szSend &szReceive
dwBuf[7] : [Input] Slot number
fBuf : Float Input/Ouput argument table.
fBuf[0] : [Input] Analog output value
szSend : [Input] Command string to be sent to I-9000 series modules.
szReceive : [Output] Result string receiving from I-9000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

```
float Valot;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
AnalogOutReadBack_8K (dwBuf, fBuf, szSend, szReceive);
Volt = fBuf[0];
Close_Com(COM3);
```

Remark:

■ ReadConfigurationStatus_9K

Description:

This function is used to read configuration status of analog output module for I-9000 series modules.

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x9024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number
dwBuf[8] :	[Output] Output range: 0x30, 0x31,0x32
dwBuf[9] :	[Output] Slew rate
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-9000 series modules.
szReceive :	[Output] Result string receiving from I-9000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD Status;
DWORD Rate;
DWORD dwBuf[12];
```



```
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
ReadConfigurationStatus_8K (dwBuf, fBuf, szSend, szReceive);
Status = dwBuf[8];
Rate = dwBuf[9];
Close_Com(COM3);
```

Remark:

■ SetStartUpValue_9K

Description:

This function is used to setting start-up value of analog output module for I-9000 series modules.

Syntax:

[C]

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

Parameter:

dwBuf: DWORD Input/Output argument talbe
dwBuf[0] : [Input] COM port number, from 1 to 255
dwBuf[1] : [Input] Module address, form 0x00 to 0xFF
dwBuf[2] : [Input] Module ID, 0x9024

dwBuf[3] : [Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] : [Input] Timeout setting , normal=100 milliseconds
dwBuf[5] : [Input] The defined analog output channel No.
dwBuf[6] : [Input] 0 → no save to szSend &szReceive
1 → Save to szSend &szReceive
dwBuf[7] : [Input] Slot number
fBuf : Not used.
szSend : [Input] Command string to be sent to I-9000 series modules.
szReceive : [Output] Result string receiving from I-9000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
SetStartUpValue_8K (dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ SetStartUpValue_9K

Description:

This function is used to read start-up value of analog output module for I-9000 series modules.

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x9024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number
fBuf :	Float input/output argument table.
fBuf[0] :	[Output] Start-Up value.
szSend :	[Input] Command string to be sent to I-9000 series modules.
szReceive :	[Output] Result string receiving from I-9000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

```
float StartUp;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
```



```
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
ReadStartUpValue_8K (dwBuf, fBuf, szSend, szReceive);
StartUp = fBuf[0];
Close_Com(COM3);
```

Remark:

■ AnalogOutReadBack_9K

Description:

This function is used to read back the analog value of analog output module for I-9000 series modules.

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x9024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
dwBuf[7] :	[Input] Slot number
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Input] Analog output value
szSend :	[Input] Command string to be sent to I-9000 series modules.
szReceive :	[Output] Result string receiving from I-9000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

```
float Volt;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
```



```
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x9024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
AnalogOutReadBack_8K(dwBuf, fBuf, szSend, szReceive);
Volt = fBuf[0];
Close_Com(COM3);
```

Remark:

6.4.2.3 I-97000 series modules

■ AnalogOut_97K

Description:

This function is used to output input value form I-97000 series analog input modules.

Syntax:

[C]

WORD AnalogOut_97K(DWORD dwBuf[], float fBuf[],char szSend[],char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x97024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Input] Channel number for multi-channel
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Output] The analog output value
szSend :	[Input] Command string to be sent to I-97000 series modules.
szReceive :	[Output] Result string receiving from I-97000 series modules .

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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



```
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
fBuf[0] = 2.55;            //+2.55V
AnalogOut_97K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ AnalogOutReadBack_97K

Description:

This function is used to read back the analog value of analog output module for I-97000 series modules. There are two types of read back functions, as described in the following:

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

Syntax:

[C]

WORD AnalogOutReadBack_97K(DWORD dwBuf[], float fBuf[],char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x97024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf :	Float Input/Ouput argument table.
fBuf[0] :	[Outut] Analog output read back value
szSend :	[Input] Command string to be sent to I-97000 series modules.
szReceive :	[Output] Result string receiving from I-97000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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



```
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
AnalogOutReadBack_97K (dwBuf, fBuf, szSend, szReceive);
Volt = fBuf[0];
Close_Com(COM3);
```

Remark:

■ ReadConfigurationStatus_97K

Description:

This function is used to read configuration status of analog output module for I-97000 series modules.

Syntax:

[C]

```
WORD ReadConfigurationStatus_97K(DWORD dwBuf[], float fBuf[],char szSend[],
                                 char szReceive[])
```

Parameter:

dwBuf: DWORD Input/Output argument talbe
dwBuf[0] : [Input] COM port number, from 1 to 255
dwBuf[1] : [Input] Module address, form 0x00 to 0xFF
dwBuf[2] : [Input] Module ID, 0x97024
dwBuf[3] : [Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] : [Input] Timeout setting , normal=100 milliseconds
dwBuf[5] : [Input] The defined analog output channel No.
dwBuf[6] : [Input] 0 → no save to szSend & szReceive

1 → Save to szSend & szReceive

dwBuf[7] : [Input] Slot number

dwBuf[8] : [Output] Output range: 0x30, 0x31,0x32

dwBuf[9] : [Output] Slew rate

fBuf : Not used.

szSend : [Input] Command string to be sent to I-97000 series modules.

szReceive : [Output] Result string receiving from I-97000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD Status;
DWORD Rate;
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
ReadConfigurationStatus_97K (dwBuf, fBuf, szSend, szReceive);
Status = dwBuf[8];
Rate = dwBuf[9];
Close_Com(COM3);
```

Remark:

■ SetStartUpValue_97K

Description:

This function is used to setting start-up value of analog output module for I-97000 series modules.

Syntax:

[C]

WORD SetStartUpValue_97K(DWORD dwBuf[], float fBuf[],char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x97024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7] :	[Input] Slot number
fBuf :	Not used.
szSend :	[Input] Command string to be sent to I-97000 series modules.
szReceive :	[Output] Result string receiving from I-97000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

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



```
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 1;
dwBuf[6] = 1;
dwBuf[7] = 1;
SetStartUpValue_97K (dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

■ ReadStartUpValue_97K

Description:

This function is used to setting start-up value of analog output module for I-97000 series modules.

Syntax:

[C]

WORD SetStartUpValue_97K(DWORD dwBuf[], float fBuf[],char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument talbe
dwBuf[0] :	[Input] COM port number, from 1 to 255
dwBuf[1] :	[Input] Module address, form 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x97024
dwBuf[3] :	[Input] 0= Checksum disable; 1= Checksum enable
dwBuf[4] :	[Input] Timeout setting , normal=100 milliseconds
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7] :	[Input] Slot number
fBuf :	Float input/output argument table.
fBuf[0] :	Start-Up value.
szSend :	[Input] Command string to be sent to I-97000 series modules.
szReceive :	[Output] Result string receiving from I-97000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

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

Examples:

```
Float StartUp;
float fBuf[12];
char szSend[80];
char szReceive[80];
DWORD dwBuf[12];
DWORD m_port=3;
DWORD m_address=1;
```



```
DWORD m_timeout=100;
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0] = m_port;
dwBuf[1] = m_address;
dwBuf[2] = 0x97024;
dwBuf[3] = m_checksum;
dwBuf[4] = m_timeout;
dwBuf[5] = 0;
dwBuf[6] = 1;
dwBuf[7] = 1;
ReadStartUpValue_97K(dwBuf, fBuf, szSend, szReceive);
StartUp = fBuf[0];
Close_Com(COM3);
```

Remark:

6.5 Error Code Explanation

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

7. Demos for LP-9x21 Modules With C Language

In this section, we will focus on examples for the description and application of the control functions on the I-7000/I-9000/I-97K series modules for use with the LP-9x21. After installing the LP-9x21 SDK, the demo programs provided below can be found in the "**c:/cygwin/LinCon8k/examples**" folder.

