LinPAC

Standard API User Manual

V1.3.2

June 2024

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

Trademark

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

Contact US

If you have any problem, please feel free to contact us.

You can count on us for quick response.

Email: service@icpdas.com

Contents

1. Getting Started

This chapter provides a guided tour that describes the steps needed to know, download, install and configure the basic procedures for the user working with the LinPAC SDK for the first time.

1.1. Introduction the LinPAC SDK

This section will discuss some of the techniques that are adopted in the LinPAC SDK, including detailed explanations that describe how to easily use the LinPAC SDK. The LinPAC 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 LinPAC applications to be quickly developed. Therefore, once an application has been created, the LinPAC SDK can be used to compile it into an executable file that can be run on the LinPAC embedded controller.

1.1.1. 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.

1.1.2 Introduction to Cross-Compilation

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

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

1.1.3. Download the LinPAC SDK

- ❒ **For Windows systems:** Extract the **.exe** file into to the **C:\ driver**.
- ❒ **For Linux systems (running a 32-bit OS): Ex**tract the **.bz2** file into to the **root (/) directory**.

LinPAC		Download Path
iMX8MM	LP-2841M	https://www.icpdas.com/en/download/index.php?model=LP-2841M
AM335x Series	LP-2x4x LP-52xx	https://www.icpdas.com/en/download/show.php?num=1195&model=LP-5231M
	LP-8x2x LP-9x2x	https://www.icpdas.com/en/download/show.php?num=915&model=LP-9821
X86/E38xx Series	LX-Series	https://www.icpdas.com/en/download/show.php?num=904&model=LX-9371

Note:

1) There are six independent LinPAC SDKs above, and different LinPAC cannot share both source files, library file and compiled files, user should be download the respective LinPAC SDK versions from the target manager and use them.
2) We recommend user to change user ID to become **root** by '**sudo**' or '**su**' command.
3) Linux 64-bit operating system lacks 32-bit support libraries. If your Linux PC is 64-bit OS, you must install 32-bit libraries on your system before you run the 32-bit version of the LinPAC SDK (Linux version).

1.2. The Architecture of LIBI8K.A in the Linux PAC

The library file **libi8k.a** is designed for I-7000/8000/9000/87000/97000 applications running on the LinPAC 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 relationships among the libi8k.a and user's applications are depicted as Figure 1.2-1:

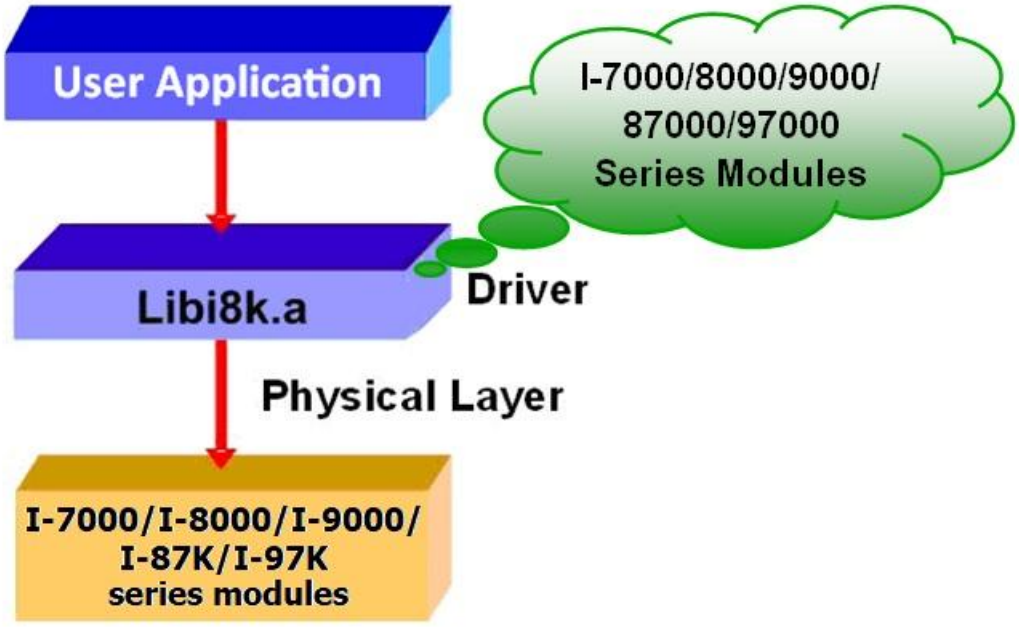


Figure 1.2-1. The relationship between the libi8k.a library and the custom applications

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

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

The functions contained in the libi8k.a library is specifically designed for the LinPAC controller, and those functions needed for specific applications can easily be determined from the descriptions provided and from the demo programs described in chapter 7.

1.3. Setting up the Development Environment

The OS of LinPAC series is Linux, and the 'LinPAC SDK' is a development toolkit provided by ICP DAS, which can be used to easily develop custom applications for the LinPAC series embedded controller platform. The toolkit consists of the following items.

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

Refer to the following chapter to download and install the appropriate SDK.

Step	LP-2241M/5231/8x2x/9x2x	LP-2841M/LX-8000/9000
0.	Download SDK on Windows or Linux PC	Download SDK on LinPAC
1.	Find demo 'helloworld.c' in SDK	Find demo 'helloworld.c' in SDK
2.	Compile the demo on Windows or Linux PC using SDK	Compile the demo on LinPAC directly
3.	Upload and execute the demo on LinPAC	Execute the application on LinPAC at boot time
4.	Execute the application on LinPAC at boot time	—

1.3.1. LinPAC PXA270 Series

The topic provides LinPAC PXA270 SDK installation instructions for the following platforms:

- Linux (running a **32-bit** operating system)

  Download/Install LinPAC PXA270 SDK on Linux

- Windows

  Download/Install LinPAC PXA270 SDK on Windows

  Integrating LinPAC PXA270 SDK with Code::Blocks IDE

❒ Download/Install LinPAC PXA270 SDK on Linux

1. Before installing the LP-8x4x SDK, several tasks must be completed, as the root user by '**sudo**' or '**su**' command.

2. Insert the installation CD into your CD-ROM driver (refer to Figures 1.3.1-1 and 1.3.1-2). Locate the '**linpac_pxa270_sdk_for_linux.tar.bz2**' file in the \napdos\lp-8x4x\SDK\ folder, or visit the ICP DAS website to download the latest version.

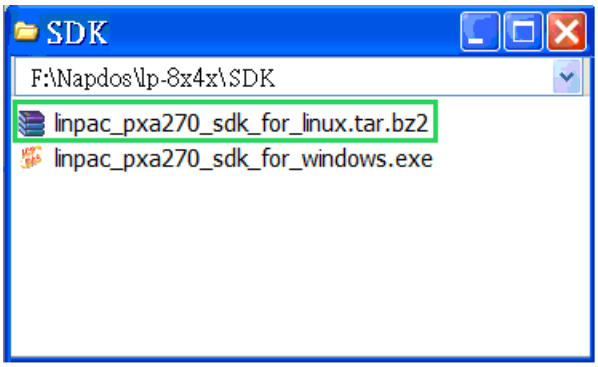


Figure 1.3.1-1.

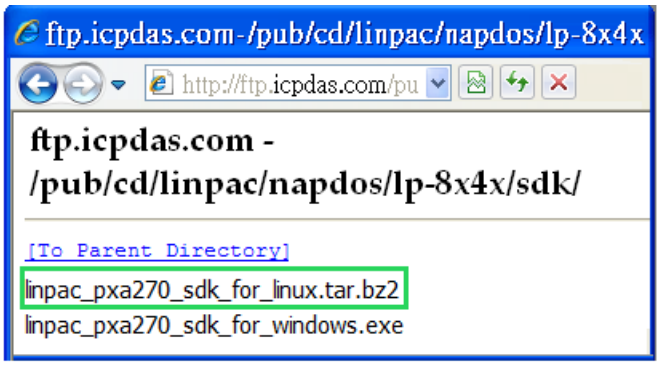


Figure 1.3.1-2.

3. Download SDK in '/ (the root directory)', and try the following command to decompress file. (refer to Figure 1.3.1-3)

```
#   tar   jxvf   linpac_pxa270_sdk_for_linux.tar.bz2
```

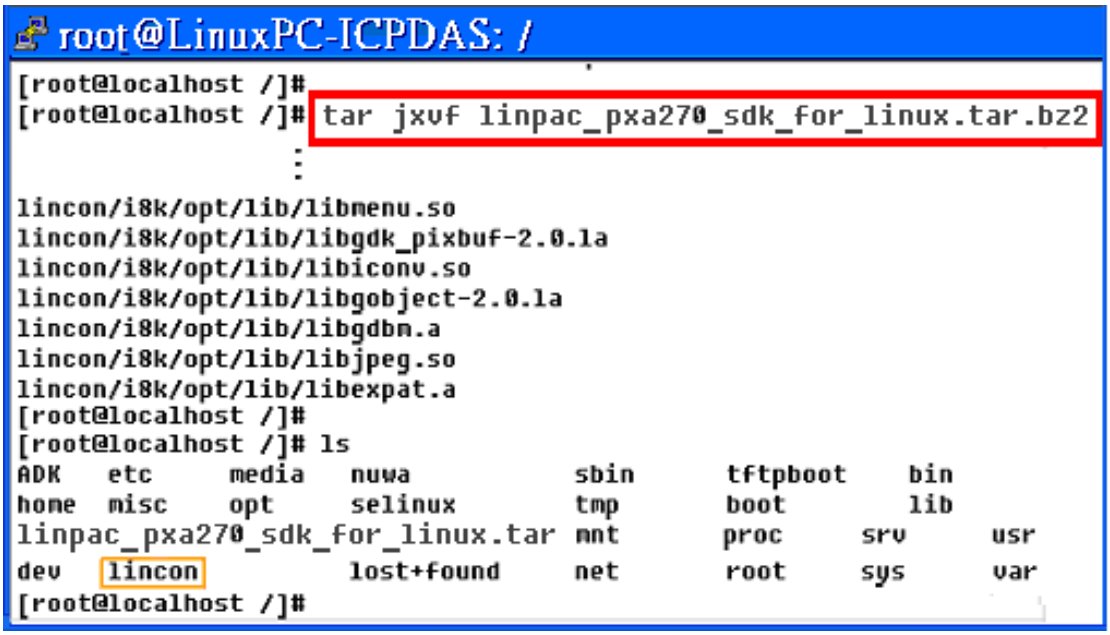


Figure 1.3.1-3. Decompress '.tar.bz2' file

4. Before compile the program, you need to set LinPAC PXA270 SDK path in environment variables. To execute the shell startup script and set the environment variables, enter the following command.

```
#   . /lincon/linpac.sh
```

5. Type '**make**' on the command line it will execute the compile command according to the Makefile. (refer to Figure 1.3.1-4)

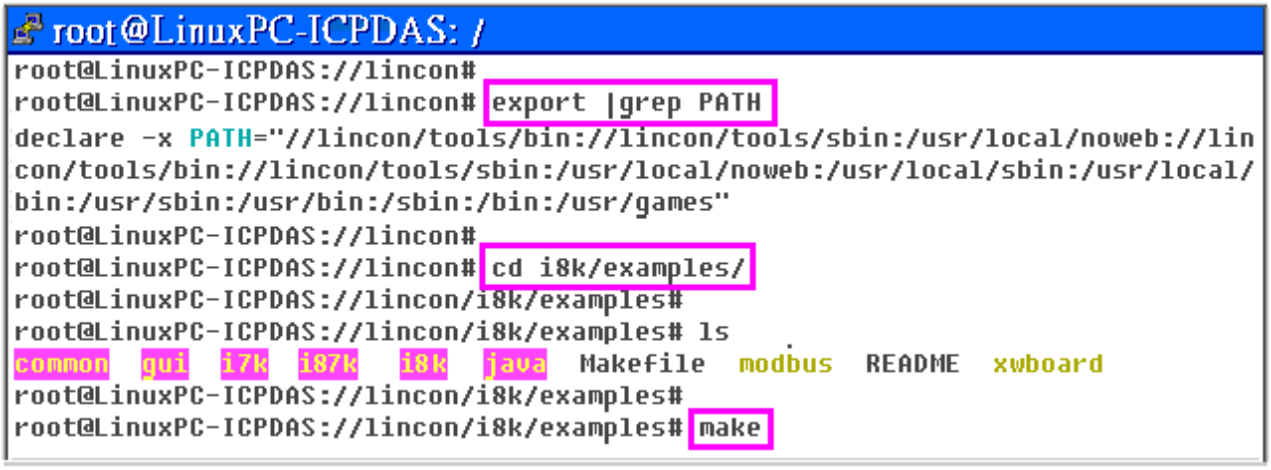


Figure 1.3.1-4. Compiling demo code according to the Makefile

❒ Download/Install LinPAC PXA270 SDK on Windows

The LinPAC_PXA270_SDK_for_Windows.exe provides compilers, library, header, examples, and IDE workspace file (for Code::Blocks project). Following the step by step procedure below will help users get started.

1. Insert the installation CD into your CD-ROM driver.

2. Open the \napdos\lp-8x4x\SDK\ folder and double-click the icon for the 'linpac_pxa270_sdk_for_windows.exe' file, when the Setup Wizard is displayed, click the 'Next>' button to continue, refer to Figure 1.3.1-5.

3. Click the 'I accept the agreement' option and then click the 'Next' button, refer to Figure 1.3.1-6 below.

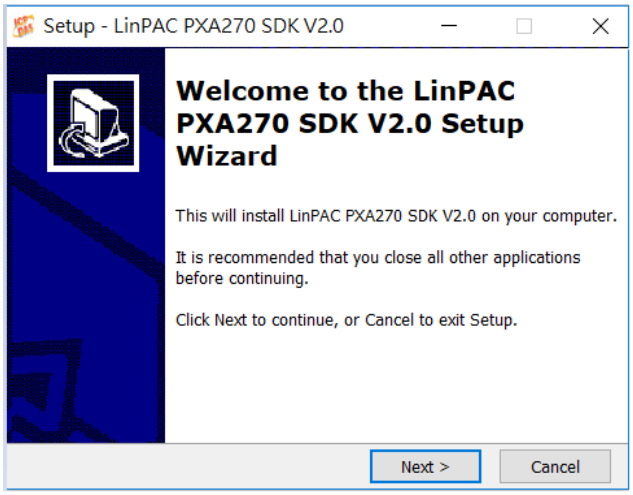


Figure 1.3.1-5.

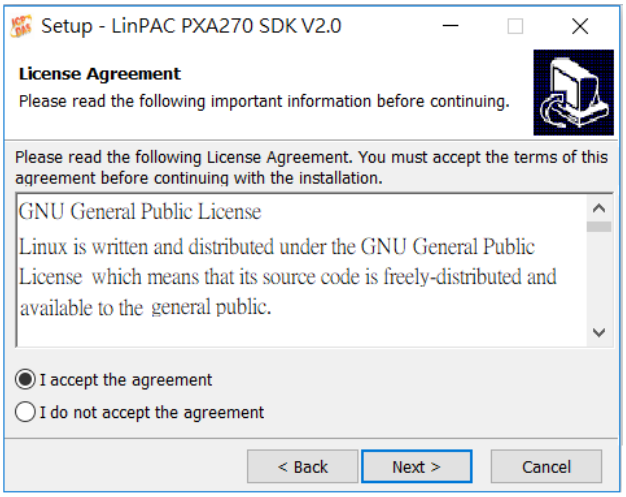


Figure 1.3.1-6.

4. The 'LinPAC PXA270 SDK' files will be extracted and installed and a progress bar will be displayed to indicate the status, refer to Figure 1.3.1-7.

5. Once the software has been successfully installed, click the 'Finish' button to complete the development toolkit installation, refer to Figure 1.3.1-8.

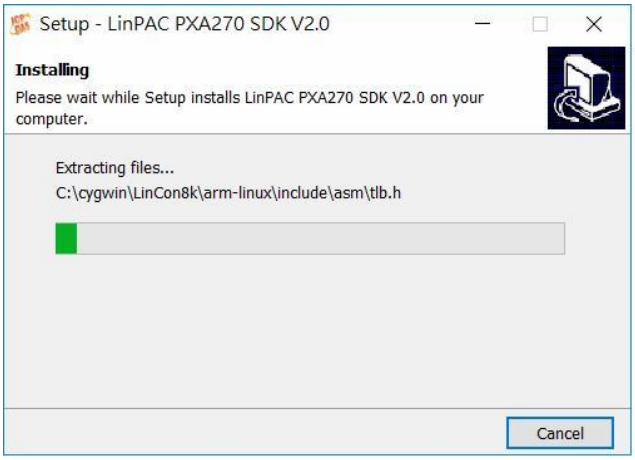


Figure 1.3.1-7.

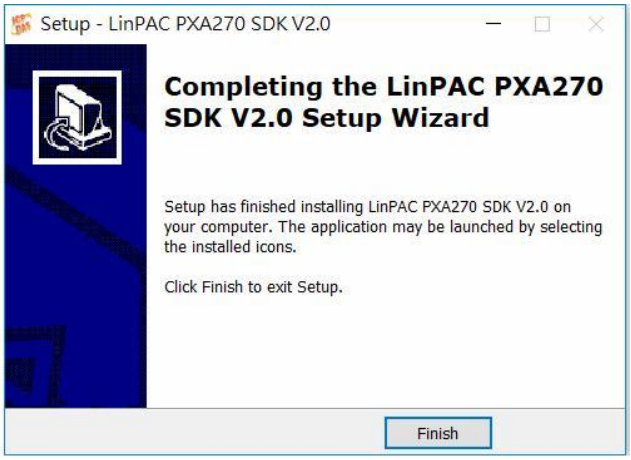


Figure 1.3.1-8.

6. Open the LinPAC PXA270 SDK installation directory, the default data directory location is '**C:\cygwin**', user can see the contents of folder. Refer to Figure 1.3.1-9.

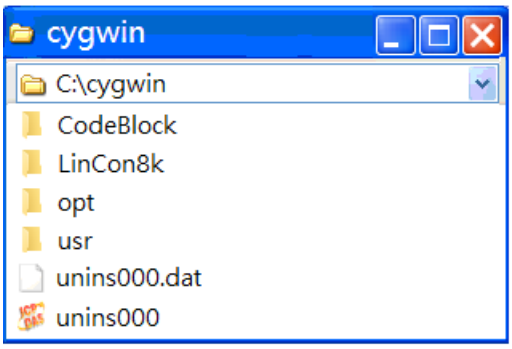


Figure 1.3.1-9. Open the folder of the LinPAC AM335x

7. Open the '**C:\cygwin\LinCon8k**' folder and see the content. Refer to Figure 1.3.1-10.

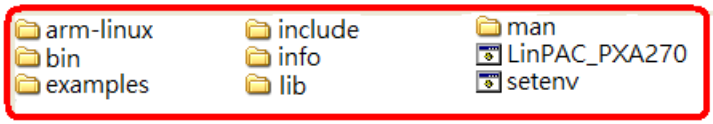


Figure 1.3.1-10. The contents of the folder

8. From the desktop, right-click the shortcut icon for the '**LinPAC PXA270 Build Environment**' and select 'Run As Administrator'. Or click the 'Start' > 'Programs' > 'ICPDAS' > 'LinPAC PXA270 Build Environment'.

   A Command Prompt window will then be displayed that allows applications for the LP-8x4x to be compiled. Refer to Figure 1.3.1-11.

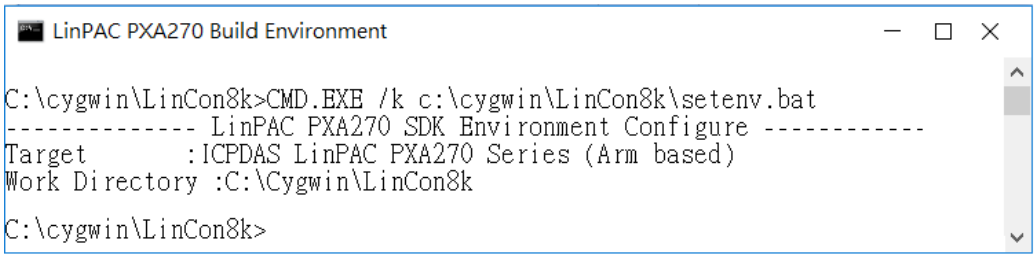


Figure 1.3.1-11. Click the 'LinPAC_AM335x Build Environment'

9. Type '**make**' command (needs run as an administrator). A Command Prompt window will then be displayed that allows applications for the LP-8x4x to be compiled. Refer to Figure 1.3.1-12.

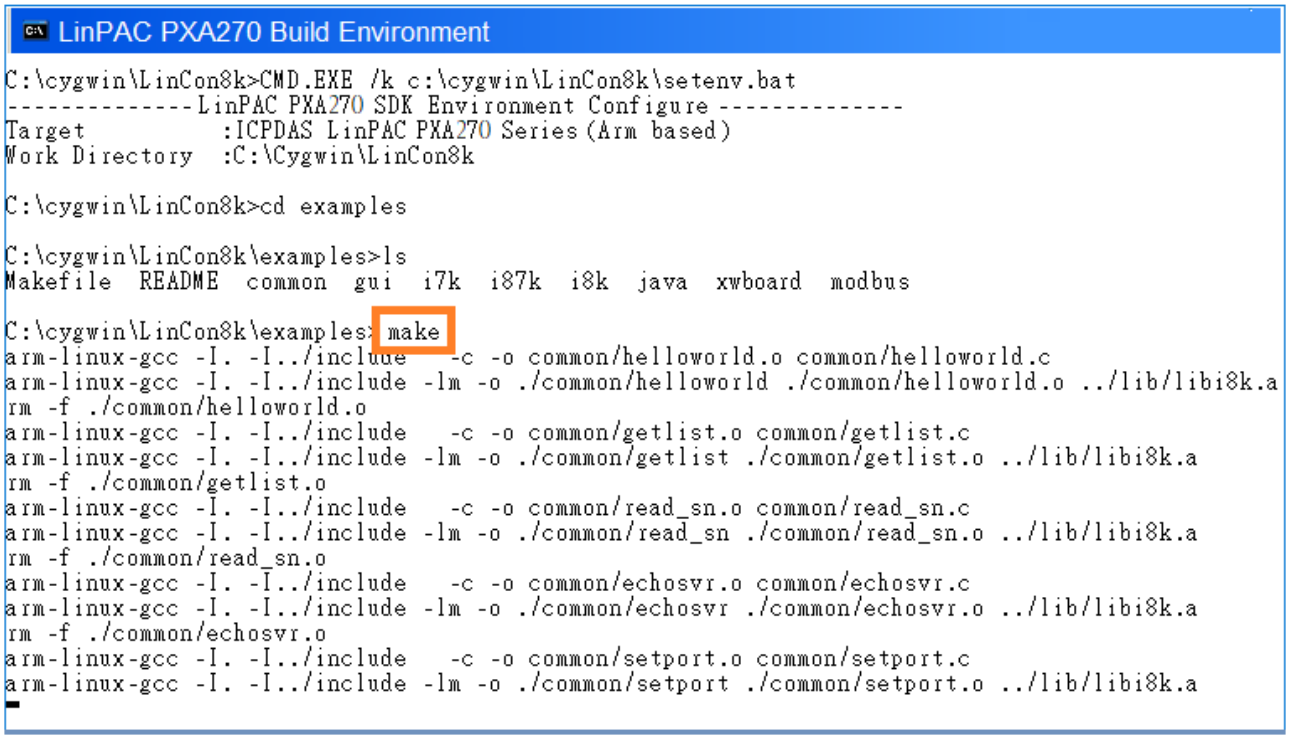


Figure 1.3.1-12. Compiling demo code according to the Makefile

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

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

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

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

❒ Integrating LinPAC PXA270 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 LinPAC PXA270 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 LinPAC_PXA270_SDK workspace file.
- Open the **C:\cygwin\CodeBlock** folder, and double click the '**LinPAC_PXA270_SDK**' as below (refer to Figure 1.3.1-13):

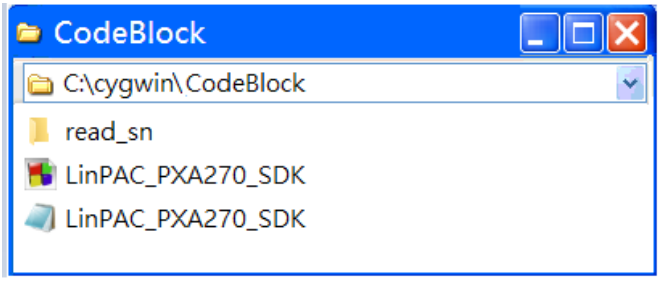


Figure 1.3.1-13. Startup the LinPAC AM335x SDK

➢ Following window will come up (refer to Figure 1.3.1-14):

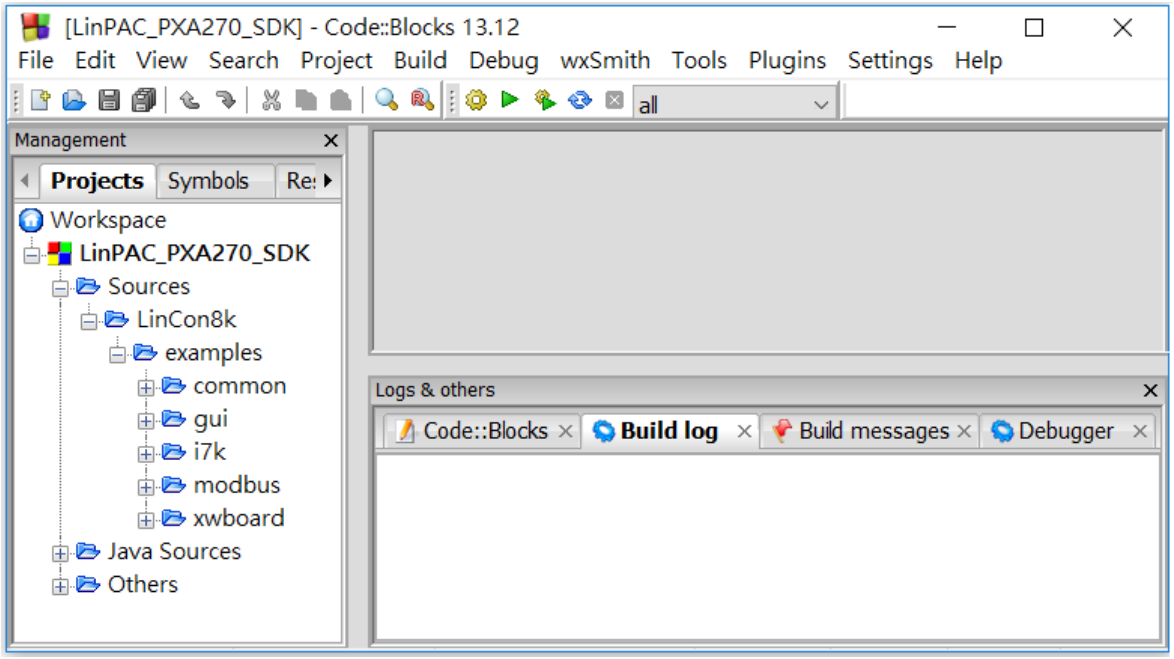


Figure 1.3.1-14. Startup the LinPAC AM335x SDK

➢ Check Compiler settings for Linaro GCC cross compiler : Click 'Settings' > 'Compiler' > 'Toolchain executables tab' (refer to Figure 1.3.1-15) :

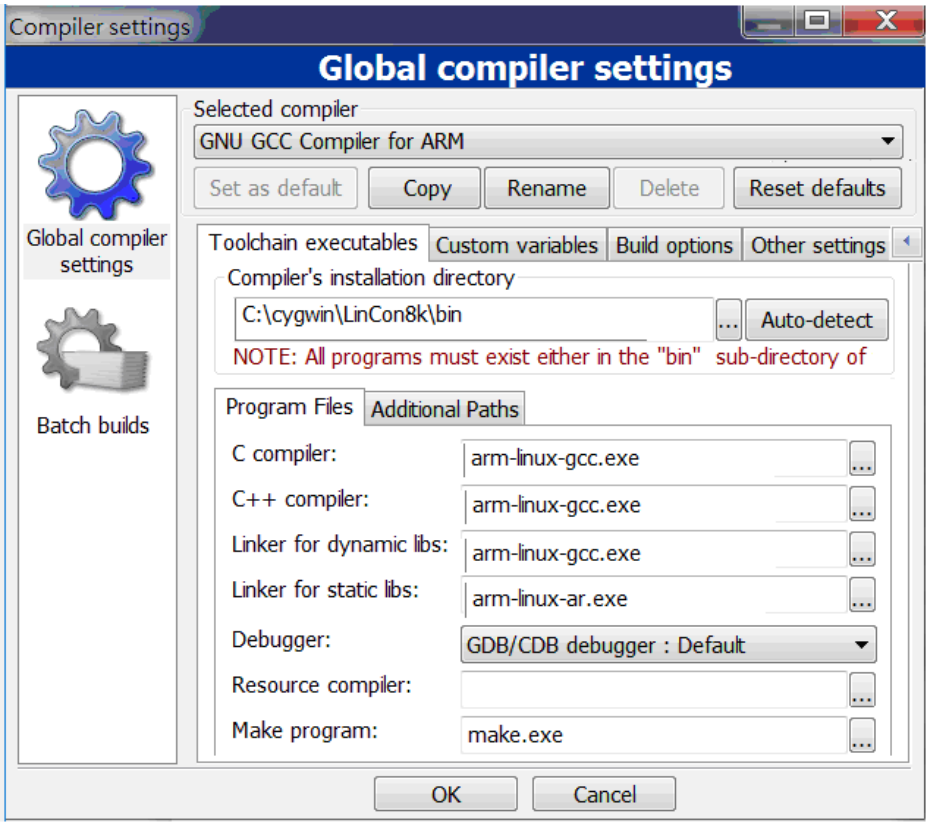


Figure 1.3.1-15. Check compiler settings

➢ Check Link libraries for Linaro GCC cross compiler : Click ‘Settings’ > ‘Compiler’ > ‘Linker Settings’ (refer to Figure 1.3.1-16) :

