Software Guide

Implement industry control with Linux Technique

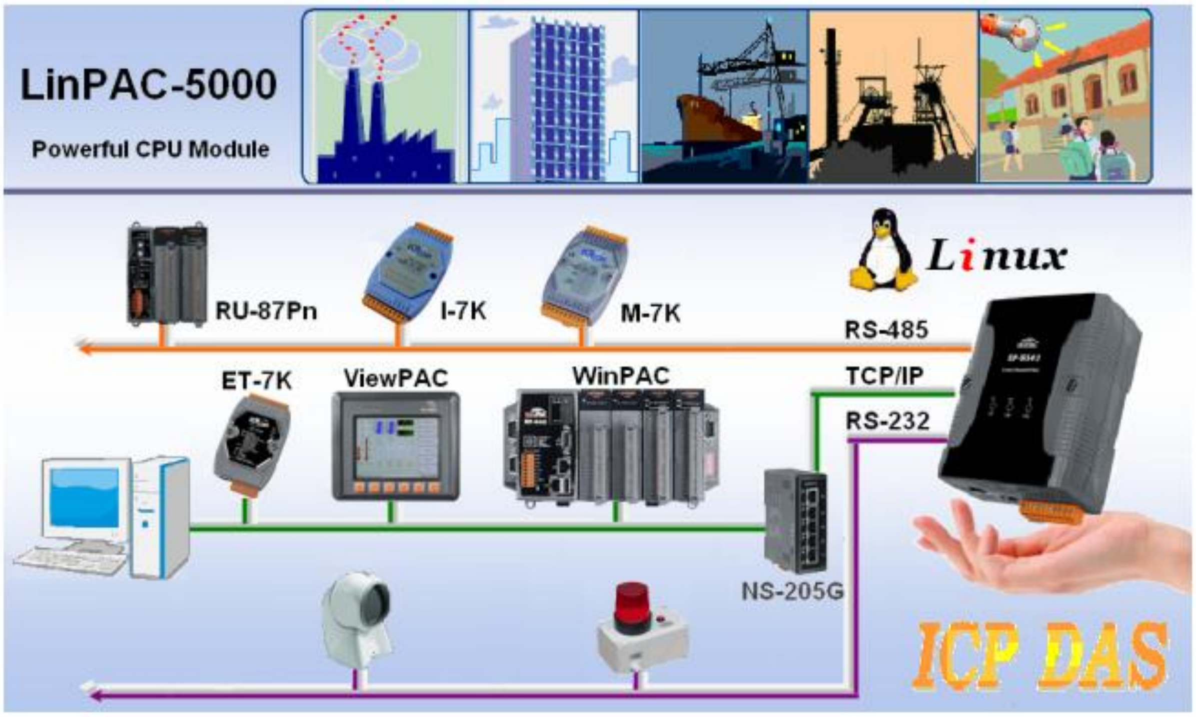
Service and usage information for LP-5231 Series LinPAC

Contact US

If you have any question, please feel free to contact us.
We will give you quick response within 2 workdays.
Email: service@icpdas.com , service.icpdas@gmail.com

Table of Contents

1. Introduction



The LP-5231 series is a new generation LinPAC from ICP DAS and is equipped with a powerful CPU module running on the open operating system, various connectivity (Ethernet, micro SD and serial port) and communication interfaces. Compared with the previous generation LP-5000 series, not only the CPU performance is higher but also more features are improved such as 512 MB fl ash, 512 MB DDR3 memory, unique 64-bit hardware serial number, and real-time clock, etc. These make the LP-5231 series becoming one of the most powerful system.

This LinPAC is designed to add Ethernet and Internet connectivity to any RS-232 and RS-422/485, and to eliminate the cable length limitation of legacy serial communication, coupled with a large built-in RAM buffer, allows for fast transmission and prevents congestion of serial data on the network. Built-in powerful 1 GHz ARM-based processor offers excellent performance at low power consumption. The preloaded high-performance operating system is open, flexible, scalable and allows user to easily add or remove application/service from configuration mechanism.

1.1 Packing List

The package includes the following items:

- One LinPAC-5231 series hardware module
- One software utility CD
- One RS-232 console/download cable, CA-0903
- One Quick Start Guide
- GPRS/3G/4G External Antenna and GPS Active External Antenna for LP-5231PM-3GWA and LP-5231PM-4GE/4GC.

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

1.2 Features

- AM3354, 1 GHz CPU
- 512 MB SRAM and 512 MB Flash
- Linux kernel 3.2.14
- Hard Real-Time Capability
- 64-bit Hardware Serial Number for Software Protection
- I/O Expansion Bus
- 10/100/1000M Ethernet Port
- 4 Serial Ports (RS-232/485)
- GSM/3G System(LP-5231PM-3GWA only)
- GSM/3G/4G System(LP-5231PM-4GE/4GC only)
- Operating Temperature: -25 ~ +75°C

1.3 Specifications

Models	LP-5231	LP-5231M
System Software		
OS	Linux Kernel 3.2.14	
Embedded Service	SFTP/FTP server, Web server, SSH	
SDK Provided	Standard LinPAC SDK for Linux by GNU C language	
CPU Module		
CPU	32-bit RISC, 1 GHz	
SDRAM	512MB	
Flash	512MB	
FRAM	64KB	
Expansion Flash Memory	microSD socket with one 4 GB microSD card (support up to 32 GB microSDHC card)	
RTC (Real Time Clock)	Provide second, minute, hour, date, day of week, month, year	
64-bit Hardware Serial Number	Yes, for Software Copy Protection	
Dual Watchdog Timers	Yes	
LED Indicators	2 LED for Power and Running; 2 LED for user defined	
Rotary Switch	Yes(0~9)	
VGA & Communication Interface		
VGA	VGA x 1 (max. resolution up to 1280 x 1024)	
USB 2.0(host)	1	
Console Port	RS-232 (RxD, TxD and GND); Non-isolated	
ttyO4	RS-232 (RxD, TxD and GND); Non-isolated	
ttyO2	RS-485 (Data+, Data-); Non-isolated	
ttyO5	RS-485 (Data+, Data-); 2500 VDC isolated	
Ethernet Port	10/100 Base-TX, RJ-45 port (Auto-negotiating, Auto MDI/MDI-X, LED indicators), PoE (IEEE 802.3af, Class 1)	
I/O Expansion Slots		
I/O Expansion Bus	Yes, one optional XV-board	
COM Port Formats		
Speed	921.6 Kbps Max.	
Data Bit	5, 6, 7, 8	
Parity	None, Even, Odd, Space, Mark	
Stop Bit	1, 1.5, 2	
Pull High/Low Resistor	1kΩ default, 150kΩ (for RS-485)	
Software		
Protocol	ICMP, IPv4/v6, TCP, UDP, DHCP, BOOTP,SSH, FTP, SFTP, DNS, DDNS, SNMP V1/V2c/V3, HTTP, SMTP, ARP, PPPoE	
Configuration method	Web, Serial Console, SSH Console	
Management	SNMP MIB-II	

Models	LP-5231	LP-5231M
Power Input		
Input Range	+12 ~ +48 VDC	
Consumption	4.8 W	
Mechanism		
Casing	Plastic	Metal
Dimensions (W x L x H)	91 mm x 132 mm x 52 mm	117 mm x 126 mm x 58 mm
Installation	DIN-Rail Mounting	DIN-Rail Mounting/Wall Mounting
Environment		
Operating Temperature	-25 ~ +75 °C	
Storage Temperature	-40 ~ +80 °C	
Humidity	10 ~ 90% RH, non-condensing	

Models	LP-5231PM-3GWA	LP-5231PM-4GE	LP-5231PM-4GC
System Software			
OS	Linux Kernel 3.2.14		
Embedded Service	SFTP/FTP server, Web server, SSH		
SDK Provided	Standard LinPAC SDK for Linux by GNU C language		
CPU Module			
CPU	32-bit RISC, 1 GHz		
SDRAM	512MB		
Flash	512MB		
FRAM	64KB		
Expansion Flash Memory	microSD socket with one 4 GB microSD card (support up to 32 GB microSDHC card)		
RTC (Real Time Clock)	Provide second, minute, hour, date, day of week, month, year		
64-bit Hardware Serial Number	Yes, for Software Copy Protection		
Dual Watchdog Timers	Yes		
LED Indicators	2 LED for Power and Running; 2 LED for user defined		
Rotary Switch	Yes(0~9)		
VGA & Communication Interface			
VGA	VGA x 1 (max. resolution up to 1280 x 1024)		
USB 2.0(host)	1		
Console Port	RS-232 (RxD, TxD and GND); Non-isolated		
ttyO4	RS-232 (RxD, TxD and GND); Non-isolated		
ttyO2	RS-485 (Data+, Data-); Non-isolated		
ttyO5	RS-485 (Data+, Data-); 2500 VDC isolated		
Ethernet Port	10/100 Base-TX, RJ-45 port (Auto-negotiating, Auto MDI/MDI-X, LED indicators), PoE (IEEE 802.3af, Class 1)		
I/O Expansion Slots			
I/O Expansion Bus	Yes, one optional XV-board		
COM Port Formats			
Speed	921.6 Kbps Max.		
Data Bit	5, 6, 7, 8		
Parity	None, Even, Odd, Space, Mark		
Stop Bit	1, 1.5, 2		
Pull High/Low Resistor	1kΩ default, 150kΩ (for RS-485)		
Software			
Protocol	ICMP, IPv4/v6, TCP, UDP, DHCP, BOOTP,SSH, FTP, SFTP, DNS, DDNS, SNMP V1/V2c/V3, HTTP, SMTP, ARP, PPPoE		
Configuration method	Web, Serial Console, SSH Console		
Management	SNMP MIB-II		
Power Input			
Input Range	+12 ~ +48 VDC		
Consumption	4.8 W		

Models	LP-5231PM-3GWA	LP-5231PM-4GE	LP-5231PM-4GC
GSM System			
Frequency Band	GSM: 850/900/1800/1900 MHz		
GPRS Connectivity	GPRS class 12/10; GPRS station class B		
Data GPRS	Downlink transfer: Max. 85.6 kbps; Uplink transfer: Max 42.8k bps		
3G System			
Frequency Band	WCDMA 850/900/1900/2100 MHz	WCDMA 850/900/2100 MHz	WCDMA 900/2100 TD-SCDMA 1900/2100 CDMA2000 (BC0) 800
Data Transmission	Downlink transfer: Max. 85.6 kbps Uplink transfer: Max 42.8k bps	DC-HSPA+ Download: Max. 42 Mbps Upload: Max 5.76Mbps TD-SCDMA Download: Max. 4.2 Mbps Upload: Max 2.2 Mbps CDMA2000 EVDO Download: Max. 14.7 Mbps Upload: Max 5.4 Mbps	
4G System			
Frequency Band	-	FDD LTE: B1/B3/B5/B7/B8/B20	FDD LTE: B1/B3/B8 TDD LTE: B38/B39/B40/B41
Data Transmission	-	Download Max 100Mbps; Upload Max 50Mbps	
GPS System(Option)			
Support Channels	32		
Protocol Support	NMEA 0183		
Mechanism			
Casing	Metal		
Dimensions (W x L x H)	117 mm x 126 mm x 58 mm		
Installation	DIN-Rail Mounting/Wall Mounting		
Environment			
Operating Temperature	-25 ~ +75 °C		
Storage Temperature	-40 ~ +80 °C		
Humidity	10 ~ 90% RH, non-condensing		

1.4 Ordering Information

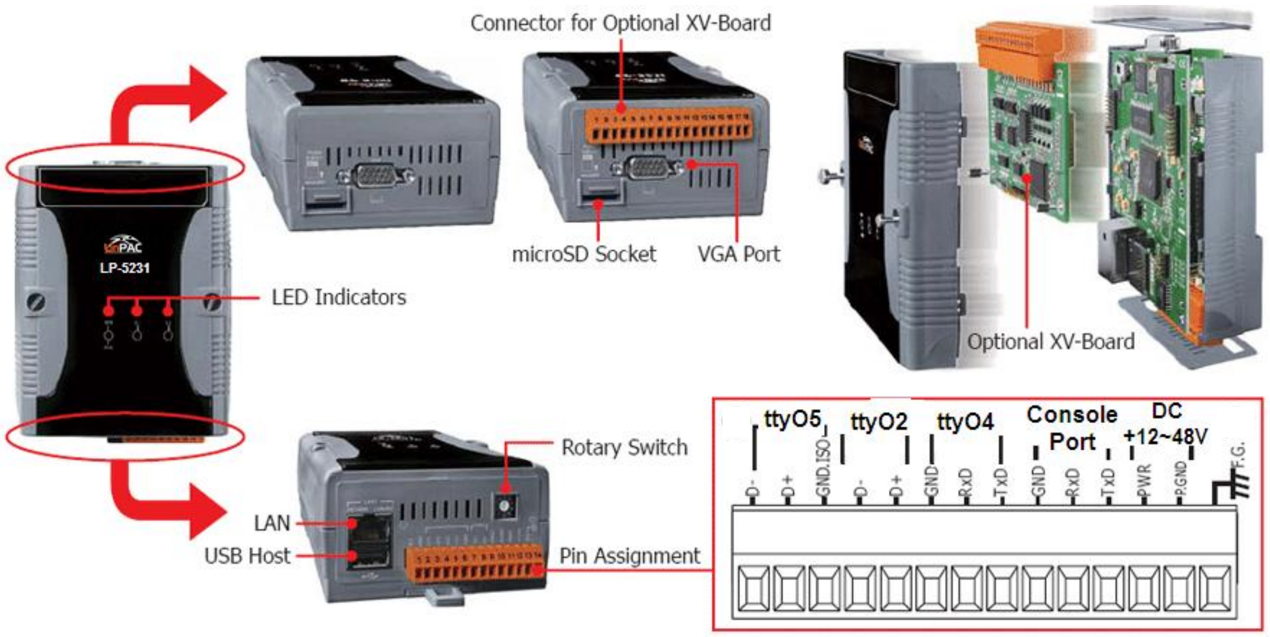
LP-5231	PAC with Linux OS and one LAN port (Plastic Case)(RoHS)
LP-5231M	PAC with Linux OS and one LAN port (Metal Case)(RoHS)
LP-5231PM-3GWA	PAC with Linux OS、one LAN port GPS and 3G module (WCDMA) (Metal Case) (RoHS)
LP-5231PM-4GE	PAC with Linux OS、one LAN port GPS and 4G module (Frequency Band for Europe) (Metal Case) (RoHS)
LP-5231PM-4GC	PAC with Linux OS、one LAN port GPS and 4G module (Frequency Band for China) (Metal Case) (RoHS)