7.1 DIO Control Demo for I-7k Modules

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

The result of executing this demo program is that DO channels 0 to 7 on the I-7050 module will be set as the output channels, and DI channel 2 on the I-7050 module will be set as the input channel. The source code for the demo program is as follows:

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

char szSend[80], szReceive[80], ans;
WORD wBuf[12];
float fBuf[12];
/* ---------------------------------------------------------------- */
int main()
{
      int   wRetVal;
      // Check Open_Com3
      wRetVal = Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed!\n");
            return (-1);
      }
      //   *****   7050 DO && DI Parameter *******
      wBuf[0] = 3;                  // COM Port
      wBuf[1] = 0x02;               // Address
      wBuf[2] = 0x7050;             // ID
      wBuf[3] = 0;                  // CheckSum disabled
      wBuf[4] = 100;                // TimeOut , 100 milliseconds
```



```
    wBuf[5] = 0x0f;                 // Set 8 DO Channels to ON
    wBuf[6] = 0;                    // Debug string
    // 7050 DO Output
    wRetVal = DigitalOut(wBuf, fBuf, szSend, szReceive);
    if (wRetVal)
            printf("DigitalOut_7050 Error !, Error Code=%d\n", wRetVal);

    printf("The DO of 7050 : %u \n", wBuf[5]);

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

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

Follow the procedure below to achieve the desired results:

STEP 1: **Write i7kdio.c**

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

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

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

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

Open the LP-9x21 Build Environment by clicking the Start > Programs > ICPDAS > LP-9x21 SDK > LP-9x21 Build Environment to open **LP-9x21 SDK** window, and change the path to C:\cygwin\LinCon8k\examples\i7k. To compile the i7kdio.c file to an executable file, type **lcc i7kdio**. (refer to Fig. 7-1)

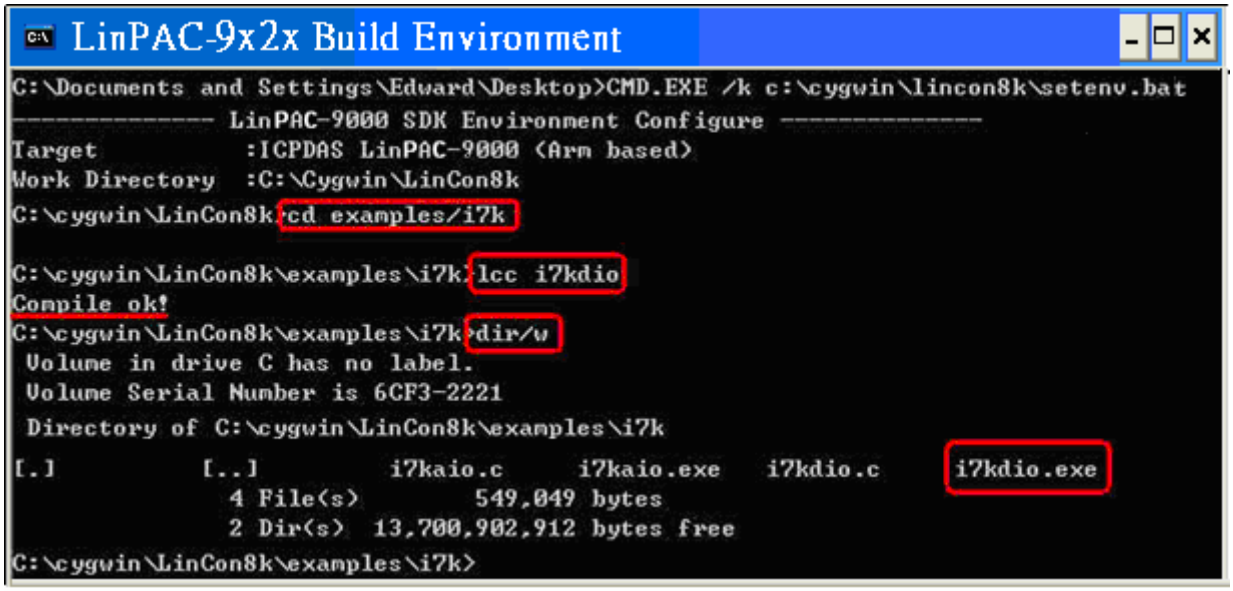


Fig. 7-1

Method Two – Using Compile Instructions

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

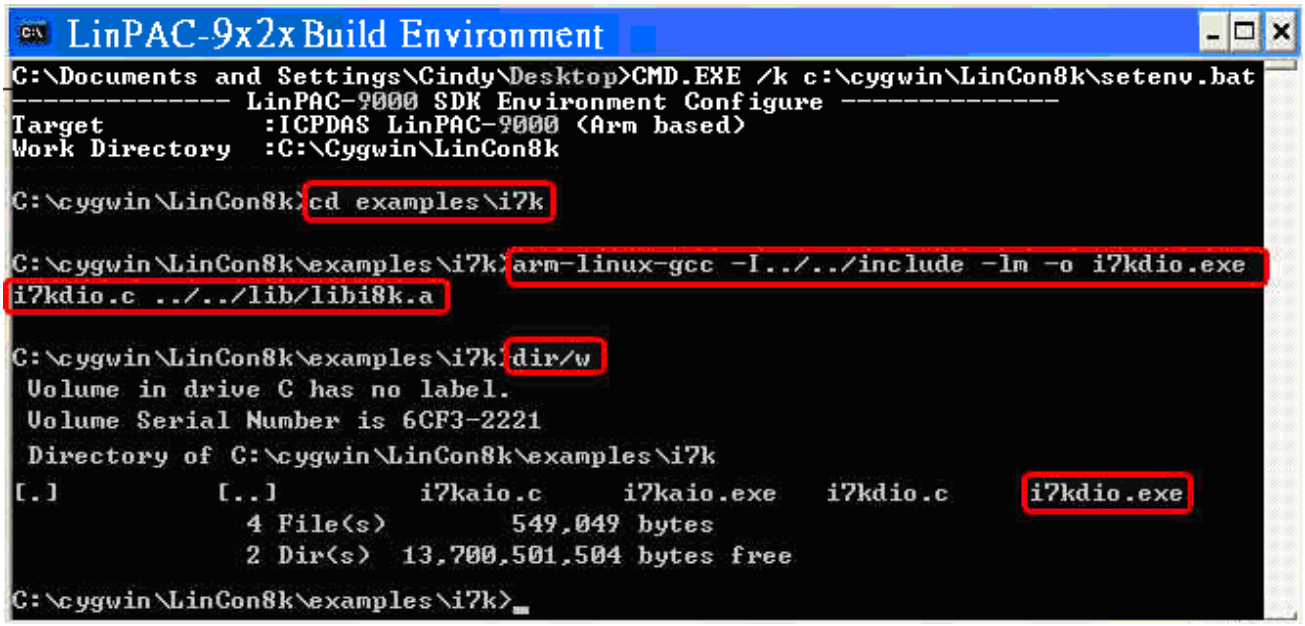


Fig. 7-2

STEP 3: **Transfer i7kdio.exe to the LP-9x21**

Two methods can be used to transfer the executable file to the LP-9x21, each of which is introduced here.

Method One – Using an FTP application

Step 1: Open a FTP application and create a new FTP connection. Enter the login details for the LP-9x21, including the Host name (or IP address), Username and Passeord. The default value for the **User_Name** is '**root**' and the **Password** is **'icpdas'**. Click the **"Connect"** button to connect to the ftp server on the LP-9x21. Refer to Fig.7-3 below for more details.

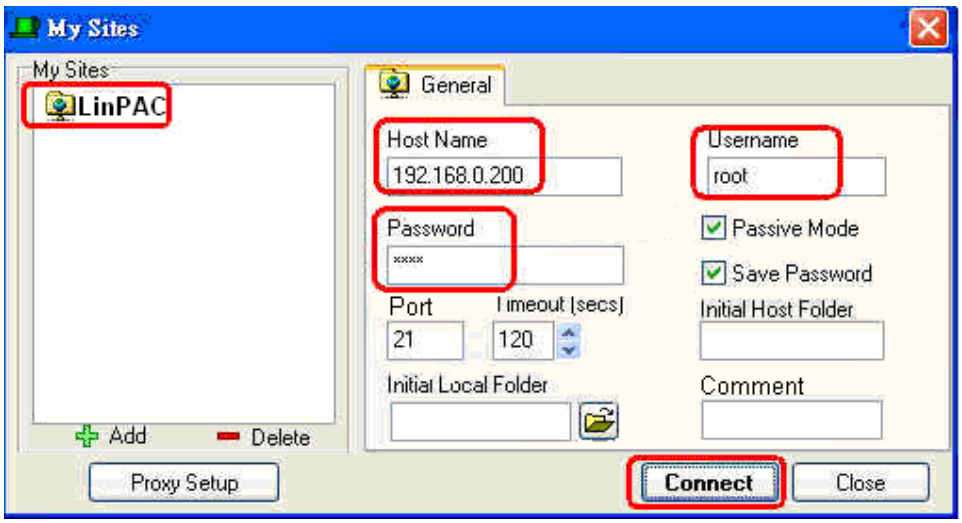


Fig.7-3

Step2: Upload the file **i7kdio.exe** file to the LP-9x21. (refer to Fig.7-4).