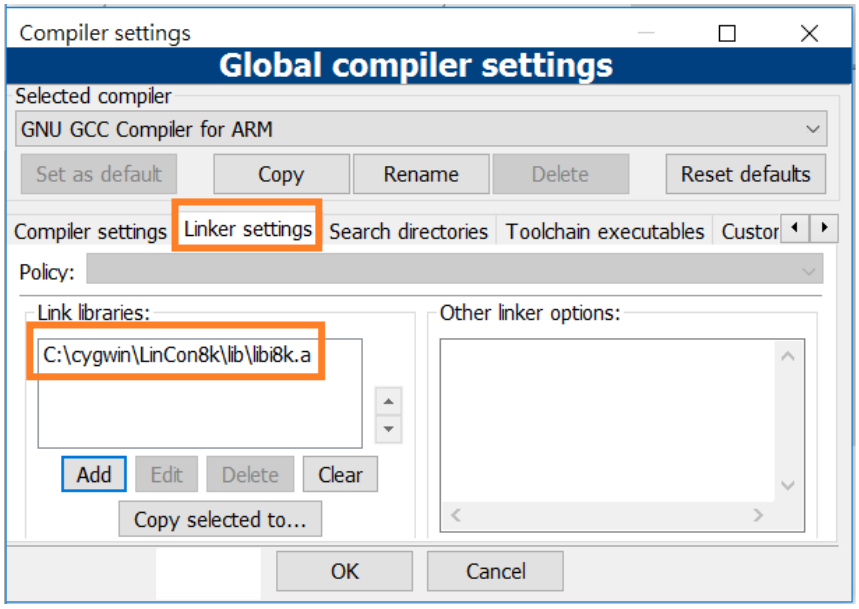


Figure 1.3.1-16. Check Link libraries for Linaro GCC cross compiler

➢ Check **Makefile** for Linaro GCC cross compiler : Click ‘Project’ > ‘Properties’ (refer to Figure 1.3.1-17) :

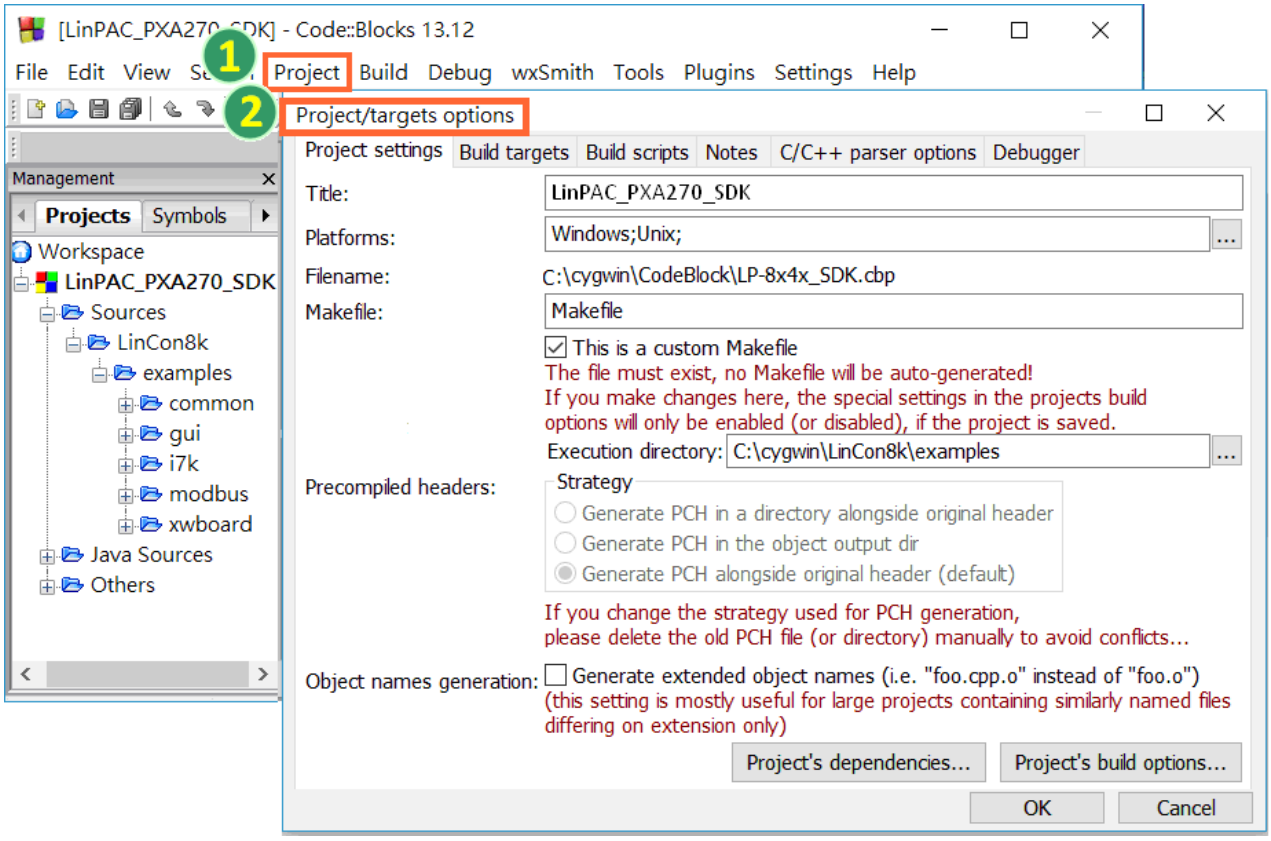


Figure 1.3.1-17. Check Makefile for Linaro GCC cross compiler

➢ Click **Build** options, and it will compile the LinPAC PXA270 project completely (refer to Figure 1.3.1-18).

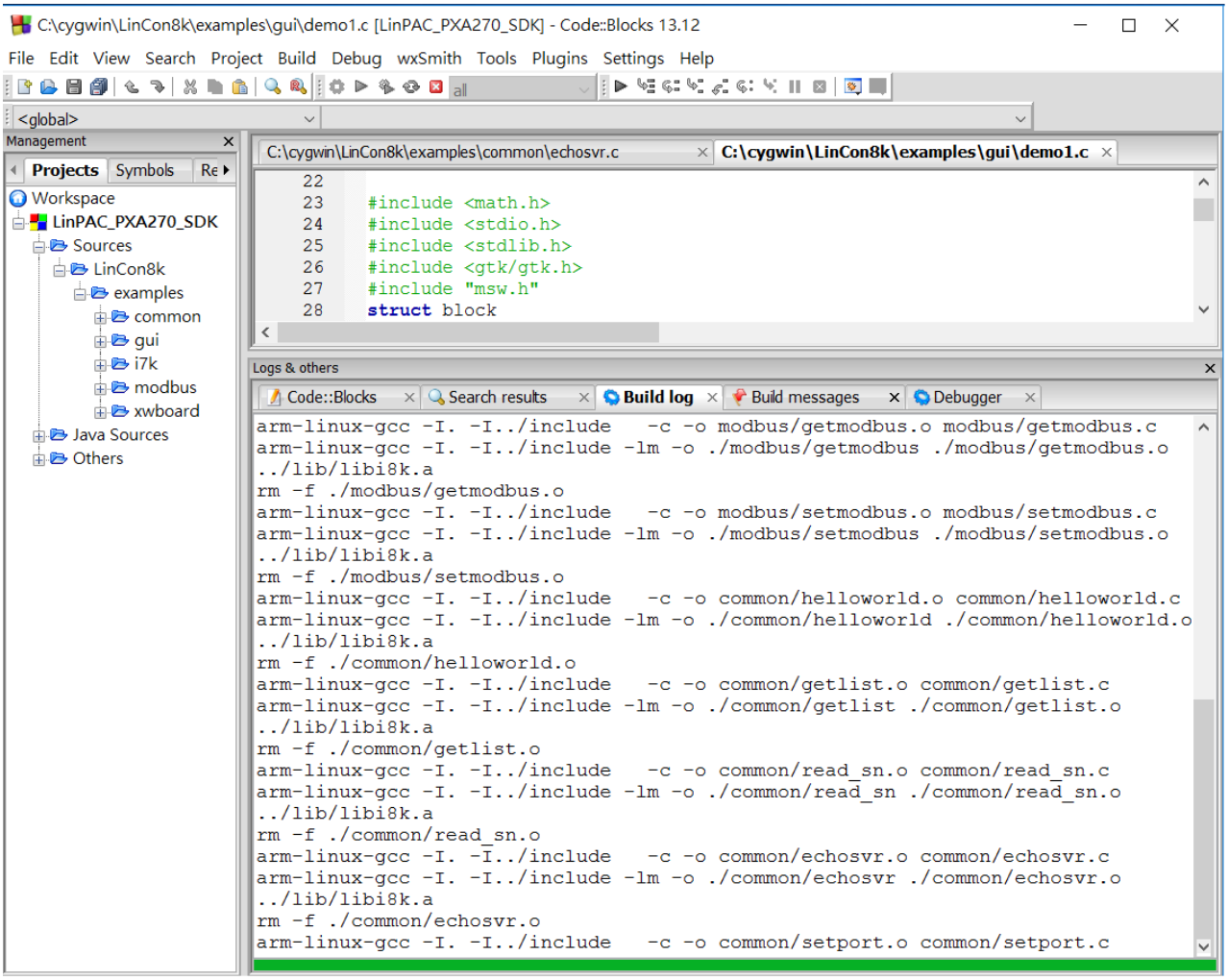


Figure 1.3.1-18. Compiling a C program

【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 PXA270 Build Environment**'.
- ❒ Type command: **chcp 65001** (Refer to Figures 1.3.1-19 and 1.3.1-20).

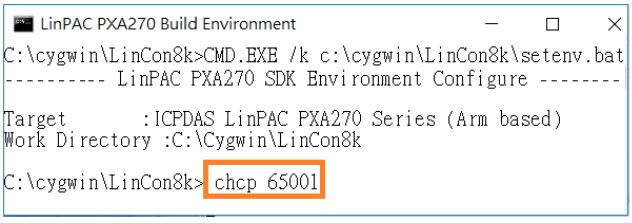


Figure 1.3.1-19.

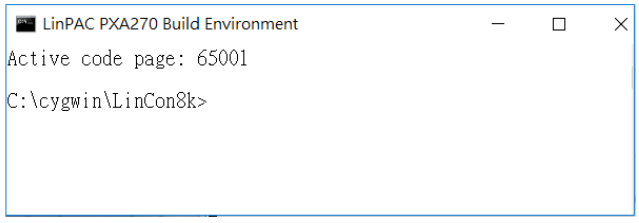


Figure 1.3.1-20.

1.3.2. LinPAC AM335x Series

The topic provides LinPAC_AM335x SDK installation instructions for the following platforms:

- Linux (running a 32-bit operating system)

  Download/Install LinPAC AM335x SDK on Linux

- Windows

  Download/Install LinPAC AM335x SDK on Windows

  Integrating LinPAC AM335x SDK with Code:: Blocks IDE

❒ Download/Install LinPAC AM335x 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 Figure 1.3.2-1)

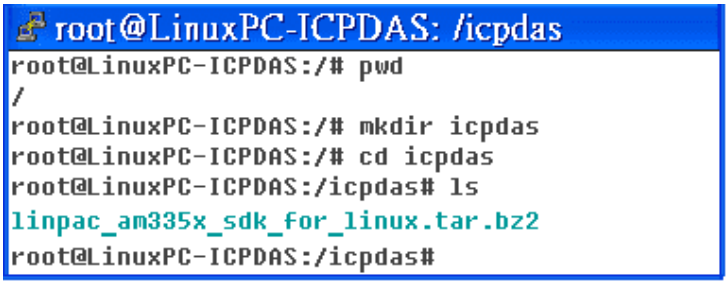


Figure 1.3.2-1. Create a directory named 'icpdas'

2. Visit the ICP DAS website to download the latest version of the LinPAC_AM335x SDK -- '**linpac_am335x_sdk_for_linux.tar.bz2**' for example.

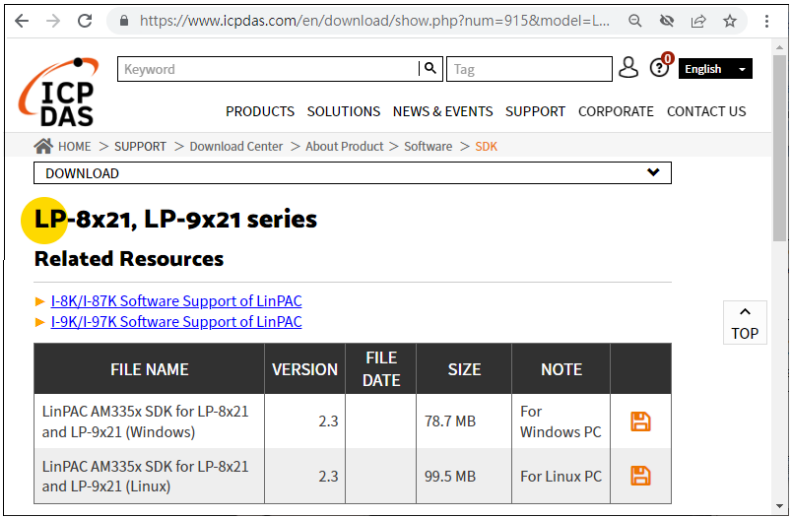


Figure 1.3.2-2.

Figure 1.3.2-3.

3. Try the following command to decompress file. (Refer to Figure 1.3.2-4)

```
#  tar  jxvf  linpac_am335x_sdk_for_linux.tar.bz2
```

```
root@LinuxPC-ICPDAS: /icpdas
root@LinuxPC-ICPDAS:/icpdas# tar jxvf linpac_am335x_sdk_for_linux.tar.bz2
linpac_am335x_sdk/
linpac_am335x_sdk/linpac_am335x.sh
linpac_am335x_sdk/tools/
linpac_am335x_sdk/tools/lib/
linpac_am335x_sdk/tools/lib/gcc/
linpac_am335x_sdk/tools/lib/gcc/arm-linux-gnueabihf/
linpac_am335x_sdk/tools/lib/gcc/arm-linux-gnueabihf/4.7.3/
linpac_am335x_sdk/tools/lib/gcc/arm-linux-gnueabihf/4.7.3/crtbeginS.o
linpac_am335x_sdk/tools/lib/gcc/arm-linux-gnueabihf/4.7.3/libgcc.a
```

Figure 1.3.2-4. Decompress '.tar.bz2' file

4. Before compiling 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 Figure 1.3.2-5).

```
root@LinuxPC-ICPDAS: /icpdas/linpac_am335x_sdk
root@LinuxPC-ICPDAS:/icpdas#
root@LinuxPC-ICPDAS:/icpdas# cd linpac_am335x_sdk
root@LinuxPC-ICPDAS:/icpdas/linpac_am335x_sdk# ls
i8k  linpac_am335x.sh  tools
root@LinuxPC-ICPDAS:/icpdas/linpac_am335x_sdk# . linpac_am335x.sh
root@LinuxPC-ICPDAS:/icpdas/linpac_am335x_sdk# export | grep PATH
declare -x PATH="/icpdas/linpac_am335x_sdk/tools/bin:/icpdas/linpac_am335x_sdk/tools/sbin:
/usr/local/noweb:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games"
root@LinuxPC-ICPDAS:/icpdas/linpac_am335x_sdk# ls  i8k/
ChangeLog  examples  include  lib  opt
root@LinuxPC-ICPDAS:/icpdas/linpac_am335x_sdk#
```

Figure 1.3.2-5. Setting environment variables for LinPAC_AM335x SDK

5. Type '**make**' on the command line it will execute the compile command according to the Makefile. (Refer to Figure 1.3.2-6)

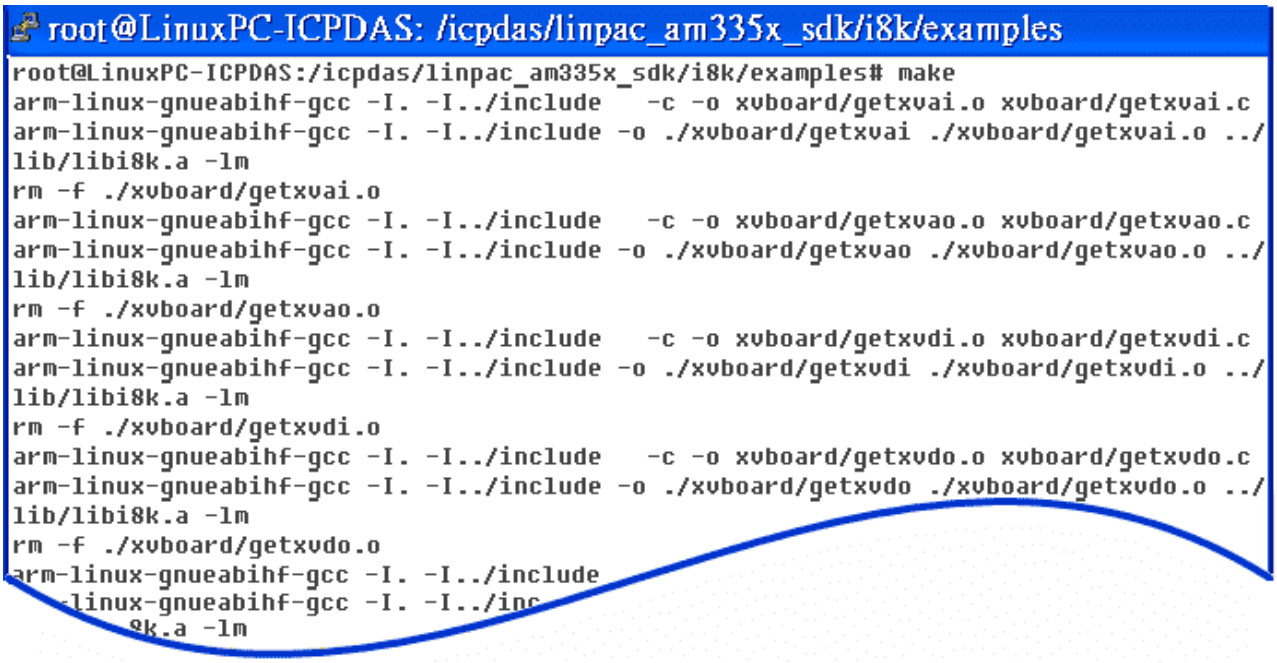


Figure 1.3.2-6. Compiling demo code according to the Makefile

❒ Download/Install LinPAC AM335x SDK on Windows

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

1. Download LinPAC AM335x SDK from website.

2. Open the '**LinPAC_AM335x_SDK_for_Windows.exe**' file, when the Setup Wizard is displayed, click the 'Next>' button to continue, refer to Figures 1.3.2-7 and 1.3.2-8.

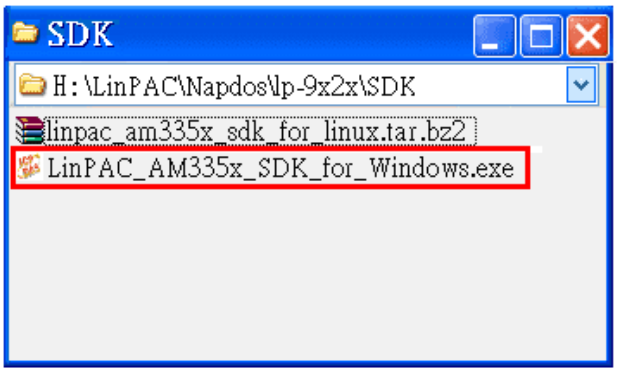


Figure 1.3.2-7.

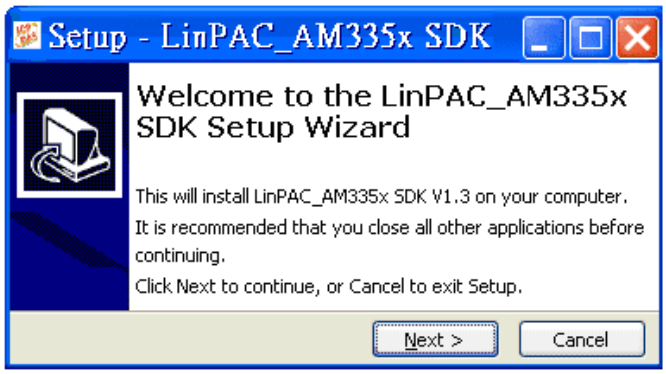


Figure 1.3.2-8.

3. Click the '**I accept the agreement**' option and then click the '**Next**' button (refer to Figure 1.3.2-9), and select Start Menu Folder option and then click the '**Next**' button, refer to Figure 1.3.2-10.

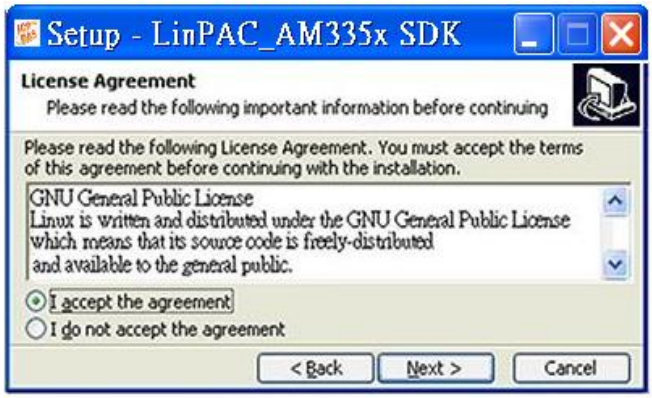


Figure 1.3.2-9.

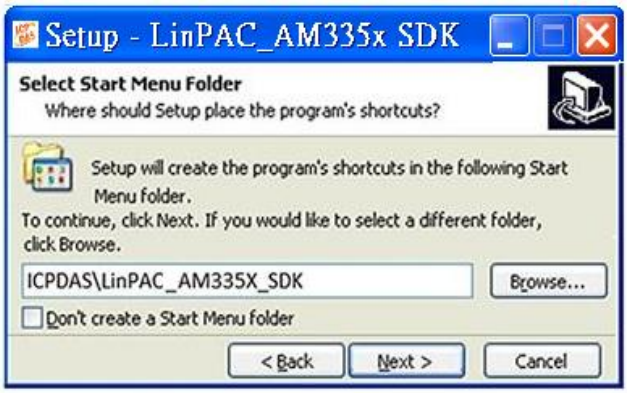


Figure 1.3.2-10.

4. The LinPAC_AM335x SDK files will be extracted and installed and a progress bar will be displayed to indicate the status, refer to Figure 1.3.2-11.

5. Once the software has been successfully installed, click the '**Finish**' button to complete the development toolkit installation, refer to Figure 1.3.2-12.

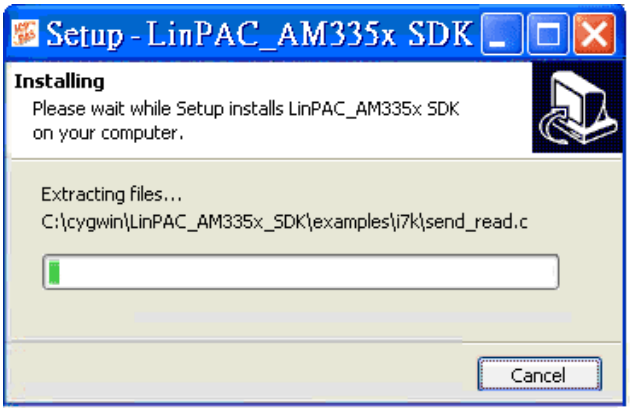


Figure 1.3.2-11.

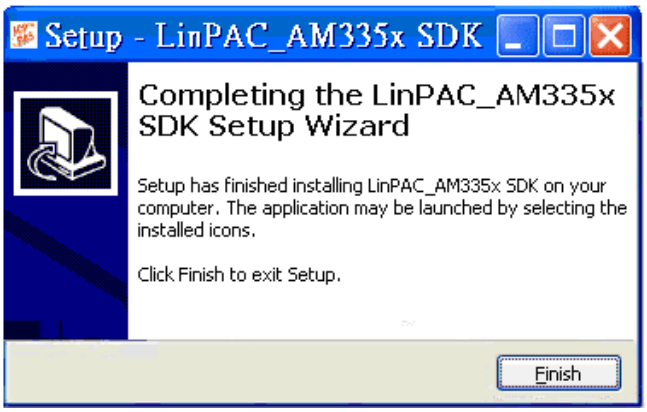


Figure 1.3.2-12.

6. Open the LinPAC_AM335x SDK installation directory, the default data directory location is '**C:\cygwin**', the user can see the contents of the folder. Refer to Figures 1.3.2-13 and 1.3.2-14.

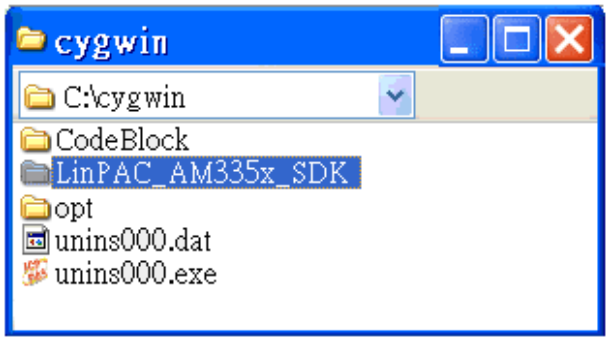


Figure 1.3.2-13.

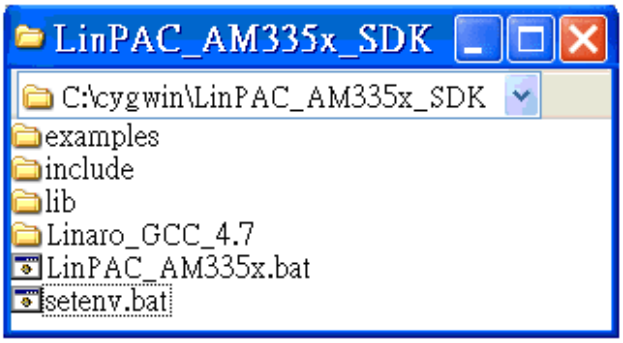


Figure 1.3.2-14.

7. From the desktop, double-click the shortcut icon for the ‘LinPAC_AM335x Build Environment’ or click the ‘Start’ > ‘Programs’ > ‘ICPDAS’ > ‘LinPAC_AM335x_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 Figures 1.3.2-15 and 1.3.2-16.

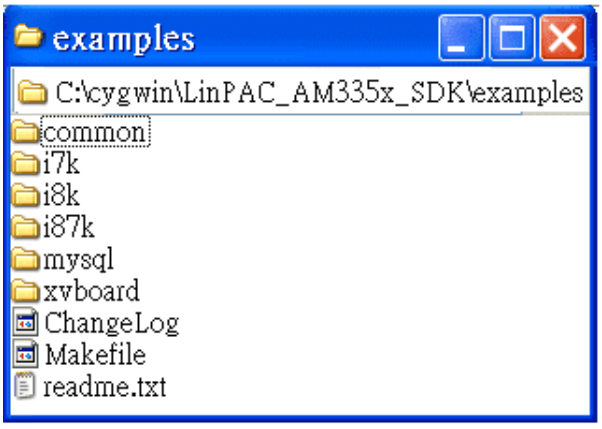


Figure 1.3.2-15.

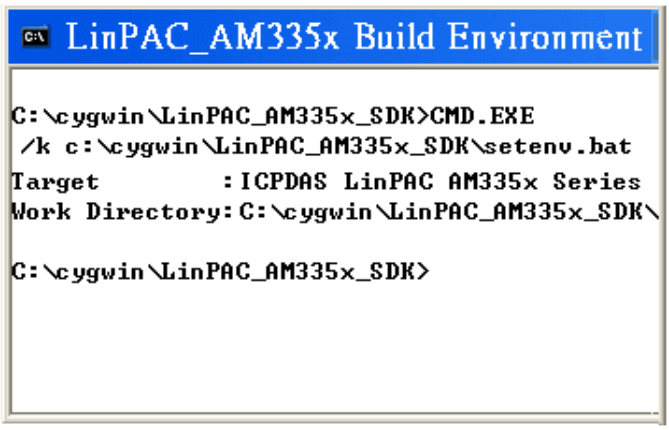


Figure 1.3.2-16.