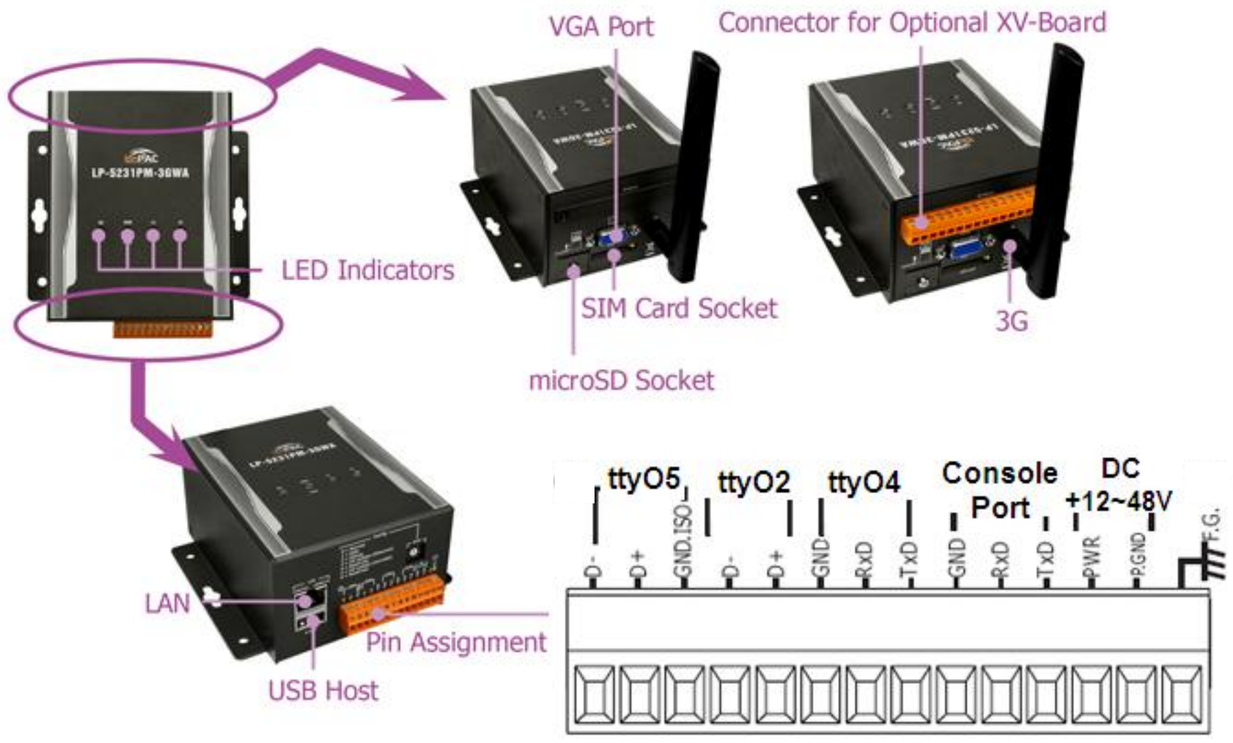
1.5 Ordering Information Option Accessories

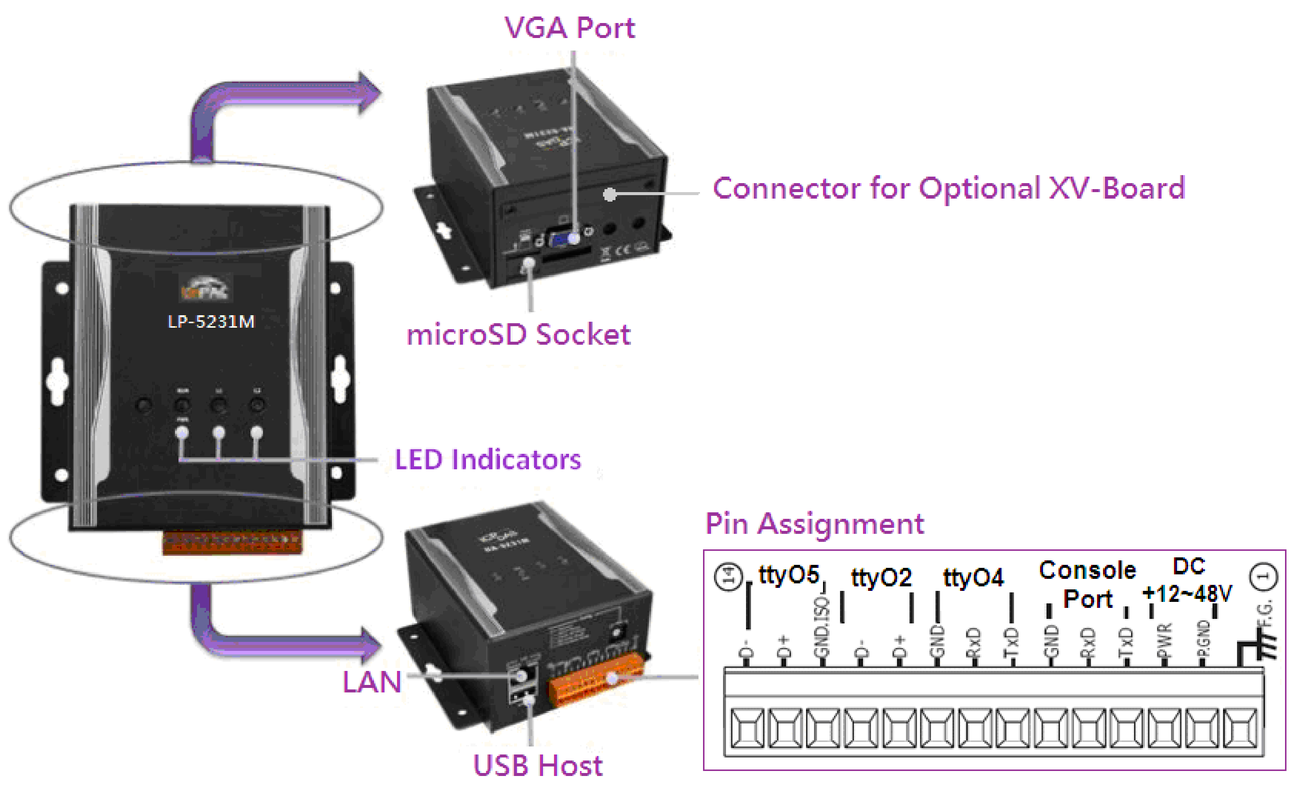
XV-Board	Add-on I/O Expansion Board
GPSU06U-6 CR	24 VDC/0.25 A, 6 W Power Supply
MDR-20-24 CR	24 VDC/1 A, 24 W Power Supply with DIN-R
DIN-KA52F-48 CR	48V/0.52A, 25 W Power Supply with DIN-Rail Mounting (RoHS, for NS-205PSE)
CA-0903	9-Pin Female D-Sub and RS-232 Connector Cable, 30 cm Cable
CA-0910	9-Pin Female D-Sub and 3-wire RS-232 Cable, 1 m Cable
NS-205 CR	Unmanaged 5-port Industrial Ethernet Switch (RoHS)
NS-205PSE CR	Unmanaged Ethernet Switch with 4 PoE Ports and 1 RJ-45 Uplink (RoHS)

2. Hardware Introduction

2.1 Hardware Feature







1. Ethernet Port

The LP-5231 contains one Ethernet port for use with network devices.

2. LED Indicators

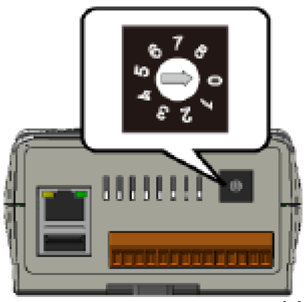
LED Indicators	Color	Meaning
PWR	Red	Power is on
RUN	Green	OS is running
L1	Green/Red	User programmable LED
L2	Green/Red	User programmable LED
3G (LP-5231PM-3GWA only)	Green	The 3G LED indicates that the antenna is connected to 3G network

3. USB Port

The LP-5231 contains one USB port that allow support for the USB devices such as mouse, keyboard or an external USB hard drive.

4. Rotary Switch

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



5. VGA Connector

A VGA connector is a 3-row 15-pin connector that can be used with a variety of supported VGA resolutions (max. resolution up to 1280x1024).

6. SIM Card Slot (LP-5231PM-3GWA/LP-5231PM-4GE/4GC Only)

The SIM card expansion slot is an interface that is used to insert SIM card for GSM/3G system on LP-5231PM-3GWA or GSM/3G/4G system on LP-5231PM-4GE/4GC.

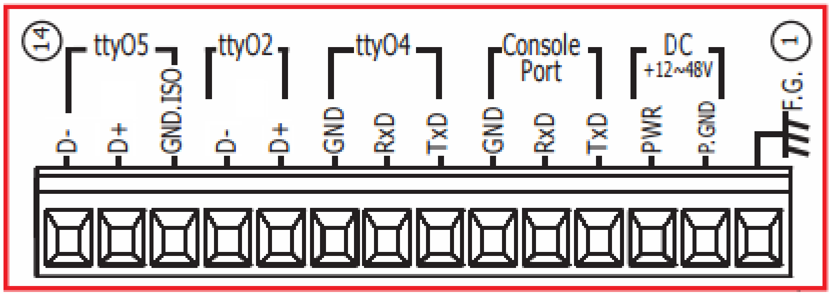
7. SD Card Slot

The SD card expansion slot is an interface that is used to access and download information on a SD card to a LP-5231.

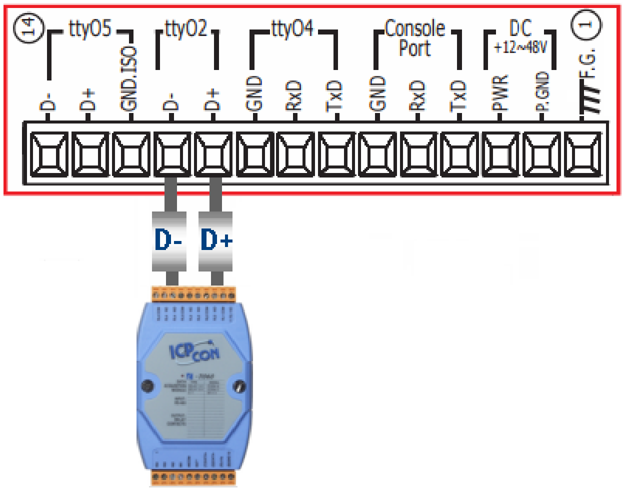
Please note:

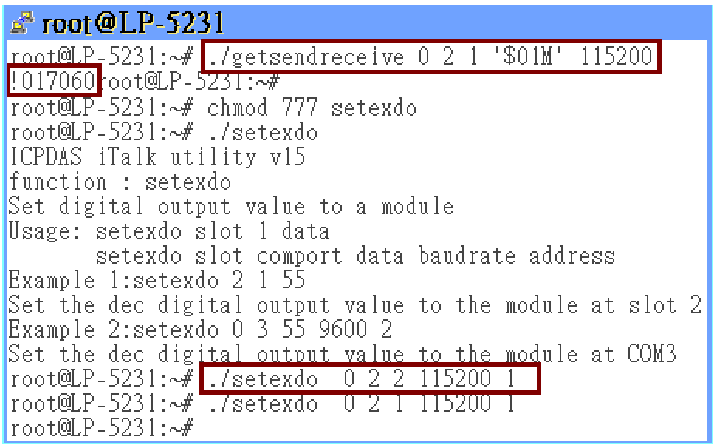
- **The flash and microSD disk have a finite number of program-erase cycles. Important information should always be backed up on other media or storage device for long-term safekeeping.**
- **The Li-batteries can continually supply power to the 512 KB SRAM to retain the data for 10 years (It is recommended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

8. Serial Ports



Device name	Definition in LP-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




```
root@LP-5231
root@LP-5231:~# ./getsendreceive 0 2 1 '$01M' 115200
!017060root@LP-5231:~#
root@LP-5231:~# chmod 777 setexdo
root@LP-5231:~# ./setexdo
ICPDAS iTalk utility v15
function : setexdo
Set digital output value to a module
Usage: setexdo slot 1 data
       setexdo slot comport data baudrate address
Example 1:setexdo 2 1 55
Set the dec digital output value to the module at slot 2
Example 2:setexdo 0 3 55 9600 2
Set the dec digital output value to the module at COM3
root@LP-5231:~# ./setexdo  0 2 2 115200 1
root@LP-5231:~# ./setexdo  0 2 1 115200 1
root@LP-5231:~#
```

9. XV-Board (optional)

LP-5231 has one expansion I/O slots to expand the functions. For more detailed information about the XV-board specifications, please refer to Appendix A. XV-Board Modules.

Model: XV107

Model: XV107A

3. Your First Program-Hello World

3.1 SDK Compiler Installation

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

❑ LinPAC AM335x SDK (Linaro GCC toolchain, Libraries, header, examples files, etc.)

❑ Code::Blocks project file (Windows platform only)

❑ Basic Linux commands (Windows platform only)

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

❑ Linux
  - ◆ Download/Install LinPAC AM335x SDK on Linux

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

The latest version of the LP-52xx SDK (hereinafter referred to as LP-52xx) can be downloaded from:

❑ **For Windows systems** (Extract the **.exe** file into to the **C:\** driver.)
Download the linpac_am335x_sdk_for_windows.exe file from:
ftp://ftp.icpdas.com.tw/pub/cd/linpac/napdos/lp-9x2x/sdk/linpac_am335x_sdk_for_windows.exe

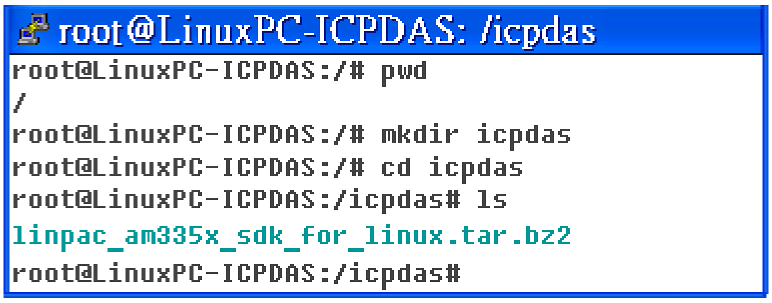
❑ **For Linux systems** (Extract the .bz2 file into to the **root (/) directory**.)
Download the linpac_am335x_sdk_for_linux.tar.bz2 file from:
ftp://ftp.icpdas.com.tw/pub/cd/linpac/napdos/lp-9x2x/sdk/linpac_am335x_sdk_for_linux.tar.bz2
Note: We recommend user to change user ID to become root by '**sudo**' or '**su**' command.

NOTE:

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

3.1.1 Download / Install SDK on Linux

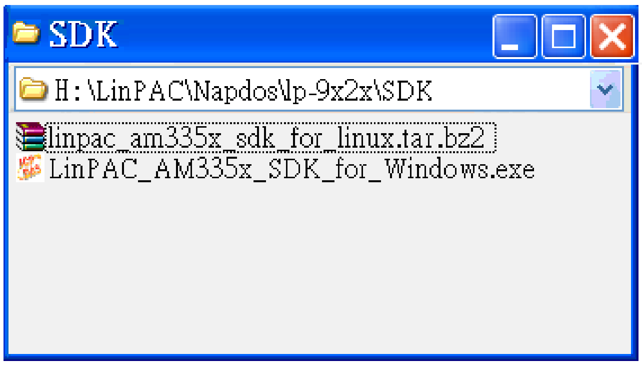
1. To create a "**icpdas**" folder in root directory, maybe you need to change the root user by '**sudo**' or '**su**' command.