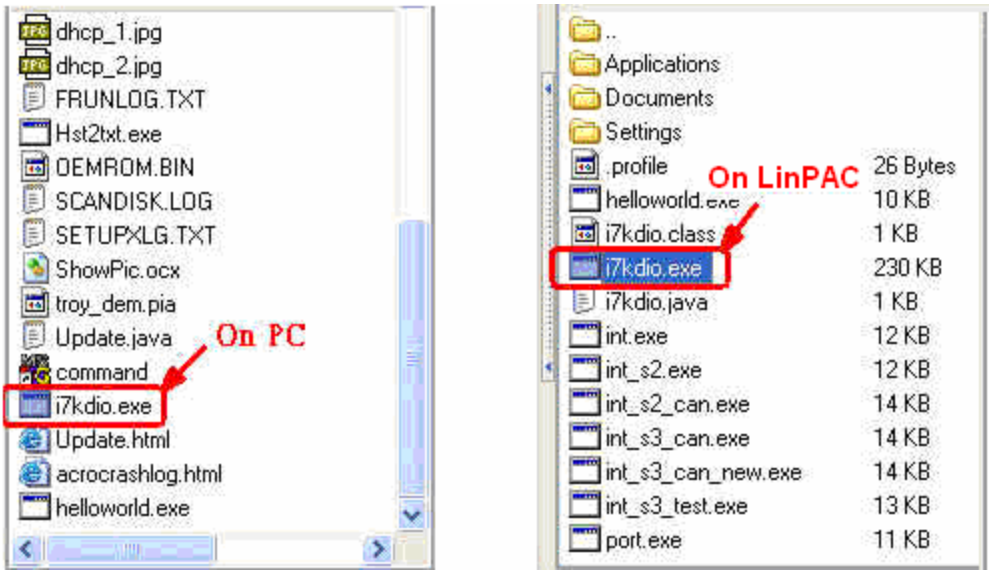


Fig.7-4

Step 3: Choose i7kdio.exe in the LP-9x21 and Click the right mouse button to select the "**Permissions**" option for the menu. Enter "777" in the Numeric textbox to set the file permissions to readable, writeable, and executable. Refer to Fig.7-5 and 7-6 below for more details.

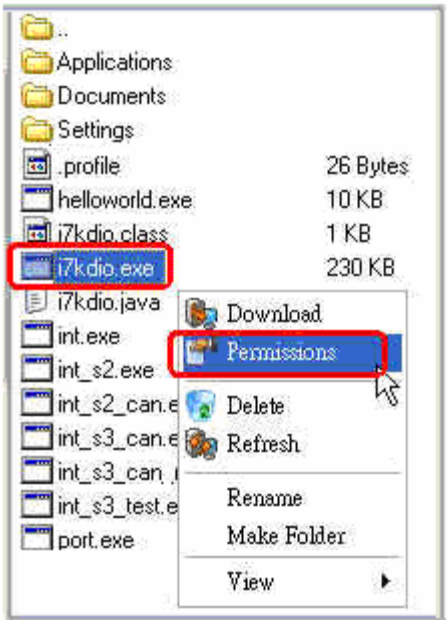


Fig.7-5

Fig.7-6

Method Two – Using a DOS Command Prompt

Open DOS Command Prompt and enter the IP Address of the server on the LP-9x21 in order to connect to the ftp server of the LP-9x21. Enter the **User Name** (the default value is root) and **Password** (the default value is icpdas) to login to the LP-9x21 ftp server.

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

At Command Prompt, type put c:/cygwin/lincon8k/examples/i7k/i7kdio.exe i7kdio.exe to transfer the i7kdio.exe file to the LP-9x21. Once the file has been transferred, the "Transfer complete" message will be displayed. Refer to Fig. 7-7 below for more details.

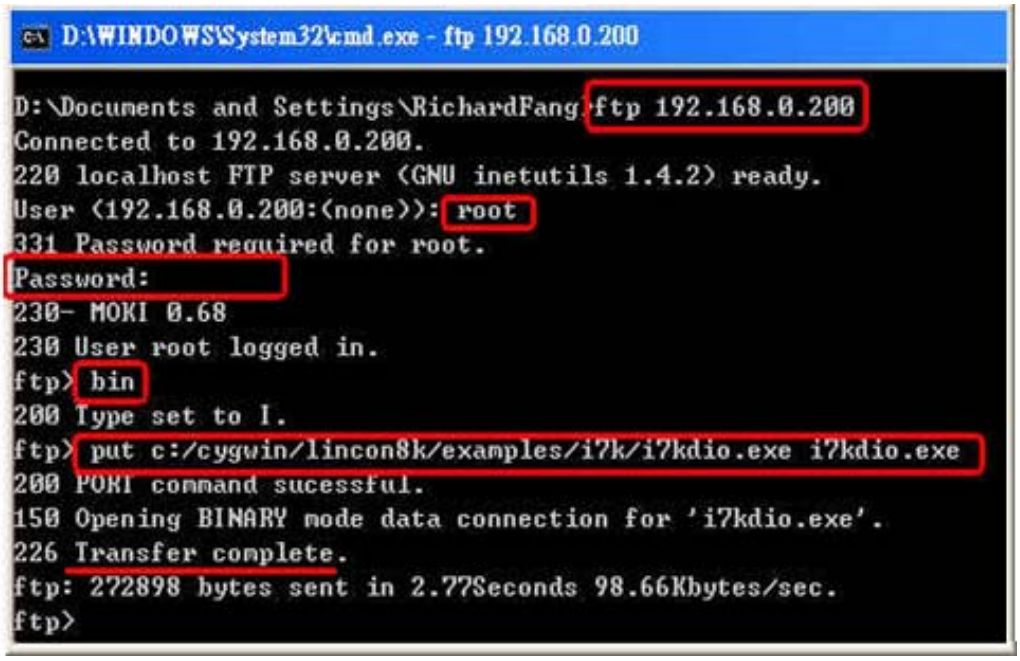


Fig. 7-7

STEP 4: **Use a free SSH software Pietty or Putty to the LP-9x21 to execute i7kdio.exe**

At the Command Prompt, type telnet IP Address of the LP-9x21 to establish a connection to the LP-9x21. Enter **User Name** (the default value is **root**) and **Password** (the default value is **icpdas**) to login to the LP-9x21. Refer to Fig. 7-8.

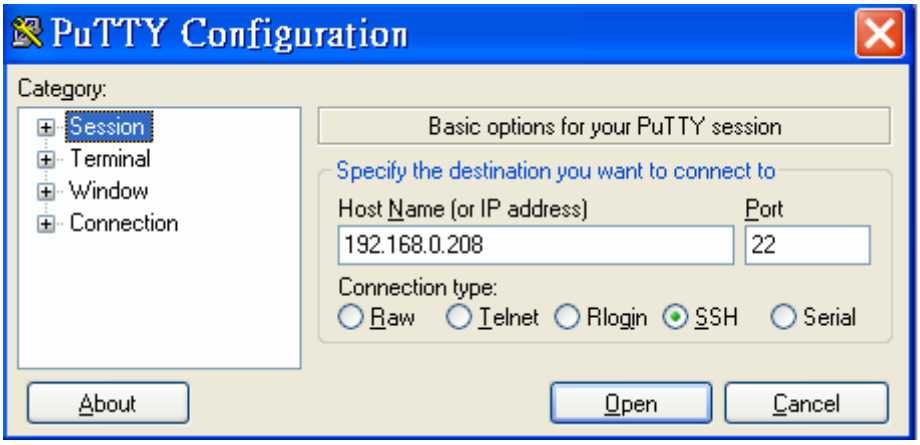


Fig. 7-8

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

Fig. 7-9

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

7.2 AIO Control Demo for I-7k Modules

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

The result of executing this demo program is that the AO channel on the I-7021 module will be set to output a voltage of 3.5V, and AI channel 2 on the I-7017 module will be set as the input channel. The source code for this demo program is as follows:

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"
char szSend[80], szReceive[80];
WORD wBuf[12];
float fBuf[12];
/* ---------------------------------------------------------------- */
int main()
{
      int i,j, wRetVal;
      DWORD temp;
```



```
    wRetVal = Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
    if (wRetVal > 0) {
        printf("open port failed!\n");
        return (-1);
    }

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

    wRetVal = AnalogOut(wBuf, fBuf, szSend, szReceive);
    if (wRetVal)                    // There was an error with the Analog Output on the I-7021
        printf("AO of 7021 Error !, Error Code=%d\n", wRetVal);
    else
        printf("AO of 7021 channel %d = %f \n",i,fBuf[0]);

    //--- Analog Input ----    ****     7017 -- AI    ****
    j = 1;
    wBuf[0] = 3;                    // COM Port
    wBuf[1] = 0x03;                 // Address
    wBuf[2] = 0x7017;               // ID
    wBuf[3] = 0;                    // CheckSum disabled
    wBuf[4] = 100;                  // TimeOut , 100 milliseconds
    wBuf[5] = j;                    // Channel of AI
    wBuf[6] = 0;                    // Debug string

    wRetVal = AnalogIn(wBuf, fBuf, szSend, szReceive);
    if (wRetVal)                    // There was an error with the Analog Input on the I-7017
        printf("AI of 7017 Error !, Error Code=%d\n", wRetVal);
    else
        printf("AI of 7017 channel %d = %f \n",j,fBuf[0]);

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

Fig. 7-10 illustrates the result of execution.