8. Type ‘**make**’. A Command Prompt window will then be displayed that allows applications for the LinPAC_AM335x to be compiled. Refer to Figure 1.3.2-17.

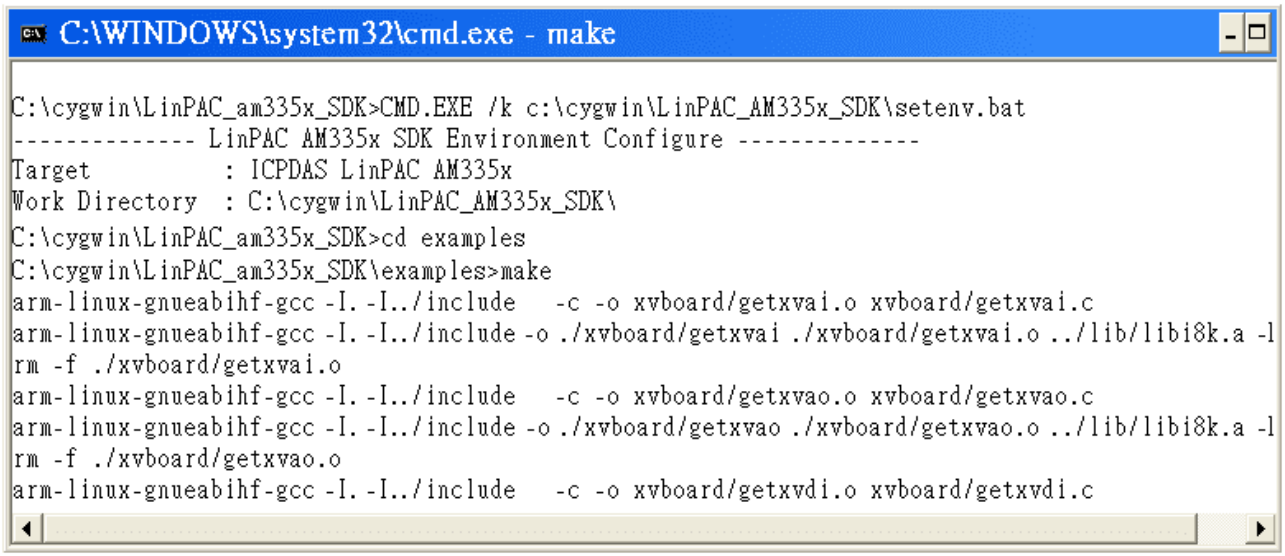


Figure 1.3.2-17. Compiling demo code according to the Makefile

❒ Integrating LinPAC AM335x 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 LinPAC 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 LinPAC_AM335x_SDK workspace file.
- Open the **C:\cygwin\CodeBlock** folder, and double click the '**LinPAC_AM335x_SDK**' as below (Refer to Figure 1.3.2-18):

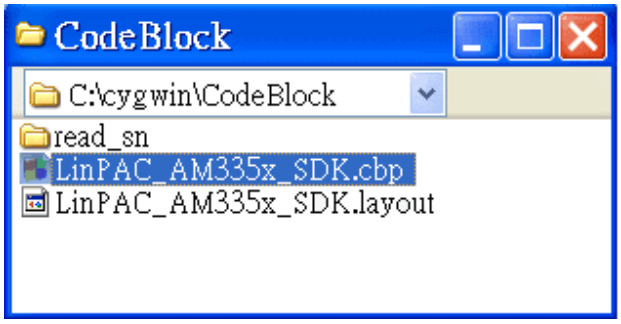


Figure 1.3.2-18. Startup the LinPAC AM335x SDK

➢ Following window will come up (Refer to Figure 1.3.2-19):

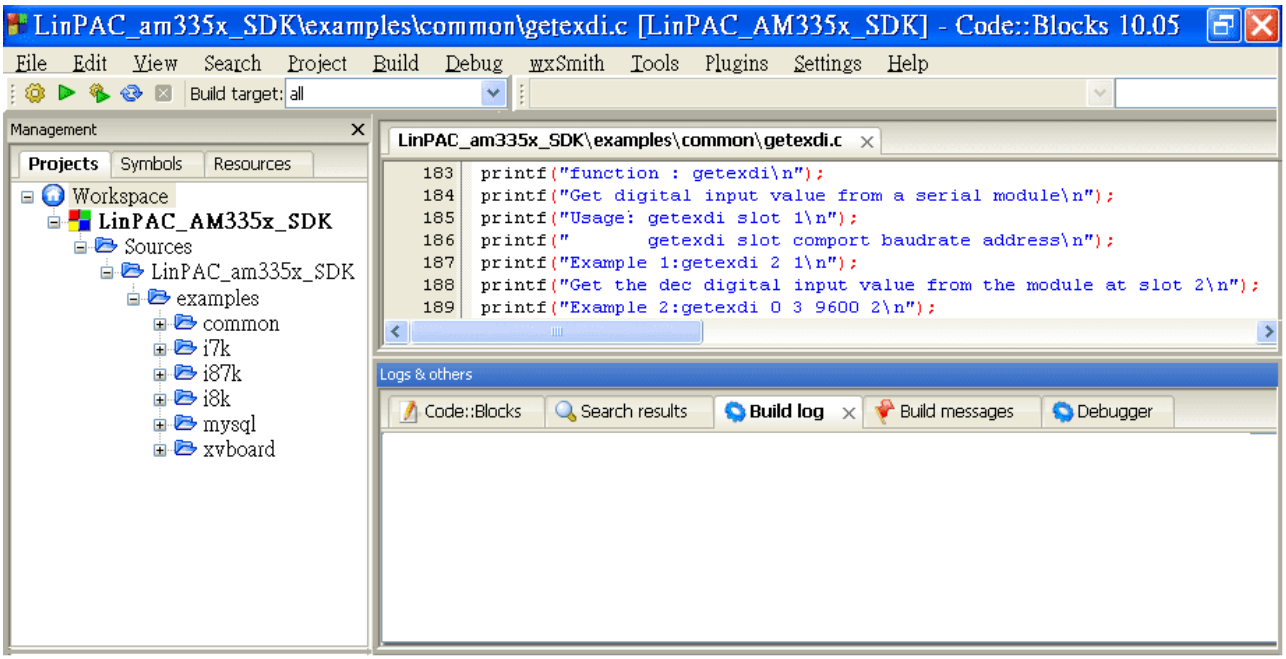


Figure 1.3.2-19. Startup the LinPAC AM335x SDK

➢ Check compiler settings for Linaro GCC cross compiler: Click 'Settings' > 'Compiler' > 'Toolchain executables tab' (Refer to Figure 1.3.2-20):

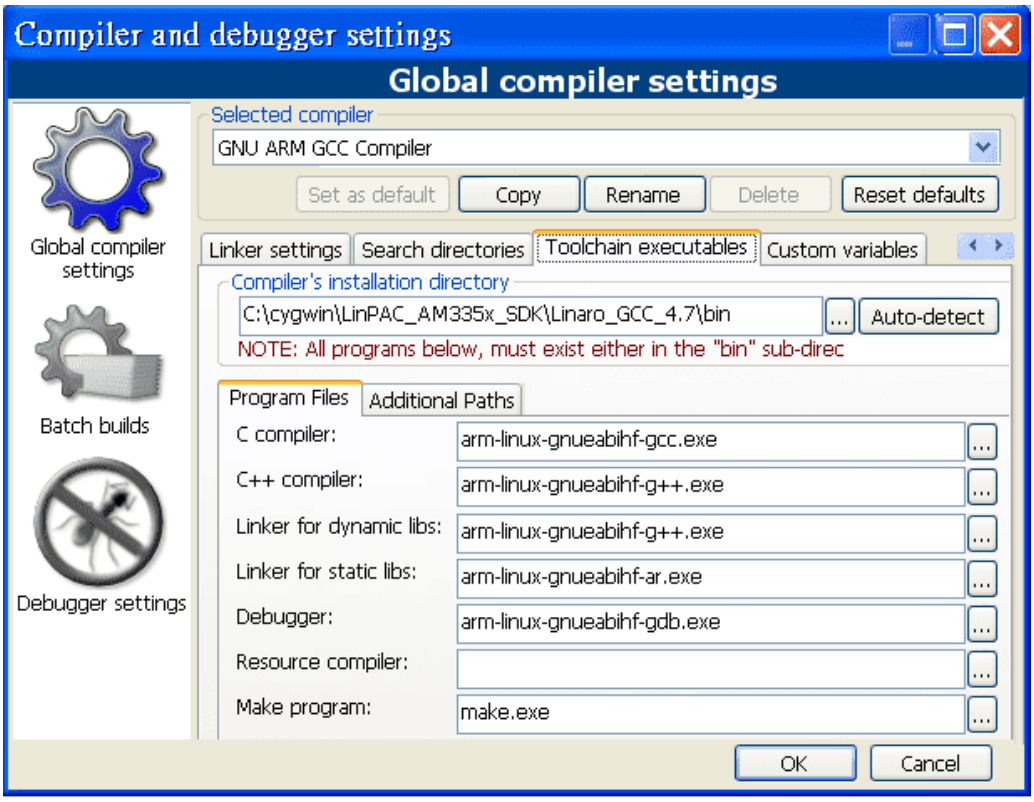


Figure 1.3.2-20. Check compiler settings

➢ Click **Build** options, and it will compile the LinPAC_AM335x project completely. (Refer to Figure 1.3.2-21)

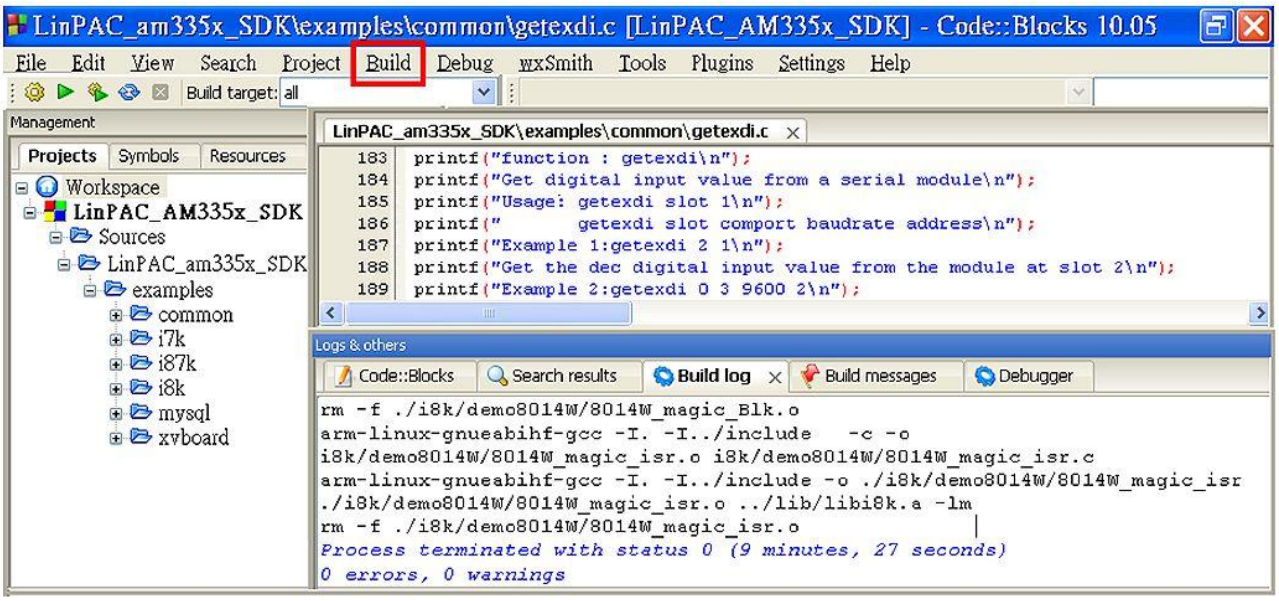


Figure 1.3.2-21. Compiling a C program

【**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 Figure 1.3.2-22)

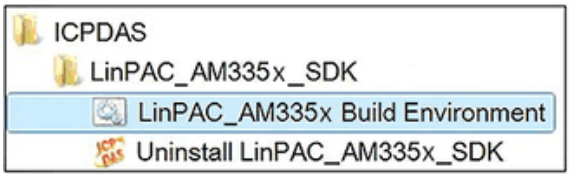


Figure 1.3.2-22. Click the 'LinPAC_AM335x Build Environment'

➢ Type command: **chcp 65001**. (Refer to Figures 1.3.2-23 and 1.3.2-24)

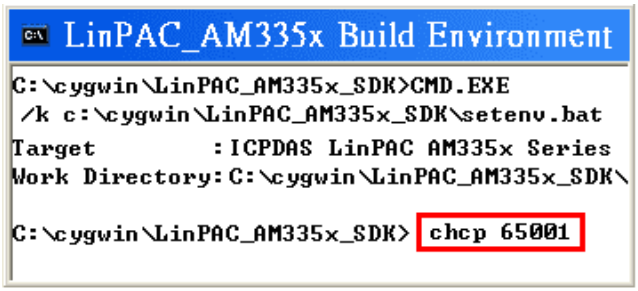


Figure 1.3.2-23.

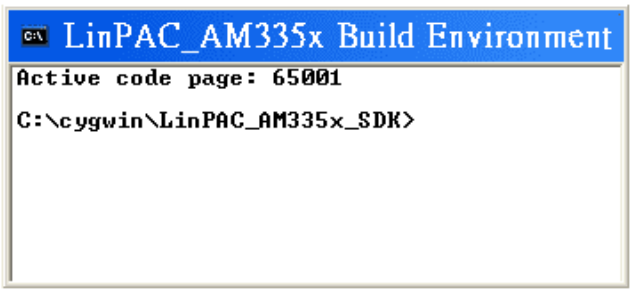


Figure 1.3.2-24.

1.3.3. LinPAC X86/E38xx/iMX8MM Series

❒ Download/Install LinPAC X86/E38xx SDK on Linux

Here is a simple application for using the LX-8000/9000 SDK.

From https://www.icpdas.com/en/download/show.php?num=904&model=LX-9371, you can download the latest version of LX-8000/9000 SDK. And then follows the below steps in order to get the development toolkit which has been provided by ICP DAS for the easy application of the LX-8000/9000 embedded controller platform.

1. User can connect to LX-8000/9000 through communication port (**Console**, **LAN1**, **LAN2)** by using '**putty**' software (refer to "CH2. LX-8000/9000 Getting Started").

2. After connecting to LX-8000/9000, the user could type the following command to get the latest version of LX-8000/9000 SDK.

```
#  wget  https://www.icpdas.com/en/download/file.php?num=1449
```

【**Note**】Please check the network can connect to the ICP DAS official website.

3. To type 'tar zxf LinPAC_X86_SDK.tar.gz' to decompress tar file and type 'make' to compile demo code.

```
root@icpdas:~# tar  zxf  LinPAC_X86_SDK.tgz
root@icpdas:~# ls  LinPAC_X86_SDK
LinPAC_X86_SDK
root@icpdas:~# cd  LinPAC_X86_SDK/
root@icpdas:~/LinPAC_X86_SDK# make
```

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

The libPAC_x86.a path is '**LinPAC_X86_SDK/lib**'.

The include files path is '**LinPAC_X86_SDK/include**'.

The LX-8000/9000 demo path is '**LinPAC_X86_SDK/examples/lx-series**'.

The LP-8x81/8x81-Atom demo path is '**LinPAC_X86_SDK/examples/lp-8x81**'.

2. System Information Functions

Supported LinPACs

The table below lists the common API of system information functions that are supported by each LinPAC. For more details, please refer to the corresponding chapters.

Functions \ Models	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-Series
Open_Slot		✔		✔	✔	✔	✔
Close_Slot		✔		✔	✔	✔	✔
Open_SlotAll		✔		✔	✔	✔	✔
Close_SlotAll		✔		✔	✔	✔	✔
ChangeToSlot		✔		✔	✔	✔	✔
GetModuleType		✔		✔	✔	✔	✔
sio_open	✔	✔	✔	✔	✔	✔	✔
sio_close	✔	✔	✔	✔	✔	✔	✔
sio_set_noncan	✔	✔	✔	✔	✔	✔	✔
Read_SN	✔	✔	✔	✔	✔	✔	✔
Open_Com	✔	✔	✔	✔	✔	✔	✔
Close_Com	✔	✔	✔	✔	✔	✔	✔
Send_Receive_Cmd	✔	✔	✔	✔	✔	✔	✔
Send_Cmd	✔	✔	✔	✔	✔	✔	✔
Receive_Cmd	✔	✔	✔	✔	✔	✔	✔
Send_Binary	✔	✔	✔	✔	✔	✔	✔
Receive_Binary	✔	✔	✔	✔	✔	✔	✔
GetBackPlaneID		✔		✔	✔	✔	✔
GetSDKversion	✔	✔	✔	✔	✔	✔	✔
GetSlotCount				✔	✔	✔	✔
GetModuleName	✔	✔	✔	✔	✔	✔	✔
GetNameOfModule_9K						✔	

Functions \ Models	LP-51xx	LP-8x4x	LP-2x4x	LP-52xx	LP-8x2x	LP-9x2x	LX-Series
GetNameOfModule_xw	✔						
GetDIPswitch		✔			✔		✔
SetLED	✔	✔	Refer to AM335x SDK /examples/led.c,				✔
SetLED_Single			or linpac-am335x_user_manual_en.pdf				✔
GetRotaryID	✔	✔					✔
rotary_switch_read			✔	✔	✔	✔	
Read_SRAM		✔					✔
Write_SRAM		✔					✔
EnableWDT	✔	✔	Refer to AM335x SDK/examples/wdt.c,				✔
DisableWDT	✔	✔	or linpac-am335x_user_manual_en.pdf				✔

Note: LX-Series includes LX-8000 and LX-9000 series.

The table below provides a summary of the various communication functions that can be used depending on the for the different locations of the I/O modules when using the ICP DAS modules in conjunction with the Linux PAC.

API( )	Open_Slot	Close_Slot	Open_Com	Close_Slot	ChangeToSlot	sio_open	sio_close
I-8K I-9K	✔	✔				✔	✔
						RS-422/485 Module	
I-87K I-97K	✔	✔	✔	✔	✔		
I-7K			✔	✔			

Note that the Open_slot()/Close_Slot() and sio_open()/sio_close() functions cannot be used for the same slot.

I/O type \ Function	I-8K or I-9K modules (Parallel Bus)		I-87K or I-97K modules (Serial Bus)	
For example	e.g., I-8050W is plug in Slot number 3.		e.g., I-87054W is plug in Slot number 3.	
Step 1	**Open_Slot( )**	➔ Open_Slot(3);	**Open_Com( )**	➔ Open_Com(1)
Step 2	**Close_Slot( )**	➔ Close_Slot(3);	**Open_Slot(0)**	➔ Open_Slot(0)
Step 3			**ChangeToSlot( )**	➔ ChangeToSlot(3)
Step 4			**Close_Slot(0)**	➔ Close_Slot(0)
Step 5			**Close_Com()**	➔ Close_Com(1)

2.1. Open_Slot

Description:

This function is used to open and initialize a specific slot on the LinPAC, and will be used by modules inserted in the LinPAC. 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: 'Error Code Definitions' for details of other returned values.

Example:

```
int slot=1;
Open_Slot(slot);          // The first slot in the LinPAC will be open and initiated.
```

Remark:

(1) The function can't be applied on PAC: LP-2x4x and LP-52xx.

2.2. Close_Slot

Description:

After using the of Open_Slot() function to open and initialize a specific slot on the LinPAC, the Close_Slot() function must also be used to close the slot. This function will be used modules inserted in the LinPAC. 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 Values:

None

Example:

```
int slot=1;
Close_Slot(slot);          // The first slot in the LinPAC will be closed.
```

Remark:

(1) The function can't be applied on PAC: LP-2x4x and LP-52xx.

2.3. Open_SlotAll

Description:

This function is used to open and initialize all slots on the LinPAC. 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 Values:

0: The slot was successfully initialized.

Other: The initialization failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Example:

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

Remark:

(1) The function can't be applied on PAC: LP-2x4x and LP-52xx.

2.4. Close_SlotAll

Description:

If you the Open_SlotAll() function was used to open and initialize all the slots on the LinPAC, 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 Values:

None

Example:

```
Close_Slot();
// All slots in the LinPAC will be closed.
```

Remark:

(1) The function can't be applied on PAC: LP-2x4x and LP-52xx.

2.5. ChangeToSlot

Description:

This function is used to assign serial control to the specified slots for to allow control of the I-87K/I-97 series. The serial bus on the backplane of the LX-series PAC is used for mapping through to **ttySA0** port, and others are **COM1** port. 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(int slot)
```

Parameter:

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

Return Values:

None

Example:

```
int slot=2;
Open_Slot(0);
Open_COM(COM1);
ChangeToSlot(slot);
Close_Com(COM1);
Close_Slot(0);
```

Remark:

(1) The function can't be applied on PAC: LP-2x4x and LP-52xx.

2.6. GetModuleType

Description:

This function is used to retrieve which type of I/O module is inserted in a specific I/O slot in the LinPAC. 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 Values:

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

(Refer to Figure 2.6-1.)

Example:

```
int slot=1;
int moduleType;
Open_Slot(slot);
printf("GetModuleType= 0x%X \n", GetModuleType(slot));
Close_Slot(slot);
// The I-8057W card is inserted in slot 1 of LP-8x4x and has a
// return Value: 0xC2.
```

Remark:

(1) The function can't be applied on PAC: LP-2x4x and LP-52xx.

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

Figure 2.6-1

2.7. sio_open

Description:

This function is used to open and initiate a specified serial port in the LinPAC. The n-port modules in the LinPAC 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:	**[LP-2x4x/52xx]** [Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5) **[LP-51xx/8x2x/8x4x/9x2x]** [Input] Device name: /dev/ttyS2, /dev/ttyS3…/dev/ttyS34 **[LX-Series]** [Input] ttyS0~ttyS34, ttySA0, ttySA1
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 Values:

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

ERR_PORT_OPEN is for Failure.

Example:

```
#define COM_M1 "/dev/ttyS2"            // Defined the first port of I-8144W in slot 1.
char fd[5];
fd[0]=sio_open(COM_M1, B9600, DATA_BITS_8, NO_PARITY,ONE_STOP_BIT);
if (fd[0]==ERR_PORT_OPEN) {
        printf("open port_m failed!\n");
        return (-1);
}
// The I-8114W is inserted in slot 1 and the first port will be open and initiated.
```

Remark:

(1) The function can be applied for all LinPAC series.

(2) This function can be applied to modules: I-8114W, I-8112iW, I-8142iW, I-8144iW, I-9114i and I-9144i.

(3) More detailed information about device node, user can refer to:
LinPAC_SDK\include\sio.h

2.8. sio_close

Description:

If you have used the function of sio_open() to open the specified serial port in the LinPAC, you need to use the sio_close() function to close the specified serial port in the LinPAC. 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: **[LP-2x4x/52xx]**

[Input] COM1, COM2, COM4, COM5

(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)

[LP-51xx/8x2x/8x4x/9x2x]

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

[LX-Series]

[Input] ttyS0~ttyS34, ttySA0, ttySA1

Return Values:

None

Example:

```
#define COM_M2 "/dev/ttyS3"     // Defined the second port of I-8144iW in slot 1.
char fd[5];
fd[0]=sio_open(COM_M2, B9600, DATA_BITS_8, NO_PARITY,ONE_STOP_BIT);
sio_close (fd[0]);
// The second port of i8144iW in slot 1 will be closed.
```

Remark:

(1) The function can be applied for all LinPAC series.

(2) This function can be applied on COM port modules.

(3) More detailed information about device node, user can refer to:

LinPAC_SDK\include\sio.h

2.9. sio_set_noncan

Description:

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

Syntax:

[C]

```
int sio_set_noncan (int port)
```

Parameter:

port: [Input] Device name: /dev/ttyS0, /dev/ttyS1…/dev/ttyS34

Return Values:

None

Example:

```
#define COM_M2 "/dev/ttyS1"     // Defined the second port for COM3 (RS-232).
char fd[5];
fd[0]=sio_open(COM_M2, B9600, DATA_BITS_8, NO_PARITY,ONE_STOP_BIT);
sio_set_noncan(fd[0]);
sio_close(fd[0]);               // The second port will be closed.
```

Remark:

(1) The function can be applied for all LinPAC series.

2.10. Read_SN

Description:

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

Syntax:

[C]

```
int Read_SN(unsigned char serial_num[])
```

Parameter:

serial_num: [Output] Receive the serial ID number

Return Values:

-6: NOT SUPPORT

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Example:

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

Remark:

(1) The function can be applied for all LinPAC series.

2.11. 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: **[LP-2x4x/52xx]**

[Input] COM1, COM2, COM4, COM5

(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)

[LP-51xx/8x2x/8x4x/9x2x]

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

[LX-Series]

[Input] ttyS0~ttyS34, ttySA0, ttySA1

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 slot was successfully initialized.

Other: The initialization failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Example:

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

Remark:

(1) The function can be applied for all LinPAC series.

(2) The user can refer to the following information that about COM port for LP-2x4x/52xx.

[LP-2x4x/52xx]

Device name	Definition in LP-2x4x/52xx SDK	Description	Default Baud rate
-	/dev/ttyO1 or COM1	Internal communication with the XV-board modules	115200
-	Console port	RS-232 (RxD, TxD and GND); Non-isolated	115200
ttyO4	/dev/ttyO4 or COM4	RS-232 (RxD, TxD and GND);Non-isolated	9600
ttyO2	/dev/ttyO2 or COM2	RS-485 (Data+, Data-); Non-isolated	9600
ttyO5	/dev/ttyO5 or COM5	RS-485 (Data+, Data-); 2500 VDC isolated	9600

2.12. Close_Com

Description:

This function is used to closes and releases the COM port which has been opened. 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: **[LP-2x4x/52xx]**

[Input] COM1, COM2, COM4, COM5

(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)

[LP-51xx/8x2x/8x4x/9x2x]

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

[LX-Series]

[Input] ttyS0~ttyS34, ttySA0, ttySA1

Return Values:

None

Example:

```
Close_Com(COM3);
```

Remark:

(1) The function can be applied for all LinPAC series.

2.13. 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.

The time-measurement between in Linux platform as follows:

Function	Argument	Unit on Linux
Send_Receive_Cmd()	wTimeOut	**0.1 s**
	wT	**1 us**

Syntax:

[C]

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

Parameter:

port: **[LP-2x4x/52xx]**

[Input] COM1, COM2, COM4, COM5

(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)

[LP-51xx/8x2x/8x4x/9x2x]

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

[LX-Series]

[Input] ttyS0~ttyS34, ttySA0, ttySA1

szCmd: [Input] Sending command string

szResult: [Input] Receiving the response string from the modules

wTimeOut: [Input] Communicating timeout setting, the unit=**0.1 s**

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

*wT: [Output] Total time expended in microsecond, the unit=**1 us**

Return Values:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

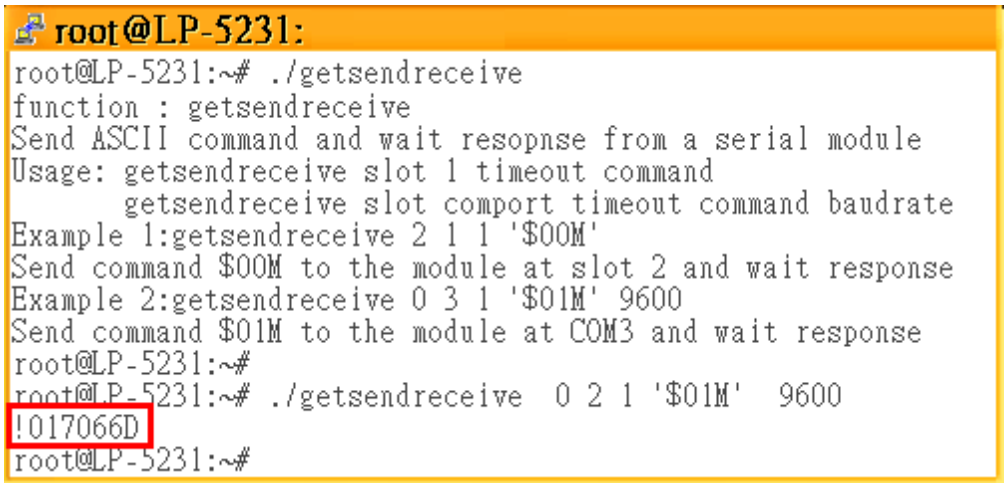
Example:

```
char m_port =1, m_szSend[40], m_szReceive[40];
DWORD m_baudrate=115200;
WORD m_timeout=30;              // the unit=0.1 s
WORD m_chksum=0;
WORD m_wT;                      // the unit=1 us
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:

(1) The function can be applied for all LinPAC series.