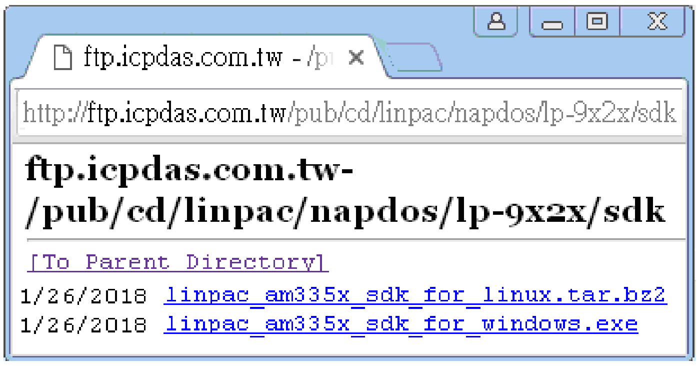
```
root@LinuxPC-ICPDAS: /icpdas
root@LinuxPC-ICPDAS:/# pwd
/
root@LinuxPC-ICPDAS:/# mkdir icpdas
root@LinuxPC-ICPDAS:/# cd icpdas
root@LinuxPC-ICPDAS:/icpdas# ls
linpac_am335x_sdk_for_linux.tar.bz2
root@LinuxPC-ICPDAS:/icpdas#
```

Fig. 3-1

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



```
SDK
H:\LinPAC\Napdos\lp-9x2x\SDK
linpac_am335x_sdk_for_linux.tar.bz2
LinPAC_AM335x_SDK_for_Windows.exe
```

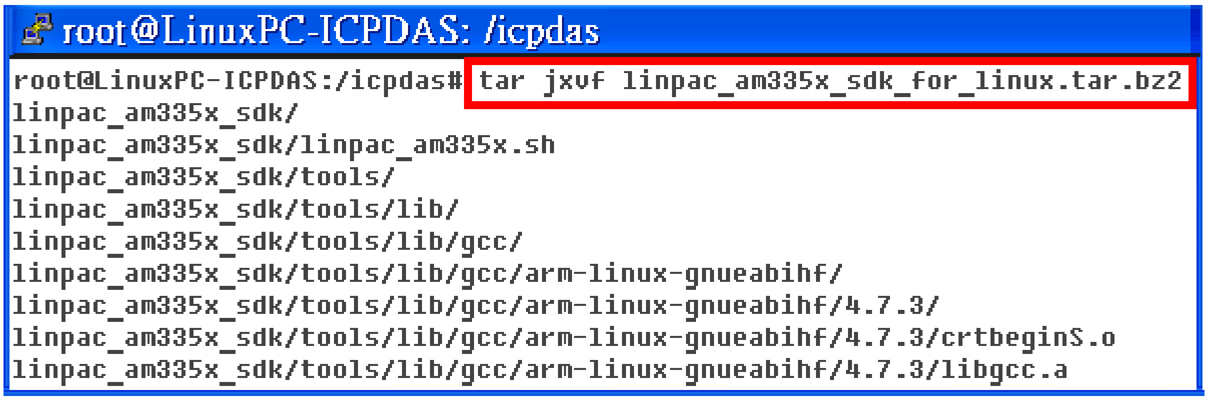


```
ftp.icpdas.com.tw - /p
http://ftp.icpdas.com.tw/pub/cd/linpac/napdos/lp-9x2x/sdk
ftp.icpdas.com.tw-
/pub/cd/linpac/napdos/lp-9x2x/sdk
[To Parent Directory]
1/26/2018 linpac_am335x_sdk_for_linux.tar.bz2
1/26/2018 linpac_am335x_sdk_for_windows.exe
```

Fig. 3-2 Fig. 3-3

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

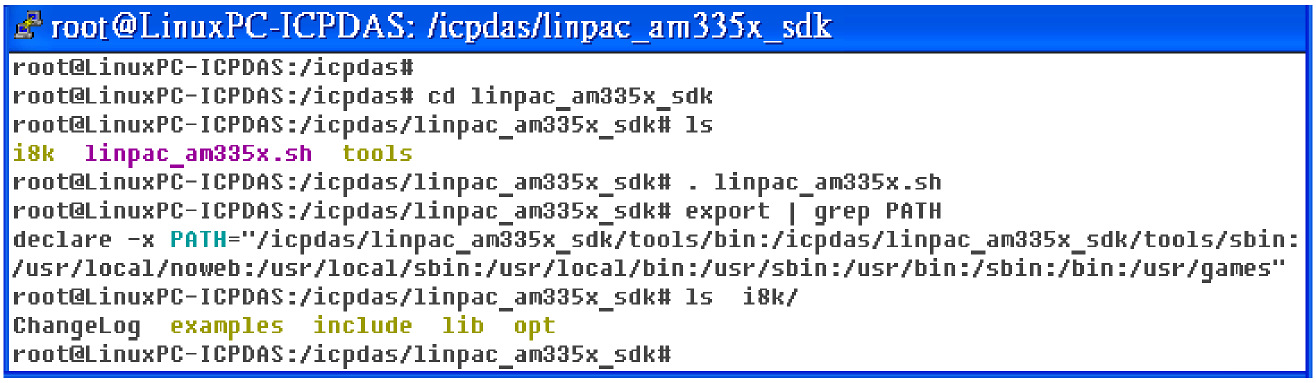
# tar **jxvf** **linpac_am335x_sdk_for_linux.tar.bz2**



```
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
```

Fig. 3-4

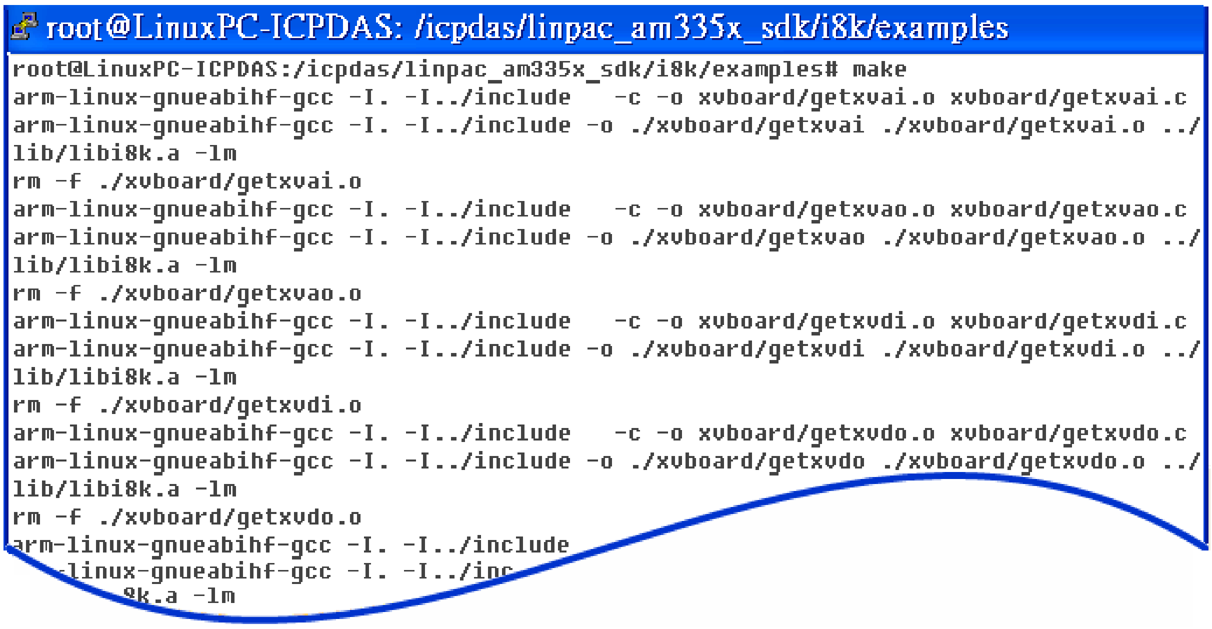
4. Before compile the program, you need to set LinPAC AM335x SDK path in environment variables: using the provided environment variable script, which is called **linpac_am335x.sh** (refer to Fig.3-5).



```
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#
```

Fig.3-5

5. Type ‘**make**’ on the command line it will execute the compile command according to the Makefile (refer to Fig.3-6).



```
root@LinuxPC-ICPDAS: /icpdas/linpac_am335x_sdk/i8k/examples
root@LinuxPC-ICPDAS:/icpdas/linpac_am335x_sdk/i8k/examples# make
arm-linux-gnueabihf-gcc -I. -I../include   -c -o xvboard/getxvai.o xvboard/getxvai.c
arm-linux-gnueabihf-gcc -I. -I../include -o ./xvboard/getxvai ./xvboard/getxvai.o ../
lib/libi8k.a -lm
rm -f ./xvboard/getxvai.o
arm-linux-gnueabihf-gcc -I. -I../include   -c -o xvboard/getxvao.o xvboard/getxvao.c
arm-linux-gnueabihf-gcc -I. -I../include -o ./xvboard/getxvao ./xvboard/getxvao.o ../
lib/libi8k.a -lm
rm -f ./xvboard/getxvao.o
arm-linux-gnueabihf-gcc -I. -I../include   -c -o xvboard/getxvdi.o xvboard/getxvdi.c
arm-linux-gnueabihf-gcc -I. -I../include -o ./xvboard/getxvdi ./xvboard/getxvdi.o ../
lib/libi8k.a -lm
rm -f ./xvboard/getxvdi.o
arm-linux-gnueabihf-gcc -I. -I../include   -c -o xvboard/getxvdo.o xvboard/getxvdo.c
arm-linux-gnueabihf-gcc -I. -I../include -o ./xvboard/getxvdo ./xvboard/getxvdo.o ../
lib/libi8k.a -lm
rm -f ./xvboard/getxvdo.o
arm-linux-gnueabihf-gcc -I. -I../include
  -linux-gnueabihf-gcc -I. -I../inc
       8k.a -lm
```

Fig. 3-6

3.1.2 Download / Install SDK on Windows

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

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

2. Open the \napdos\LP-9x21\SDK\ folder and double-click the icon for the "LinPAC_AM335x_SDK_for_Windows.exe" file, when the Setup Wizard is displayed, click the "Next>" button to continue, refer to Fig. 3-7 and Fig.3-8.

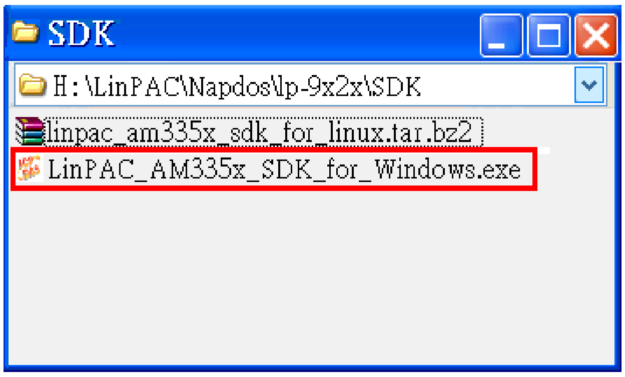
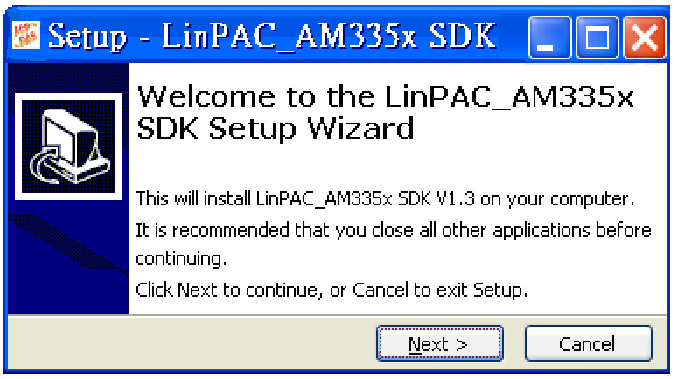

SDK
H:\LinPAC\Napdos\lp-9x2x\SDK
linpac_am335x_sdk_for_linux.tar.bz2
LinPAC_AM335x_SDK_for_Windows.exe

Fig. 3-7



Welcome to the LinPAC_AM335x SDK Setup Wizard

This will install LinPAC_AM335x SDK V1.3 on your computer.

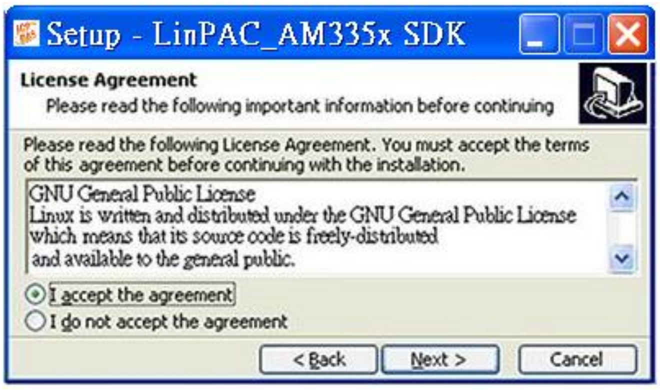
It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Fig. 3-8

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

4. Select Start Menu Folder option and then click the "Next" button, refer to Fig. 3-10.



License Agreement

Please read the following important information before continuing

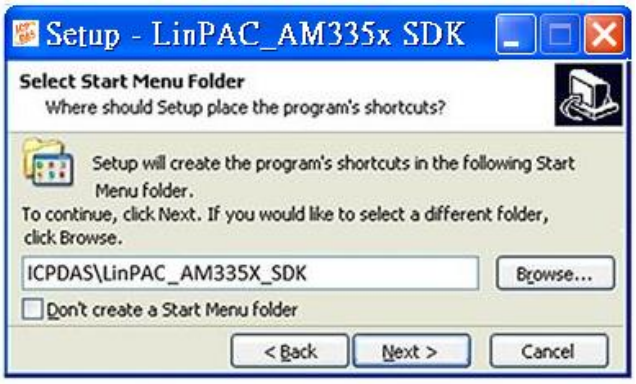
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

GNU General Public License
Linux is written and distributed under the GNU General Public License which means that its source code is freely-distributed and available to the general public.

[✓ I accept the agreement]

I do not accept the agreement

Fig. 3-9



Select Start Menu Folder

Where should Setup place the program's shortcuts?

Setup will create the program's shortcuts in the following Start Menu folder.

To continue, click Next. If you would like to select a different folder, click Browse.