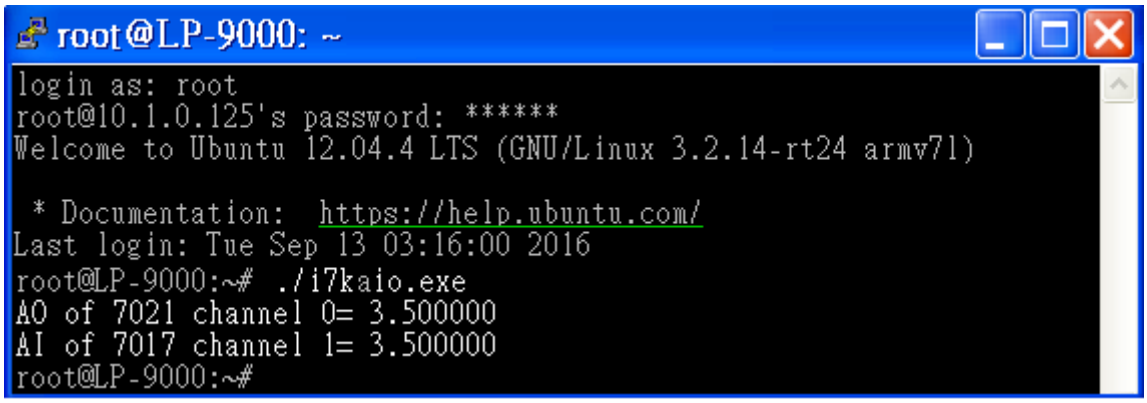


Fig. 7-10

7.3 DIO Control Demo for I-97K Modules

When using I-97K DIO modules to perform I/O control on the LP-9x21, the program will be slightly different, depending on the location of the I-97K modules. The code will need to be modified depending on the configuration of the I-97K modules.

If there are I-97KW DIO modules inserted **in the slots on the LP-9x21**, the “Open_Slot()” and “ChangeToSlot()” functions, must be called before other functions for the I-97KW modules and used, and the “Close_Slot()“ function also needs to be called at the end of the program.

The **i97Kdio.c** demo program will illustrates how to control the DI/DO function using an I-97054 module (8 DO channels and 8 DI channels). The module is in slot 3 on the LP-9x21. The address and baudrate in the LP-9x21 are 00 and 115200 respectively, they were fixed by library.

The result of this demo program is that DO channels 0 to 7 on the I-97054 module will be se as the output channels, and DI channel 1 on the I-97054 module will be set as the input channel. The source code for this demo program is as follows:

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

char szSend[80], szReceive[80];
DWORD dwBuf[12];
```



```
float fBuf[12];

int main()
{
    int i, wRetVal;
    DWORD temp;
    //Check Open_Slot
    wRetVal = Open_Slot(0);
    if (wRetVal > 0) {
        printf("open Slot failed. \n");
        return (-1);
    }

    //Check Open_Com1
    wRetVal = Open_Com(COM1, 115200, Data8Bit, NonParity, OneStopBit);
    if (wRetVal > 0) {
        printf("open port failed. \n");
        return (-1);
    }

    //Choose Slot3
     ChangeToSlot(3);

    //--- Digital Output ----    **(DigitalOut_97K()**)
    dwBuf[0] = 1;                   // COM Port
    dwBuf[1] = 00;                  // Address
    dwBuf[2] = 0x97054;             // ID
    dwBuf[3] = 0;                   // CheckSum disabled
    dwBuf[4] = 100;                 // TimeOut , 100 milliseconds
    dwBuf[5] = 0xff;                // Set digital output
    dwBuf[6] = 0;                   // Debug string
     wRetVal = DigitalOut_97K(dwBuf, fBuf, szSend, szReceive);     // DO Output
  printf("DO Value= %u", dwBuf[5]);

  //--- digital Input ----      **(DigitalIn_97K()**)
    dwBuf[0] = 1;                   // COM Port
    dwBuf[1] = 00;                  // Address
    dwBuf[2] = 0x97054;             // ID
    dwBuf[3] = 0;                   // CheckSum disabled
    dwBuf[4] = 100;                 // TimeOut , 100 milliseconds
    dwBuf[6] = 0;                   // Debug string
    getch();
    DigitalIn_97K(dwBuf, fBuf, szSend, szReceive);     // DI Input
   printf("DI= %u",dwBuf[5])

    //--- digital output ----     ** Close DO **
    dwBuf[0] = 1;                   // COM Port
    dwBuf[1] = 00;                  // Address
    dwBuf[2] = 0x97054;             // ID
    dwBuf[3] = 0;                   // CheckSum disabled
    dwBuf[4] = 100;                 // TimeOut , 100 milliseconds
    dwBuf[5] = 0x00;                // Digital output
    dwBuf[6] = 0;                   // Debug string
```



```
    getch();                                // Press any key to continue
    wRetVal = DigitalOut_97K(dwBuf, fBuf, szSend, szReceive);

    Close_Com(COM1);
    Close_SlotAll();
    return 0;
}
```

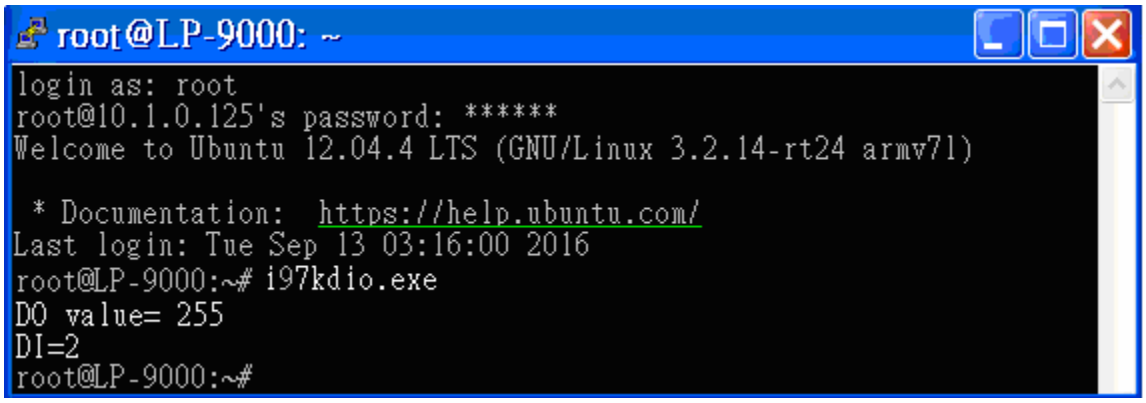


Fig. 7-11

7.4 AIO Control Demo for I-97K Modules

When using I-97K modules to perform I/O control on the LP-9x21, the program code will be slightly different depending on the location of the I-97K modules.The code will need to be modified depending on the configuration of the I-97K modules.

If there are I-97K AIO modules inserted **in the slots on the LP-9x21**, the “Open_Slot” and “ChangeToSlot” functions must be called before other functions for the I-97K modules are used, and the “Close_Slot()“ function also needs to be called at the end of the program.

The **i97Kaio.c** demo program illustrates how to control the AI/AO using an the **I-97022** module (2 AO channels) and an **I-97017** module (8 AI channels). The I-97022 and I-97017 modules are inserted into slots 2 and 3 of the LP-9x21 reseparately. The addresses and baudrate fo both modules in the LP-9x21 are 00 and 115200 bps reseparately, they were fixed by library.

The result of executing this demo program is that AO channel 0 on the I-97022 module will be set to output a voltage of 2.5V, and AI channel 1 on the I-97017 module will be set as the input channel. The source code for this demo program is as follows:


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

char szSend[80], szReceive[80];
DWORD wBuf[12];
DWORD    wBuf7[12];
float fBuf[12];
int main()
{
      int i,j, wRetVal;
      DWORD temp;

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

      //Check Open_Com1
      wRetVal = Open_Com(COM1, 115200, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed. \n");
            return (-1);
      }

      ChangeToSlot(2);
      //--- Analog output ----    ****     87022 -- AO    ****
      i=0;
      wBuf[0] = 1;                  // COM Port
      wBuf[1] = 0x00;               // Address
      wBuf[2] = 0x97022;            // ID
      wBuf[3] = 0;                  // CheckSum disable
      wBuf[4] = 100;                // TimeOut , 100 milliseconds
      wBuf[5] = i;                  // Channel Number of AO
      wBuf[6] = 0;                  // string debug
      fBuf[0] = 2.5;                 // AO Value
      wRetVal = AnalogOut_97K(wBuf, fBuf, szSend, szReceive);
      if (wRetVal)                  // There was an error with the Analog Output on the I-97022W
            printf("AO of 97022 Error, Error Code=%d\n", wRetVal);
      else
            printf("AO of 97022 channel %d = %f \n",i,fBuf[0]);

      ChangeToSlot(3);
      //--- Analog Input ----    ****     97017 -- AI    ****
      j=1;
      wBuf7[0] = 1;                   // COM Port
      wBuf7[1] = 0x00;                // Address
      wBuf7[2] = 0x97017;             // ID
      wBuf7[3] = 0;                   // CheckSum disabled
      wBuf7[4] = 100;                 // TimeOut , 100 milliseconds
      wBuf7[5] = j;                   //Channel Number of AI
```



```
    wBuf7[6] = 0;                   // Debug string
    wRetVal = AnalogIn_97K(wBuf7, fBuf, szSend, szReceive);
    if (wRetVal)                    // There was an error with the Analog Output on the I-97017
        printf("AI of 97017 Error, Error Code=%d\n", wRetVal);
    else
        printf("AI of 97017 channel %d = %f \n",j,fBuf[0]);

    Close_Com(COM1);
    Close_SlotAll();
    return 0;
}
```

Fig. 7-12. illustrates the result of execution.