(2) For example, user can refer to LP-52xx SDK, locate the ‘getsendreceive.c’ file in the C:\cygwin\LP-52xx_SDK\examples\common\ folder.



(3) If user want to read or write I-87K modules which is pluggid into a specific I/O Slot in LinPAC, the addresses and baudrate in the LinPAC are 00 and 115200 bps reseparately, they were fixed by library.

2.14. 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'.

The time-measurement between in Linux platform as follows:

Function	Argument	Unit on Linux
Send_Cmd()	wTimeOut	**0.1 s**

Syntax:

[C]

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

Parameter:

port: **[LP-2x4x/52xx]**

[Input] COM1, COM2, COM4, COM5

(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)

[LP-51xx/8x2x/8x4x/9x2x]

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

[LX-Series]

[Input] ttyS0~ttyS34, ttySA0, ttySA1

szCmd: [Input] Sending command string

wTimeOut: [Input] Communicating timeout setting, the unit=**0.1 s**

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

Return Values:

None

Example:

```
char m_port=1, m_szSend[40];
DWORD m_baudrate=115200;
WORD m_timeout=50;            // the unit=0.1 s
WORD 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:

(1) The function can be applied for all LinPAC series.

(2) If user want to read or write I-87K modules which is pluggid into a specific I/O Slot in LinPAC, the addresses and baudrate in the LinPAC are 00 and 115200 bps reseparately, they were fixed by library.

2.15. 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].

The time-measurement between in Linux platform as follows:

Function	Argument	Unit on Linux
Receive_Cmd()	wTimeOut	**0.1 s**

Syntax:

[C]

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

Parameter:

port: **[LP-2x4x/52xx]**

[Input] COM1, COM2, COM4, COM5

(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)

[LP-51xx/8x2x/8x4x/9x2x]

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

[LX-Series]

[Input] ttyS0~ttyS34, ttySA0, ttySA1

szResult: [Output] Sending command string

wTimeOut: [Input] Communicating timeout setting, the unit = **0.1 s**

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

Return Values:

None

Example:

```
char m_port=3;
char m_Send[40], m_szResult[40] ;
DWORD m_baudrate=115200;
WORD m_timeout=50;          // the unit = 0.1 s
WORD 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:

(1) The function can be applied for all LinPAC series.

(2) If user want to read or write I-87K modules which is pluggid into a specific I/O Slot in LinPAC, the addresses and baudrate in the LinPAC are 00 and 115200 bps reseparately, they were fixed by library.

2.16. Send_Binary

Description:

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

Syntax:

[C]

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

Parameter:

port: **[LP-2x4x/52xx]**

[Input] COM1, COM2, COM4, COM5

(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)

[LP-51xx/8x2x/8x4x/9x2x]

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

[LX-Series]

[Input] ttyS0~ttyS34, ttySA0, ttySA1

szCmd: [Input] Sending command string

iLen: [Input] The length of command string

Return Values:

None

Example:

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

Remark:

(1) The function can be applied for all LinPAC series.

(2) If user want to read or write I-87K modules which is pluggid into a specific I/O Slot in LinPAC, the addresses and baudrate in the LinPAC are 00 and 115200 bps reseparately, they were fixed by library.

2.17. Receive_Binary

Description:

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

The time-measurement between in Linux platform as follows:

Function	Argument	Unit on Linux
Receive_Binary()	wTimeOut	**0.1 s**
	wT	**1 us**

Syntax:

[C]

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

Parameter:

port: **[LP-2x4x/52xx]**

[Input] COM1, COM2, COM4, COM5

(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)

[LP-51xx/8x2x/8x4x/9x2x]

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

[LX-Series]

[Input] ttyS0~ttyS34, ttySA0, ttySA1

szResult: [Input] Receiving the response string from the modules

wTimeOut: [Input] Communicating timeout setting, the unit=**0.1s**

wLen: [Input] The length of command string

*wT: [Output] Total time expended in microsecond, unit=**1 us**

Return Values:

None

Example:

```
int m_length=10;
char m_port=3, m_szSend[40], m_szReceive[40];
DWORD m_baudrate=115200;
WORD m_wt, m_timeout=10, m_wlength=10;                    // the unit is 0.1 s for m_timeout
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:

(1) The function can be applied for all LinPAC series.

(2) If user want to read or write I-87K modules which is pluggid into a specific I/O Slot in LinPAC, the addresses and baudrate in the LinPAC are 00 and 115200 bps reseparately, they were fixed by library.

2.18. GetBackPlaneID

Description:

This function is used to retrieve the back plane ID number in the LinPAC.

Syntax:

[C]

```
int GetBackPlaneID()
```

Parameter:

None

Return Values:

Backplane ID number.

Example:

```
int id;
id=GetBackPlaneID();
printf("GetBackPlanel =%d \n", id);
// Get the LinPAC backplane id.
```

Remark:

(1) The function can't be applied on PAC: LP-2x4x and LP-52xx.

2.19. GetSDKversion

Description:

This function is used to retrieve the version of LinPAC SDK.

Syntax:

[C]

```
float GetSDKversion(void)
```

Parameter:

None

Return Values:

Version number.

Example:

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

Remark:

(1) The function can be applied for all LinPAC series.

2.20. GetSlotCount

Description:

This function is used to retrieve the number of slot in the LinPAC.

Syntax:

[C]

```
int GetSlotCount();
```

Parameter:

None

Return Values:

[LP-8x2x/8x4x/9x2x]	Number of slot.
[LX-Series]	Number of slot, and add 1.

Example:

```
int number;
number=GetSlotCount();
printf("GetSlotCount=%d \n", number);
// Get the LinPAC-8841/9821 slot count.
// Returned Value: GetSlotCount=8.
```

Remark:

(1) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

(2) Refer to LinPAC_X86_SDK/examples/lx_8k_9k/common/getlist.c for details of LX-Series PAC.

Module / Value	LP-8841	LX-9781
API Return value	8	8
Slot number	8	7

2.21. GetModuleName

Description:

This function is used to retrieve the name of an I/O module, which is plugged into a specific I/O slot in the LP-8000 or LP-9000. This function supports the collection of system hardware configurations.

Syntax:

[C]

```
int GetModuleName (int slot, char* module)
```

Parameter:

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

module: [Output] A pointer to a buffer that receives the name of the specified I/O module.

Return Values:

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

Example:

```
int slot=1;
char module[10];
Open_Slot(slot);
GetModuleName(slot, module);
Close_Slot(slot);
// The I-8017HW module is inserted in slot 1 of LP-8x4x.
// Returned Value: moduleName=' I-8017HW '.
```

Remark:

(1) The function can't be applied on PAC: LP-2x4x and LP-52xx.

2.22. GetNameOfModule_xw

Description:

This function is used to retrieve the name of an XW-Board series I/O module, which is plugged into a slot in the LP-5000. This function supports the collection of system hardware configurations.

Syntax:

[C]

```
int GetNameOfModule_xw()
```

Parameter:

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

Return Value:

I/O module ID. For Example, the XW107 module will return XW107.

Example:

```
int slot;
int moduleID;
Open_Slot(1);
moduleID=GetNameOfModule_xw();
Close_Slot(1);
// The XW107 card plugged in slot 1 of LP-51xx.
// Returned Value: moduleName='XW107'.
```

Remark:

(1) One LP-5000 can only plug only one XW-board.

(2) The function only for applied on PAC: LP-51xx.

2.23. GetBattery

Description:

This function retrieves the battery status of the backplane for LP-2841M.

Syntax:

[C]

```
int GetBattery(unsigned int *value)
```

Parameter:

value: [Output] Specifies

Return Value:

0: No error.

Other: No device, failed.

Example:

```
int ret, value;
ret = GetBattery (&value);
if(value)
        printf("Battery Status = FAIL\n");
else
        printf("Battery Status = OK\n");
```

Remark:

(1) The function only for applied on PAC: LP-2841.

2.24. GetDIPswitch

Description:

This function is used to retrieve the DIP switch value in the LinPAC.

Syntax:

[C]

```
int GetDIPswitch()
```

Parameter:

None

Return Values:

DIP switch value.

Example:

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

Remark:

(1) The functioncan be applied on PAC: LP-8x2x, LP-8x4x and LX-Series.

2.25. SetLED

Description:

This function is used to turn the LinPAC LED's on/off.

(Both LP-8x21 and LX-9x21 series please refer to SDK/examples/common/led.c directly.)

[LP-8x4x Series]

Syntax:

[C]

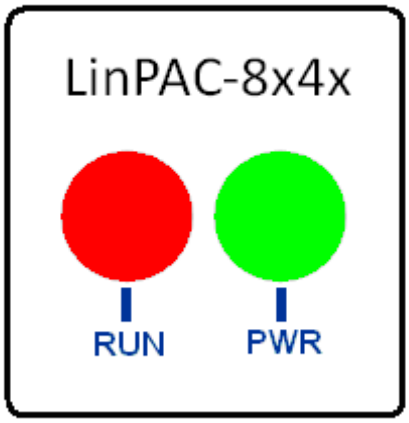
```
void SetLED(unsigned int led)
```

Parameter:

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

0: Turn off the LED

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



Return Values:

None

Example:

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

Remark:

[LP-2000 Series]

Syntax:

[C]

```
void SetLED(unsigned int addr,unsigned int value)
```

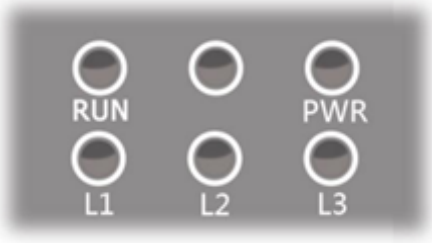
Parameter:

addr: [Input] Range of programmable LED display is 0 to 3

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

0: Turn off the LED

Address	RUN (0)	PWR	L1 (1)	L2 (2)	L3 (3)
Color	Green	Red	Green	Orange	Red
Programmable	Yes	No	Yes	Yes	Yes
Function	Start	Power	None	None	None



Return Values:

None

Example:

```
unsigned int addr,value;
addr=3;
value=1;
SetLED(addr, value);
// Turn on the LED3.
```

Remark:

(1) The function can be applied on PAC: LP-2241 and LP-2841.

[LP-51xx Series]

Syntax:

[C]

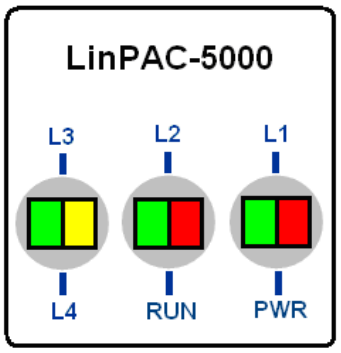
```
void SetLED(unsigned int addr,unsigned int value)
```

Parameter:

addr: [Input] Range of programmable LED display is 1 to 5

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

0: Turn off the LED



Address	L4	L3	L2	RUN (L5)	PWR	L1
Color	Green	Yellow	Green	Red	Green	Red
Programmable	Yes	Yes	Yes	Yes	No	Yes
Function	None	None	None	Start	Power	None

Return Values:

None

Example:

```
unsigned int addr,value;
addr=4;
value=1;
SetLED(addr, value);
// Turn on the LED4.
```

Remark:

[LX-Series]

Syntax:

[C]

```
void SetLED(unsigned int bFlag)
```

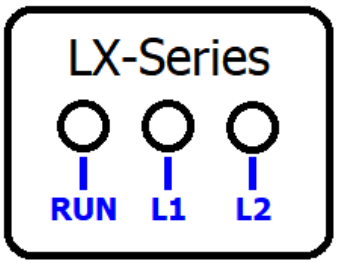
Parameter:

bFlag : [Input] Select one number to control LED status

There are eight options in below:

1. Led 'RUN' ON
2. Led 'L1' ON
3. Led 'RUN' and 'L1' ON
4. Led 'L2' ON
5. Led 'RUN' and 'L2' ON
6. Led 'L1' and 'L2' ON
7. All led ON
8. All led OFF

Options / Address	1	2	3	4	5	6	7	8
RUN	ON	OFF	ON	OFF	ON	OFF	ON	OFF
L1	OFF	ON	ON	OFF	OFF	ON	ON	OFF
L2	OFF	OFF	OFF	ON	ON	ON	ON	OFF



Return Values:

None

Example:

```
unsigned int option;
scanf("%d",&option);
SetLED(option);
```

Remark:

(1) Refer to LinPAC_X86_SDK/examples/lx_8k_9k/common/led.c for more details.

2.26. SetLED_Single

Description:

This function is used to turn on/off a single LED. Support for LX-Series.

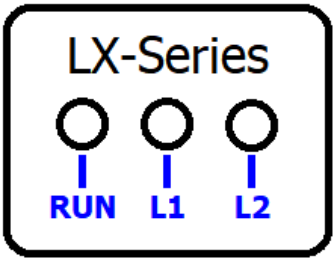
Syntax:

[C]

void SetLED_Single(char led, char status)

Parameter:

led:	[Input]	**0**: RUN
		1: L1
		2: L2
status:	[Input]	**0**: Turn off the led
		1: Turn on the led



Return Values:

None

Example:

```
led = atoi(argv[1]);
status = atoi(argv[2]);
SetLED_Single(led, status);
```

Remark:

(1) The function only for applied on PAC: LX-Series.

(2) Refer to LinPAC_X86_SDK/examples/lx_8k_9k/common/led_single.c for more details.

2.27. GetRotaryID

Description:

This function is used to retrieve the rotary ID number in the LP-51xx, LP-8x4x and LX-Series.

Syntax:

[C]

```
int GetRotaryID(int type, unsigned int * id)        // LP-8x4x and LP-51xx
int GetRotaryID(int slot)                            // LX-Series
```

Parameter:

type:	[Input] Type definition: LP-8x4x is type 1, LP-51xx is type 2
id:	[Output] Rotary switch ID number
slot:	[Input] Slot definition: LX-Series is slot 9

Return Values:

0: The slot was successfully initialized.

Other: The initialization failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Example:

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

Remark:

(1) The function can be applied on PAC: LP-51xx, LP-8x4x and LX-Series.

(2) Refer to LinPAC_X86_SDK/examples/lx_8k_9k/common/**rotary_sw.c** for details of LX-Series PAC.

(3) The following is the rotary ID number table of LP-8x4x and LP-51xx:

[LP-8x4x]

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

[LP-51xx]

SW	0	1	2	3	4	5	6	7	8	9
ID	47	46	45	44	43	42	41	40	39	38

2.28. rotary_switch_read

Description:

This function is used to retrieve the rotary ID number in the LP-2x4x, LP-52xx, LP-8x2x and LP-9x2x.

Syntax:

[C]

```
int rotary_switch_read (unsigned int *value)
```

Parameter:

value: [Output] Rotary switch ID number

Return Values:

0: The slot was successfully initialized.

Other: The initialization failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Example:

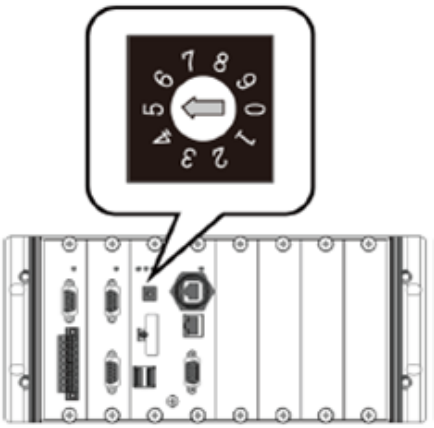
```
int result=0 ;
unsigned char value=0;
rotary_switch_read(&value);
if(result) {
        printf("rsw(%d) : rotary switch read error\n",result);
        return FAILURE;
}
else {
        printf("%d\n", value);          // Get the LP-9x21 rotary id.
}
// If user turn the rotary switch to the 1 position, would get the returned value: 1.
```

Remark:

(1) The function can't be applied on PAC: LP-51xx, LP-8x4x and LX-Series.

(2) The following is the rotary ID number table of LP-2x4x, LP-52xx, LP-8x2x and LP-9x2x:

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



2.29. Read_SRAM

Description:

This function is used to read mram data in LP-8x41 and LX-Series PAC.

Syntax:

[C]

```
unsigned char Read_SRAM(int offset);
```

Parameter:

offset: [input] Get mram offset address value

Return Values:

Mram value of offset address.

Example:

```
int offset=0;          // Read mram address 0 value.
Open_Slot(0);
Read_SRAM(offset);
Close_Slot(0);
```

Remark:

2.30. Write_SRAM

Description:

This function is used to write mram data to LP-8x41 and LX-Series PAC.

Syntax:

[C]

```
void Write_SRAM(int offset, unsigned char data);
```

Parameter:

offset: [input] Mram offset address to write

data: [input] Data you want to write to mram

Return Values:

None

Example:

```
int offset=0;          // Read mram address 0 value.
int data=1;
Open_Slot(0);
Write_SRAM(offset, data&0xff);
Close_Slot(0);
```

Remark:

2.31. EnableWDT

Description:

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

Syntax:

[C]

```
int EnableWDT(unsigned int timer)
```

Parameter:

timer: [Input] LinPAC will reset in the assigned time if users don't reset WDT.

LP-2841: the unit is **second**, other LinPACs is **milliseconds (ms)**.

For LP-2841, each unit is about 4s, unit :'2' is about 8s (range: 1~ 127).

second,mseconds: The unit is mini-second.

Return Values:

0: No Error

-6: NOT SUPPORT

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Example:

```
EnableWDT(10000);                    //Enable WDT interval 10000ms=10s for LP-8421
while (getchar()==10)
{
      printf("Refresh WDT\n");
      EnableWDT(10000);              //Refresh WDT 10s
}
printf("Disable WDT\n");
DisableWDT();
```

Remark:

(1) The function can be applied for all LinPAC series.

2.32. DisableWDT

Description:

This function is used to disable WDT.

Syntax:

[C]

```
int DisableWDT(void)
```

Parameter:

None

Return Values:

0: No Error

-6: NOT SUPPORT

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Example:

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

Remark:

(1) The function can be applied for all LinPAC series.

2.33. WatchDogSWEven

Description:

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

Syntax:

[C]

```
unsigned int WatchDogSWEven (void)
```

Parameter:

None

Return Values:

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

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

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

Bit 2 : WDT Reset (Like : Use "EnableWDT(1000)")

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

Example:

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

Remark:

(1) The function can be applied for LinPAC **PXA270** series.

2.34. ClearWDTSWEven

Description:

This function is used to clear RCSR value.

Syntax:

[C]

```
void ClearWDTSWEven (unsigned int rcsr)
```

Parameter:

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

1 : clear bit 0

2 : clear bit 1

4 : clear bit 2

8 : clear bit 3

F : clear bit 0 to bit 3

Return Values:

None

Example:

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

Remark:

(1) The function can be applied for LinPAC **PXA270** series.

2.35. RefreshWDT

Description:

This function is used to refresh WDT for Linux **AM335x** PAC, refer to user manual for detailed.

Syntax:

[C]

```
int RefreshWDT ()
```

Parameter:

None

Return Values:

None

Example:

```
int main(int argc, char **argv)
{
	int wdt_en = 0, wdt_refresh = 0, c, time_sec = 10;		// default 10 secs
	Open_Slot(SLOT0);						// open device file
	while((c=getopt(argc, argv, "dehrs:")) != -1) {
		switch(c)	{
			case 'd':
				wdt_en = 0;
				break;
			case 'e':
				wdt_en = 1;
				break;
			case 'r':
				RefreshWDT();				//refresh watchdog timer
				Close_Slot(SLOT0);
				return SUCCESS;
		}
	}
}
```

Remark:

The function can be applied for Linux AM335x PAC series, please download **Linux AM335x SDK**.

3. Digital Input/Output Functions

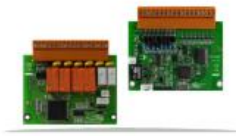
Supported LinPACs

The table below lists the common functions of digital input/output modules that are supported by each LinPAC. For more details, please refer to the corresponding chapters.

I-9K/I-97K Series

I-8K/I-87K Series (High Profile)

I-8K/I-87K Series (Low Profile)



XW-board

❒ I-8000/9000 modules via parallel port

Functions \ Models	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-Series
DO_8				✔	✔	✔	✔
DO_16				✔	✔	✔	✔
DO_32				✔	✔	✔	✔
ReadDI				✔	✔	✔	✔
DI_8				✔	✔	✔	✔
DI_16				✔	✔	✔	✔
DI_32				✔	✔	✔	✔
DIO_DO_8				✔	✔	✔	✔
DIO_DO_16				✔	✔	✔	✔
DIO_DI_8				✔	✔	✔	✔
DIO_DI_16				✔	✔	✔	✔
DO_8_RB DO_16_RB DO_32_RB DIO_DO_8_RB, DIO_DO_16_RB				✔	✔	✔	✔
DO_8_BW DO_16_ BW DO_32_ BW DIO_DO_8_ BW DIO_DO_16_ BW				✔	✔	✔	✔

Functions \ Models	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-Series
DI_8_BW、DI_16_ BW、DI_32_ BW				✔	✔	✔	✔
UDIO_WriteConfig_16				✔	✔	✔	✔
UDIO_ReadConfig_16				✔	✔	✔	✔
UDIO_DO16				✔	✔	✔	✔
UDIO_DI16				✔	✔	✔	✔

Note: LX-Series includes LX-8000 and LX-9000 series.

❒ I-7000 modules via serial port

Functions \ Models	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-Series
DigitalOut	✔	✔	✔	✔	✔	✔	✔
DigitalBitOut	✔	✔	✔	✔	✔	✔	✔
DigitalOutReadBack	✔	✔	✔	✔	✔	✔	✔
DigitalOut_7016	✔	✔	✔	✔	✔	✔	✔
DigitalIn	✔	✔	✔	✔	✔	✔	✔
DigitalInLatch	✔	✔	✔	✔	✔	✔	✔
ClearDigitalInLatch	✔	✔	✔	✔	✔	✔	✔
DigitalInCounterRead	✔	✔	✔	✔	✔	✔	✔
ClearDigitalInCounter	✔	✔	✔	✔	✔	✔	✔
ReadEventCounter	✔	✔	✔	✔	✔	✔	✔
ClearEventCounter	✔	✔	✔	✔	✔	✔	✔

Note: LX-Series includes LX-8000 and LX-9000 series.

❒ I-8000 modules via serial port

Functions \ Models	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-8000
DigitalOut_8K				✔	✔		✔
DigitalBitOut_8K				✔	✔		
DigitalIn_8K				✔	✔		✔
DigitalInCounterRead_8K				✔	✔		
ClearDigitalInCounter_8K				✔	✔		
DigitalInLatch_8K				✔	✔		
ClearDigitalInLatch_8K				✔	✔		

❒ I-9000 modules via serial port

Functions \ Models	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-9000
DigitalOut_9K						✔	✔
DigitalBitOut_9K						✔	
DigitalIn_9K						✔	✔
DigitalInCounterRead_9K						✔	
ClearDigitalInCounter_9K						✔	
DigitalInLatch_9K						✔	
ClearDigitalInLatch_9K						✔	

❒ I-87000 modules via serial port

Functions \ Models	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-8000
DigitalOut_87K				✔	✔		✔
DigitalOutReadBack_87K				✔	✔		
DigitalBitOut_87K				✔	✔		
DigitalIn_87K				✔	✔		✔
DigitalInLatch_87K				✔	✔		
ClearDigitalInLatch_87K				✔	✔		
DigitalInCounterRead_87K				✔	✔		
ClearDigitalInCounter_87K				✔	✔		

3.1. For I-8000/9000 modules via parallel port

3.1.1 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;
Open_Slot(slot);
DO_8(slot, data);
Close_Slot(slot);
// The I-8064W is inserted in slot 1 of LinPAC and can turn on channel 0 and 1.
```

Remark:

(1) This function can be applied on modules: I-8060W, I-8064W, I-8065, I-8066, I-8068W, I-8069W and I-9064.

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.2 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;
Open_Slot(slot);
DO_16(slot, data);
Close_Slot(slot);
// The I-8057W is inserted in slot 1 of LinPAC and can turn on channel 0 and 1.
```

Remark:

(1) This function can be applied on modules: I-8037W, I-8056W, I-8057W, I-8046W and I-9057P.

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.3. 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;
Open_Slot(slot);
DO_32(slot, data);
Close_Slot(slot);
// The I-8041W is inserted in slot 1 of LinPAC and can turn on channel 0 and 1.
```

Remark:

(1) This function can be applied on module: I-8041W and I-9041P.

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.4 ReadDI

Description:

This function is used to obtain input data from a digital input module, supports 8/16/32-bit digital input.

Syntax:

[C]

```
unsigned long ReadDI(int slot)
```

Parameter:

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

Return Value:

Input data.

Examples:

```
int slot=1;
Open_Slot(slot);
printf("Read DI value: %d\n", ReadDI(slot));
Close_Slot(slot);
```

Remark:

(1) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.5 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;
Open_Slot(slot);
data=DI_8(slot);
Close_Slot(slot);
// The I-8058W is inserted in slot 1 of LinPAC and has inputs in channel 0 and 1.
// Returned value: Data=0xfC.
```

Remark:

(1) 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-8058W
2 (Wet contact)	LED On, Readback as 0	LED Off, Readback as 1	I-8048W, I-8052W I-9048

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.6 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;
Open_Slot(slot);
data=DI_16(slot);
Close_Slot(slot);
// The I-8053W is inserted in slot 1 of LinPAC and has inputs in channel 0 and 1.
// Returned value: Data=0xfffC.
```

Remark:

(1) 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-8046W
2 (Wet contact)	LED On, Readback as 0	LED Off, Readback as 1	I-8051W, I-8053W, I-8053PW, I-9053P

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.7 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;
Open_Slot(slot);
data=DI_32(slot);
Close_Slot(slot);
// The I-8040W is inserted in slot 1 of LinPAC and has inputs in channels 0 and 1.
// Returned value: Data=0xfffffffC.
```

Remark:

(1) There is one kind of Input type:

Input Type	On State	Off State	Modules
1 (Wet contact)	LED On, Readback as 0	LED Off, Readback as 1	I-8040W, I-9040P

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.8 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;
Open_Slot(slot);
DIO_DO_8(slot, data);
Close_Slot(slot);
// The I-8054W is inserted in slot 1 of LinPAC and can turn on channels 0 and 1.
// It not only outputs a value, but also shows 16LEDs.
```

Remark:

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

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.9 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;
Open_Slot(slot);
DIO_DO_16(slot, data);
Close_Slot(slot);
// The I-8042W is inserted in slot 1 of LinPAC and can turn on the channels 0 and 1.
// It not only outputs a value, but also shows 32LEDs.
```

Remark:

(1) This function can be applied on modules: I-8042W and I-8050W.

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.10 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 input data, are mapped onto the 0 to 7 input 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;
Open_Slot(slot);
data=DIO_DI_8(slot);
Close_Slot(slot);
// The I-8054W is inserted in slot 1 of LinPAC and has inputs in channel 0 and 1.
// Returned value: Data=0xfC.
```

Remark:

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

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.11 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 input data are mapped onto the 0 to 15 input 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;
Open_Slot(slot);
data=DIO_DI_16(slot);
Close_Slot(slot);
// The I-8042W is inserted in slot 1 of LinPAC and has inputs in channel 0 and 1.
// Returned value: Data=0xfffC.
```

Remark:

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

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.12 DO_8_RB, DO_16_RB, DO_32_RB, DIO_DO_8_RB, DIO_DO_16_RB

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

All DO channel status.

Examples:

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

Remark:

(1) These functions can be applied on modules:

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

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

DO 32 channel: I-8041W and I-9041P.

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.13 DO_8_BW, DO_16_ BW, DO_32_ BW, DIO_DO_8_ BW, DIO_DO_16_ BW

Description:

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

Syntax:

[C]

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

Parameter:

slot:	[Input] Specifies the slot where the I/O module is inserted
bit:	[Input] Channel of module
data:	[Input] Channel status [on: 1 / off : 0]

Return Value:

None

Examples:

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

Remark:

(1) These functions can be applied on modules:

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

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

DO 32 channel: I-8041W and I-9041P.

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.14 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));      // (0: ON, 1: OFF).
Close_Slot(slot);
// The I-8040W module is inserted in slot 1 of LinPAC and return channel 0 status.
```

Remark:

(1) These functions can be applied on modules:

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

DI 16 channel: I-8051W and I-8053W.

DI 32 channel: I-8040W and I-9040P.

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.15 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

data: [Input] Channel status [DO: 1/DI: 0]

Return Value:

None

Examples:

```
int slot=1;
unsigned short config=0xffff;
Open_Slot(slot);
UDIO_WriteConfig_16(slot, config);
Close_Slot(slot);
// WriteConfig: 0xffff (ch 0 to ch15 is DO mode).
```

Remark:

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

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.16 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;
Open_Slot(slot);
UDIO_WriteConfig_16(slot, config);
ret=UDIO_ReadConfig_16(slot);
printf("Read the I/O Type is: 0x%04lx \n\r",ret);
Close_Slot(slot);
// The I-8050W is inserted in slot 1 of LinPAC.
// WriteConfig: 0x0000 (ch 0 to ch15 is DI mode)
// Read the I/O Type is: 0x0000
```

Remark:

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

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.17 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

data: [Input] Output data

Return Value:

None

Examples:

```
int slot=1;                    // The I-8050W is inserted in slot 1 of LinPAC.
unsigned int data;
unsigned short config =0x00ff;
Open_Slot(slot);
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);
Close_Slot(slot);
// 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:

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

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

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

Remark:

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

(2) The function can't be applied on PAC: LP-2x4x, LP-51xx and LP-52xx.