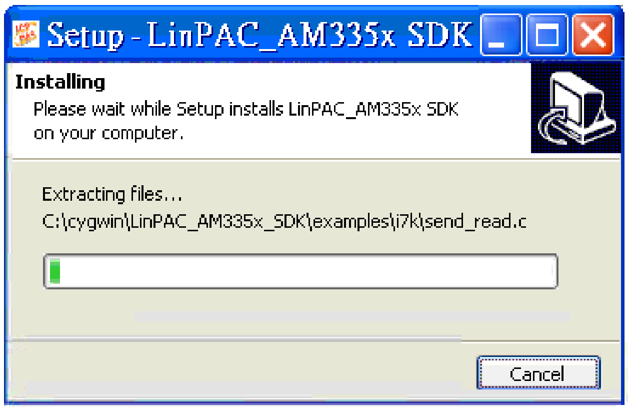
ICPDAS\LinPAC_AM335X_SDK

Don't create a Start Menu folder

Fig. 3-10

5. The LinPAC AM335x SDK files will be extracted and installed and a progress bar will be displayed to indicate the status, refer to Fig 3-11.

6. Once the software has been successfully installed, click the "Finish" button to complete the development toolkit installation, refer to Fig. 3-12.



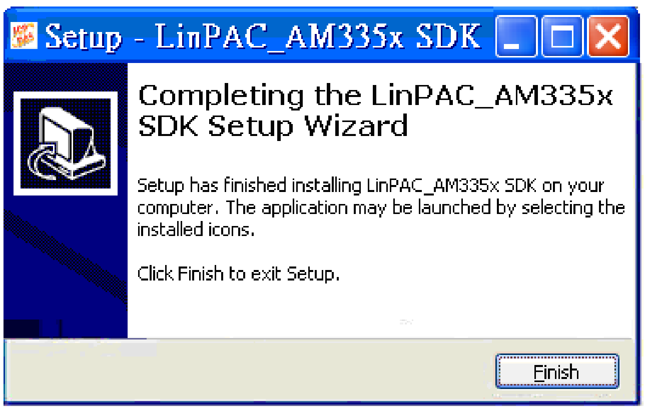
Installing

Please wait while Setup installs LinPAC_AM335x SDK on your computer.

Extracting files...

C:\cygwin\LinPAC_AM335x_SDK\examples\i7k\send_read.c

Fig. 3-11



Completing the LinPAC_AM335x SDK Setup Wizard

Setup has finished installing LinPAC_AM335x SDK on your computer. The application may be launched by selecting the installed icons.

Click Finish to exit Setup.

Fig. 3-12

7. Open the LinPAC AM335x SDK installation directory, the default data directory location is **"C:\cygwin\"**, user can see the contents of folder. Refer to Fig 3-13 and Fig 3-14.

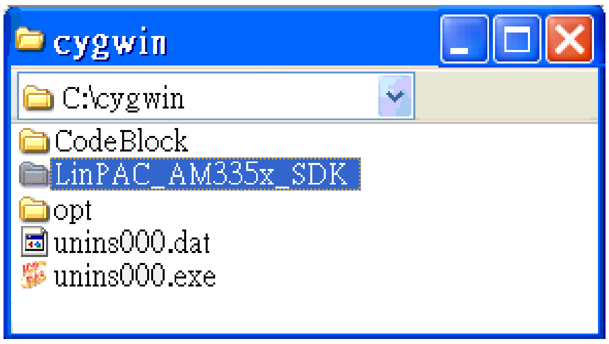


Fig. 3-13

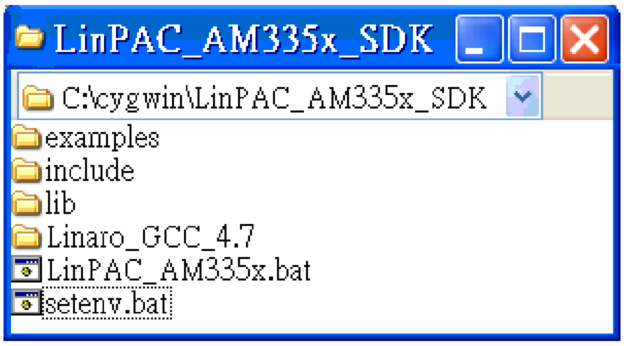


Fig. 3-14

8. 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 LP-52xx to be compiled. Refer to Fig. 3-15 and Fig 3-16.

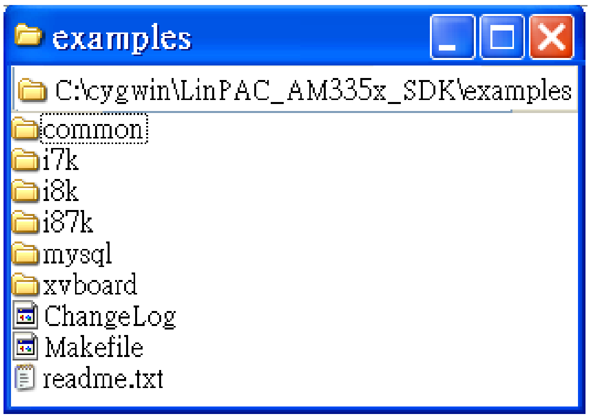
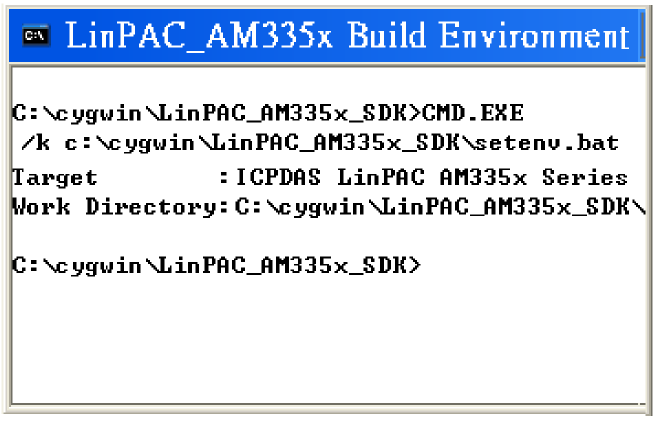


Fig. 3-15

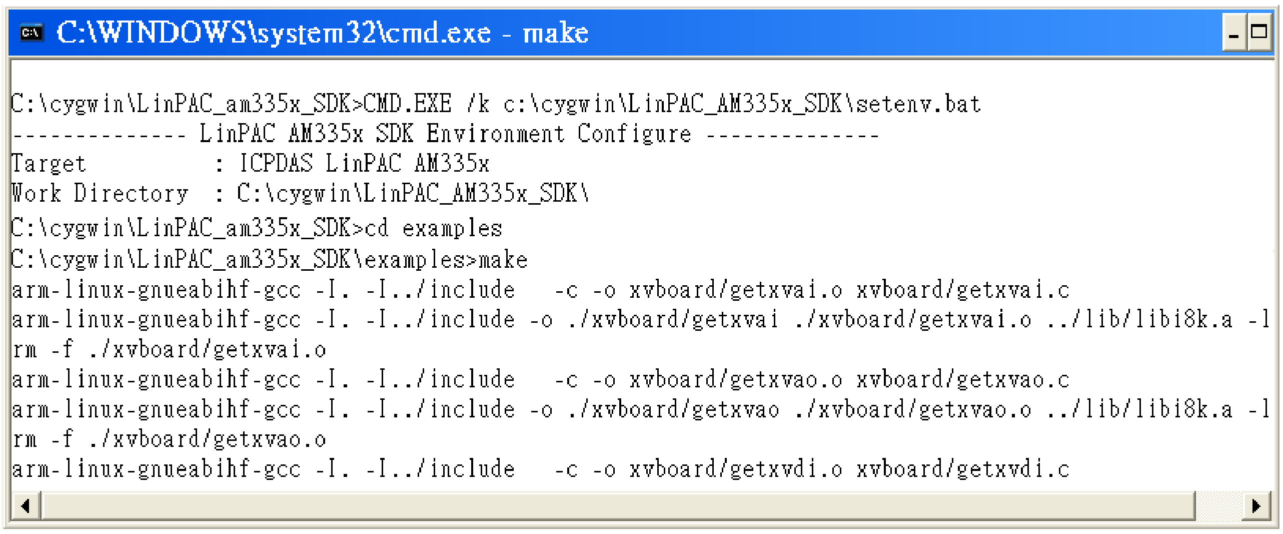


```
C:\cygwin\LinPAC_AM335x_SDK>CMD.EXE
 /k c:\cygwin\LinPAC_AM335x_SDK\setenv.bat
Target         :ICPDAS LinPAC AM335x Series
Work Directory:C:\cygwin\LinPAC_AM335x_SDK\

C:\cygwin\LinPAC_AM335x_SDK>
```

Fig. 3-16

9. Type "**make**". A Command Prompt window will then be displayed that allows applications for the LP-52xx to be compiled. Refer to Fig. 3-17


C:\WINDOWS\system32\cmd.exe - make

```
C:\cygwin\LinPAC_am335x_SDK>CMD.EXE /k c:\cygwin\LinPAC_AM335x_SDK\setenv.bat
-------------- LinPAC AM335x SDK Environment Configure --------------
Target          : ICPDAS LinPAC AM335x
Work Directory  : C:\cygwin\LinPAC_AM335x_SDK\
C:\cygwin\LinPAC_am335x_SDK>cd examples
C:\cygwin\LinPAC_am335x_SDK\examples>make
arm-linux-gnueabihf-gcc -I. -I../include   -c -o xvboard/getxvai.o xvboard/getxvai.c
arm-linux-gnueabihf-gcc -I. -I../include -o ./xvboard/getxvai ./xvboard/getxvai.o ../lib/libi8k.a -l
rm -f ./xvboard/getxvai.o
arm-linux-gnueabihf-gcc -I. -I../include   -c -o xvboard/getxvao.o xvboard/getxvao.c
arm-linux-gnueabihf-gcc -I. -I../include -o ./xvboard/getxvao ./xvboard/getxvao.o ../lib/libi8k.a -l
rm -f ./xvboard/getxvao.o
arm-linux-gnueabihf-gcc -I. -I../include   -c -o xvboard/getxvdi.o xvboard/getxvdi.c
```

Fig. 3-17

3.1.3 Integrating SDK with Code::Blocks IDE

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

Step 1: Download Code::Blocks IDE

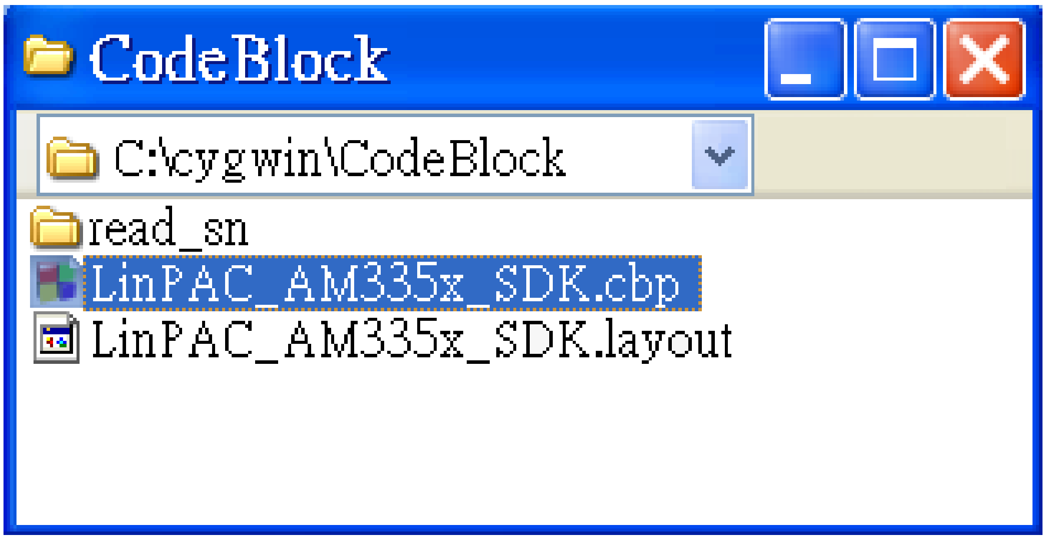
- ❑ Go to this website: http://www.codeblocks.org/downloads/binaries
- ❑ Go to the Windows 2000/ XP / Vista / 7 section, and download Windows version.