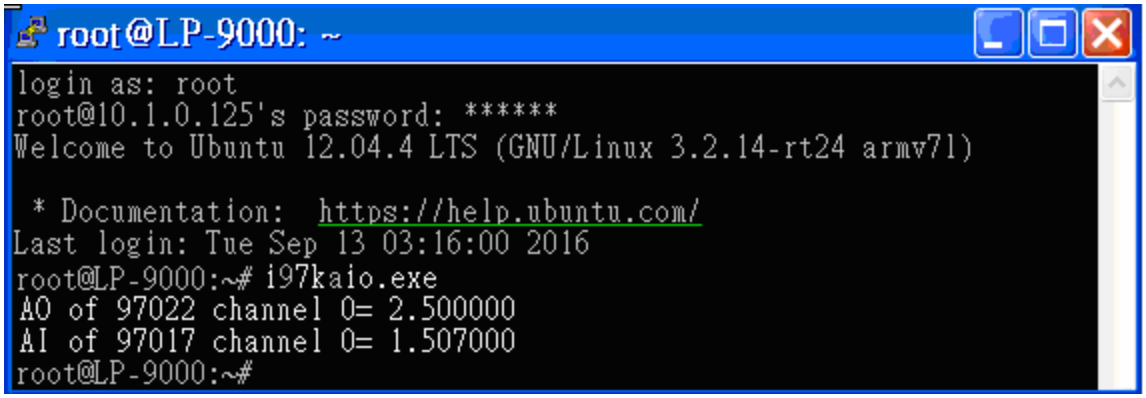


Fig. 7-12

7.5 DIO Control Demo for I-9K Modules

The **i9kdio.c** demo program illustrates how to control the DI/DO functions using an I-9055 modules (8 DO channels and 8 DI channels) that is inserted into slot 3 on the LP-9x21. The address and baudrate for the LP-9x21 are 00 and 115200 bps reseparately, and they were fixed by library.

The result of executing this demo program is that DO channels 0 to 7 on the I-9055 module to will be set as the output channels, and DI channel 0 on I-9055 module will be set as the input channel. The source code for this demo program is as follows:

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



```
char szSend[80], szReceive[80];
DWORD dwBuf[12];
float fBuf[12];
/* ---------------------------------------------------------------------- */
int main()
{
	int i,j, wRetVal;
	WORD DOval,temp;
	wRetVal = Open_Slot(3);
	if (wRetVal > 0) {
		printf("open Slot failed. \n");
		return (-1);
	}

	//I-9055_DO
	DO_8(3,255);
	printf("DO of I-9055 = 0x%x \n", 255);

	//I-9055_DI
	printf("DI of I-9055 = %x",DI_8(3));
	Close_Slot(3);

	return 0;
}
```

Fig. 7-13 illustrates the result of execution.

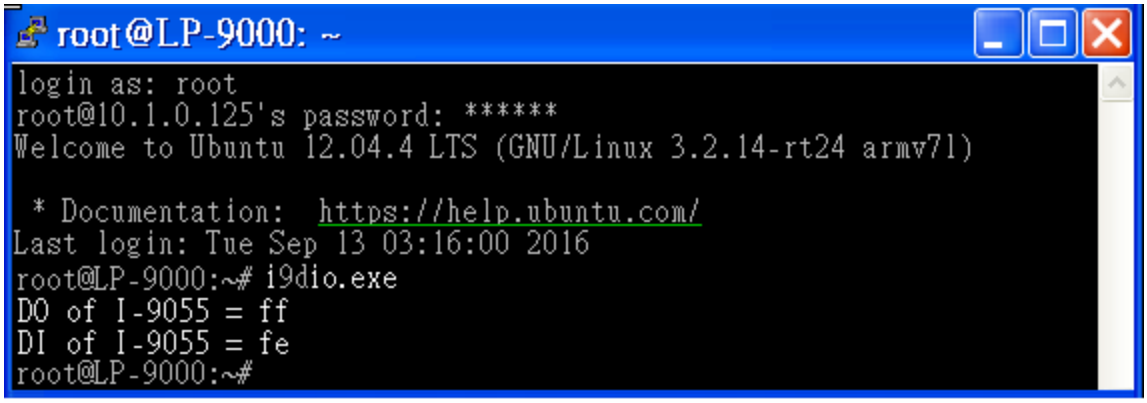


Fig. 7-13

7.6 AIO Control Demo for I-9K Modules

The **i9kaio.c** demo program illustrates how to control the AI/AO functions using the I-9024 (4 AO channels) and I-9017 (8 AI channels) modules, which are inserted in slot 1 and slot 2 on the LP-9x21 reseparately.

The address and baudrate in the LP-9x21 are 00 and 115200 bps reseparately, and they were fixed by library. The result of executing this demo is that AO voltage channel 0 on the I-9024 module to will be set to output 5.5V and AI channel 2 on the I-9017 module to will be set as the input channel. The source code for this demo program is as follows:

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

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

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

    //I-9024
    wRetVal = Open_Slot(1);
    if (wRetVal > 0) {
        printf("open Slot failed. \n");
        return (-1);
    }

    //I9024 Initial
    I9024_Initial(1);
    //I9024_AO Output
    I9024_VoltageOut(1,0,5.5);
    Close_Slot(1);

    //I-9017
```



```
    wRetVal = Open_Slot(2);
    if (wRetVal > 0) {
        printf("open Slot failed. \n");
        return (-1);
    }
    //I9017 Initial
    I9017_Init(2);
    //I9017 _Channel Setup
    I9017_SetChannelGainMode(2,2,0,0);

    // First Method：Get AI Value
    hAi = I9017_GetCurAdChannel_Hex(2);     //Get the uncalibrated AI Hex Value
    printf("9017_AI_not_Cal_Hex =%x\n",hAi);
    fAi = HEX_TO_FLOAT_Cal(hAi,2,0);        //Uncalibrated AI Hex Value and then
modify it to a calibrated AI Float Value
    printf("9017_AI_Cal_Float =%f\n\n",fAi);

    // Second Method：Get AI Value
    hAi = I9017_GetCurAdChannel_Hex_Cal(2);   //Get the Calibrated AI Hex Value
    printf("9017_AI_Cal_Hex =%x\n",hAi);
    fAi = CalHex_TO_FLOAT(hAi,0);
    //Modify the Calibrated AI Hex Value modify to a Calibrated AI Float Value
    printf("9017_AI_Cal_Float =%f\n\n",fAi);

    // Third Method：Get AI Value
    fAi = I9017_GetCurAdChannel_Float_Cal(2);   //Get the Calibrated AI Float Value
    printf("9017_AI_Cal_Float =%f\n\n\n",fAi);

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

Fig. 7-14 illustrates the result of execution.

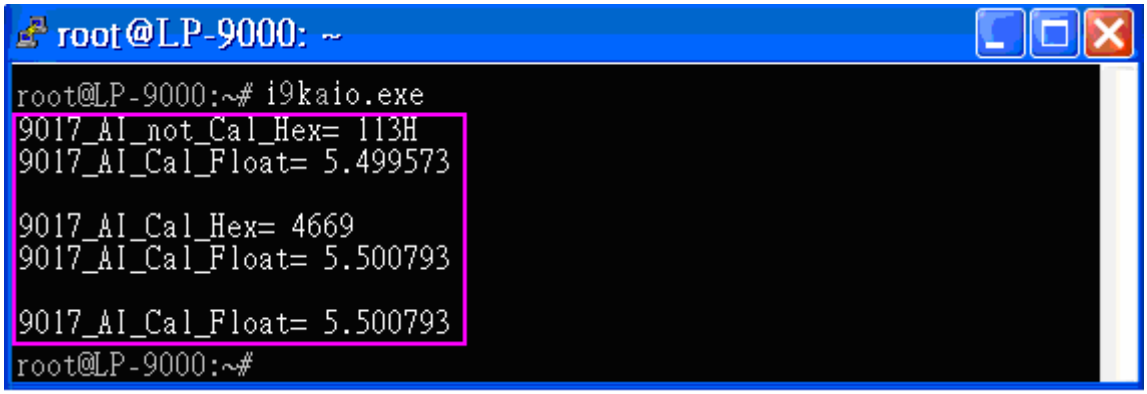


Fig. 7-14

7.7 Overview of the Module Control Demo Program

Fig. 7-15 provides a summary of the various communication functions that can be used depending on the for the different locations of the I-7000/I-9000/I-97000 modules when using the ICP DAS modules in conjunction with the LP-9x21, which can be helpful in understanding which communication functions should be used.