3.1.19 ReadDI_LPF

Description:

This function is used to obtain response value from a Low Pass Filter module.

Syntax:

[C]

short ReadDI_LPF (int slot, DWORD *lpf_Value, int type)

Parameter:

slot:	**[Input]** Specifies the slot where the I/O module is inserted
lpf_Value:	**[Output]** Response time for Low Pass Filter
type:	**[Input]** Low Pass Filter Selection:
	1. Setting by Hardware(0ms/1ms/5ms/10ms/20ms/40ms/70ms by Jumper select)
	2. Setting by Software (Set Jumper position to CPU)

Return Value:

Low Pass Filter data.

Examples:

```
int    RetValue, slot, type;
DWORD read_lpf_Value;
Open_Slot(slot);
RetValue= ReadDI_LPF(slot, &read_lpf_Value, type); // type: 0 --> Hardware, 1 --> Software
printf("ReadDI_LPF Value: Response time = %d ms \n", read_lpf_Value);
Close_Slot(slot);
```

Remark:

(1) This function can be applied on modules: I-9053P.

(2) Here is the result of demo – 9053_lpf.exe

```
root@icpdas:~# ./9053_lpf 1 0 90
ReadDI_LPF Value:
    Response time = 70 ms

If you would like to setup LPF by software, please input <type> parameter = 1
root@icpdas:~#
```

3.1.20 WriteDI_LPF

Description:

This function is used to configure the Low Pass Filter value which range is 0 ~ 100 ms.

Syntax:

[C]

short WriteDI_LPF (int slot, DWORD *lpf_Value)

Parameter:

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

lpf_Value: **[Input]** Output response time for Low Pass Filter (Range: 0~ 100 ms)

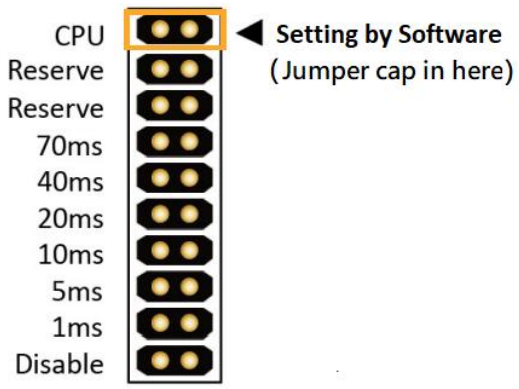
Return Value:

0: The function was successfully processed.

Other: The processing failed. Refer to 'Error Code Definitions' for details of other returned values.

Examples:

```
int    RetValue, slot;
DWORD write_lpf_Value;
Open_Slot(slot);
RetValue=WriteDI_LPF(slot, write_lpf_Value);
printf("WriteDI_LPF = %d ms \n", write_lpf_Value);
Close_Slot(slot);
```



Remark:

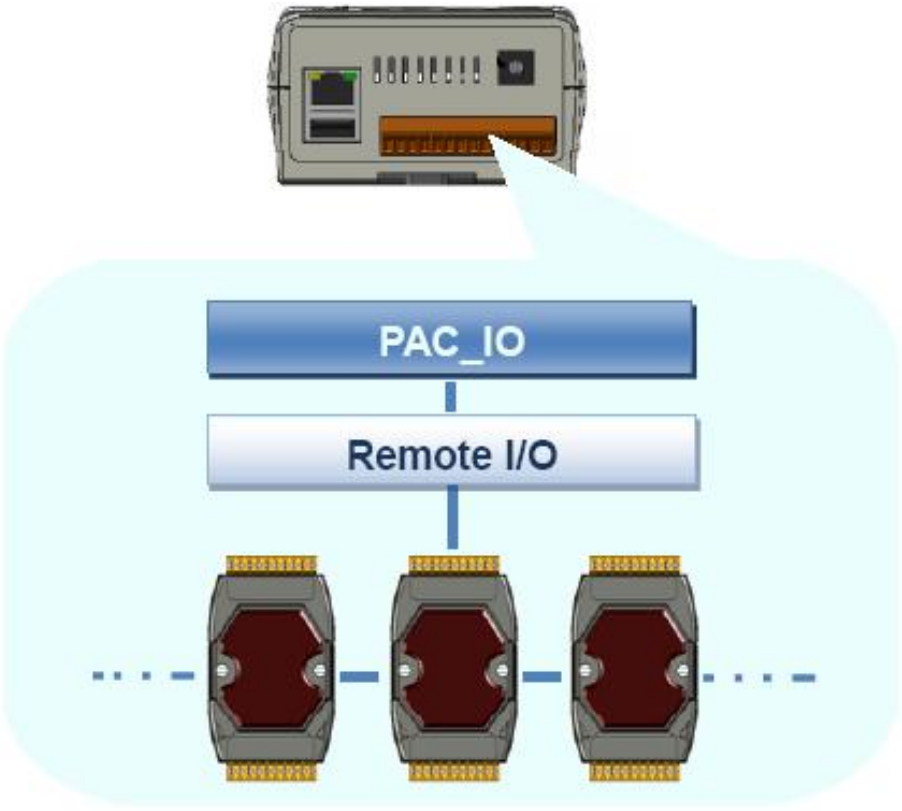
(1) If you would like to setup LPF by software, it is necessary to move the jumper cap into **CPU** position.

(2) This function can be applied on modules: I-9053P.

```
root@icpdas:~# ./9053_lpf 1 1 30
ReadDI_LPF Value:
    Response time = 0 ms
The LPF is Software mode now.
>>> WriteDI_LPF = 30 ms
ReadDI_LPF Value:
    Response time = 30 ms
root@icpdas:~#
root@icpdas:~# ./9053_lpf 1 1 49
ReadDI_LPF Value:
    Response time = 30 ms
The LPF is Software mode now.
>>> WriteDI_LPF = 49 ms
ReadDI_LPF Value:
    Response time = 49 ms
root@icpdas:~#
```

3.2. For I-7000/I-8000/I-9000/I-87000 modules via serial port

3.2.1. 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 table
wBuf[0]:	**[LP-2x4x/52xx]**
	[Input] COM1, COM2, COM4, COM5
	(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
	[LP-51xx/8x2x/8x4x/9x2x]
	[Input] COM port number, from 1 to 255
	[LX-Series]
	[Input] ttyS0~ttyS34, ttySA0, ttySA1
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, the unit = **0.1 s**
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: 'Error Code Definitions' for details of other returned values.

Examples:

```
char szSend[80], szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=3, m_address=1;
WORD m_timeout=50, m_checksum=0;                    // the unit is 0.1 s for m_timeout
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:

(1) The function can be applied for all LinPAC series.

■ 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 table
wBuf[0]:	**[LP-2x4x/52xx]** [Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5) **[LP-51xx/8x2x/8x4x/9x2x]** [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, the unit=**0.1 s**
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: '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=10;                // the unit=0.1 s
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:

(1) The function can be applied for all LinPAC series.

■ 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 table
wBuf[0]:	**[LP-2x4x/52xx]**
	[Input] COM1, COM2, COM4, COM5
	(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
	[LP-51xx/8x2x/8x4x/9x2x]
	[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, the unit=**0.1 s**
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: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=50;                  // the unit=0.1 s
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:

(1) The function can be applied for all LinPAC series.

■ 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 table
wBuf[0]:	**[LP-2x4x/52xx]** [Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5) **[LP-51xx/8x2x/8x4x/9x2x]** [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, the unit=**0.1 s**
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: '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=50;                  // the unit=0.1 s
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:

(1) The function can be applied for all LinPAC series.

■ 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 table
wBuf[0]:	**[LP-2x4x/52xx]**
	[Input] COM1, COM2, COM4, COM5
	(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
	[LP-51xx/8x2x/8x4x/9x2x]
	[Input] COM port number, from 1 to 255
	[LX-Series]
	[Input] ttyS0~ttyS34, ttySA0, ttySA1
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, the unit=**0.1 s**
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: '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=10;                // the unit=0.1 s
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:

(1) The function can be applied for all LinPAC series.

■ DigitalInLatch

Description:

This function is used to obtain the latch value of the high or low latch mode of the I-7000 digital input module.

Syntax:

[C]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	**[LP-2x4x/52xx]** [Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5) **[LP-51xx/8x2x/8x4x/9x2x]** [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, the unit=**0.1 s**
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: '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=10;                // the unit=0.1 s
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:

(1) The function can be applied for all LinPAC series.

■ ClearDigitalInLatch

Description:

This function is used to clear the latch status of I-7000 digital input module when latch function has been enabling.

Syntax:

[C]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	**[LP-2x4x/52xx]** [Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5) **[LP-51xx/8x2x/8x4x/9x2x]** [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, the unit=**0.1 s**
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: '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=20;                  // the unit=0.1 s
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:

(1) The function can be applied for all LinPAC series.

■ DigitalInCounterRead

Description:

This function is used to obtain the counter event value of the channel number of the I-7000 digital input module.

Syntax:

[C]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	**[LP-2x4x/52xx]** [Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5) **[LP-51xx/8x2x/8x4x/9x2x]** [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, the unit=**0.1 s**
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: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=10;                  // the unit=0.1 s
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:

(1) The function can be applied for all LinPAC series.

■ ClearDigitalInCounter

Description:

This function is used to clear the counter value of the channel number of the I-7000 digital input module.

Syntax:

[C]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	**[LP-2x4x/52xx]** [Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5) **[LP-51xx/8x2x/8x4x/9x2x]** [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, the unit=**0.1 s**
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: '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=50;                  // the unit=0.1 s
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:

(1) The function can be applied for all LinPAC series.

■ 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 table
wBuf[0]:	**[LP-2x4x/52xx]** [Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5) **[LP-51xx/8x2x/8x4x/9x2x]** [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, the unit=**0.1 s**
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: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=50;                // the unit=0.1 s
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:

(1) The function can be applied for all LinPAC series.

■ 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 table
wBuf[0]:	**[LP-2x4x/52xx]** [Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5) **[LP-51xx/8x2x/8x4x/9x2x]** [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, the unit=**0.1 s**
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: '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=50;                  // the unit=0.1 s
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:

(1) The function can be applied for all LinPAC series.

3.2.2. I-8000 series modules

■ DigitalOut_8K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	WORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8041/42/54/55/56/57/60/63/64/65/66/68
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] 16-bit digital output data
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=50;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8041;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=10;         // Digital output.
dwBuf[6]=0;
dwBuf[7]=m_slot;
DigitalOut_8K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x, LP-8x4x and LX-8000.

■ DigitalBitOut_8K

Description:

This function is used to set the digital value of the digital output channel No. of I-8000 series modules. The output value is '**0**' or '**1**'.

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8041/42/54/55/56/57/60/63/64/65/66/68
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] 16-bit digital output data
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit
dwBuf[8]:	[Input] The output channel No.
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=10;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8041;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=10;         // Digital output.
dwBuf[6]=0;
dwBuf[7]=m_slot;
dwBuf[8]=3;
DigitalBitOut_8K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ DigitalIn_8K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8040/42/51/52/54/55/58/63
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Output] 16-bit digital output data
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=50;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8040;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=10;         // Digital output.
dwBuf[6]=0;
dwBuf[7]=m_slot;
DigitalIn_8K(dwBuf, fBuf, szSend, szReceive);
DI=dwBuf[5];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x, LP-8x4x and LX-8000.

■ 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 table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8040/51/52/53/54/55/58/63
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] Channel No.
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit
dwBuf[8]:	[Output] DigitalIn counter value
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=50;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8040;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=0;
dwBuf[6]=0;
dwBuf[7]=m_slot;
DigitalInCounterRead_8K(dwBuf, fBuf, szSend, szReceive);
DI_counter=dwBuf[8];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ ClearDigitalInCounter_8K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8040/51/52/53/54/55/58/63
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] Channel No.
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=50;        // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8040;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=0;
dwBuf[6]=0;
dwBuf[7]=m_slot;
ClearDigitalInCounter_8K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ DigitalInLatch_8K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8040/51/52/53/54/55/58/63
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] 0 → select to latch low 1 → select to latch high
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit
dwBuf[8]:	[Output] Latched data
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=50;                    // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8040;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=0;
dwBuf[6]=0;
dwBuf[7]=m_slot;
DigitalInLatch_8K(dwBuf, fBuf, szSend, szReceive);
DI_latch=dwBuf[8];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ 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 table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8040/51/52/53/54/55/58/63
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	Not used
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=50;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8040;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=0;
dwBuf[6]=0;
dwBuf[7]=m_slot;
ClearDigitalInLatch_8K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

3.2.3. I-9000 series modules

■ DigitalOut_9K

Description:

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

Syntax:

[C]

WORD DigitalOut_9K(DWORD dwBuf[],float fBuf[],char szSend[],char szReceive[])

Parameter:

dwBuf:	WORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x9041/57/64
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] 16-bit digital output data
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-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: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=50;              // the unit=0.1 s
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_9K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-9x2x and LX-9000.

■ DigitalBitOut_9K

Description:

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

Syntax:

[C]

WORD DigitalBitOut_9K(DWORD dwBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x9041/57/64
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] 16-bit digital output data
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit
dwBuf[8]:	[Input] The output channel No.
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-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: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=50;                    // the unit=0.1 s
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_9K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

(1) The function only for applied on PAC: LP-9x2x.

■ DigitalIn_9K

Description:

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

Syntax:

[C]

WORD DigitalIn_9K(DWORD dwBuf[],float fBuf[],char szSend[],char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
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
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Output] 16-bit digital output data
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-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: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=50;              // the unit=0.1 s
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_9K(dwBuf, fBuf, szSend, szReceive);
DI=dwBuf[5];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-9x2x and LX-9000.

■ DigitalInCounterRead_9K

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_9K(DWORD dwBuf[],float fBuf[],char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x9040/53
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] Channel No.
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit
dwBuf[8]:	[Output] DigitalIn counter value
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-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: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=50;              // the unit=0.1 s
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_9K(dwBuf, fBuf, szSend, szReceive);
DI_counter=dwBuf[8];
Close_Com(COM3);
```

Remark:

(1) The function only for applied on PAC: LP-9x2x.

■ ClearDigitalInCounter_9K

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_9K(DWORD dwBuf[],float fBuf[],char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x9040/53
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] Channel No.
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-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: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=50;              // the unit=0.1 s
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_9K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

(1) The function only for applied on PAC: LP-9x2x.

■ DigitalInLatch_9K

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_9K(DWORD dwBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x9040/53
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] 0 → select to latch low 1 → select to latch high
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit
dwBuf[8]:	[Output] Latched data
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-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: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=10;              // the unit=0.1 s
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_9K(dwBuf, fBuf, szSend, szReceive);
DI_latch=dwBuf[8];
Close_Com(COM3);
```

Remark:

(1) The function only for applied on PAC: LP-9x2x.

■ ClearDigitalInLatch_9K

Description:

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

Syntax:

[C]

WORD ClearDigitalInLatch_9K(DWORD dwBuf[],float fBuf[],char szSend[],char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x9040/53
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	Not used
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-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: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=10;              // the unit=0.1 s
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_9K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

(1) The function only for applied on PAC: LP-9x2x.

3.2.4. I-87000 series modules

■ DigitalOut_87K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87041/54/55/57/58/63/64/66/68
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] 16-bit digital output data
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=10;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM1, 115200, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87054;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=3;
dwBuf[6]=0;
DigitalOut_87K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM1);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x, LP-8x4x and LX-Series.

■ DigitalOutReadBack_87K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87041/54/55/57/58/63/64/66/68
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Output] 16-bit digital output data
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=50;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM1, 115200, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87054;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[6]=0;
DigitalOutReadBack_87K(dwBuf, fBuf, szSend, szReceive);
DO=dwBuf[5];
Close_Com(COM1);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ DigitalBitOut_87K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87041/54/55/57/58/63/64/66/68
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] 1-bit digital output data
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] The digital output channel No.
dwBuf[8]:	[Input] Data to output (0 or 1)
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=50;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM1, 115200, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87054;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=0;
dwBuf[7]=1;
dwBuf[8]=1;
DigitalBitOut_87K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM1);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ DigitalIn_87K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87040/51/52/53/54/55/58/63
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Output] 16-bit digital input data
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=10;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM1, 115200, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87054;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[6]=0;
DigitalIn_87K(dwBuf, fBuf, szSend, szReceive);
DI=dwBuf[5];
Close_Com(COM1);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x, LP-8x4x and LX-Series.

■ DigitalInLatch_87K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87040/51/52/53/54/55/58/63
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] 0: low latch mode, 1: high latch mode
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Output] Latch value
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=10;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM1, 115200, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87051;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=0;
DigitalInLatch_87K(dwBuf, fBuf, szSend, szReceive);
DI_latch=dwBuf[7];
Close_Com(COM1);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ ClearDigitalInLatch_87K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87040/51/52/53/54/55/58/63
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	Not used
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=10;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM1, 115200, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87051;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[6]=0;
ClearDigitalInLatch_87K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM1);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ DigitalInCounterRead_87K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87040/51/52/53/54/55/58/63
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] The digital input channel No.
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Output] Counter value of the digital input channel No.
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=10;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM1, 115200, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87051;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=0;
DigitalInCounterRead_87K(dwBuf, fBuf, szSend, szReceive);
DI_counter=dwBuf[7];
Close_Com(COM1);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ ClearDigitalInCounter_87K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87040/51/52/53/54/55/58/63
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] The digital input channel No.
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
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=10;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM1,115200, Data8Bit, NonParity, OneStopBit);
dwBuf [0]=m_port;
dwBuf [1]=m_address;
dwBuf [2]=0x87051;
dwBuf [3]=m_checksum;
dwBuf [4]=m_timeout;
dwBuf [5]=1;
dwBuf [6]=0;
ClearDigitalInCounter_87K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM1);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

4. Analog Input Functions

Supported LinPACs

The table below lists the common functions of analog input modules that are supported by each LinPAC. For more details, please refer to the corresponding chapters.

❒ I-8000/9000 modules via parallel port

About special applications of API function for I-8000/9000 modules, please visit to

✓ I-9K Series I/O Module ✓ I-8K Series I/O Modules

Note: For more details about old version I-8017 API, please refer to Appendix C1.

❒ I-7000 modules via serial port

Functions \ Models	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-Series
AnalogIn	✔	✔	✔	✔	✔	✔	✔
AnalogInHex	✔	✔	✔	✔	✔	✔	
AnalogInFsr	✔	✔	✔	✔	✔	✔	
AnalogInAll	✔	✔	✔	✔	✔	✔	
ThermocoupleOpen_7011	✔	✔	✔	✔	✔	✔	
SetLedDisplay	✔	✔	✔	✔	✔	✔	
GetLedDisplay	✔	✔	✔	✔	✔	✔	

Note: LX-Series includes LX-8000 and LX-9000 series.

❒ I-8000 modules via serial port

Functions \ Models	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-8000
AnalogIn_8K				✔	✔		✔
AnalogInHex_8K				✔	✔		
AnalogInFsr_8K				✔	✔		
AnalogInAll_8K				✔	✔		

❒ I-9000 modules via serial port

Models / Functions	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-9000
AnalogIn_9K						✔	✔
AnalogInHex_9K						✔	
AnalogInFsr_9K						✔	
AnalogInAll_9K						✔	

❒ I-87000 modules via serial port

Models / Functions	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-8000
AnalogIn_87K				✔	✔		✔
AnalogInHex_87K				✔	✔		
AnalogInFsr_87K				✔	✔		
AnalogInAll_87K				✔	✔		

❒ I-97000 modules via serial port

Models / Functions	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-9000
AnalogIn_97K						✔	
AnalogInHex_97K						✔	
AnalogInFsr_97K						✔	
AnalogInAll_97K						✔	

4.1. 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 table
wBuf[0]:	**[LP-2x4x/52xx]**
	[Input] COM1, COM2, COM4, COM5
	(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
	[LP-51xx/8x2x/8x4x/9x2x]
	[Input] COM port number, from 1 to 255
	[LX-Series]
	[Input] ttyS0~ttyS34, ttySA0, ttySA1
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, the unit=**0.1 s**
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: ‘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: ‘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=10;                  // the unit=0.1 s
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:

(1) The function can be applied for all LinPAC series.

■ 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 table

wBuf[0]: **[LP-2x4x/52xx]**

[Input] COM1, COM2, COM4, COM5

(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)

[LP-51xx/8x2x/8x4x/9x2x]

[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, the unit=**0.1 s**

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

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

1 → Save to szSend & szReceive

wBuf[7]: [Output] 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: '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=10;                // the unit=0.1 s
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:

(1) The function can't be applied on PAC: LX-Series.

■ 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 table
wBuf[0]:	**[LP-2x4x/52xx]** [Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5) **[LP-51xx/8x2x/8x4x/9x2x]** [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, the unit=**0.1 s**
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: '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=30;                    // the unit=0.1 s
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:

(1) The function can't be applied on PAC: LX-Series.

■ 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 table
wBuf[0]:	**[LP-2x4x/52xx]**
	[Input] COM1, COM2, COM4, COM5
	(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
	[LP-51xx/8x2x/8x4x/9x2x]
	[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, the unit=**0.1 s**
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: 'Error Code Definitions' for details of other returned values.

Examples:

```
float AI[12] , fBuf[12];
char szSend[80], szReceive[80];
WORD wBuf[12], m_port=3 ,m_address=1;
WORD m_timeout=10, m_checksum=0;              // the unit is 0.1 s for m_timeout
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);
```

■ 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 table
wBuf[0]:	**[LP-2x4x/52xx]**
	[Input] COM1, COM2, COM4, COM5
	(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
	[LP-51xx/8x2x/8x4x/9x2x]
	[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, the unit=**0.1 s**
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: '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=50;          // the unit is 0.1 s
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:

(1) The function can't be applied on PAC: LX-Series.

■ 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 table
wBuf[0]:	**[LP-2x4x/52xx]** [Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5) **[LP-51xx/8x2x/8x4x/9x2x]** [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, the unit=**0.1 s**
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: '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=50;                  // the unit=0.1 s
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:

(1) The function can't be applied on PAC: LX-Series.

■ 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 table
wBuf[0]:	**[LP-2x4x/52xx]** [Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5) **[LP-51xx/8x2x/8x4x/9x2x]** [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, the unit=**0.1 s**
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=50;                // the unit=0.1 s
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:

(1) The function can't be applied on PAC: LX-Series.

4.2. I-8000 series modules

■ AnalogIn_8K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8017
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
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] Analog input value
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=50;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8017;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=1;
dwBuf[7]=1;
AnalogIn_8K(dwBuf, fBuf, szSend, szReceive);
AI=fBuf[0];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x, LP-8x4x and LX-8000.

■ AnalogInHex_8K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8017
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting , the unit=**0.1 s**
dwBuf[5]:	[Input] Channel number of analog input module
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number
dwBuf[8]:	[Output] The analog input value in Hex format
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=50;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8017;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=1;
dwBuf[7]=1;
AnalogInHex_8K(dwBuf, fBuf, szSend, szReceive);
AI=dwBuf[8];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ AnalogInFsr_8K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8017
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] Channel number of analog input module
dwBuf[6]:	[Input] 0 → No save to szSend &szReceive
	1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number
fBuf:	Float input/Output argument table
fBuf[0]:	[Output] The analog input value
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=50;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8017;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=1;
dwBuf[7]=1;
AnalogInFsr_8K(dwBuf, fBuf, szSend, szReceive);
AI=fBuf[0];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ AnalogInAll_8K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8017
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number
fBuf:	Float input/Output argument table
fBuf[0]:	[Output] Analog input value of channel 0
fBuf[1]:	[Output] Analog input value of channel 1
fBuf[2]:	[Output] Analog input value of channel 2
fBuf[3]:	[Output] Analog input value of channel 3
fBuf[4]:	[Output] Analog input value of channel 4
fBuf[5]:	[Output] Analog input value of channel 5
fBuf[6]:	[Output] Analog input value of channel 6
fBuf[7]:	[Output] Analog input value of channel 7
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
float AI[12], fBuf[12];
char szSend[80], szReceive[80];
DWORD dwBuf[12], m_port=3, m_address=1;
DWORD,m_timeout=50, m_checksum=0;                    // the unit is 0.1 s for m_timeout
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8017;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[6]=1;
dwBuf[7]=1;
AnalogInAll_8K(dwBuf, fBuf, szSend, szReceive);
AI[0]=fBuf[0];
AI[1]=fBuf[1];
AI[2]=fBuf[2];
AI[3]=fBuf[3];
AI[4]=fBuf[4];
AI[5]=fBuf[5];
AI[6]=fBuf[6];
AI[7]=fBuf[7];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

4.3. I-9000 series modules

■ AnalogIn_9K

Description:

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

Syntax:

[C]

WORD AnalogIn_9K(DWORD dwBuf[],float fBuf[],char szSend[],char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
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, the unit=**0.1 s**
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] 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: '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=30;              // the unit is 0.1 s
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_9K(dwBuf, fBuf, szSend, szReceive);
AI=fBuf[0];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-9x2x and LX-9000.

■ AnalogInHex_9K

Description:

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

Syntax:

[C]

WORD AnalogInHex_9K(DWORD dwBuf[],float fBuf[],char szSend[],char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x9017
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting , the unit=**0.1 s**
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: '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=30;              // the unit is 0.1 s
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_9K(dwBuf, fBuf, szSend, szReceive);
AI=dwBuf[8];
Close_Com(COM3);
```

Remark:

(1) The function only for applied on PAC: LP-9x2x.

■ AnalogInFsr_9K

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_9K(DWORD dwBuf[],float fBuf[],char szSend[],char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
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, the unit=**0.1 s**
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: '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=10;              // the unit is 0.1 s
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_9K(dwBuf, fBuf, szSend, szReceive);
AI=fBuf[0];
Close_Com(COM3);
```

Remark:

(1) The function only for applied on PAC: LP-9x2x.

■ AnalogInAll_9K

Description:

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

Syntax:

[C]

WORD AnalogInAll_9K(DWORD dwBuf[],float fBuf[],char szSend[],char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
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, the unit=**0.1 s**
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: 'Error Code Definitions' for details of other returned values.

Examples:

```
float AI[12], fBuf[12];
char szSend[80], szReceive[80];
DWORD dwBuf[12], m_port=3, m_address=1;
DWORD m_timeout=20, m_checksum=0;          // the unit is 0.1 s for m_timeout
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_9K(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:

(1) The function only for applied on PAC: LP-9x2x.

4.4. I-87000 series modules

■ AnalogIn_87K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87013/15/16/17/18/19
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
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/Output argument table
fBuf[0]:	[Output] The analog input value return
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=20;              // the unit is 0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87017;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=1;
AnalogIn_87K(dwBuf, fBuf, szSend, szReceive);
AI=fBuf[0];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x, LP-8x4x and LX-Series.

■ AnalogInHex_87K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf: DWORD Input/Output argument table

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

dwBuf[1]: [Input] Module address, form 0x00 to 0xff

dwBuf[2]: [Input] Module ID, 0x87013/15/16/17/18/19

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

dwBuf[4]: [Input] Timeout setting, the unit=**0.1 s**

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

dwBuf[6]: [Input] 0 → No save to szSend & szReceive

1 → Save to szSend & szReceive

dwBuf[7]: [Output] The analog input value in 'Hex' format

fBuf: Not used

szSend: [Input] Command string to be sent to I-87000 series modules

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=10;                    // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87017;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=1;
AnalogInHex_87K(dwBuf, fBuf, szSend, szReceive);
AI=dwBuf[8];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ AnalogInFsr_87K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87013/15/16/17/18/19
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
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/Output argument table
fBuf[0]:	[Output] The analog input value
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=50;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87017;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=1;
AnalogInHex_87K(dwBuf, fBuf, szSend, szReceive);
AI=fBuf[0];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ AnalogInAll_87K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87013/15/16/17/18/19
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[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 of channel 0
fBuf[1]:	[Output] Analog input value of channel 1
fBuf[2]:	[Output] Analog input value of channel 2
fBuf[3]:	[Output] Analog input value of channel 3
fBuf[4]:	[Output] Analog input value of channel 4
fBuf[5]:	[Output] Analog input value of channel 5
fBuf[6]:	[Output] Analog input value of channel 6
fBuf[7]:	[Output] Analog input value of channel 7
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

```
float AI[12], fBuf[12];
DWORD AI;
char szSend[80], szReceive[80];
DWORD dwBuf[12], m_port=3, m_address=1;
DWORD m_timeout=50, m_checksum=0;                    // the unit is 0.1 s for m_timeout
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87017;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[6]=1;
AnalogInAll_87K(dwBuf, fBuf, szSend, szReceive);
AI[0]=fBuf[0];
AI[1]=fBuf[1];
AI[2]=fBuf[2];
AI[3]=fBuf[3];
AI[4]=fBuf[4];
AI[5]=fBuf[5];
AI[6]=fBuf[6];
AI[7]=fBuf[7];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

4.5. 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 table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x97015/17/18/19
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
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/Output 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: '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=30;        // the unit is 0.1 s
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:

(1) The function only for applied on PAC: LP-9x2x.

■ 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 table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x97015/17/18/19
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
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: '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=30;              // the unit is 0.1 s
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:

(1) The function only for applied on PAC: LP-9x2x.

■ 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 table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x97015/17/18/19
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
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/Output 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: '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=10;              // the unit is 0.1 s
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:

(1) The function only for applied on PAC: LP-9x2x.

■ 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 table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x97015/17/18/19
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[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 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: 'Error Code Definitions' for details of other returned values.

Examples:

```
float AI[12], fBuf[12];
DWORD AI;
char szSend[80], szReceive[80];
DWORD dwBuf[12], m_port=3, m_address=1;
DWORD m_timeout=50, m_checksum=0;          // the unit is 0.1 s for m_timeout
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:

(1) The function only for applied on PAC: LP-9x2x.