Step 2: Install Code::Block IDE

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

Step 3: Running in Code::Block IDE

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



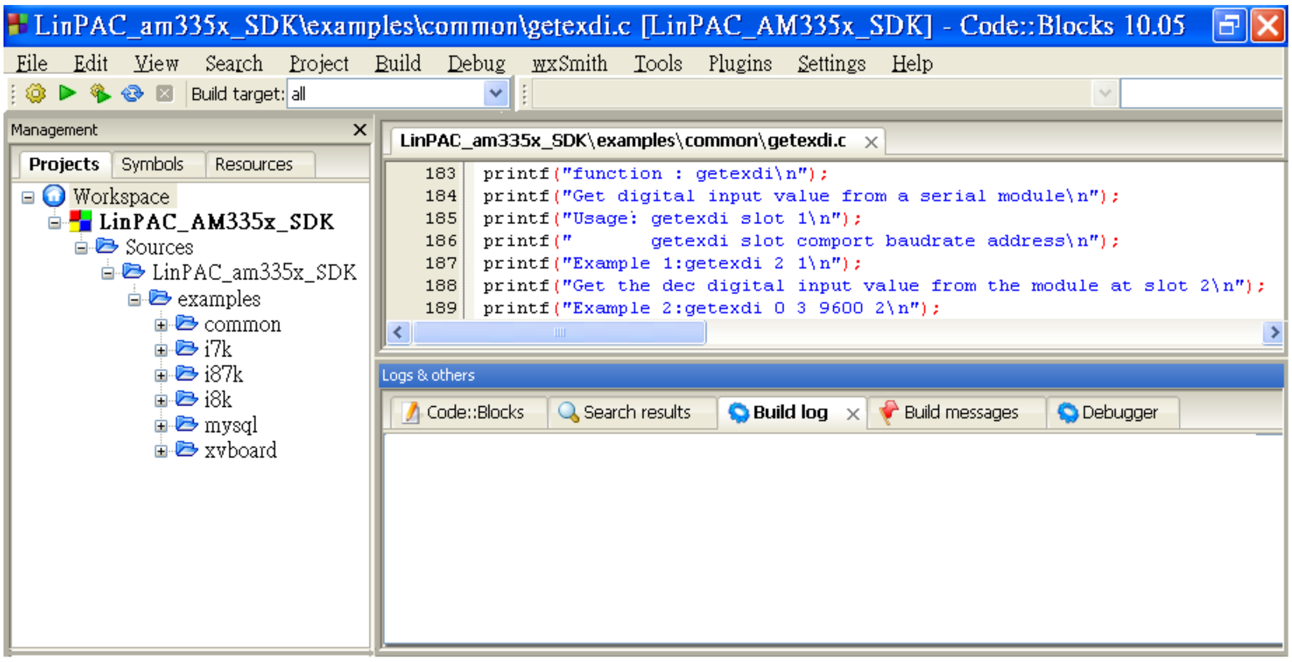
CodeBlock

C:\cygwin\CodeBlock

read_sn
LinPAC_AM335x_SDK.cbp
LinPAC_AM335x_SDK.layout

Fig. 3-18

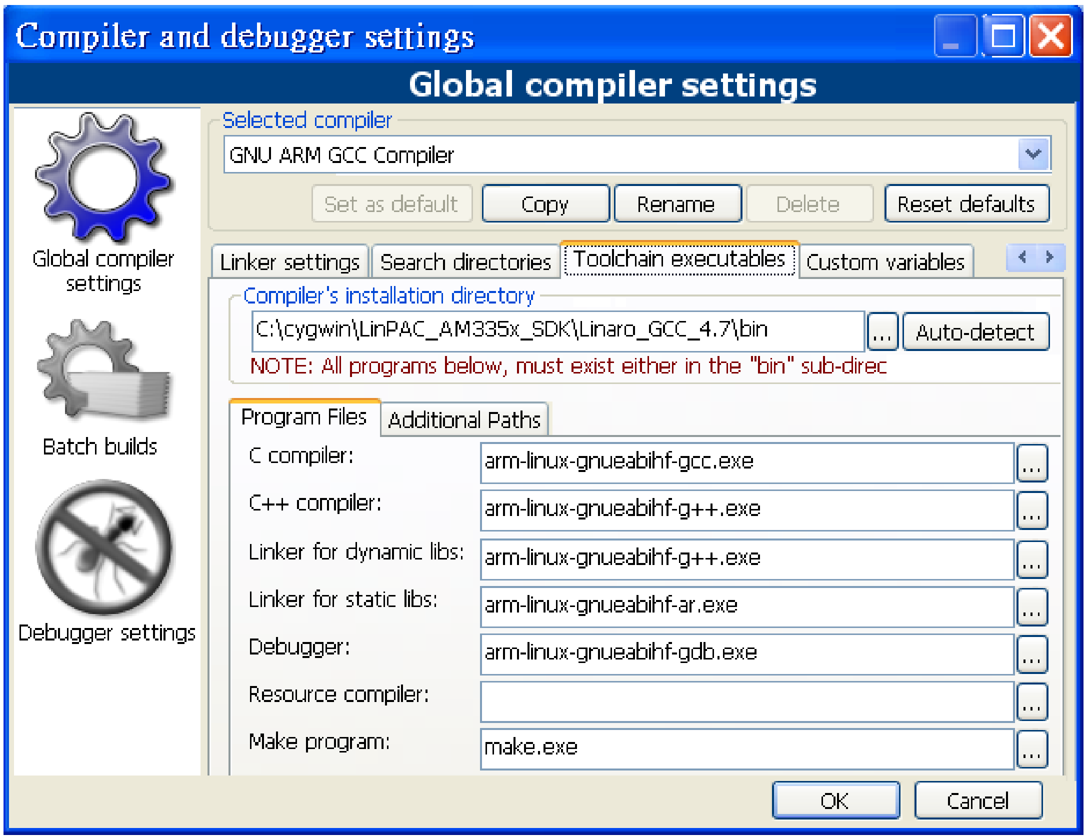
- ❑ Following window will come up (refer to Fig.3-19):


LinPAC_am335x_SDK\examples\common\getexdi.c [LinPAC_AM335x_SDK] - Code::Blocks 10.05
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help
Build target: all
Management
Projects Symbols Resources
Workspace
LinPAC_AM335x_SDK
Sources
LinPAC_am335x_SDK
examples
common
i7k
i87k
i8k
mysql
xvboard
LinPAC_am335x_SDK\examples\common\getexdi.c
Logs & others
Code::Blocks Search results Build log Build messages Debugger

```
printf("function : getexdi\n");
printf("Get digital input value from a serial module\n");
printf("Usage: getexdi slot 1\n");
printf("       getexdi slot comport baudrate address\n");
printf("Example 1:getexdi 2 1\n");
printf("Get the dec digital input value from the module at slot 2\n");
printf("Example 2:getexdi 0 3 9600 2\n");
```

Fig. 3-19

- ❑ Check Compiler settings for Linaro GCC cross compiler : Click "Settings" > "Compiler" > "Toolchain executables tab" (refer to Fig.3-20) :



Selected compiler: GNU ARM GCC Compiler

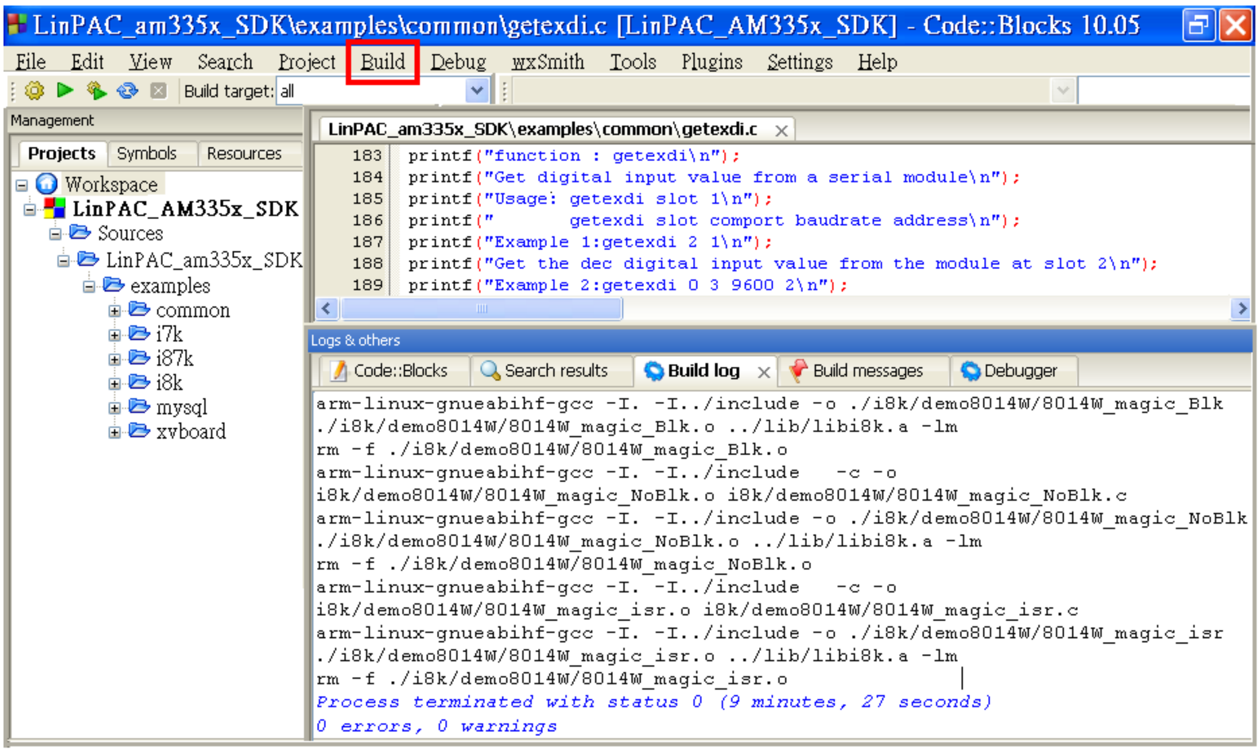
Compiler's installation directory: C:\cygwin\LinPAC_AM335x_SDK\Linaro_GCC_4.7\bin

NOTE: All programs below, must exist either in the "bin" sub-direc

C compiler:	arm-linux-gnueabihf-gcc.exe
C++ compiler:	arm-linux-gnueabihf-g++.exe
Linker for dynamic libs:	arm-linux-gnueabihf-g++.exe
Linker for static libs:	arm-linux-gnueabihf-ar.exe
Debugger:	arm-linux-gnueabihf-gdb.exe
Resource compiler:	
Make program:	make.exe

Fig. 3-20

- ❑ Click Build options, and it will compile the LinPAC_AM335x project completely (refer to Fig.3-21).


LinPAC_am335x_SDK\examples\common\getexdi.c [LinPAC_AM335x_SDK] - Code::Blocks 10.05
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help
Build target: all
Management
Projects Symbols Resources
Workspace
LinPAC_AM335x_SDK
Sources
LinPAC_am335x_SDK
examples
common
i7k
i87k
i8k
mysql
xvboard
LinPAC_am335x_SDK\examples\common\getexdi.c
Logs & others
Code::Blocks Search results Build log Build messages Debugger

```
printf("function : getexdi\n");
printf("Get digital input value from a serial module\n");
printf("Usage: getexdi slot 1\n");
printf("       getexdi slot comport baudrate address\n");
printf("Example 1:getexdi 2 1\n");
printf("Get the dec digital input value from the module at slot 2\n");
printf("Example 2:getexdi 0 3 9600 2\n");
```

```
arm-linux-gnueabihf-gcc -I. -I../include -o ./i8k/demo8014W/8014W_magic_Blk
./i8k/demo8014W/8014W_magic_Blk.o ../lib/libi8k.a -lm
rm -f ./i8k/demo8014W/8014W_magic_Blk.o
arm-linux-gnueabihf-gcc -I. -I../include   -c -o
i8k/demo8014W/8014W_magic_NoBlk.o i8k/demo8014W/8014W_magic_NoBlk.c
arm-linux-gnueabihf-gcc -I. -I../include -o ./i8k/demo8014W/8014W_magic_NoBlk
./i8k/demo8014W/8014W_magic_NoBlk.o ../lib/libi8k.a -lm
rm -f ./i8k/demo8014W/8014W_magic_NoBlk.o
arm-linux-gnueabihf-gcc -I. -I../include   -c -o
i8k/demo8014W/8014W_magic_isr.o i8k/demo8014W/8014W_magic_isr.c
arm-linux-gnueabihf-gcc -I. -I../include -o ./i8k/demo8014W/8014W_magic_isr
./i8k/demo8014W/8014W_magic_isr.o ../lib/libi8k.a -lm
rm -f ./i8k/demo8014W/8014W_magic_isr.o
Process terminated with status 0 (9 minutes, 27 seconds)
0 errors, 0 warnings
```

Fig. 3-21

【Note】

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

- ❑ Double-click the shortcut icon for the "LP-52xx Build Environment" (Refer to Fig. 3-22).

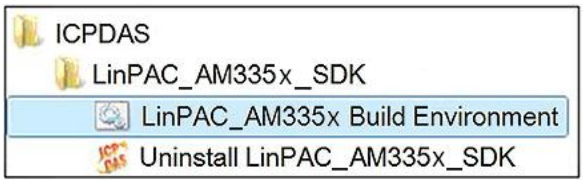
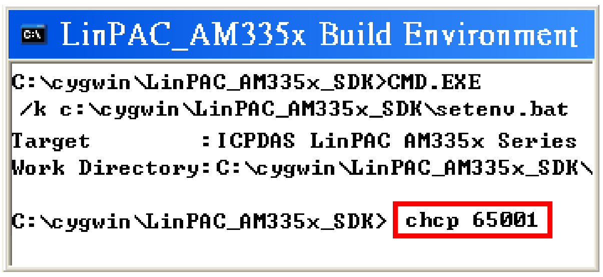


Fig. 3-22

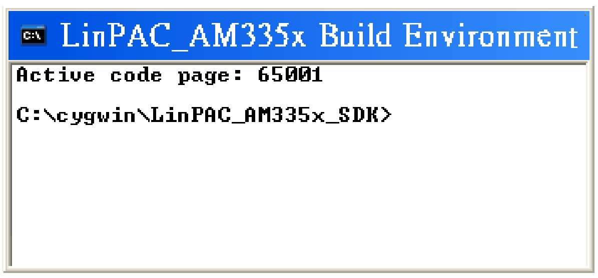
- ❑ Type command: chcp 65001 (Refer to Fig. 3-23 and Fig 3-24).



```
C:\cygwin\LinPAC_AM335x_SDK>CMD.EXE
 /k c:\cygwin\LinPAC_AM335x_SDK\setenv.bat
Target         :ICPDAS LinPAC AM335x Series
Work Directory:C:\cygwin\LinPAC_AM335x_SDK\

C:\cygwin\LinPAC_AM335x_SDK> chcp 65001
```

Fig. 3-23



```
Active code page: 65001

C:\cygwin\LinPAC_AM335x_SDK>
```

Fig. 3-24

3.2 First Program in LP-52xx

This section will discuss some of the techniques that are adopted in the LinPACAM335x SDK, including detailed explanations that describe how to easily use the SDK. The LinPAC_AM335x SDK is based on Cygwin and is also a Linux-like environment for Microsoft Windows systems, and provides a powerful GCC cross-compiler and an IDE (Integrated Development Environment) that enables LP-52xx applications to be quickly developed. Therefore, once an application has been created, the LinPAC_AM335x SDK can be used to compile it into an executable file that can be run on the LP-52xx embedded controller.