Note that the Open_slot()/Close_Slot() and sio_open()/sio_close() functions cannot be used for the same slot.

Module Location / Communication Functions	I-9K module In LP-9x21	I-97K module In LP-9x21	I-97K module In Expansion Unit	I-7K module
Open_Slot( )	✓	✓		
Open_Com( )		✓	✓	✓
Close_Slot( )	✓	✓		
Close_Com( )		✓	✓	✓
ChangeToSlot( )		✓		
sio_open( ) (I-911x and I-914x only)	✓			
sio_close( ) (I-911x and I-914x only)	✓			

Fig. 7-15

Fig. 7-16 provides an overview of the source files from the **libi8k.a** library that can be used depending on the different locations of I-7000/I-9000/I-97000 modules when using ICP DAS modules in conjunction with the LP-9x21, which can be helpful in understanding which source files should be calld.

Source File / Module Location	sio.c (serial module)	I7000.c	I8000.c	I87000.c	slot.c
I-7K		✓			
I-9K or I-97K in I-9000 Controller			✓		
I-97K in Expansion Unit				✓	
I-97K in LinPAC				✓	
I-9K in LinPAC	✓				✓

Fig. 7-16

8. Overview of the Serial Ports on the LP-9x21

The following is a description of the functionlity for the three serial ports contained in the LP-9x21 embedded controller, and are based on the RS-232 or RS-485 interfaces. Fig 8-1 illustrates the ports contained on the LP-9821. The information in this section is organized as follows:

- ttyO4─ Internal communication with the I-97KW modules in slots
- ttyO5 ─ RS-232/RS-485; Non-isolation; Console
- ttyS0 ─ RS-485 (D2+, D2-; self-tuner ASIC inside)
- ttyS1 ─ RS-232/RS-485
  (RXD, TXD, CTS, RTS and GND for RS-232, Data+ and Data- for RS-485)
- ttyS34 ─ RS-232 (RXD, TXD, CTS, RTS, DSR, DTR, CD, RI and GND)

Device name	Definition in LP-9x21 SDK	Default baudrate
ttyO4	COM1	115200
ttyO5 (RS-232/485/console)	None	115200
ttyS0 (RS-485)	COM2	9600
ttyS1 (RS-232/485)	COM3(LP-9421/9821 only)	9600
ttyS34 (RS-232)	COM36(LP-9421/9821 only)	9600

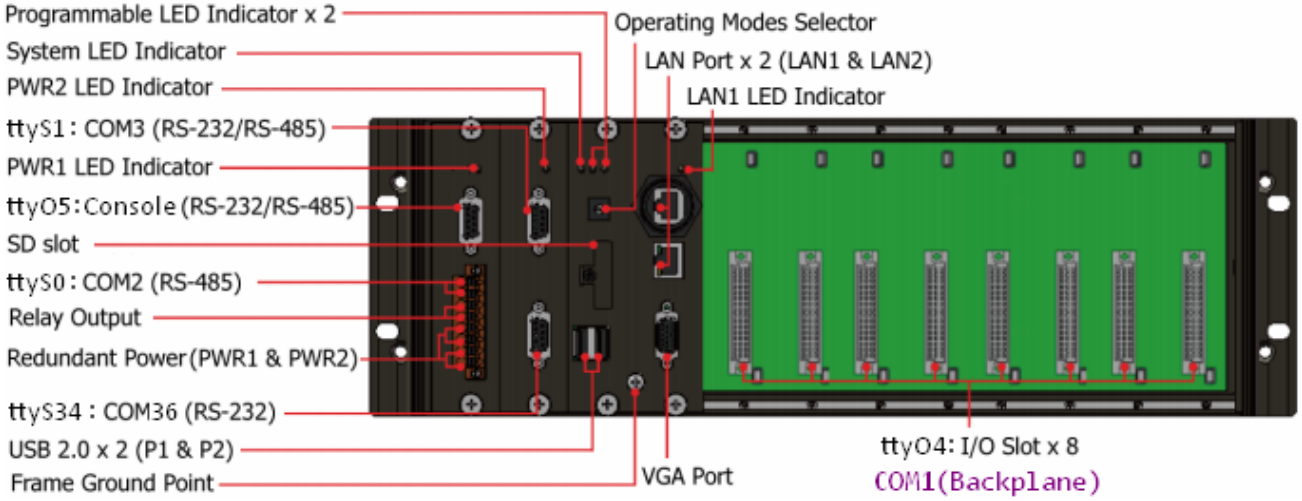


Fig. 8-1

Use the **stty** command to query or configure the COM port. For example, to modify the baudrate 9600 to 115200 bps via /dev/ttyS1 port:

stty -F /dev/ttyS1 ispeed 115200 ospeed 115200

Use the '**getsendreceive**' command to query or configure the COM port (Refert to Fig 8-2).

For example, sending the command '$01M' to query the module name which baudrate is 115200 bps connect with /dev/ttyS0 port, it will get a response: '!017060'.

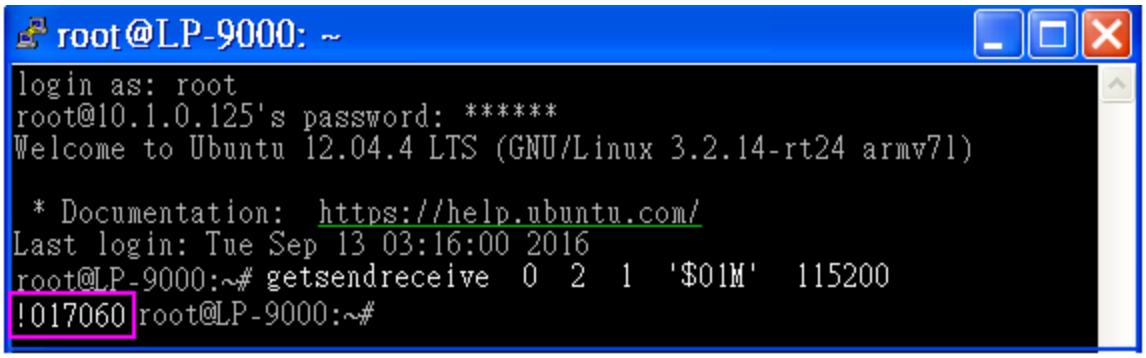


Fig. 8-2

8.1 ttyO4 Port (COM1)

The ttyO4 port is an the internal I/O expansion port on the LP-9x21, and is used to connect to an I-97K series module inserted into the LP-9x21 embedded controller. A serial command must be used to control the I-97K series module.

To control the I-97K module, the Com port parameters and call the **Open_Com()** function to open the COM1 port based on the appropriate settings. Finally, call the **ChangeToSlot(slot)** function to specify which slot will be controlled. This is like the serial address, meaning that control commands can be sent to an I/O module that is inserted in the specified slot. Consequently, the serial address for the slot that contains the module is **0**. A detailed example is provided below:

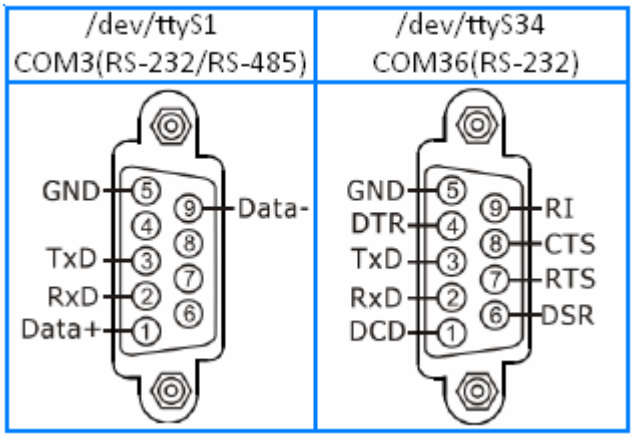
For Example:

```
int slot=1; char data=8, parity=0, stopbit=1 ;
unsigned char port=1;                          // for all modules in COM1 port of LP-9x21
DWORD baudrate=115200;
Open_Slot(slot);
Open_Com(port, baudrate, data, parity, stopbit);
ChangeSlotTol-97K(slot);
// send command...
Close_Com(port);
Close_Slot(slot);
```


8.2 ttyS1 OR ttyS34 Port (COM3 or COM36)

This ttyS1 or ttyS34 port is located on the right-upper corner on the LP-8421/8821, and is a standard **RS-232** serial port that provides TxD, RxD, RTS, CTS, GND, non-isolated and a maximum speed of 115200 bps.