5. Analog Output Functions

Supported LinPACs

The table below lists the common functions of analog output modules that are supported by each LinPAC. For more details, please refer to the corresponding chapters.

❒ I-8000/9000 modules via parallel port

About special applications of API function for I-8000/9000 modules, please visit to

http://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x4x/user_manual/

Note: For more details about old version I-8024 API, please refer to Appendix C2.

❒ I-7000 modules via serial port

Functions \ Models	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-Series
AnalogOut	✔	✔	✔	✔	✔	✔	✔
AnalogOutReadBack	✔	✔	✔	✔	✔	✔	
AnalogOutHex	✔	✔	✔	✔	✔	✔	
AnalogOutFsr	✔	✔	✔	✔	✔	✔	
AnalogOutReadBackHex	✔	✔	✔	✔	✔	✔	
AnalogOutReadBackFsr	✔	✔	✔	✔	✔	✔	

Note: LX-Series includes LX-8000 and LX-9000 series.

❒ I-8000 modules via serial port

Functions \ Models	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-8000
AnalogOut_8K				✔	✔		✔
AnalogOutReadBack_8K				✔	✔		
ReadConfigurationStatus_8K				✔	✔		
SetStartUpValue_8K				✔	✔		
ReadStartUpValue_8K				✔	✔		

❒ I-9000 modules via serial port

Functions \ Models	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-9000
AnalogOut_9K						✔	✔
AnalogOutReadBack_9K						✔	
ReadConfigurationStatus_9K						✔	
SetStartUpValue_9K						✔	
ReadStartUpValue_9K						✔	

❒ I-87000 modules via serial port

Functions \ Models	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-8000
AnalogOut_87K				✔	✔		✔
AnalogOutReadBack_87K				✔	✔		
ReadConfigurationStatus_87K				✔	✔		
SetStartUpValue_87K				✔	✔		
ReadStartUpValue_87K				✔	✔		

❒ I-97000 modules via serial port

Functions \ Models	LP-2x4x	LP-51xx	LP-52xx	LP-8x2x	LP-8x4x	LP-9x2x	LX-9000
AnalogOut_97K						✔	✔
AnalogOutReadBack_97K						✔	
ReadConfigurationStatus_97K						✔	
SetStartUpValue_97K						✔	
ReadStartUpValue_97K						✔	

5.1. 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 table
wBuf[0]:	**[LP-2x4x/52xx]** [Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5) **[LP-51xx/8x2x/8x4x/9x2x]** [Input] COM port number, from 1 to 255 **[LX-Series]** [Input] ttyS0~ttyS34, ttySA0, ttySA1
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, n the unit=**0.1 s**
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/Output 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: '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=50;                // the unit is 0.1 s
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:

(1) The function can be applied for all LinPAC series.

■ 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 table

wBuf[0]: **[LP-2x4x/52xx]**

[Input] COM1, COM2, COM4, COM5

(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)

[LP-51xx/8x2x/8x4x/9x2x]

[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, the unit=**0.1 s**

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/Output 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: 'Error Code Definitions' for details of other returned values.

Examples:

```
float Volt, fBuf[12];
char szSend[80], szReceive[80];
WORD wBuf[12], m_port=3, m_address=1;
WORD m_timeout=50, m_checksum=0;                    // the unit=0.1 s
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:

(1) The function can't be applied on PAC: LX-Series.

■ 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 table
wBuf[0]:	**[LP-2x4x/52xx]**
	[Input] COM1, COM2, COM4, COM5
	(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
	[LP-51xx/8x2x/8x4x/9x2x]
	[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, the unit=**0.1 s**
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: '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=30;                // the unit=0.1 s
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:

(1) The function can't be applied on PAC: LX-Series.

■ 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 table
wBuf[0]:	**[LP-2x4x/52xx]** [Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5) **[LP-51xx/8x2x/8x4x/9x2x]** [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, the unit=**0.1 s**
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: '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=50;          // the unit=0.1 s
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:

(1) The function can't be applied on PAC: LX-Series.

■ 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 table
wBuf[0]:	**[LP-2x4x/52xx]**
	[Input] COM1, COM2, COM4, COM5
	(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
	[LP-51xx/8x2x/8x4x/9x2x]
	[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, the unit=**0.1 s**
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: 'Error Code Definitions' for details of other returned values.

Examples:

```
WORD Volt;
float fBuf[12];
char szSend[80], szReceive[80];
WORD wBuf[12], m_port=3, m_address=1;
WORD m_timeout=50, m_checksum=0;                    // the unit is 0.1 s for m_timeout
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:

(1) The function can't be applied on PAC: LX-Series.

■ AnalogOutReadBackFsr

Description:

This function is used to obtain read back the analog value of analog output modules through % 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 table
wBuf[0]:	**[LP-2x4x/52xx]** [Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5) **[LP-51xx/8x2x/8x4x/9x2x]** [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, the unit=**0.1 s**
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: 'Error Code Definitions' for details of other returned values.

Examples:

```
float Volt, fBuf[12];
char szSend[80], szReceive[80];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=50;                    // the unit=0.1 s
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:

(1) The function can't be applied on PAC: LX-Series.

5.2. I-8000 series modules

■ AnalogOut_8K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8024
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
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]:	[Input] Analog output value
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=50;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8024;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=1;
dwBuf[7]=1;
fBuf[0]=2.55;
AnalogOut_8K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x, LP-8x4x and LX-8000.

■ AnalogOutReadBack_8K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8024
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
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]:	[Input] Analog output value
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=50;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8024;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=1;
dwBuf[7]=1;
AnalogOutReadBack_8K(dwBuf, fBuf, szSend, szReceive);
Volt=fBuf[0];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ ReadConfigurationStatus_8K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8024
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] The defined analog output channel No.
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number
dwBuf[8]:	[Output] Output range: 0x30, 0x31,0x32
dwBuf[9]:	[Output] Slew rate
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=50;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8024;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=1;
dwBuf[7]=1;
ReadConfigurationStatus_8K(dwBuf, fBuf, szSend, szReceive);
Status=dwBuf[8];
Rate=dwBuf[9];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ SetStartUpValue_8K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8024
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] The defined analog output channel No.
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=50;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8024;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=1;
dwBuf[7]=1;
SetStartUpValue_8K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ ReadStartUpValue_8K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x8024
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] The defined analog output channel No.
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number
fBuf:	Float input/output argument table
fBuf[0]:	[Output] Start-Up value
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=50;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x8024;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=1;
dwBuf[7]=1;
ReadStartUpValue_8K(dwBuf, fBuf, szSend, szReceive);
StartUp=fBuf[0];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

5.3. 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_9K(DWORD dwBuf[],float fBuf[],char szSend[],char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
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, the unit=**0.1 s**
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]:	[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: '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=50;              // the unit=0.1 s
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_9K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-9x2x and LX-9000.

■ 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_9K(DWORD dwBuf[],float fBuf[],char szSend[],
char szReceive[]

Parameter:

dwBuf:	DWORD Input/Output argument table
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, the unit=**0.1 s**
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]:	[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: '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=30;              // the unit=0.1 s
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_9K(dwBuf, fBuf, szSend, szReceive);
Volt=fBuf[0];
Close_Com(COM3);
```

Remark:

(1) The function only for applied on PAC: LP-9x2x.

■ ReadConfigurationStatus_9K

Description:

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

Syntax:

[C]

WORD ReadConfigurationStatus_9K(DWORD dwBuf[],float fBuf[],char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
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, the unit=**0.1 s**
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: '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=30;              // the unit=0.1 s
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_9K(dwBuf, fBuf, szSend, szReceive);
Status=dwBuf[8];
Rate=dwBuf[9];
Close_Com(COM3);
```

Remark:

(1) The function only for applied on PAC: LP-9x2x.

■ 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_9K(DWORD dwBuf[],float fBuf[],char szSend[],char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
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, the unit=**0.1 s**
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: '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=10;              // the unit=0.1 s
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_9K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

(1) The function only for applied on PAC: LP-9x2x.

■ ReadStartUpValue_9K

Description:

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

Syntax:

[C]

```
WORD ReadStartUpValue_9K(DWORD dwBuf[],float fBuf[],char szSend[],char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument table
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, the unit=**0.1 s**
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: '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=30;                    // the unit=0.1 s
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_9K(dwBuf, fBuf, szSend, szReceive);
StartUp=fBuf[0];
Close_Com(COM3);
```

Remark:

(1) The function only for applied on PAC: LP-9x2x.

5.4. I-87000 series modules

■ AnalogOut_87K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87024
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
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/Output argument table
fBuf[0]:	[Output] The analog output value
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=50;                    // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87024;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=1;
fBuf[0]=2.55;          // +2.55V.
AnalogOut_87K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x, LP-8x4x and LX-8000.

■ AnalogOutReadBack_87K

Description:

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

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87024
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
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/Output argument table
fBuf[0]:	[Output] Analog output read back value
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=30;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87024;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=1;
AnalogOutReadBack_87K(dwBuf, fBuf, szSend, szReceive);
Volt=fBuf[0];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ ReadConfigurationStatus_87K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87024
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting , the unit=**0.1 s**
dwBuf[5]:	[Input] The defined analog output channel No.
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number
dwBuf[8]:	[Output] Output range: 0x30, 0x31,0x32
dwBuf[9]:	[Output] Slew rate
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=30;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87024;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=1;
dwBuf[7]=1;
ReadConfigurationStatus_87K(dwBuf, fBuf, szSend, szReceive);
Status=dwBuf[8];
Rate=dwBuf[9];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ SetStartUpValue_87K

Description:

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

Syntax:

[C]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87024
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] The defined analog output channel No.
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=10;                    // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87024;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=1;
dwBuf[6]=1;
dwBuf[7]=1;
SetStartUpValue_87K(dwBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

■ ReadStartUpValue_87K

Description:

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

Syntax:

[C]

WORD ReadStartUpValue_87K(DWORD dwBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, from 1 to 255
dwBuf[1]:	[Input] Module address, form 0x00 to 0xff
dwBuf[2]:	[Input] Module ID, 0x87024
dwBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
dwBuf[4]:	[Input] Timeout setting, the unit=**0.1 s**
dwBuf[5]:	[Input] The defined analog output channel No.
dwBuf[6]:	[Input] 0 → No save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number
fBuf:	Float input/output argument table
fBuf[0]:	Start-Up value
szSend:	[Input] Command string to be sent to I-87000 series modules
szReceive:	[Output] Result string receiving from I-87000 series modules

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: '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=30;              // the unit=0.1 s
DWORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
dwBuf[0]=m_port;
dwBuf[1]=m_address;
dwBuf[2]=0x87024;
dwBuf[3]=m_checksum;
dwBuf[4]=m_timeout;
dwBuf[5]=0;
dwBuf[6]=1;
dwBuf[7]=1;
ReadStartUpValue_87K(dwBuf, fBuf, szSend, szReceive);
StartUp=fBuf[0];
Close_Com(COM3);
```

Remark:

(1) The function can be applied on PAC: LP-8x2x and LP-8x4x.

5.5. 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 table
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, the unit=**0.1 s**
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/Output 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: '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=30;              // the unit=0.1 s
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:

(1) The function can be applied on PAC: LP-9x2x and LX-9000.

■ 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 table
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, the unit=**0.1 s**
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/Output argument table
fBuf[0]:	[Output] 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: '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=30;              // the unit=0.1 s
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:

(1) The function only for applied on PAC: LP-9x2x.

■ 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 table
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, the unit=**0.1 s**
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: '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=30;              // the unit=0.1 s
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:

(1) The function only for applied on PAC: LP-9x2x.

■ 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 table
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, the unit=**0.1 s**
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: '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=30;              // the unit=0.1 s
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:

(1) The function only for applied on PAC: LP-9x2x.

■ ReadStartUpValue_97K

Description:

This function is used to setting start-up value of analog output module for I-97000 series modules.

Syntax:

[C]

WORD ReadStartUpValue_97K(DWORD dwBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
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, the unit=**0.1 s**
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: '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=30;              // the unit=0.1 s
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:

(1) The function only for applied on PAC: LP-9x2x.

6. Error Code Explanation

Error Code	Explanation	Error Code	Explanation
0	NoError	-1	ID_ERROR
1	FunctionError	-2	SLOT_ERROR
2	PortError	-3	CHANNEL_ERROR
3	BaudrateError	-4	HARDWARE_LPF_ERROR
4	DataError	-5	SOFTWARE_LPF_ERROR
5	StopError	-6	NOT_SUPPORT_ERROR
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 I/O Modules using C Language

In this section, we will focus on examples for the description and application of the control functions on the I-7k/I-8k/I-9k/I-87k/I-97k series modules for use with the LinPAC. For Windows platform of the PXA270 series, after installing the LinPAC SDK, the demo programs provided below can be found in the **'C:/cygwin/LinCon8k/examples'** folder in Windows PC.

7.1. DI/DO Control Demo

7.1.1. 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 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 , the unit=0.1s
    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 LinPAC Build Environment by clicking the Start > Programs > ICPDAS > LinPAC SDK > LinPAC Build Environment to open **LinPAC 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 Figure 7.1.1-1).

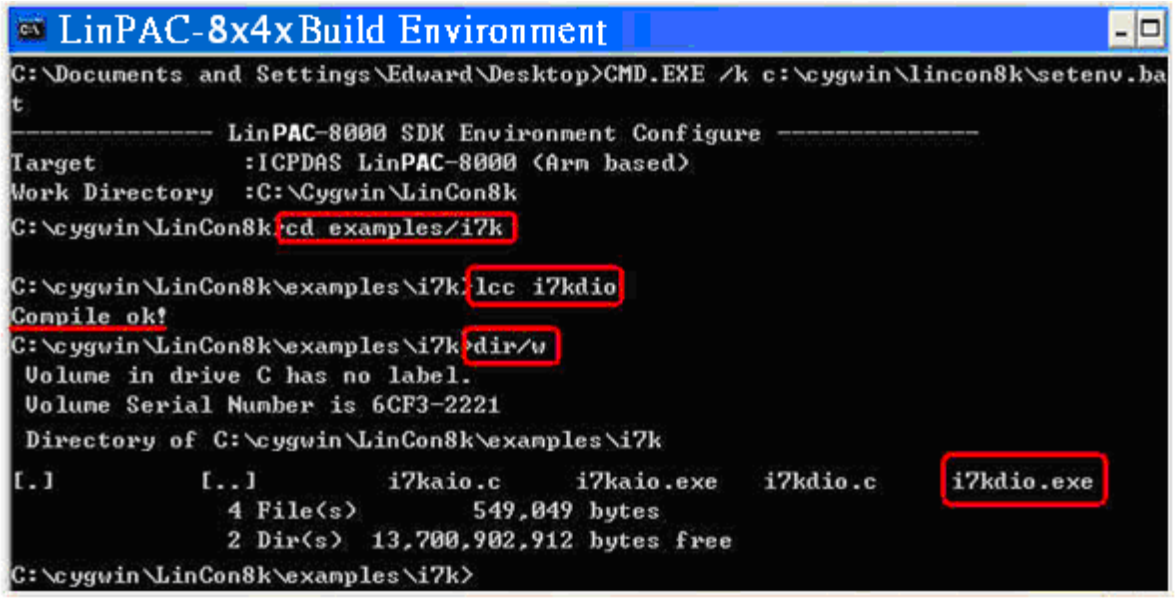


Figure 7.1.1-1. Using a Batch File to compile i7kdio.c to an executable file

Method Two – Using Compile Instructions

When using this method, type cd C:\cygwin\LinCon8k\examples\i7k command prompt to change the path. To compile i7kdio.c to an executable file, type arm-linux-gcc –I../../include –lm –o i7kdio.exe i7kdio.c ../../lib/libi8k.a (refer to Figure 7.1.1-2).

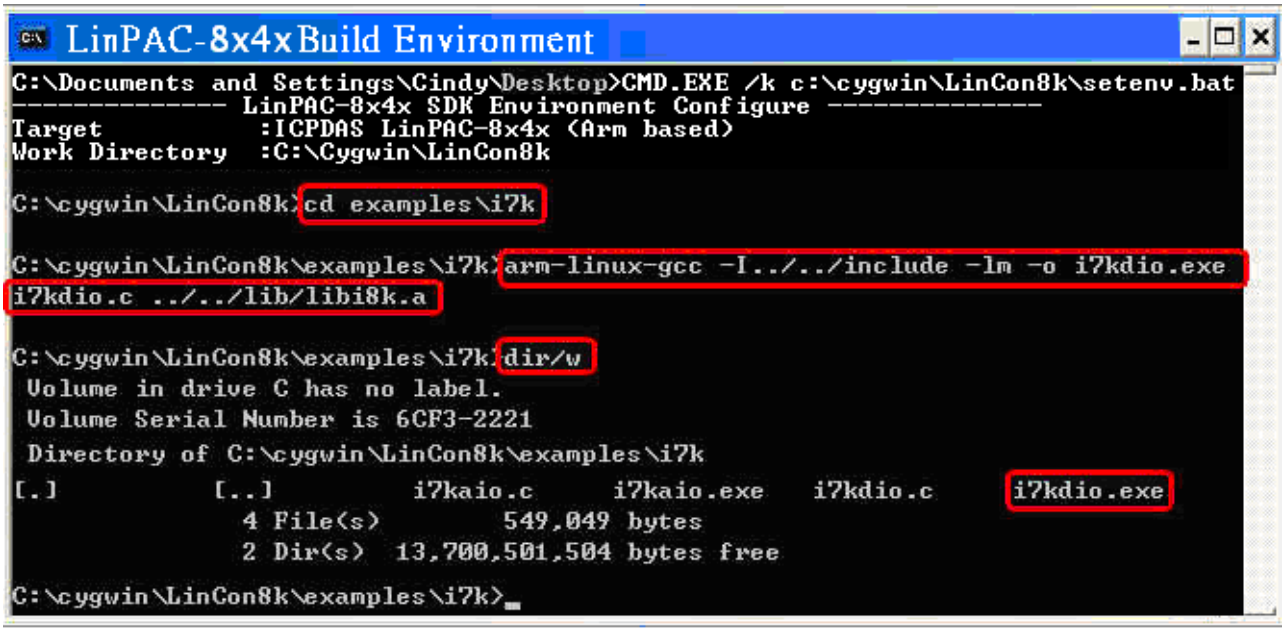


Figure 7.1.1-2. Using Compile Instructions to compile i7kdio.c to an executable file

STEP 3: **Transfer i7kdio.exe to the LinPAC**

Two methods can be used to transfer the executable file to the LinPAC, each of which is introduced here.

Method One – Using an FTP application

(1) Open a FTP application and create a new FTP connection. Enter the login details for the LinPAC, including the Host name (or IP address), Username and Password. The default value for both the **User_Name** and the **Password** is **'root'**. Click the **'Quickconnect'** button to connect to the ftp server on the LinPAC. Refer to Figure 7.1.1-3 below for more details.

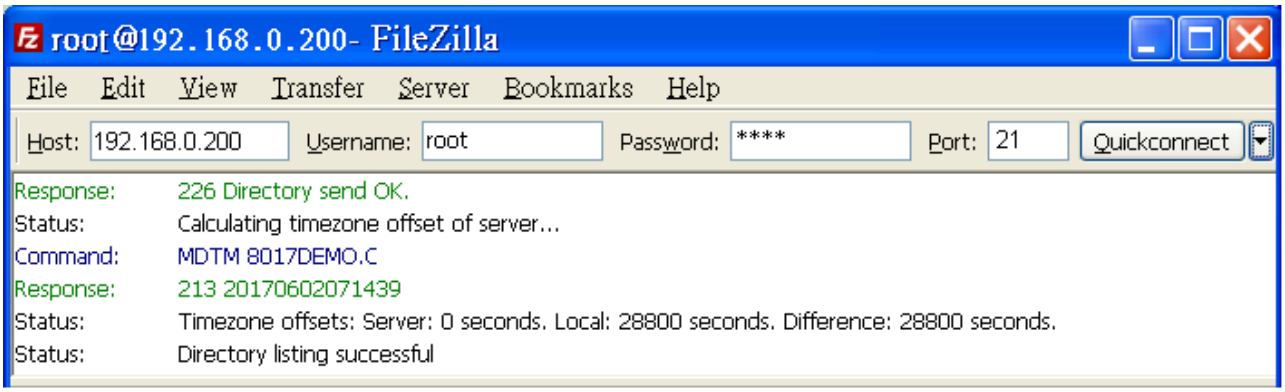


Figure 7.1.1-3. Using an FTP application

(2) Upload the file **i7kdio.exe** file to the LinPAC (refer to Figure 7.1.1-4).

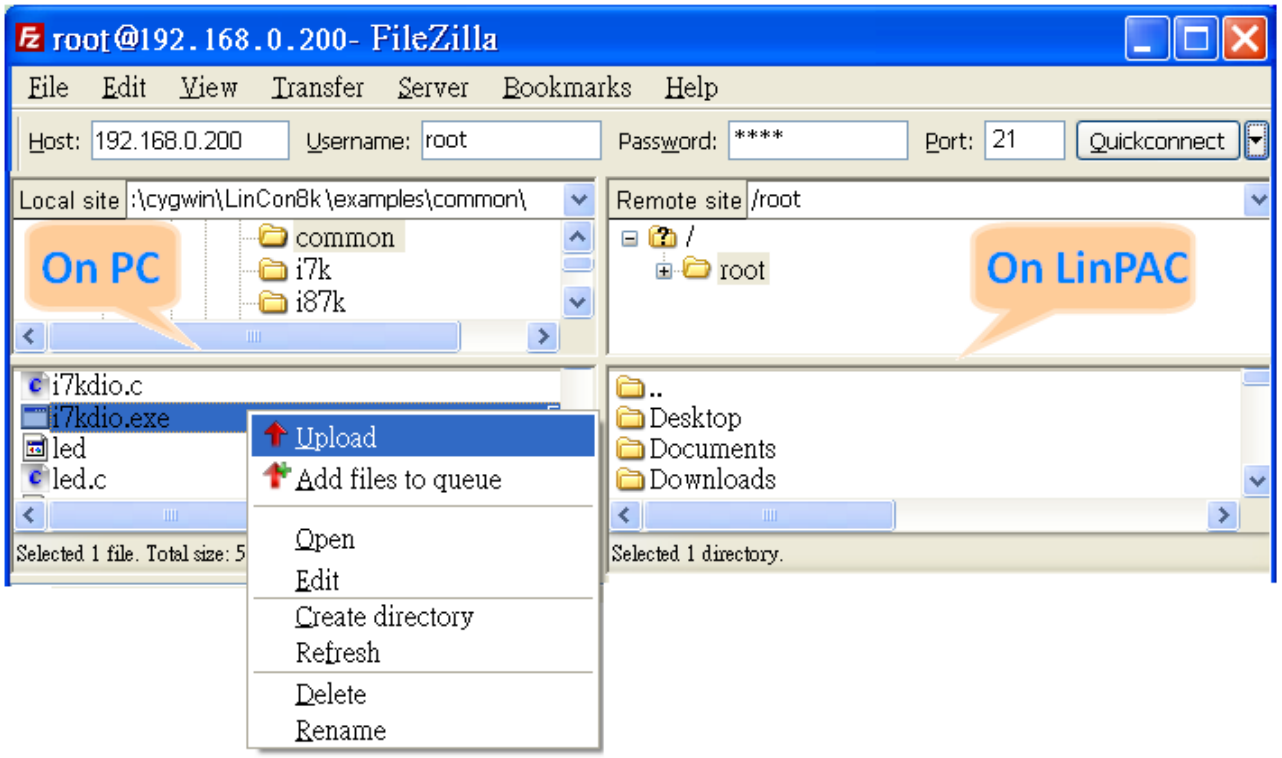


Figure 7.1.1-4. Upload the i7kdio.exe file

(3) Choose i7kdio.exe in the LinPAC 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 Figures 7.1.1-5 and 7.1.1-6 below for more details.

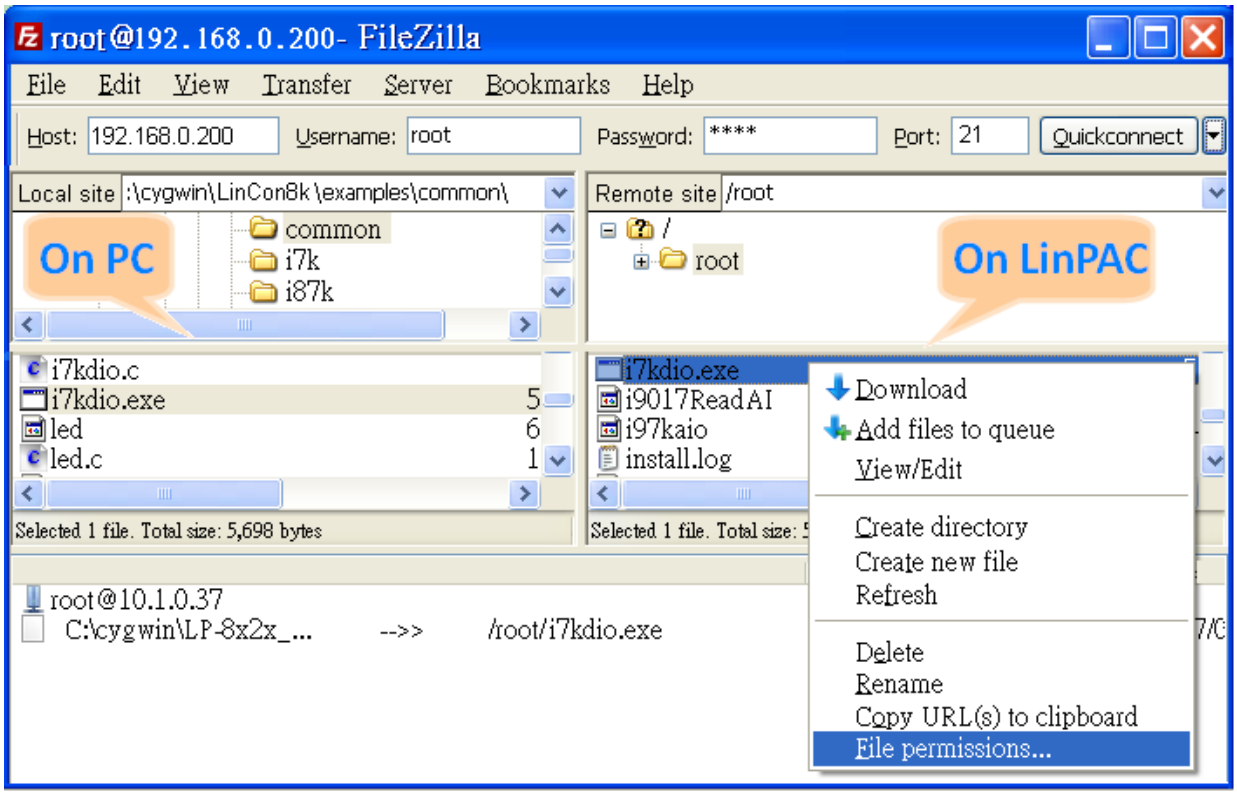


Figure 7.1.1-5. Set the file permissions

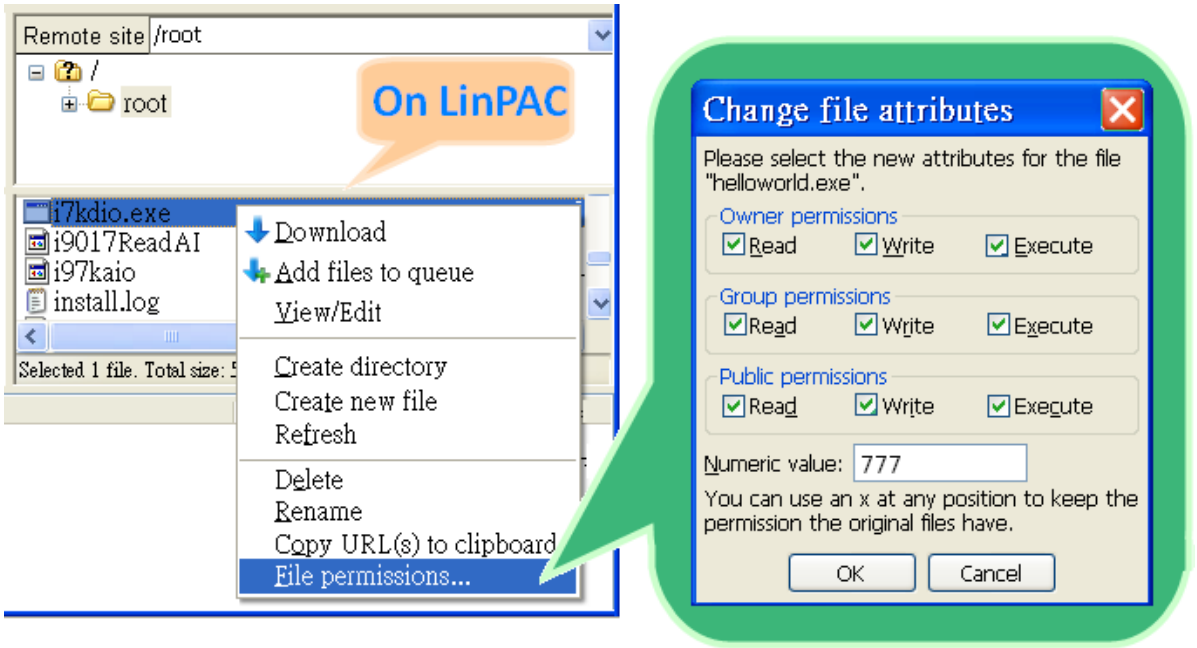


Figure 7.1.1-6. Enter '777' in the Numeric textbox

Method Two – Using a DOS Command Prompt

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

(2) 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 LinPAC. Once the file has been transferred, the 'Transfer complete' message will be displayed. Refer to Figure 7.1.1-7 below for more details.