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

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

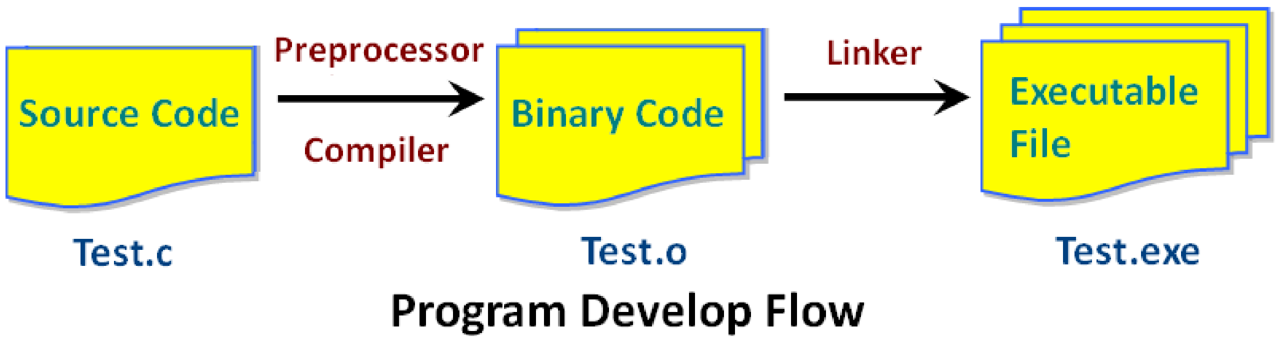
Type the following code and save as helloworld.c. Note that the code is case-sensitive.

```
#include <stdio.h>       /* Include the header file that allows functions to be used */

int main()
{
        printf("ICPDAS LP5231 hello world!\n");  /* Print the message on the screen */
        return 0;
}
```

3.2.1 Compile helloworld.c file to executable file

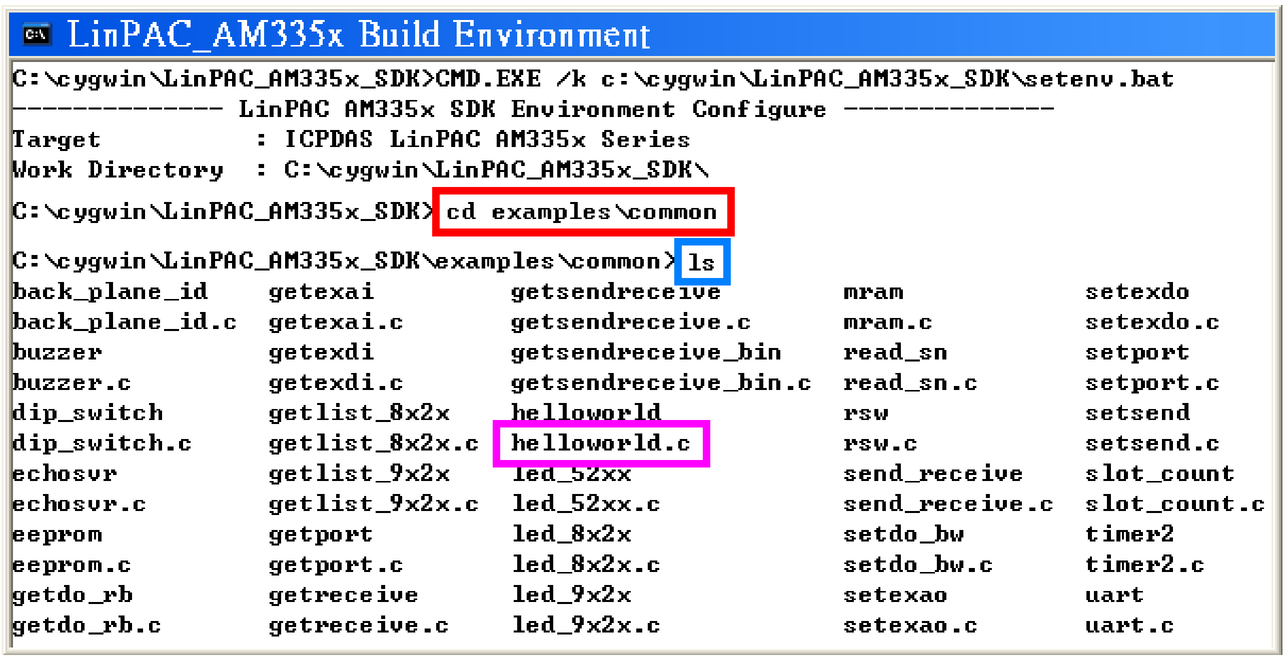
This section describes how to 1) compile the helloworld.c file to the executable file, 2) transfer the executable file (helloworld.exe or helloworld) to the LP-52xx using FTP, and 3) execute this file via the SSH Server on the LP-52xx.



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

1. Open the LinPAC_AM335x SDK (refer to step 8 in section 2.1) and then type "**cd examples/common**" to change the path to C:/cygwin/LinPAC_AM335x_SDK/examples/common.

2. Type "dir/w" or "ls" command and to display the contents of the directory and confirm that the helloworld.c file is present. Refer to Fig. 3-25 for more details.



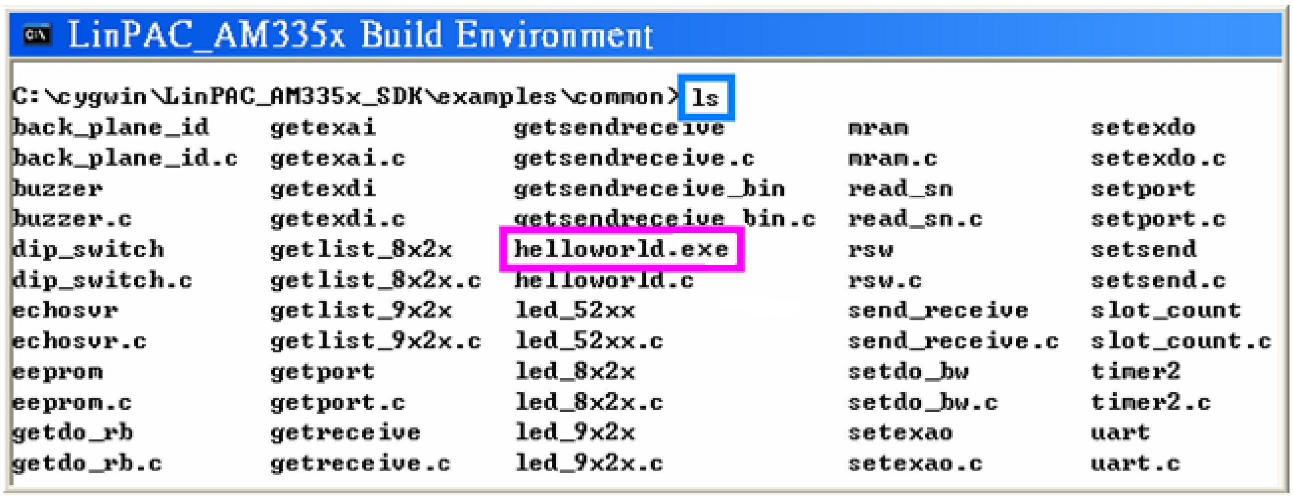
```
LinPAC_AM335x Build Environment
C:\cygwin\LinPAC_AM335x_SDK>CMD.EXE /k c:\cygwin\LinPAC_AM335x_SDK\setenv.bat
-------------- LinPAC AM335x SDK Environment Configure --------------
Target            : ICPDAS LinPAC AM335x Series
Work Directory    : C:\cygwin\LinPAC_AM335x_SDK\

C:\cygwin\LinPAC_AM335x_SDK>cd examples\common

C:\cygwin\LinPAC_AM335x_SDK\examples\common>ls
back_plane_id     getexai          getsendreceive          mram             setexdo
back_plane_id.c   getexai.c        getsendreceive.c        mram.c           setexdo.c
buzzer            getexdi          getsendreceive_bin      read_sn          setport
buzzer.c          getexdi.c        getsendreceive_bin.c    read_sn.c        setport.c
dip_switch        getlist_8x2x     helloworld              rsw              setsend
dip_switch.c      getlist_8x2x.c   helloworld.c            rsw.c            setsend.c
echosvr           getlist_9x2x     led_52xx                send_receive     slot_count
echosvr.c         getlist_9x2x.c   led_52xx.c              send_receive.c   slot_count.c
eeprom            getport          led_8x2x                setdo_bw         timer2
eeprom.c          getport.c        led_8x2x.c              setdo_bw.c       timer2.c
getdo_rb          getreceive       led_9x2x                setexao          uart
getdo_rb.c        getreceive.c     led_9x2x.c              setexao.c        uart.c
```

Fig. 3-25

3. Type the command "arm-linux-gnueabihf-gcc –o helloworld.exe helloworld.c" to compile helloworld.c into helloworld.exe, then type "dir/w" or "ls" command to display the contents of the directory and confirm that the helloworld.exe file has been created. (Refer to Fig. 3-26)



LinPAC_AM335x Build Environment

```
C:\cygwin\LinPAC_AM335x_SDK\examples\common> ls
back_plane_id     getexai          getsendreceive          mram            setexdo
back_plane_id.c   getexai.c        getsendreceive.c        mram.c          setexdo.c
buzzer            getexdi          getsendreceive_bin      read_sn         setport
buzzer.c          getexdi.c        getsendreceive_bin.c    read_sn.c       setport.c
dip_switch        getlist_8x2x     helloworld.exe          rsw             setsend
dip_switch.c      getlist_8x2x.c   helloworld.c            rsw.c           setsend.c
echosvr           getlist_9x2x     led_52xx                send_receive    slot_count
echosvr.c         getlist_9x2x.c   led_52xx.c              send_receive.c  slot_count.c
eeprom            getport          led_8x2x                setdo_bw        timer2
eeprom.c          getport.c        led_8x2x.c              setdo_bw.c      timer2.c
getdo_rb          getreceive       led_9x2x                setexao         uart
getdo_rb.c        getreceive.c     led_9x2x.c              setexao.c       uart.c
```

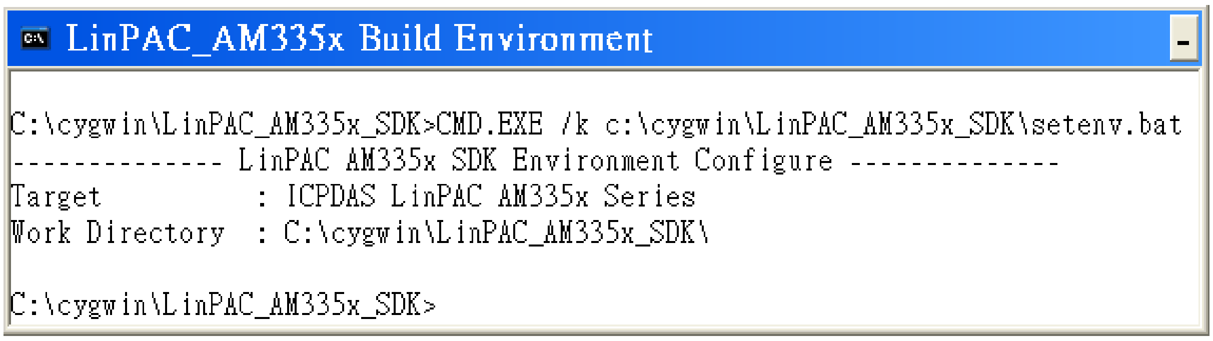
Fig. 3-26

3.2.2 Transfer helloworld.exe to the LP-52xx

Two methods can be used to transfer files to the LP-52xx:

＜Method one＞ Using a "<u>Command Prompt</u>"

(1) Open a "DOS Command Prompt" or LP-52xx Build Environment (C:\cygwin\LP-52xx_SDK\LP-52xx.bat) and type the ftp IP Address of the LP-52xx, for example: ftp 192.168.0.200 to establish a connection to the FTP Server on the LP-52xx. When prompted, type the User_Name (default value is "**root**") and Password (default value is "**icpdas**") to establish a connection to the LP-52xx.



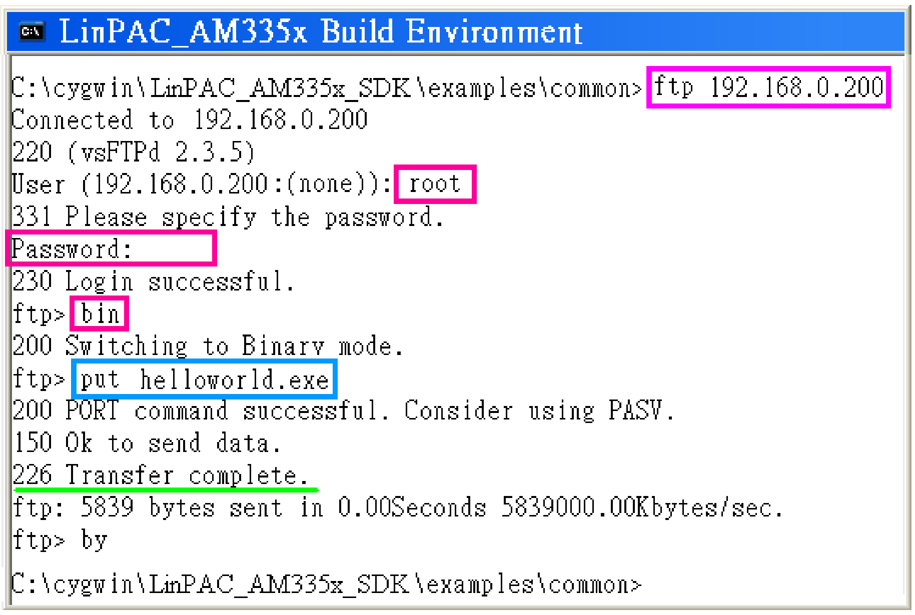
```
C:\cygwin\LinPAC_AM335x_SDK>CMD.EXE /k c:\cygwin\LinPAC_AM335x_SDK\setenv.bat
-------------- LinPAC AM335x SDK Environment Configure --------------
Target          : ICPDAS LinPAC AM335x Series
Work Directory  : C:\cygwin\LinPAC_AM335x_SDK\

C:\cygwin\LinPAC_AM335x_SDK>
```

Fig. 3-27

(2) Before transferring the files to the LP-52xx, type the "**bin**" command to ensure that the file is transferred to the LP-52xx in binary mode.

(3) Type the command "put **helloworld.exe**" to transfer the helloworld.exe file to the LP-52xx. Once the message "Transfer complete" is displayed, then transfer process has been completed. To disconnect from the LP-52xx, type the "bye" command to return to the PC console (refer to Fig. 3-28).



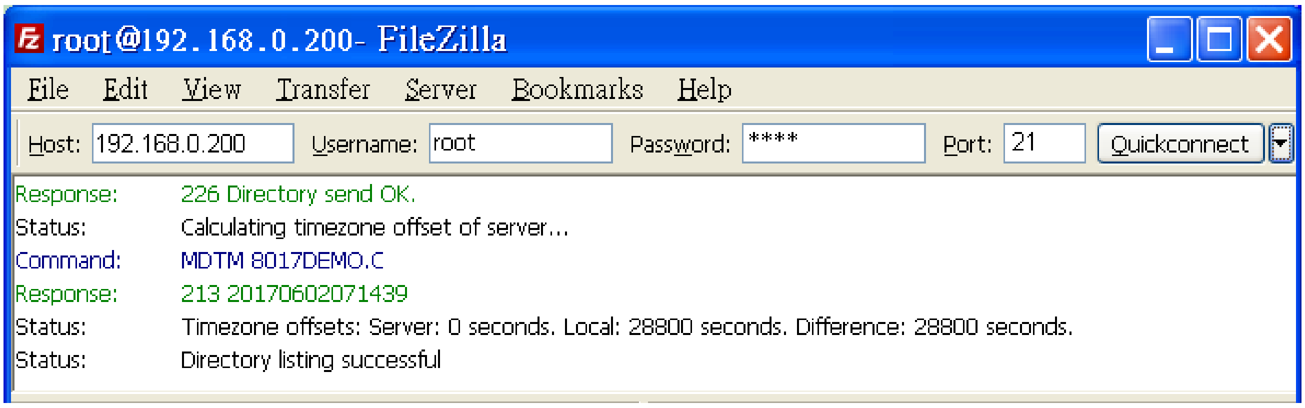
```
C:\cygwin\LinPAC_AM335x_SDK\examples\common>ftp 192.168.0.200
Connected to 192.168.0.200
220 (vsFTPd 2.3.5)
User (192.168.0.200:(none)): root
331 Please specify the password.
Password:
230 Login successful.
ftp> bin
200 Switching to Binary mode.
ftp> put helloworld.exe
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
ftp: 5839 bytes sent in 0.00Seconds 5839000.00Kbytes/sec.
ftp> by
C:\cygwin\LinPAC_AM335x_SDK\examples\common>
```

Fig. 3-28

＜**Method two**＞ **Using an FTP Client:**

(1) Open the FTP Software and add an FTP Host to the LP-52xx (for example, FileZilla - The free FTP solution for both client and server, https://filezilla-project.org/).