This port can also be used to connect to an I-7520 module in order to provide general RS-485 communication functionality. The ttyS1 or ttyS34 port can also be used to connect to a wireless modem so that the module can be controlled from a remote device. The application example and code is demonstrated below:



- Test using C language:

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

- Test using the command line interface: (PC connected to ttyS1 on the LP-9x21)

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

  B) Send data via ttyS1 port:

  On the LP-9x21:

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

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

  C) Receive data via the ttyS1 port:

  On the LP-9x21:

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

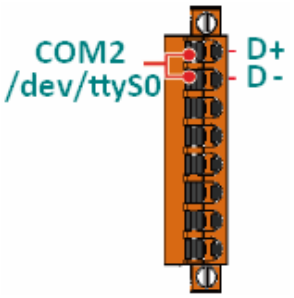
  On the PC:

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

  Check that the some words on the LP-9x21.

8.3 ttyS0 OR ttyS1 Port (COM2 or COM3)

The ttyS0 or /ttyS1 port provides **RS-485** serial communication functionality (DATA+ and DATA-) and is located on the bottom-right corner on the LP-9x21. This port allow a connection to be made to modules that contain an RS-485 interface such as the I-7000, I-97000 and I-9000 serial modules (DCON Module), meaning that ICP DAS I-7000/I-97K/I-9000 series modules can be directly controlled via this port with any converter. ICP DAS provides a very easy to use library of functions (libi8k.a) that can used to easily communicate with I-7000/I-97000/I-9000 series modules. Below is an application example of the program code demo.



- Test using C language:

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

- Test using command line: (PC ←→ i-7520 ←→ ttyS0 on the LP-9x21 - see Fig 8-3)

  A) Open "**Hyper Terminal**" on the Host PC to monitor the test process. The default settings for the ttyS0 port are 9600, 8, N, 1

  B) Send data via ttyS0 port:

  On the LP-9x21:

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

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

  C) Receive data via the ttyS0 port:

  On the LP-9x21:

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

  On the PC:

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

  Check that the same text displayed on the LP-9x21.

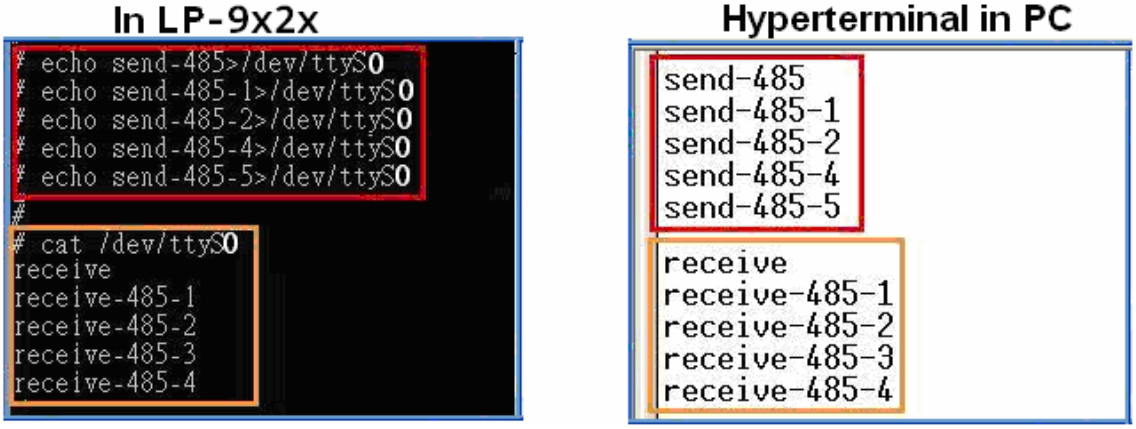


Fig. 8-3

9. Additional Support

This chapter provides additional information related to the modules supported, together with instructions that can be used to enhance the functionality and efficiency of the LP-9x21 module.

9.1 Support for N-Port Modules (I-9114, I-9144, etc.)

N-port communication modules provide **two** or **four serial ports** and can be inserted into the slot of an LP-9x21 embedded controller. In this way, additional serial ports can be used on the LP-9x21 embedded controller, meaning that the maximum number of serial ports available on the LP-9x21 will be expanded to **thirty-four**.

The LP-9x21 embedded controller is a multi-tasking uint, meaning that all the serial ports can be controlled simultaneously. **The number** of **each serial port on the** I-9114 and I-9144 modules are presented in Fig.10-1, I-9142 modules are presented in Fig.10-2, and is **fixed** based on their slot position.

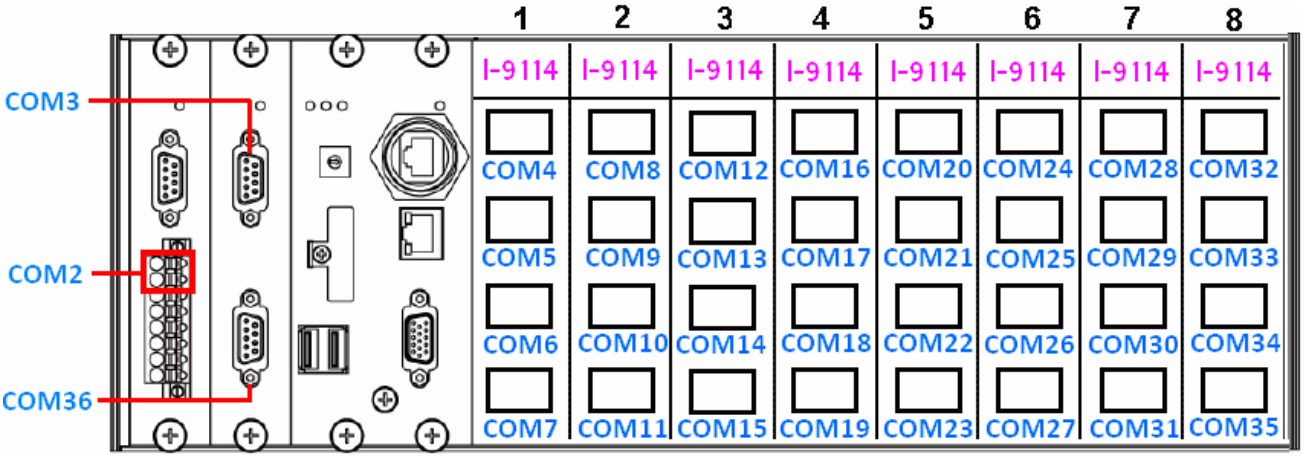


Fig.10-1

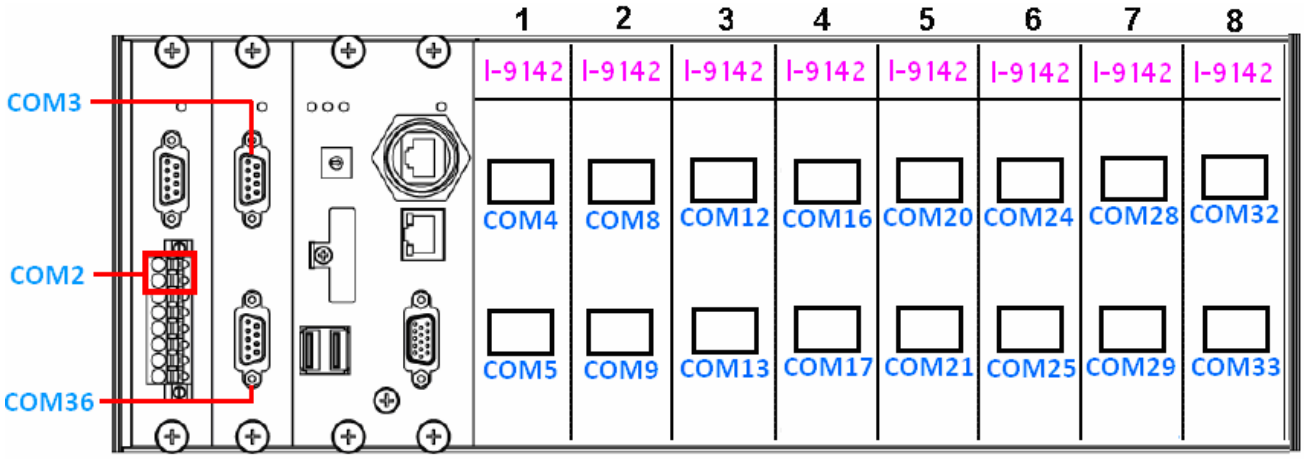


Fig.10-2

Fig.10-3 and 10-4 illustrated the serial port numbers that correspond to the **device name** on the LP-9x21.