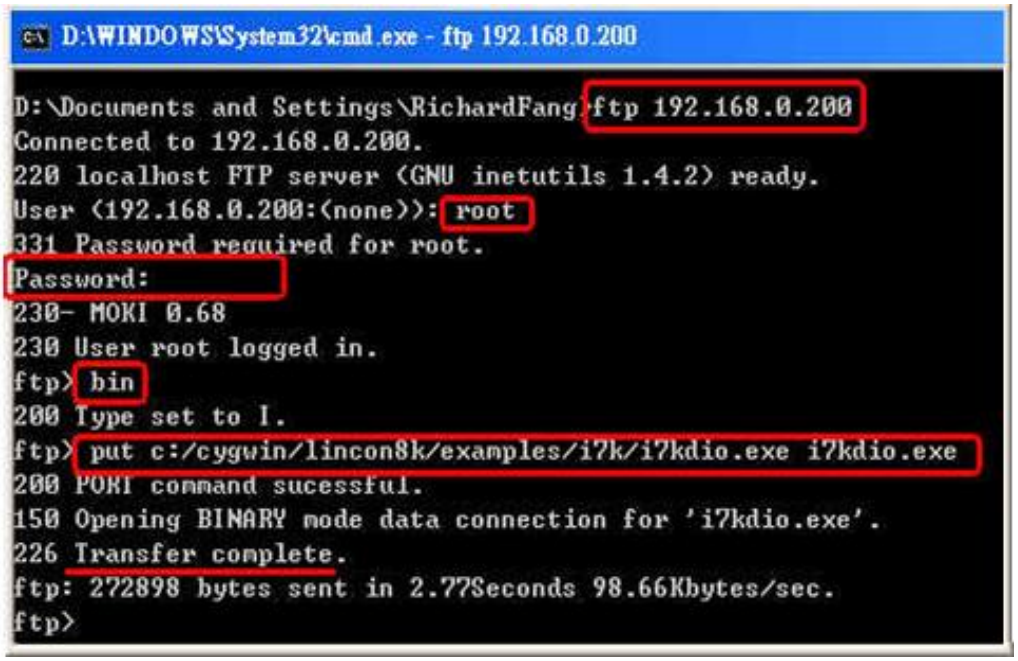


Figure 7.1.1-7. Using a DOS Command to transferred file

STEP 4: **Use Telnet to the LinPAC to execute i7kdio.exe**

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

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 Figures 7.1.1-8 and 7.1.1-9 below for more details.

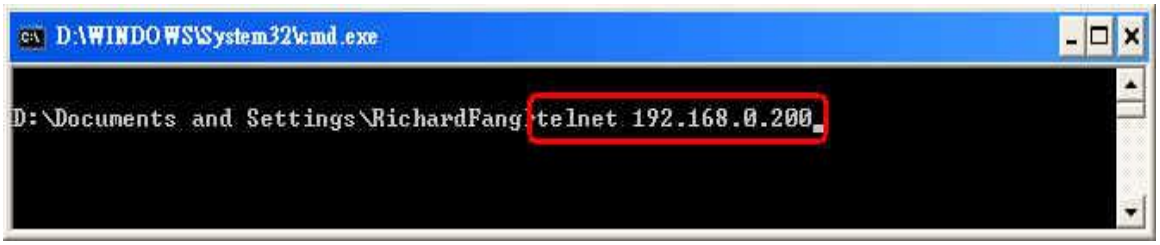


Figure 7.1.1-8. Type telnet IP Address and to establish a connection with the LinPAC

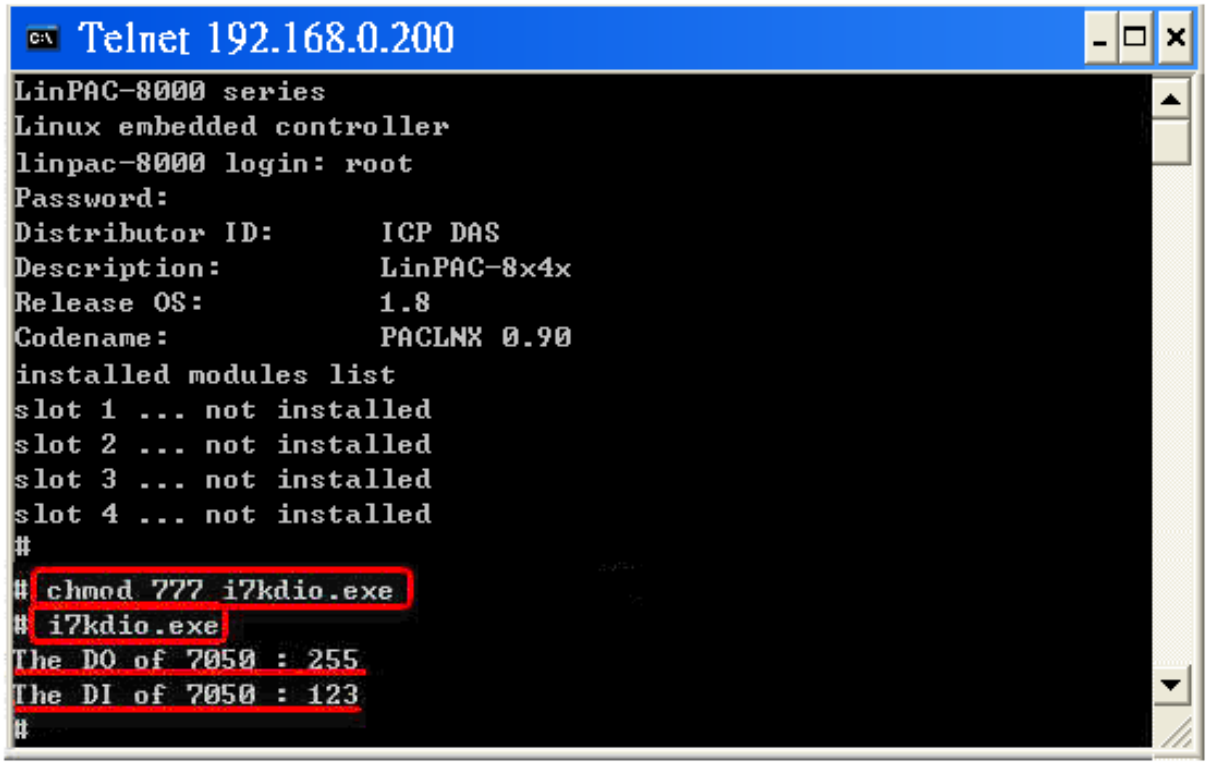


Figure 7.1.1-9. Execute the i7kdio.exe file

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.

7.1.2. I-87K Modules

If there are I-87KW DIO modules inserted in the slots on the LP-8000, the '**Open_Slot()**' and '**ChangeToSlot()**' functions, must be called before other functions for the I-87KW modules and used, and the '**Close_Slot()**' function also needs to be called at the end of the program.

The **i87kdio.c** demo program will illustrate how to control the DI/DO function using an I-87054W module (8 DO channels and 8 DI channels). The module is in slot 3 on the LP-8000. **The address and baudrate in the LP-8000 are 00 and 115200 respectively**, they were fixed by the library. The result of this demo program is that DO channels 0 to 7 on the I-87054W module will be set as the output channels, and DI channel 1 on the I-87054W module will be set as the input channel. The source code for this demo program is as follows:

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

char szSend[80], szReceive[80];
DWORD dwBuf[12];
float fBuf[12];
int main()
{
    int i, wRetVal;
    DWORD temp;
    //Check Open_Slot
    wRetVal = Open_Slot(0);
    if (wRetVal > 0) {
        printf("open Slot failed. \n");
        return (-1);
    }
    //Check Open_Com1
    wRetVal = Open_Com(COM1, 115200, Data8Bit, NonParity, OneStopBit);
    if (wRetVal > 0) {
        printf("open port failed. \n");
        return (-1);
    }


    //Choose Slot3
    ChangeToSlot(3);
```

```
    //--- Digital Output ----    **(DigitalOut_87k()**)
    dwBuf[0] = 1;                       //COM Port.
    dwBuf[1] = 00;                      //Address.
    dwBuf[2] = 0x87054                  //ID.
    dwBuf[3] = 0;                       //CheckSum disabled.
    dwBuf[4] = 100;                     //TimeOut , the unit=0.1s
    dwBuf[5] = 0xff;                    //Set digital output.
    dwBuf[6] = 0;                       //Debug string.
    wRetVal = DigitalOut_87k(dwBuf, fBuf, szSend, szReceive);      //DO Output.
  printf("DO Value= %u", dwBuf[5]);

  //--- digital Input ----      **(DigitalIn_87k()**)
    dwBuf[0] = 1;                       //COM Port.
    dwBuf[1] = 00;                      //Address.
    dwBuf[2] = 0x87054;                 //ID.
    dwBuf[3] = 0;                       //CheckSum disabled.
    dwBuf[4] = 100;                     //TimeOut , the unit=0.1s
    dwBuf[6] = 0;                       //Debug string.
    getch();

    DigitalIn_87k(dwBuf, fBuf, szSend, szReceive);   //DI Input.
    printf("DI= %u",dwBuf[5])

 //--- digital output ----      ** Close DO **
    dwBuf[0] = 1;                       //COM Port.
    dwBuf[1] = 00;                      //Address.
    dwBuf[2] = 0x87054;                 //ID.
    dwBuf[3] = 0;                       //CheckSum disabled.
    dwBuf[4] = 100;                     //TimeOut, the unit=0.1s
    dwBuf[5] = 0x00;                    //Digital output.
    dwBuf[6] = 0;                       //Debug string .
    getch();                            //Press any key to continue.
    wRetVal = DigitalOut_87k(dwBuf, fBuf, szSend, szReceive);

    Close_Com(COM1);
    Close_SlotAll();
    return 0;
}
```

7.1.3. I-8K Modules

The **i8kdio.c** demo program illustrates how to control the DI/DO functions using I-8055W modules (8 DO channels and 8 DI channels) that are inserted into slot 3 on the LinPAC. **The address and baudrate for the LinPAC are 00 and 115200 bps separately**, and they were fixed by library. The result of executing this demo program is that DO channels 0 to 7 on the I-8055W module to will be set as the output channels, and DI channel 0 on I-8055W module will be set as the input channel. The source code for this demo program is as follows:

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"
char szSend[80], szReceive[80];
DWORD dwBuf[12];
float fBuf[12];
/* ---------------------------------------------------------------- */

int main()
{
    int i,j, wRetVal;
    WORD Doval,temp;
    wRetVal = Open_Slot(3);
    if (wRetVal > 0) {
        printf("open Slot failed. \n");
        return (-1);
    }
    //I-8055W_DO
    DO_8(3,255);
    printf("DO of I-8055 = 0x%x \n", 255);
    //I-8055W_DI
    printf("DI of I-8055 = %x",DI_8(3));
    Close_Slot(3);
    return 0;
}
```

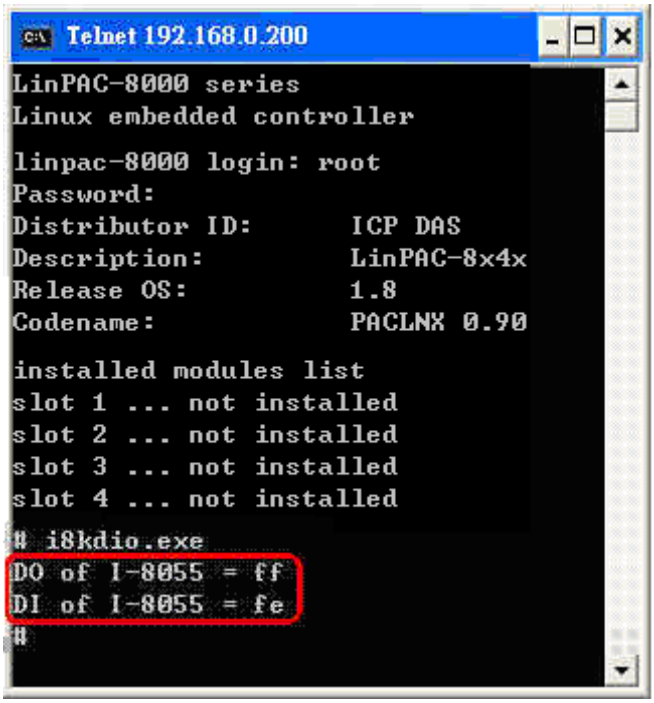


Figure 7.1.3-1. Execute the i8kdio.exe file

For this example, the programming and execution procedures are the same as those described in section 7.1.1. Figure 7.1.3-1 above illustrates the result of the execution.

7.2. AI/AO Control Demo

7.2.1. 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 , the unit=0.1s
```

```
    //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 , the unit=0.1s
    wBuf[5] = j;                //Channel of AI.
    wBuf[6] = 0;                //Debug string.

    wRetVal = AnalogIn(wBuf, fBuf, szSend, szReceive);
    if (wRetVal)                //There was an error with the Analog Input on the I-7017.
        printf("AI of 7017 Error !, Error Code=%d\n", wRetVal);
    else
        printf("AI of 7017 channel %d = %f \n",j,fBuf[0]);
    Close_Com(COM3);
    return 0;
}
```

For this example, the programming and execution procedures are the same as those described in the section 7.1.1. Figure 7.2.1-1 below illustrates the result of execution.

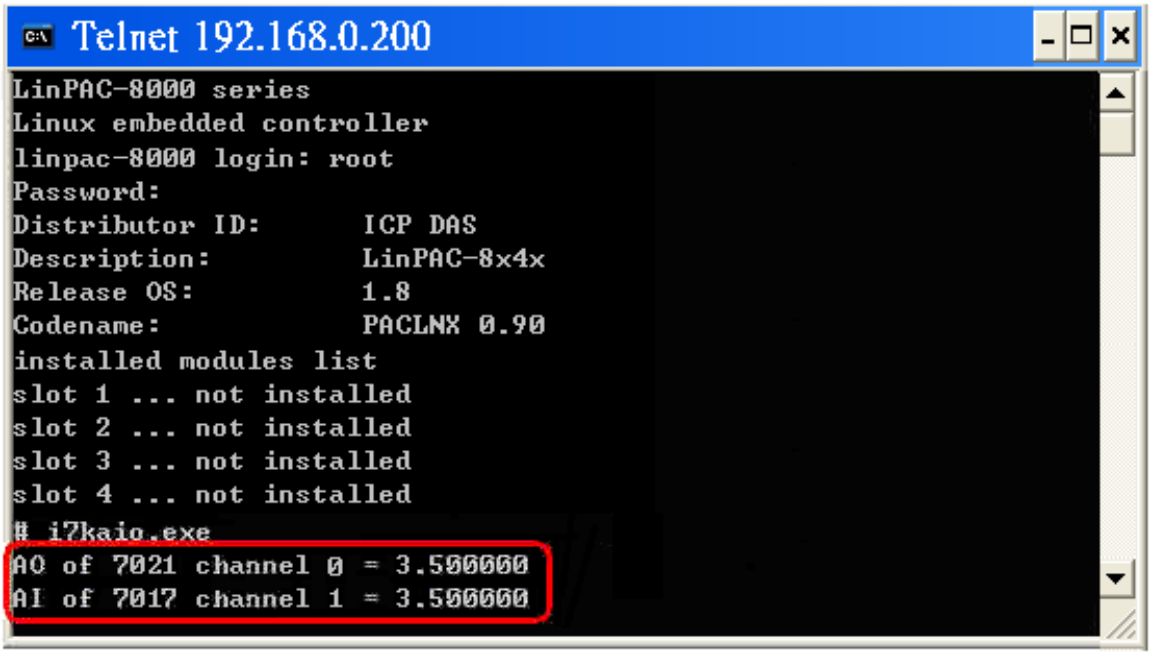


Figure 7.2.1-1. Execute the i7kaio.exe file

7.2.2. I-87K/97K Modules

If there are I-87KW/97K AIO modules inserted in the slots on the LinPAC, the '**Open_Slot**' and '**ChangeToSlot**' functions must be called before other functions for the I-87KW/97K modules are used, and the '**Close_Slot()**' function also needs to be called at the end of the program.

The **i87kaio.c** demo program illustrates how to control the AI/AO using an the **I-87022W** module (2 AO channels) and an **I-87017W** module (8 AI channels). The I-87022W and I-87017W modules are inserted into slots 2 and 3 of the LinPAC separately. **The addresses and baudrate for both modules in the LinPAC are 00 and 115200 bps separately**, they were fixed by the library. The result of executing this demo program is that AO channel 0 on the I-87022W module will be set to output a voltage of 2.5V, and AI channel 1 on the I-87017W module will be set as the input channel. The source code for this demo program is as follows:

```
#include<stdio.h>
#include<stdlib.h>
#include "msw.h"
char szSend[80], szReceive[80];
DWORD wBuf[12];
DWORD    wBuf7[12];
float fBuf[12];
int main()
{
      int i,j, wRetVal;
      DWORD temp;
    //Check Open_Slot
      wRetVal = Open_Slot(0);
      if (wRetVal > 0) {
            printf("open Slot failed. \n");
            return (-1);
      }

      //Check Open_Com1
      wRetVal = Open_Com(COM1, 115200, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed. \n");
            return (-1);
      }
```

```
    ChangeToSlot(2);
    //--- Analog output ----    ****      87022 – AO    ****
    i=0;
    wBuf[0] = 1;                          //COM Port.
    wBuf[1] = 0x00;                       //Address.
    wBuf[2] = 0x87022;                    //ID.
    wBuf[3] = 0;                          //CheckSum disable.
    wBuf[4] = 100;                        //TimeOut , the unit=0.1s
    wBuf[5] = i;                          //Channel Number of AO.
    wBuf[6] = 0;                          //String debug.
    fBuf[0] = 2.5;                        //AO Value.

    wRetVal = AnalogOut_87k(wBuf, fBuf, szSend, szReceive);
    if (wRetVal)
    // There was an error with the Analog Output on the I-87022W.
        printf("AO of 87022 Error, Error Code=%d\n", wRetVal);
    else
        printf("AO of 87022 channel %d = %f \n",i,fBuf[0]);
    ChangeToSlot(3);
    //--- Analog Input ----    ****      87017 – AI    ****
    j=1;
    wBuf7[0] = 1;                         //COM Port
    wBuf7[1] = 0x00;                      //Address
    wBuf7[2] = 0x87017;                   //ID.
    wBuf7[3] = 0;                         //CheckSum disabled.
    wBuf7[4] = 100;                       //TimeOut , the unit=0.1s
    wBuf7[5] = j;                         //Channel Number of AI.
    wBuf7[6] = 0;                         //Debug string.
    wRetVal = AnalogIn_87k(wBuf7, fBuf, szSend, szReceive);

    if (wRetVal)
    //There was an error with the Analog Output on the I-87017W.
        printf("AI of 87017 Error, Error Code=%d\n", wRetVal);
    else
        printf("AI of 87017 channel %d = %f \n",j,fBuf[0]);

    Close_Com(COM1);
    Close_SlotAll();
    return 0;
}
```

7.2.3. I-8K/9K Modules

The **i8kaio.c** demo program illustrates how to control the AI/AO functions using the I-8024W/9024 (4 AO channels) and I-8017HW/9017 (8 AI channels) modules, which are inserted in slot 1 and slot 2 on the LinPAC separately. **The address and baudrate in the LinPAC are 00 and 115200 bps separately**, and they were fixed by library. The result of executing this demo is that AO voltage channel 0 on the I-8024W/9024 module to will be set to output 5.5 V and AI channel 2 on the I-8017HW/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 wRetVal;
	short jumper;
	int hAi, chAi, Arr_hAi[5];
	float fVal, Arr_fAi[5];

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

	//I8024 Initial
	I8024_Initial(1);

	//I8024_AO Output
	I8024_VoltageOut(1,0,5.5);
	printf("Slot1: I8024 Set CH0= %f\n",fVal);
	Close_Slot(1);
```

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

    //I8017H Initial
    I8017_Init(2);
    I8017_GetSingleEndJumper(slot,&jumper);          //Read Jumper status
    //printf("Jumper mode: %d\n",jumper);

    // First Method：Get AI Value: I8017_ReadAI
    I8017_ReadAI(2,2,1,&fVal);   //I8017_ReadAI(slot,channel,iGain,&fVal);
    printf("Slot2: I8017_ReadAI CH2= %f\n",fVal);

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

For this example, the programming and execution procedures are the same as those described in section 7.1.1. Figure 7.2.3-1 below illustrates the result of the execution.

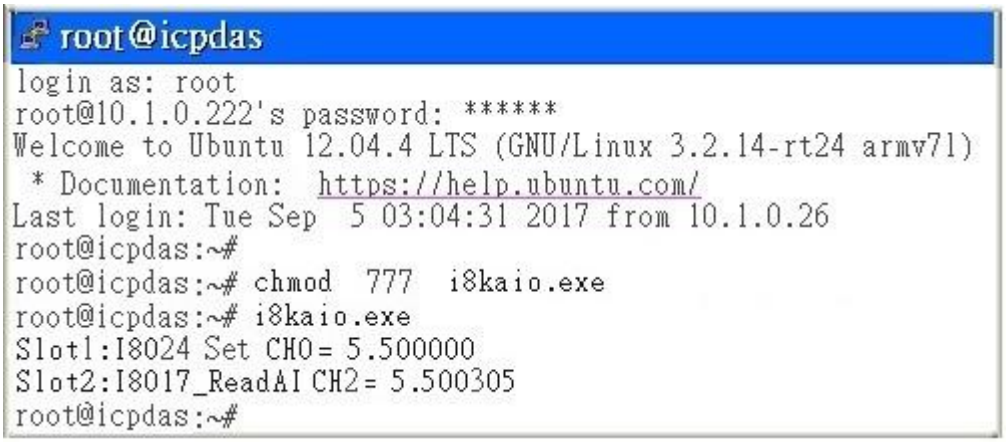


Figure 7.2.3-1. Execute the i8kaio.exe file

Appendix

A. Demo for I/O Modules in slots on an I-87K I/O expansion unit

A1. DIO Control Demo for I-87K Modules

If the I-87KW DIO modules are inserted in the slots on an I-87KW I/O expansion unit, three parts of the program illustrated in section 7.1.2 above will need to be modified as follows:

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

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

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

The I-87054W is connected to an RS-485 network where the address is set to be 06 and the baudrate is 9600 bps, which must be configured using the DCON Utility. The source code for the i87kdio_87k.c demo program –is as follows:

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

char szSend[80], szReceive[80];
DWORD dwBuf[12];
float fBuf[12];
/* ---------------------------------------------------------------- */
int main()
{
    int i, wRetVal;
    DWORD temp;
    //Check Open_Com3
    wRetVal = Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
```

```
    if (wRetVal > 0) {
        printf("open port failed. \n");
        return (-1);
    }
    //--- digital output ----    **(DigitalOut_87k()**)
    dwBuf[0] = 3;                   //COM Port.
    dwBuf[1] = 06;                  //Address.
    dwBuf[2] = 0x87054;             //ID.
    dwBuf[3] = 0;                   //CheckSum disable.
    dwBuf[4] = 100;                 //TimeOut, the unit=0.1s
    dwBuf[5] = 0xff;                //Digital output.
    dwBuf[6] = 0;                   //String debug.

    wRetVal = DigitalOut_87k (dwBuf, fBuf, szSend, szReceive);      //DO Output.
    printf("DO Value= %u", dwBuf[5]);

    //--- digital Input ----      **(DigitalIn_87k()**)
    dwBuf[0] = 3;                   //COM Port.
    dwBuf[1] = 06;                  //Address.
    dwBuf[2] = 0x87054;             //ID.
    dwBuf[3] = 0;                   //CheckSum disabled.
    dwBuf[4] = 100;                 //TimeOut , the unit=0.1s
    dwBuf[6] = 0;                   //Debug string.

    getch();
    DigitalIn_87k(dwBuf, fBuf, szSend, szReceive);          //DI Input.
    printf("DI= %u",dwBuf[5]);

    //--- digital output ----      ** Close DO **
    dwBuf[0] = 3;                   //COM Port.
    dwBuf[1] = 06;                  //Address.
    dwBuf[2] = 0x87054;             //ID.
    dwBuf[3] = 0;                   //CheckSum disabled.
    dwBuf[4] = 100;                 //TimeOut , the unit=0.1s
    dwBuf[5] = 0x00;                //Digital output.
    dwBuf[6] = 0;                   //Debug string.
    getch();                        //Press any key to continue.
    wRetVal = DigitalOut_87k(dwBuf, fBuf, szSend, szReceive);
    Close_Com(COM3);
    return 0;
}
```

For this example, the programming and execution procedures are the same as those described in section 7.1.1. Figure 8.1.1-1 below illustrates the result of the execution.

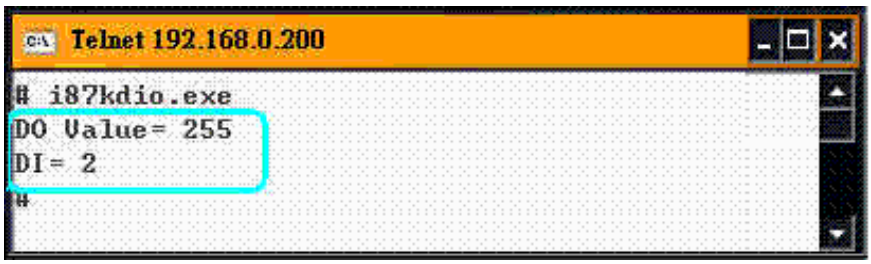


Figure 8.1.1-1. Execute the i87kdio.exe file

A2. AIO Control Demo for I-87K Modules

If the I-87KW/97K modules are inserted in slots on an I-87KW/97K I/O expansion unit, the three parts of the program illustrated in Section 7.2.2 above will need to be modified, as follows:

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

(2) The **address** and **baudrate** of any I-87KW/97K modules connected to the RS-485 network will need to be configured using the DCON Utility, which can be downloaded from http://www.icpdas.com/products/dcon/introduction.htm.

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

The I-87022W/97022 and I-87017W/97017 addresses are connected to the RS-485 network and the addresses are set to 01 and 02 separately, with the baudrate for both modules set to 9600 bps, which must be configured using the DCON Utility. The source code for the **i87kaio_87k.c/i97kaio_97k.c** demo program is as follows:

```
#include<stdio.h>
#include<stdlib.h>
#include “msw.h”