(2) Type the User_Name (default value is "**root**") and Password (default value is "**icpdas**"). Then click the "**Quickconnect** Connect" button to establish a connection to the ftp server on the LP-52xx (refer to Fig. 3-29).

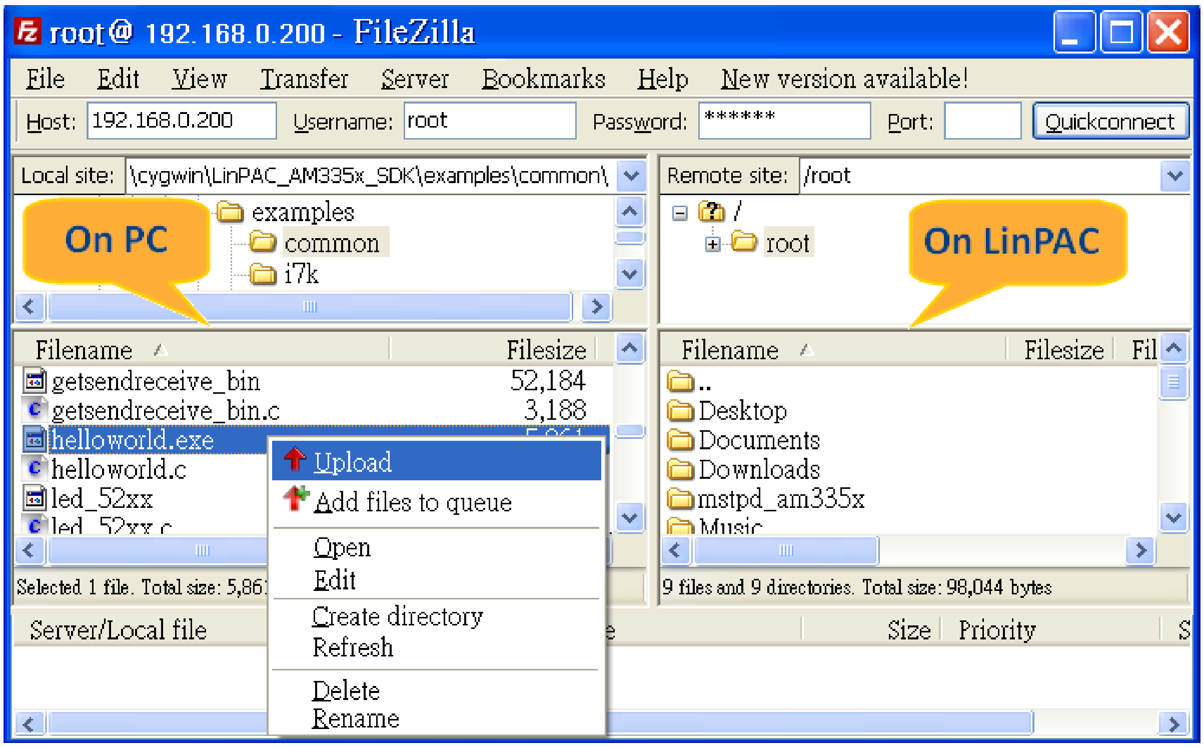


root@192.168.0.200- FileZilla
File Edit View Transfer Server Bookmarks Help
Host: 192.168.0.200 Username: root Password: ***** Port: 21 Quickconnect

```
Response:   226 Directory send OK.
Status:     Calculating timezone offset of server...
Command:    MDTM 8017DEMO.C
Response:   213 20170602071439
Status:     Timezone offsets: Server: 0 seconds. Local: 28800 seconds. Difference: 28800 seconds.
Status:     Directory listing successful
```

Fig. 3-29

(3) Upload the "Helloworld.exe" file to the LP-52xx (refer to Fig. 3-30).



root@ 192.168.0.200 - FileZilla
File Edit View Transfer Server Bookmarks Help New version available!
Host: 192.168.0.200 Username: root Password: ****** Port: Quickconnect
Local site: \cygwin\LinPAC_AM335x_SDK\examples\common\
Remote site: /root
On PC
On LinPAC
examples
common
i7k
/
root
Filename Filesize
getsendreceive_bin 52,184
getsendreceive_bin.c 3,188
helloworld.exe
helloworld.c
led_52xx
led_52xx.c
Upload
Add files to queue
Open
Edit
Create directory
Refresh
Delete
Rename
Filename Filesize Fil
..
Desktop
Documents
Downloads
mstpd_am335x
Music
Selected 1 file. Total size: 5,861
9 files and 9 directories. Total size: 98,044 bytes
Server/Local file Size Priority

Fig. 3-30

(4) Click the helloworld.exe file in the LP-52xx to select it and then right click the file icon and click the “File Permissions” option. In the Properties dialog box, type 777 into the Numeric textbox, and then click the OK button. Refer to Fig. 3-31 and Fig. 3-32 for more details.

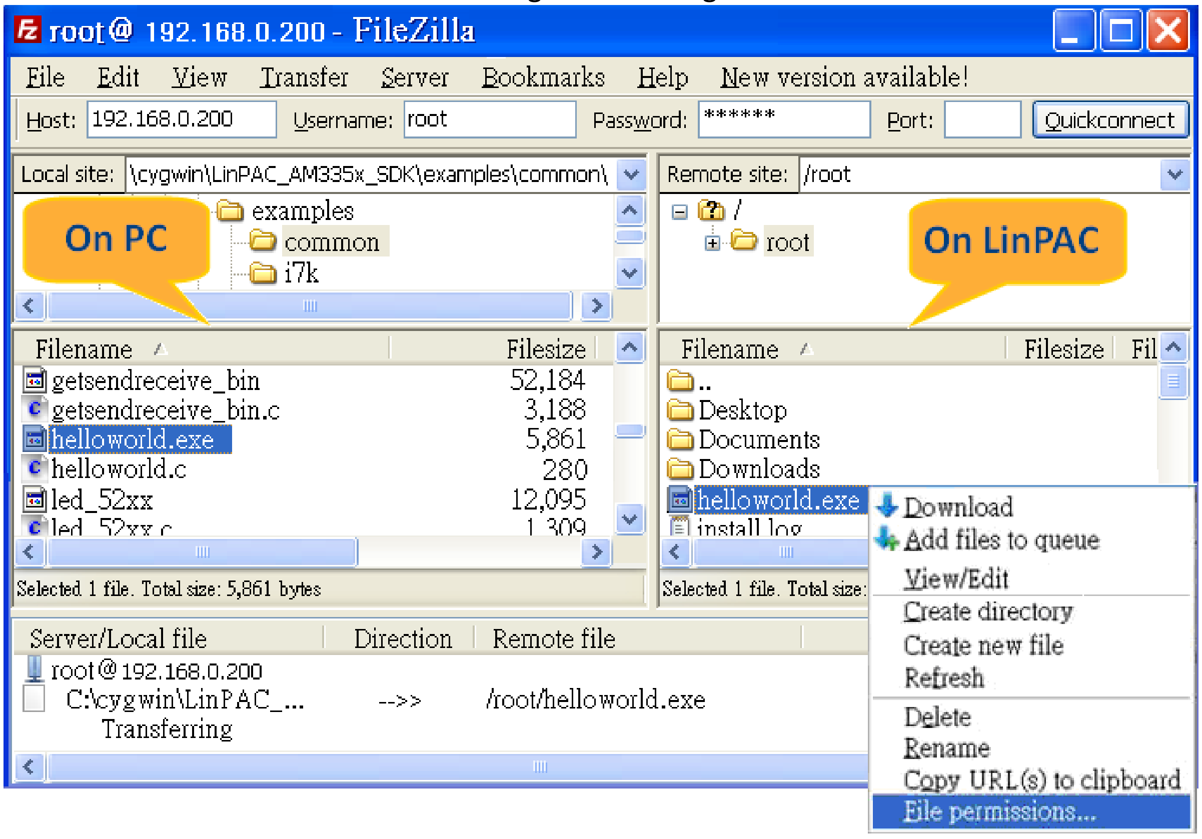
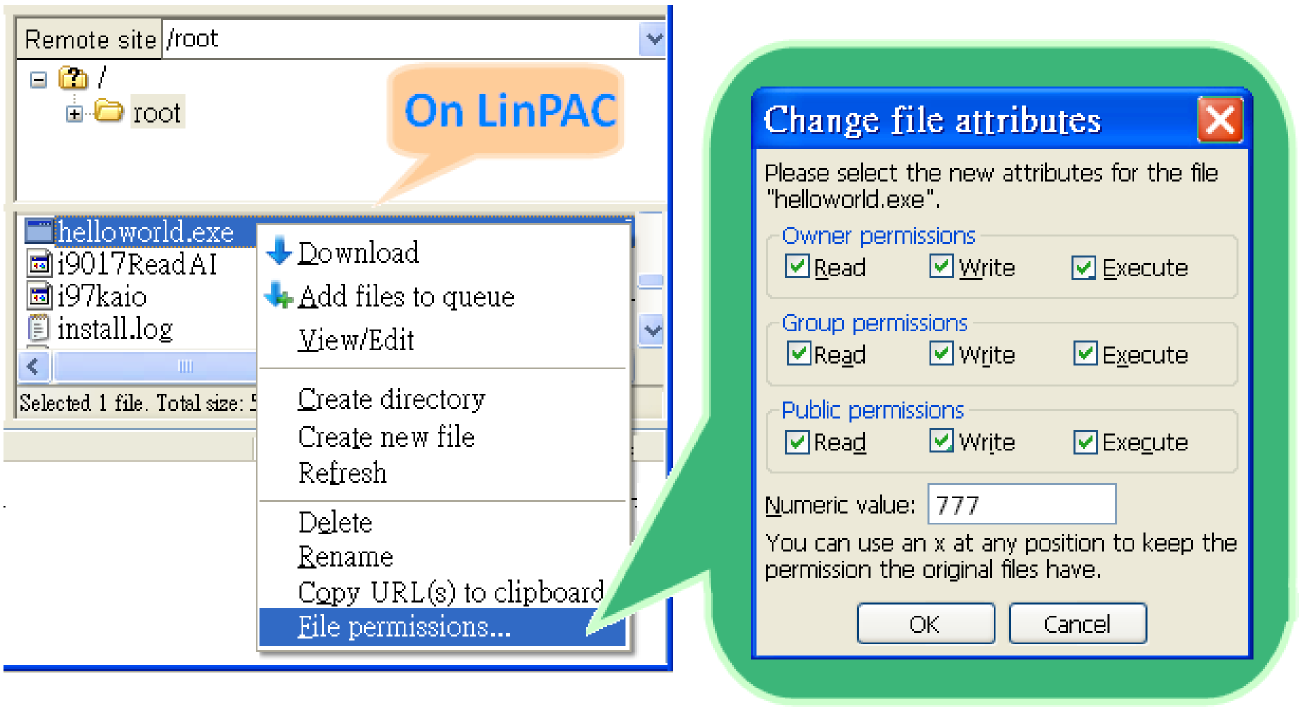

root@ 192.168.0.200 - FileZilla
File Edit View Transfer Server Bookmarks Help New version available!
Host: 192.168.0.200 Username: root Password: ****** Port: Quickconnect
Local site: \cygwin\LinPAC_AM335x_SDK\examples\common\
Remote site: /root
On PC
On LinPAC
examples
common
i7k
root
Filename Filesize
getsendreceive_bin 52,184
getsendreceive_bin.c 3,188
helloworld.exe 5,861
helloworld.c 280
led_52xx 12,095
Desktop
Documents
Downloads
helloworld.exe
install.log
Selected 1 file. Total size: 5,861 bytes
Server/Local file Direction Remote file
root@192.168.0.200
C:\cygwin\LinPAC_... -->> /root/helloworld.exe
Transferring
Download
Add files to queue
View/Edit
Create directory
Create new file
Refresh
Delete
Rename
Copy URL(s) to clipboard
File permissions...

Fig.3-31



Please select the new attributes for the file "helloworld.exe".

Owner permissions: ☑ Read ☑ Write ☑ Execute

Group permissions: ☑ Read ☑ Write ☑ Execute

Public permissions: ☑ Read ☑ Write ☑ Execute

Numeric value: 777

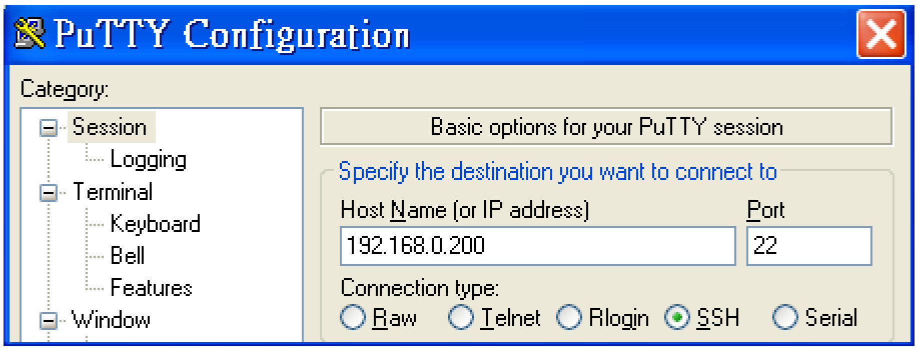
You can use an x at any position to keep the permission the original files have.

Fig.3-32

3.2.3 Use SSH to access LP-52xx and execute program

(1) Putty – the free PuTTY is an SSH and telnet client. Download PuTTY tool: http://www.putty.org/

(2) Open a "Putty Prompt" and type the IP Address of the LP-52xx, and the connection type is set to SSH. When prompted, type the User_Name and Password to establish a connection to the LP-52xx. If the "**#**" prompt character is displayed, it signifies that a connection to the telnet server on the LP-52xx has been successfully established (refer to Fig. 3-33).


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Basic options for your PuTTY session
Specify the destination you want to connect to

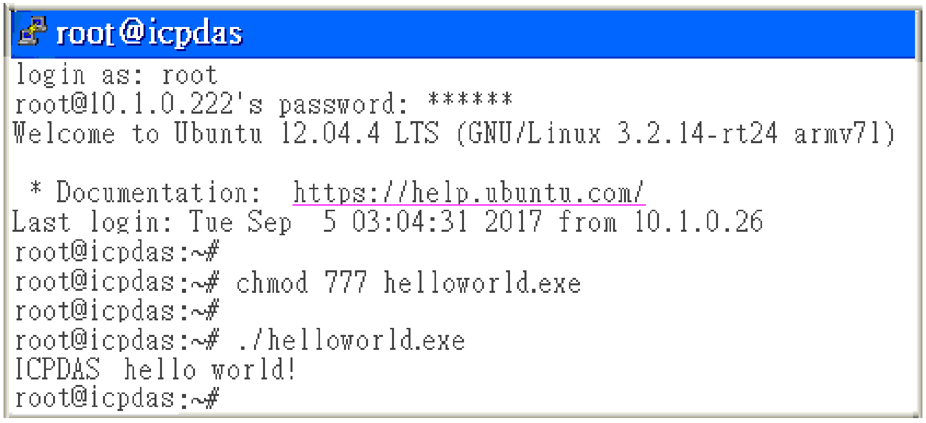
Host Name (or IP address): 192.168.0.200
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Fig.3-33

(2) Type the "ls -l" command to list all the files in the /root directory and verify that the helloworld.exe file is present.