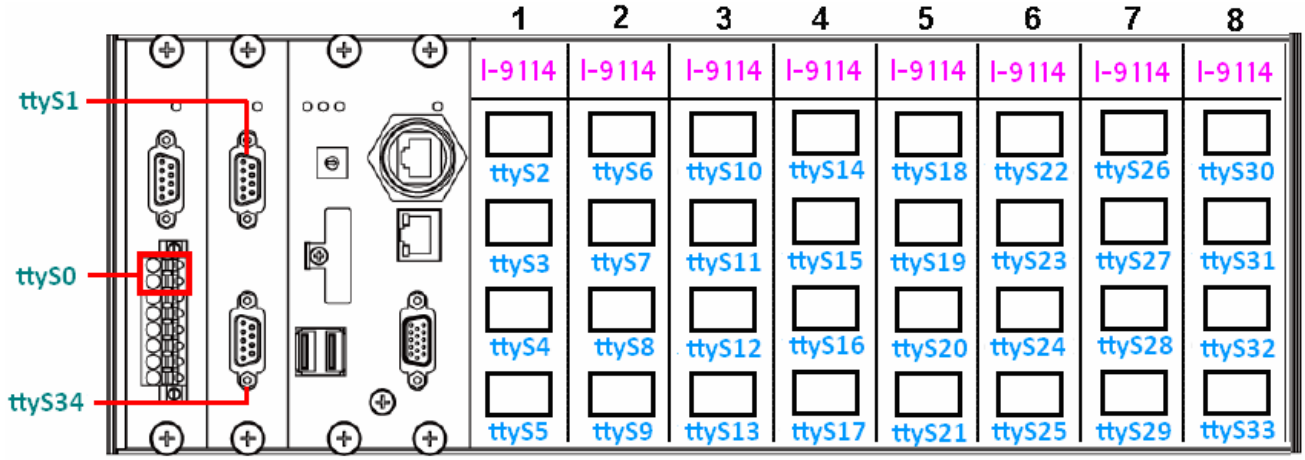


Fig.10-3

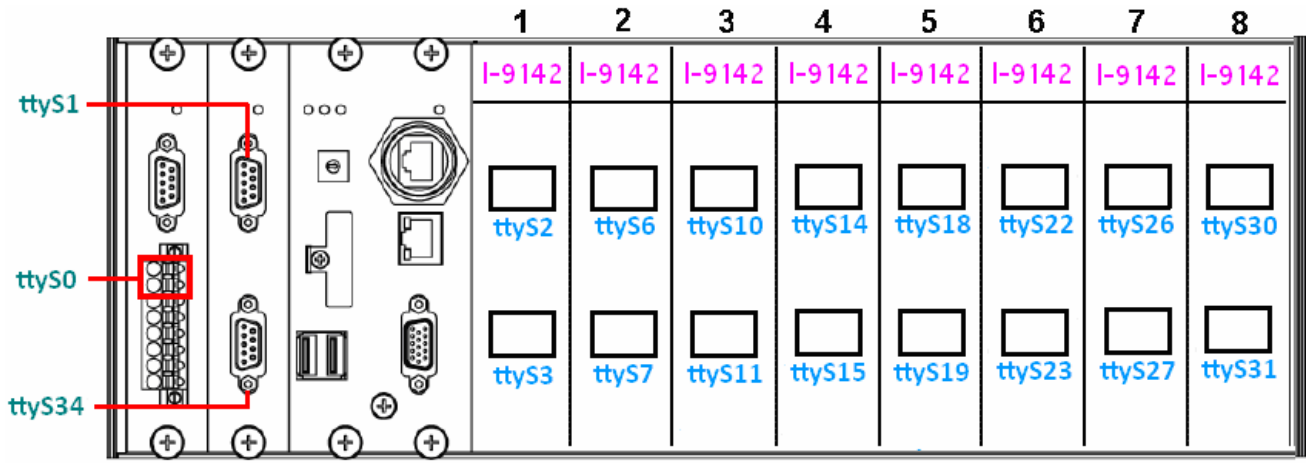


Fig.10-4

Selection guide for Hight-profile I-9K modules:

Module	Interface	Ports	Max. Speed (Kbps)	Isolation (Vrms)
I-9114	RS-232	4	115.2	2500
I-9144	RS-422/RS-485	4	115.2	2500

For more information relating to these modules, refer to:
http://www.icpdas.com/root/product/solutions/remote_io/i-9k_i-97k/i-9k_i-97k_selection.html

The **i7kdio_9114.c** demo program illustrates how to use an I-9114 module that is inserted into an LP-9x21 embedded controller. In this demo program, the I-7044 module (8 DO and 4 DI channels) is controlled through the second serial port on the I-9114 module that is inserted into the slot 2 on the LP-9x21, which, in turn, is connected to an RS-485 network. The address of the I-7044 module is 02 and the baudrate is 115200 bps. Fig.10-5 provides an illustration of the control diagram.

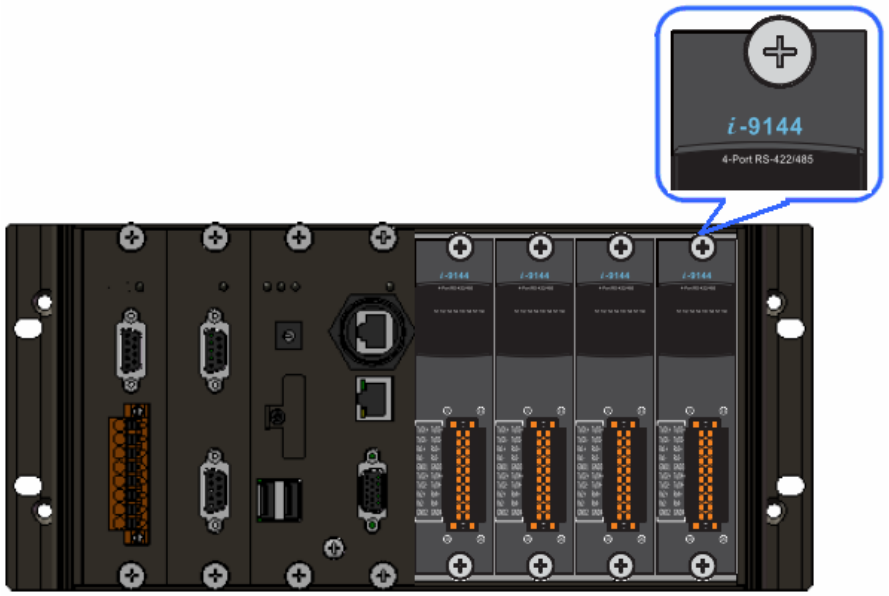


Fig.10-5

The result of executing this demo program is that the state of the DO channels can be controlled, and the program returns the state of the DI channels. The source code for the demo program is as follows:

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



```
char szSend[80], szReceive[80], ans;
WORD wBuf[12];
float fBuf[12];
int main()
{
      int wRetVal, j=0;
      char i[10];

      // Check Open_Com9 on the I-9114
      wRetVal = Open_Com(COM9, 115200, Data8Bit, NonParity, OneStopBit);

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

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

    // 7044 DO
    while(j!=113) {
            printf("Enter the DO value, or press 'q' to quit -> ");
            scanf("%s",i);

            if (i[0]=='q') {
                  wBuf[5] = 0;                  // All DO Channels OFF
                  wRetVal = DigitalOut(wBuf, fBuf, szSend, szReceive);
                  break;
            }
            j=atoi(i);
            if (j>=0 & j<=255)
                  wBuf[5] = j;                  // DO Channels ON
            else if (j>255)
                  wBuf[5] = 255;

            wRetVal = DigitalOut(wBuf, fBuf, szSend, szReceive);
            if (wRetVal)          // There was an error with the Digital Output on the I-7044
                  printf("Digital Output of 7044 is error, Error Code=%d\n", wRetVal);
            else
                  printf("The DO value of 7044 is: %u \n", wBuf[5]);

            // 7044 DI
            DigitalIn(wBuf, fBuf, szSend, szReceive);
            printf("The DI of 7044 : %u \n", wBuf[5]);
    }
    Close_Com(COM9);
    return 0;
}
```


For this example, the programming and execution procedures are the same as those described in Section 7.1. Fig. 10-6 below illustrates the result of the execution.

Fig. 10-6

Appendix A. Service Information

This appendix will show how to contact ICP DAS when you have problems in the LP-9x21 or other products.

Internet Service :

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

1. ICP DAS Web Site :

http://www.icpdas.com

2. Software Download :

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

3. E-mail for Technical Support :

service@icpdas.com

service.icpdas@gmail.com

Appendix B. Redundant Power

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

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