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

```
DWORD    wBuf7[12];
float fBuf[12];

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

      //Check Open_Com3
      wRetVal = Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed. \n");
            return (-1);
      }

      //--- Analog output ----    ****     87022 – AO    ****
      i=0;
      wBuf[0] = 3;                        //COM Port.
      wBuf[1] = 0x01;                     //Address.
      wBuf[2] = 0x87022;                  //ID.
      wBuf[3] = 0;                        //CheckSum disabled.
      wBuf[4] = 100;                      //TimeOut , the unit=0.1s
      wBuf[5] = i;                        //Channel Number of AO.
      wBuf[6] = 0;                        //Debug string.
      fBuf[0] = 2.5;                      //AO Value.

      wRetVal = AnalogOut_87k(wBuf, fBuf, szSend, szReceive);
      if (wRetVal)
                  printf("AO of 87022 Error , Error Code=%d\n", wRetVal);
      else
                  printf("AO of 87022 channel %d = %f \n",i,fBuf[0]);

      //--- Analog Input ----    ****     87017 – AI    ****
      j=1;
      wBuf7[0] = 3;                       //COM Port.
      wBuf7[1] = 0x02;                    //Address.
      wBuf7[2] = 0x87017;                 //ID.
      wBuf7[3] = 0;                       //CheckSum disabled.
      wBuf7[4] = 100;                     //TimeOut , the unit=0.1s
      wBuf7[5] = j;                       //Channel Number of AI.
      wBuf7[6] = 0;                       //Debug string.
```

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

    Close_Com(COM3);

    return 0;
}
```

For this example, the programming and execution procedures are the same as those described in section 7.1.1. Figure 8.1.2-1 below illustrates the result of the execution.

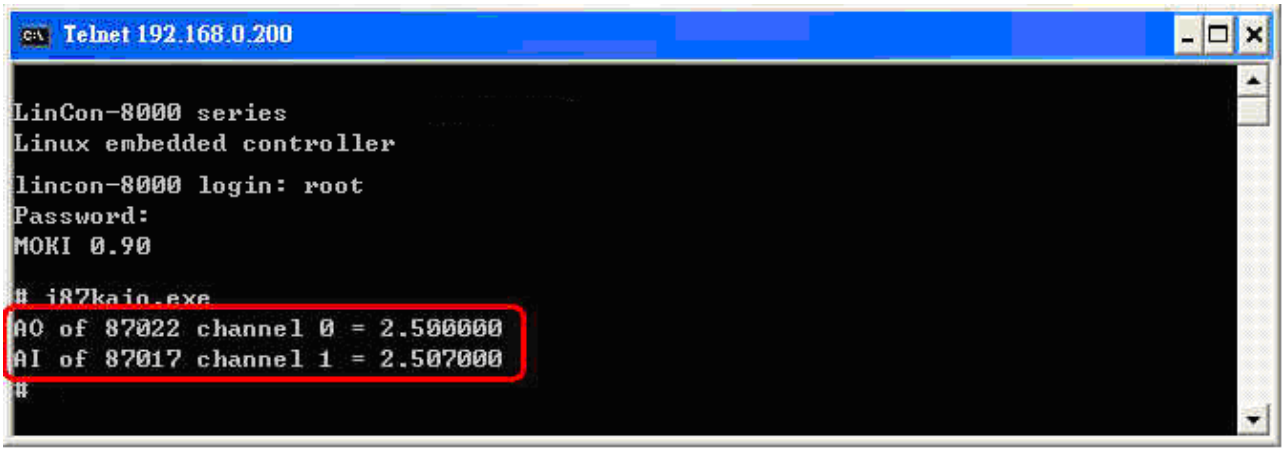


Figure 8.1.2-1. Execute the i87kaio.exe file

B. Demo for I/O Modules in slots on an I-8000 Controller

B1. DIO Control Demo for I-87K Modules

If the I-87KW DIO modules are inserted the slots on an I-8000 controller, the I-87KW modules will be regarded as I-8KW modules. For more details, refer to the description of how to perform DI/DO control on I-8KW modules provided in Appendix B3.

B2. AIO Control Demo for I-87K Modules

If the I-87KW AIO modules are inserted in slots on an I-8000 controller, the modules will be regarded as I-8KW modules. For more details, refer to the description of how to perform AI/AO control on I-8KW modules provided in Appendix B4.

B3. DIO Control Demo for I-8K Modules

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

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

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

The address of the I-8000 controller is 01 and the baudrate is 115200 bps, which must be configured using the DCON Utility. The result of executing this demo program is that DO channels 0 to 7 on the I-8055W module to will be set to the output channel, and DI channel 0 on the I-8055W module will be set as input channel. The source code for this demo program is as follows:

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

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

    //Check Open_Com3
    wRetVal = Open_Com(COM3, 115200, Data8Bit, NonParity, OneStopBit);
    if (wRetVal > 0) {
        printf("open port failed. \n");
        return (-1);
    }

    //--- digital output ----    **(DigitalOut_8K()**)
    dwBuf[0] = 3;                   //COM Port.
    dwBuf[1] = 01;                  //Address.
    dwBuf[2] = 0x8055;              //ID.
    dwBuf[3] = 0;                   //CheckSum disabled.
    dwBuf[4] = 100;                 //TimeOut , 1 the unit=0.1s
    dwBuf[5] = 0xff;                //Digital output.
    dwBuf[6] = 0;                   //Debug string.
    dwBuf[7] = 1;                   //Slot number.
    wRetVal = DigitalOut_8K(dwBuf, fBuf, szSend, szReceive);
    if (wRetVal)

    // There was an error with the Analog Output on the I-8055
        printf("DO of I-8055 Error , Error Code=%d\n", wRetVal);
    else
        printf("DO of I-8055 = 0x%x" ,dwBuf[5]);

    //--- Digital Input ----     **(DigitalIn_8K()**)
    dwBuf[0] = 3;                   //COM Port.
    dwBuf[1] = 01;                  //Address.
    dwBuf[2] = 0x8055;              //ID.
    dwBuf[3] = 0;                   //CheckSum
    dwBuf[4] = 100;                 //TimeOut , the unit=0.1s
```

```
    dwBuf[6] = 0;                    //Debug string.
    dwBuf[7] = 1;                    //Slot number.
    getch();
    DigitalIn_8K(dwBuf, fBuf, szSend, szReceive);
    printf("DI = %u",dwBuf[5]);

    //--- Digital output ----     ** Close DO **
    dwBuf[0] = 3;                    //COM Port.
    dwBuf[1] = 01;                   //Address.
    dwBuf[2] = 0x8055;               //ID.
    dwBuf[3] = 0;                    //CheckSum disabled.
    dwBuf[4] = 100;                  //TimeOut , the unit=0.1s
    dwBuf[5] = 0x00;                 //Digital output.
    dwBuf[6] = 0;                    //Debug string.
    dwBuf[7] = 1;                    //Slot number.
    getch()                          //Push any key to continue.
    wRetVal = DigitalOut_8K(dwBuf, fBuf, szSend, szReceive);
    Close_Com(COM3);
    return 0;
}
```

For this example, the programming and execution procedures are the same as those described in section 7.1.1. Figure 8.2.3-1 below illustrates the result of the execution.

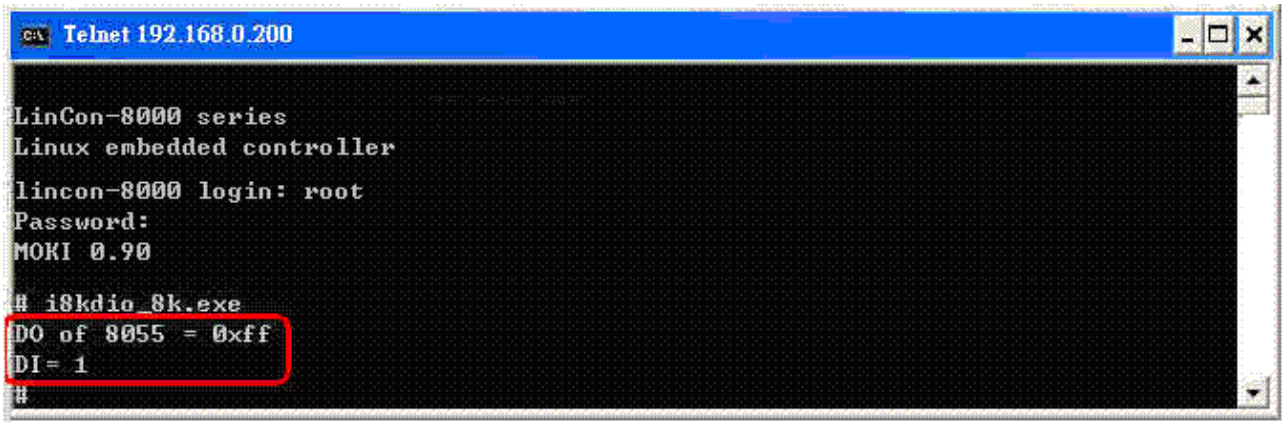


Figure 8.2.3-1. Execute the i8kdio.exe file

B4. AIO Control Demo for I-8K Modules

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

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

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

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

The address of the I-8000 controller is 01 and baudrate is 115200 bps, which must be configured using the DCON Utility. The result of executing this demo program is that AO voltage channel 0 on the I-8024W module to will be set to output 3.5 V, and AI channel 2 on the I-8017HW module will be set as the input channel. The source code for this demo program is as follows:

```
#include<stdio.h>
#include<stdlib.h>
#include “msw.h”
char szSend[80], szReceive[80];
DWORD wBuf[12];
float fBuf[12];

int main()
{
    int i=0, j=2, wRetVal;
    DWORD temp;
    wRetVal = Open_Com(COM3, 115200, Data8Bit, NonParity, OneStopBit);
    if (wRetVal > 0) {
        printf(“open port failed. \n”);
        return (-1);
    }
    //--- Analog output ----   ****    8024 – AO    ****
    wBuf[0] = 3;                    //COM Port.
    wBuf[1] = 0x01;                 //Address.
    wBuf[2] = 0x8024;               //ID.
    wBuf[3] = 0;                    //CheckSum disabled.
    wBuf[4] = 100;                  //TimeOut , the unit=0.1s
    wBuf[5] = i;                    //Channel No. of AO.
```

```
    wBuf[6] = 0;                        //Debug string.
    wBuf[7] = 0;                        //Slot Number.
    fBuf[0] = 3.5;
    wRetVal = AnalogOut_8K(wBuf, fBuf, szSend, szReceive);
    if (wRetVal)
        printf("AO of 8024 Error, Error Code=%d\n", wRetVal);
    else
        printf("AO of 8024 channel %d = %f \n",i,fBuf[0]);

    //--- Analog Input ----    ****     8017H – AI    ****
    wBuf[0] = 3;                        //COM Port.
    wBuf[1] = 0x01;                     //Address.
    wBuf[2] = 0x8017;                   //ID.
    wBuf[3] = 0;                        //CheckSum disabled.
    wBuf[4] = 100;                      //TimeOut , the unit=0.1s
    wBuf[5] = j;                        //Channel of AI.
    wBuf[6] = 0;                        //Debug string.
    wBuf[7] = 1;                        //Slot Number.
    wRetVal = AnalogIn_8K(wBuf, fBuf, szSend, szReceive);
    if (wRetVal)
        printf("AI of 8017H Error, Error Code=%d\n", wRetVal);
    else
        printf("AI of 8017H channel %d = %f \n",j,fBuf[0]);

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

For this example, the programming and execution procedures are the same as those described in section 7.1.1. Figure 8.2.4-1 below illustrates the result of the execution.

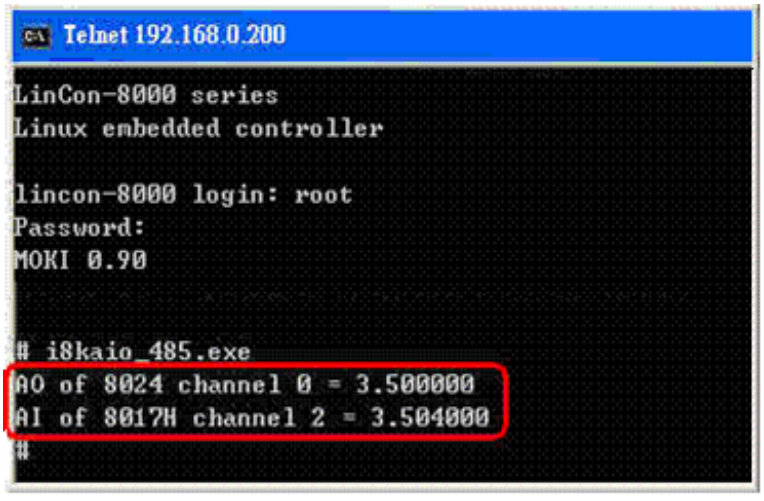


Figure 8.2.4-1. Execute the i8kaio.exe file

C. The old version of the API function

The table below lists the old version of the API function for AIO modules via a parallel port that are supported by each LinPAC. For more details, please refer to the corresponding chapters.

C1. System Function

■ GetNameOfModule

Description:

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

Syntax:

[C]

```
int GetNameOfModule(int slot)
```

Parameter:

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

Return Values:

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

Example:

```
int slot=1;
int moduleID;
Open_Slot(slot);
moduleID=GetNameOfModule(slot);
Close_Slot(slot);
// The I-8017W module is inserted in slot 1 of LP-8x4x.
// Returned Value: moduleName='8017 '.
```

■ 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 Values:

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

Example:

```
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 '.
```

C2. I-8017 API Function

■ I8017_Init

Description:

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

Syntax:

[C]

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

Parameter:

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

Modules	LP-8x2x	LP-8x4x	LP-8x8x	LX-Series
Slot range	1 ~ 8	1 ~ 8	1 ~ 7	2 ~ 8

Return Value:

The version of library.

Examples:

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

■ I8017_SetLed

Description:

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

Syntax:

[C]

```
void I8017_SetLed(int slot,unsigned int led)
```

Parameter:

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

led: [Input] Range from 0 to 0xffff

Modules	LP-8x2x	LP-8x4x	LP-8x8x	LX-Series
Slot range	1 ~ 8	1 ~ 8	1 ~ 7	2 ~ 8

Return Value:

None

Examples:

```
int slot=1;       // slot=1 or 2.
unsigned int led=0x0001;
I8017_SetLed(slot, led);
// There will be a LED light on channel 0 of the I-8017HW card
// which is inserted in slot 1 (or 2) on the LinPAC.
```

■ I8017_SetChannelGainMode

Description:

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

Syntax:

[C]

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

Parameter:

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

Modules	LP-8x2x	LP-8x4x	LP-8x8x	LX-Series
Slot range	1 ~ 8	1 ~ 8	1 ~ 7	2 ~ 8

ch: [Input] Differential mode → Range 0 to 7 (I-8017H: Range 0 to 7)

Others: Single-ended mode → Range 0 to 15

gain: [Input] input range:

0: +/- 10.0V

1: +/- 5.0V

2: +/- 2.5V

3: +/- 1.25V

4: +/- 20mA

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

Return Value:

None

Examples:

```
int slot=1, ch=0, gain=0;
I8017_SetChannelGainMode(slot, ch, gain,0);
// The I-8017HW card is inserted in slot 1 or 2 of LinPAC, and the range of the data
// value from channel 0 for I-8017H will be -10 to +10V.
```

■ I8017_GetFirmwareVersion

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

>0: Version No.

<=0: Error.

Examples:

```
int slot=1, version;
version = I8017_ GetFirmwareVersion(slot);
printf("I-8017 at Slot%d, firmware version= %d",slot, version);
// The I-8017HW card is inserted in slot 1 of LinPAC and initializes the module.
```

■ I8017_GetSingleEndJumper

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

1: Single-Ended mode.

0: Differential mode.

Examples:

```
int slot=1;
I8017_Init(slot);
printf("mode=%d", I8017_GetSingleEndJumper(slot));
```

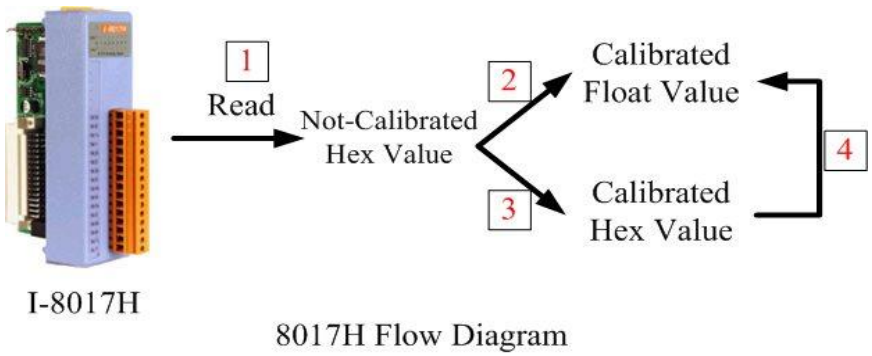


Figure 8.3.1-1. I-8017H Flow Diagram

In order to provide convenience for the user, ICP DAS released a new version of the SDK for Linux PAC at 2018, the new version API- **I8017_ReadAI()** and **I8017_ReadAIHex()** functions have replaced following:

I8017_GetCurAdChannel_Hex (int slot)

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

I8017_HEX_TO_FLOAT_Cal(int HexValue, int slot, int gain)

I8017_ARRAY_HEX_TO_FLOAT_cal(int *HexValue, float *FloatValue, int slot, int gain, int len)

I8017_Hex_Cal(int data)

I8017_Hex_Cal_Slot_Gain(int slot, int gain, int data)

I8017_CalHex_TO_FLOAT(int HexValue, int gain)

I8017_ARRAY_CalHex_TO_FLOAT(int *HexValue, float *FloatValue, int gain, int len)

I8017_GetCurAdChannel_Hex_Cal(int slot)

I8017_AD_POLLING_Cal(int slot, int ch, int gain, unsigned int datacount, int *DataPtr)

I8017_GetCurAdChannel_Float_Cal(int slot)

For more details about new version API, please refer to the following website link:

https://www.icpdas.com/en/download/show.php?num=1869&model=I-8017HW-G

Function of [1]

■ I8017_GetCurAdChannel_Hex

Description:

Obtains the non-calibrated analog input value in the Hex format from the analog input I-8017HW modules.

Syntax:

[C]

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

Parameter:

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

Return Value:

The analog input value in Hex format.

Examples:

```
int slot=1,ch=0,gain=4;
int data;
I8017_SetChannelGainMode(slot, ch, gain,0);
data= I8017_GetCurAdChannel_Hex(slot);
// The I-8017HW is inserted in slot 1 of LinPAC and the range of the data
// value from channel 0 in I-8017H is +/-20Ma.
```

■ I8017_AD_POLLING

Description:

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

Syntax:

[C]

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

Parameter:

slot:	[Input] Specifies the slot where the I/O module is inserted
ch:	[Input] I-8017H: Range 0 to 7
	Others: Single-ended mode → Range 0 to 15
	Differential mode → Range 0 to 7
gain:	[Input] Input range:
	0: +/- 10.0V ; 1: +/- 5.0V ; 2: +/- 2.5V ; 3: +/- 1.25V ; 4:+/- 20Ma
datacount:	[Input] Range from 1 to 8192, total ADCs number
*DataPtr:	[Output] The starting address of data array[ ] and the array size must be equal to or bigger than the datacount

Return Value:

0: Indicates success.

Not 0: Indicates failure.

Examples:

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

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

Function of [2]

■ I8017_HEX_TO_FLOAT_Cal

Description:

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

Syntax:

[C]

```
float I8017_HEX_TO_FLOAT_Cal(int HexValue, int slot, int gain)
```

Parameter:

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

slot: [Input] Specified slot of the LinPAC system

gain: [Input] Input range:

0: +/- 10.0V

1: +/- 5.0V

2: +/- 2.5V

3: +/- 1.25V

4: +/- 20Ma

Return Value:

The Calibrated Float Value.

Examples:

```
int slot=1, ch=0, gain=0, hdata;
float fdata;
I8017_SetChannelGainMode(slot, ch, gain,0);
hdata=I8017_I8017_GetCurAdChannel_Hex(slot);
fdata=I8017_HEX_TO_FLOAT_Cal(hdata, slot, gain);
// You can convert not-calibrated Hex Value to calibrated Float Value.
```

■ I8017_ARRAY_HEX_TO_FLOAT_Cal

Description:

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

Syntax:

[C]

void I8017_ARRAY_HEX_TO_FLOAT_cal(int *HexValue,float *FloatValue,int slot, int gain,int len)

Parameter:

*HexValue:	[Input] Data array in not-calibrated Hex type before converting
*FloatValue:	[Output] Converted data array in calibrated float type
slot:	[Input] Specifies the slot where the I/O module is inserted
gain:	[Input] Input range: 0: +/- 10.0V 1: +/- 5.0V 2: +/- 2.5V 3: +/- 1.25V 4: +/- 20Ma
len:	[input] ADC data length

Return Value:

None

Examples:

```
int slot=1, ch=0, gain=0, datacount=10, hdata[10];
float fdata[10];
I8017_SetChannelGainMode(slot, ch, gain,0);
I8017_AD_POLLING(slot, ch, gain, datacount, data);
I8017_ARRAY_HEX_TO_FLOAT_Cal(data, fdata, slot, gain, len);
// You can convert ten not-calibrated Hex values to ten calibrated Float values.
```

■ HEX_TO_FLOAT_Cal

Description:

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

Syntax:

[C]

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

Parameter:

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

slot: [Input] Specified slot of the LinPAC Series system

gain: [Input] Input range:

0: +/- 10.0V

1: +/- 5.0V

2: +/- 2.5V

3: +/- 1.25V

4: +/- 20mA

Return Value:

The Calibrated Float Value.

Examples:

```
int slot=2, ch=0, gain=0, hdata;
float fdata;
I8017_SetChannelGainMode(slot, ch, gain,0);
hdata=I8017_GetCurAdChannel_Hex(slot);
fdata=HEX_TO_FLOAT_Cal(hdata, slot, gain);
// You can convert not-calibrated Hex Value to calibrated Float Value.
```

■ ARRAY_HEX_TO_FLOAT_Cal

Description:

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

Syntax:

[C]

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

Parameter:

*HexValue:	[Input] Data array in not-calibrated Hex type before converting
*FloatValue:	[Output] Converted data array in calibrated float type
slot:	[Input] Specified slot of the LinPAC Series system
gain:	[Input] Input range:
len:	[input] ADC data length

Return Value:

None

Examples:

```
int slot=2, ch=0, gain=0, datacount=10, hdata[10];
float fdata[10];
I8017_SetChannelGainMode(slot, ch, gain,0);
I8017_AD_POLLING(slot, ch, gain, datacount, data);
ARRAY_HEX_TO_FLOAT_Cal(data, fdata, slot, gain, len);
// You can convert ten not-calibrated Hex values to ten calibrated Float values.
```

Function of [3]

■ I8017_Hex_Cal

Description:

This function is used to convert the data from non-calibrated hex values to calibrated hex values (Voltage or current). Please refer to Figure 8.3.1-1.

Syntax:

[C]

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

Parameter:

data : [Input] Specified not-calibrated hex value

Return Value:

The Calibrated Hex Value.

Examples:

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

■ I8017_Hex_Cal_Slot_Gain

Description:

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

Syntax:

[C]

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

Parameter:

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

gain: [Input] Input range:

0: +/- 10.0V

1: +/- 5.0V

2: +/- 2.5V

3: +/- 1.25V

4: +/- 20Ma

data: [Input] Specified not-calibrated hex value

Return Value:

The Calibrated Hex Value.

Examples:

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

Function of [4]

■ I8017_CalHEX_TO_FLOAT

Description:

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

Syntax:

[C]

```
float I8017_CalHex_TO_FLOAT(int HexValue,int gain)
```

Parameter:

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

gain: [Input] Input range:

0: +/- 10.0V

1: +/- 5.0V

2: +/- 2.5V

3: +/- 1.25V

4: +/- 20Ma

Return Value:

The Calibrated Float Value.

Examples:

```
int slot=1, ch=0, gain=0, hdata, hdata_cal;
float fdata;
I8017_SetChannelGainMode(slot, ch, gain,0);
hdata=I8017_GetCurAdChannel_Hex(slot);
hdata_cal=I8017_HEX_Cal(hdata);
fdata=I8017_CalHex_TO_FLOAT(hdata_cal, gain);
// You can convert calibrated Hex Value to calibrated Float Value.
```

■ I8017_ARRAY_CalHEX_TO_FLOAT

Description:

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

Syntax:

[C]

void I8017_ARRAY_CalHex_TO_FLOAT(int *HexValue,float *FloatValue,int gain,int len)

Parameter:

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

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

gain: [Input] Input range:

0: +/- 10.0V

1: +/- 5.0V

2: +/- 2.5V

3: +/- 1.25V

4: +/- 20Ma

len: [input] ADC data length

Return Value:

The Calibrated Float Value.

Examples:

```
int slot=1, ch=0, gain=0, hdata_cal[10];
float fdata[10];
fdata=I8017_ARRAY_CalHex_TO_FLOAT(hdata_cal, fdata, gain, len);
// You can convert ten calibrated Hex Values to ten calibrated Float Values.
```

■ CalHEX_TO_FLOAT

Description:

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

Syntax:

[C]

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

Parameter:

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

gain: [Input] Input range:

0: +/- 10.0V

1: +/- 5.0V

2: +/- 2.5V

3: +/- 1.25V

4: +/- 20mA

Return Value:

The Calibrated Float Value.

Examples:

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

■ ARRAY_CalHEX_TO_FLOAT

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

The Calibrated Float Value.

Examples:

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

Function of [1]+[2]

■ I8017_GetCurAdChannel_Float_Cal

Description:

Obtains the calibrated analog input value in the Float format directly from the analog input modules. This function is a combination of the '**I8017_GetCurAdChannel_Hex**' and the '**Hex_TO_FLOAT_Cal**' function.

Syntax:

[C]

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

Parameter:

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

Return Value:

The analog input value in Calibrated Float format.

Examples:

```
int slot=1,ch=0,gain=0;
float data;
I8017_SetChannelGainMode(slot, ch, gain,0);
data=I8017_GetCurAdChannel_Float_Cal(slot);
// The I-8017HW is inserted in slot 1 of LinPAC and the range of the
// data value from channel 0 in I-8017H is -10V to +10V.
```

Function of [1]+[3]

■ I8017_GetCurAdChannel_Hex_Cal

Description:

Obtain the calibrated analog input values in the Hex format directly from the analog input modules. This function is a combination of the '**I8017_GetCurAdChannel_Hex**' and the '**I8017_Hex_Cal**' function.

Syntax:

[C]

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

Parameter:

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

Return Value:

The analog input value in Calibrated Hex format.

Examples:

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

■ I8017_AD_POLLING_Cal

Description:

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

Syntax:

[C]

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

Parameter:

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

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

Others: Single-ended mode → Range 0 to 15

Differential mode → Range 0 to 7

gain: [Input] Input range:

0: +/- 10.0V; 1: +/- 5.0V; 2:+/- 2.5V; 3: +/- 1.25V; 4: +/- 20Ma

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

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 6: 'Error Code Definitions' for details of other returned values.

Examples:

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

C3. I-8024 API Function

■ I8024_Initial

Description:

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

Syntax:

[C]

```
void I8024_Initial(int slot)
```

Parameter:

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

Modules	LP-8x2x	LP-8x4x	LP-8x8x	LX-Series
Slot range	1 ~ 8	1 ~ 8	1 ~ 7	2 ~ 8

Return Value:

None

Examples:

```
int slot=1;
I8024_Initial(slot);
// The I-8024W is inserted in slot 1 of LinPAC and initializes the I-8024W module.
```

■ I8024_VoltageOut

Description:

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

Syntax:

[C]

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

Parameter:

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

Modules	LP-8x2x	LP-8x4x	LP-8x8x	LX-Series
Slot range	1 ~ 8	1 ~ 8	1 ~ 7	2 ~ 8

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

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

Return Value:

None

Examples:

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

■ I8024_CurrentOut

Description:

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

Syntax:

[C]

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

Parameter:

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

Modules	LP-8x2x	LP-8x4x	LP-8x8x	LX-Series
Slot range	1 ~ 8	1 ~ 8	1 ~ 7	2 ~ 8

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

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

Return Value:

None

Examples:

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

■ I8024_VoltageHexOut

Description:

This function is used to send the voltage value in the Hex format to the specified channel in the I-8024W module, which is inserted in the slot in the LinPAC system.

Syntax:

[C]

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

Parameter:

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

Modules	LP-8x2x	LP-8x4x	LP-8x8x	LX-Series
Slot range	1 ~ 8	1 ~ 8	1 ~ 7	2 ~ 8

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

hdata: [Input] Output data with hexadecimal

(data range: 0h to 3FFFh → Voltage Output: -10 to 10V)

Return Value:

None

Examples:

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

■ I8024_CurrentHexOut

Description:

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

Syntax:

[C]

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

Parameter:

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

Modules	LP-8x2x	LP-8x4x	LP-8x8x	LX-Series
Slot range	1 ~ 8	1 ~ 8	1 ~ 7	2 ~ 8

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

hdata: [Input] Output data with hexadecimal

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

Return Value:

None

Examples:

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

D. Revision History

This chapter provides revision history information to this document.

The table below shows the revision history.

Revision	Date	Description
V1.0.0	Apr 2019	Initial issue
V1.1.0	May 2021	Update SDK download link and wdt function
V1.1.1	October 2021	Update WDT function
V1.2.0	December 2021	1. SDK for Linux platform is running for 32 bit OS 2. Modify unit for **wTimeout** and ***wT** parameter
V1.2.1	December 2022	Add two API function for Low Pass Filter Module of I-9000

 E-mail: service@icpdas.com