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

The compile, transfer and execution processes are now complete (refer to Fig. 3-34).



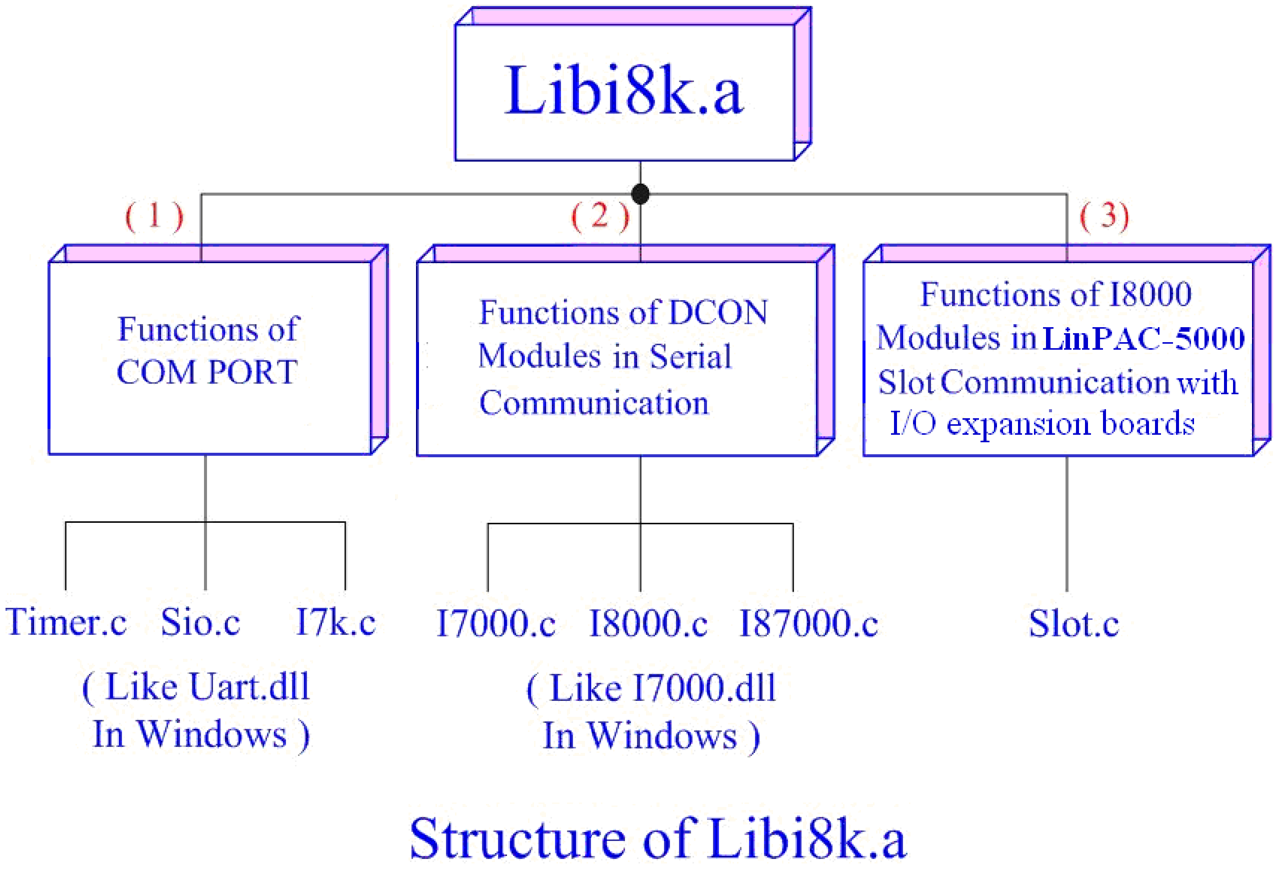
```
login as: root
root@10.1.0.222's password: ******
Welcome to Ubuntu 12.04.4 LTS (GNU/Linux 3.2.14-rt24 armv7l)

 * Documentation:  https://help.ubuntu.com/
Last login: Tue Sep  5 03:04:31 2017 from 10.1.0.26
root@icpdas:~#
root@icpdas:~# chmod 777 helloworld.exe
root@icpdas:~#
root@icpdas:~# ./helloworld.exe
ICPDAS hello world!
root@icpdas:~#
```

Fig.3-34

4. APIs and Demo References

In this section, we will focus on examples for the description of and application of the functions found in the Libi8k.a. The Libi8k.a functions can be clarified into 3 groups which are listed in Fig. 4-1.



Libi8k.a
(1)
(2)
(3)
Functions of COM PORT
Functions of DCON Modules in Serial Communication
Functions of I8000 Modules in LinPAC-5000 Slot Communication with I/O expansion boards
Timer.c
Sio.c
I7k.c
I7000.c
I8000.c
I87000.c
Slot.c
(Like Uart.dll In Windows)
(Like I7000.dll In Windows)
Structure of Libi8k.a

Fig.4-1

The following is an introduction to the functions for slot.c, which can be arranged into four main categories:

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

Note that when using a development tool to create develop applications, the msw.h file must be included in the header of the source program, and the Libi8k.a library file must also be linked. To control ICP DAS remote I/O modules such as the I-7K series modules via the /dev/ttyO2 (COM2), /dev/ttyO4 (COM4) or /dev/ttyO5 (COM5) ports on the LP-52xx, the functions are the same as those included in the DCON DLL, the functions are different and they are described in more detail below:

4.1 System Information Function

4.1.1 Open_Com

Description:

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

Syntax:

```
[ C ]

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

Parameter:

port :	[Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
baudrate:	[Input] 1200/2400/4800/9600/19200/38400/57600/115200
cDate :	[Input] Data5Bit, Data6Bit, Dat7Bit, Data8Bit
cParity :	[Input] NonParity, OddParity, EvenParity
cStop :	[Input] OneStopBit, TwoStopBit

Return Values:

0: The com port was successfully initialized.
Other: The initialization failed.
Refer to Chapter 4.5: "Error Code Definitions" for details of other returned values.

Examples:

```
Open_Com(COM2, 9600, Data8Bit, NonParity, OneStopBit);  //RS-485 port
```

Remark:

Device name	Definition in LP-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

4.1.2 Close_Com

Description:

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

Syntax:

```
                                        [ C ]

   BOOL Close_Com(char port)
```

Parameter:

port :	[Input] COM1, COM2, COM4, COM5 (1=COM1,2=COM2=/dev/ttyO2, 4=COM3=/dev/ttyO4, 5=COM5=/dev/ttyO5)

Return Value:

None

Example:

```
Close_Com (COM3);
```

Remark:

4.1.3 Send_Receive_Cmd

Description:

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

Syntax:

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

Parameter:

port :	[Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
szCmd:	[Input] Sending command string
szResult :	[Input] Receiving the response string from the modules
wTimeOut :	[Input] Communicating timeout setting, the unit=1ms
wChkSum :	[Input] 0=Disable, 1=Enable
*wT:	[Output] Total time of send/receive interval, unit=1 ms

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: “Error Code Definitions” for details of other returned values.

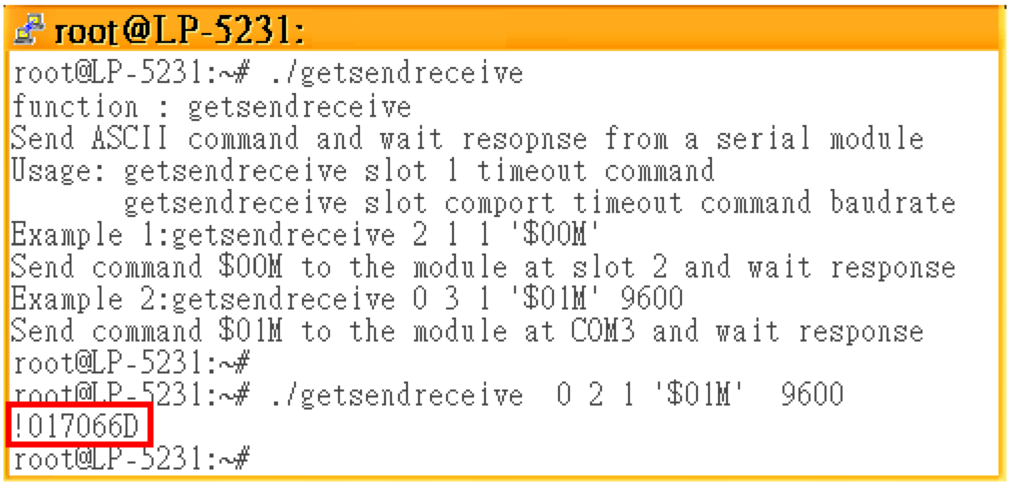
Examples:

```
char m_port =2;                    //I-7066D connect to LP-52xx via RS-485 port ; /dev/ttyO2
DWORD m_baudrate=9600;
WORD m_timeout=100;
WORD m_chksum=0;
WORD m_wT;
char m_szSend[40], m_szReceive[40];
int RetVal;
m_szSend[0] = '$';
m_szSend[1] = '0';
m_szSend[2] = '1';
```

```
m_szSend[3] = 'M';
m_szSend[4] = 0;
/* open device file */
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:

User can refer to LP-52xx SDK, locate the "getsendreceive.c" file in the C:\cygwin\LP-52xx_SDK\examples\common\ folder.



```
root@LP-5231:
root@LP-5231:~# ./getsendreceive
function : getsendreceive
Send ASCII command and wait resopnse from a serial module
Usage: getsendreceive slot 1 timeout command
       getsendreceive slot comport timeout command baudrate
Example 1:getsendreceive 2 1 1 '$00M'
Send command $00M to the module at slot 2 and wait response
Example 2:getsendreceive 0 3 1 '$01M' 9600
Send command $01M to the module at COM3 and wait response
root@LP-5231:~#
root@LP-5231:~# ./getsendreceive  0 2 1 '$01M'  9600
!017066D
root@LP-5231:~#
```

4.1.4 Send_Cmd

Description:

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

Syntax:

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

Parameter:

port :	[Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
szCmd :	[Input] Sending command string
wTimeOut :	[Input] Communicating timeout setting, the unit=1ms
wChkSum :	[Input] 0=Disable, 1=Enable

Return Value:

None

Examples:

```
char m_port=1, m_szSend[40];
DWORD m_baudrate=115200;
WORD m_timeout=100, m_chksum=0;
m_szSend[0] = '$';
m_szSend[1] = '0';
m_szSend[2] = '0';
m_szSend[3] = 'M';
Open_Com(m_port, m_baudrate, Data8Bit, NonParity, OneStopBit);
Send _Cmd(m_port, m_szSend, m_timeout, m_chksum);
Close_Com (m_port);
```

Remark:

4.1.5 Receive_Cmd

Description:

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

Syntax:

```
[ C ]

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

Parameter:

port :	[Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
szResult :	[Output] Sending command string
wTimeOut :	[Input] Communicating timeout setting, the unit=1ms
wChkSum :	[Input] 0=Disable, 1=Enable

Return Value:

None

Examples:

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

Remark:

4.1.6 Send_Binary

Description:

Send out the command string by fix length, which is controlled by the parameter “iLen”. The difference between this function and Send_cmd is that Send_Binary terminates the sending process by the string length “iLen” instead of the character "CR"(Carry return). Therefore, this function can send out command string with or without null character under the consideration of the command length. Besides, because of this function without any error checking mechanism (Checksum, CRC, LRC... etc.), users have to add the error checking information to the raw data by themselves if communication checking system is required.

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

Syntax:

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

Parameter:

port :	[Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
szCmd :	[Input] Sending command string
iLen :	[Input] The length of command string.

Return Value:

None

Examples:

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

Remark:

4.1.7 Receive_Binary

Description:

This function is applied to receive the fix length response. The length of the receiving response is controlled by the parameter “iLen”. The difference between this function and Receive_cmd is that Receive_Binary terminates the receiving process by the string length “iLen” instead of the character "CR"(Carry return). Therefore, this function can be used to receive the response string data with or without null character under the consideration of receiving length. Besides, because of this function without any error checking mechanism (checksum, CRC, LRC... etc.), users have to remove from the error checking information from the raw data by themselves if communication checking system is used.

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

Syntax:

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

Parameter:

port :	[Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
szResult :	[Input] Receiving the response string from the modules
wTimeOut :	[Input] Communicating timeout setting, the unit=1ms
wLen :	[Input] The length of command string.
*wT:	[Output] Total time of send/receive interval, unit=1 ms

Return Value:

None

Examples:

```
int m_length=10;
char m_port=2;
char m_szSend[40];
char m_szReceive[40];
DWORD m_baudrate=115200;
WORD m_wt;
WORD m_timeout=10;
WORD m_wlength=10;
```

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

Remark:

4.1.8 sio_open

Description:

This function is used to open and initiate a specified serial port in the LP-52xx. The n-port modules in the LP-52xx will use this function. For example, if you want to send or receive data from a specified serial port, this function must be called first. Then the other functions can be used later.

Syntax:

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

Parameter:

port :	[Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
baudrate:	[Input] B1200/ B2400/ B4800/ B9600/ B19200/ B38400/ B57600/ B115200
date :	[Input] DATA_BITS_5/ DATA_BITS_6/ DATA_BITS_7/ DATA_BITS_8
parity :	[Input] NO_PARITY / ODD_PARITY / EVEN_PARITY
stop :	[Input] ONE_STOP_BIT / TWO_STOP_BITS

Return Value:

This function returns int port descriptor for the port opened successfully.
ERR_PORT_OPEN is for Failure.

Examples:

```
#define  COM_M1  "/dev/ttyO2"      // Defined the /dev/ttyO2 port
 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/dev/ttyO2 port will be open and initiated.
```

Remark:

1) This function can be applied on COM port modules.
2) More detailed information about device node, user can refer to:
   C:\cygwin\LinPAC_AM335x_SDK\include\sio_52xx.h.

4.1.9 sio_close

Description:

If you have used the function of sio_open() to open the specified serial port in the LP-52xx, you need to use the sio_close() function to close the specified serial port in the LP-52xx. For example, once you have finished sending or receiving data from a specified serial port, this function would then need to be called.

Syntax:

```
[ C ]

int sio_close(int port)
```

Parameter:

port :	[Input] COM1, COM2, COM4, COM5
	(1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)

Return Value:

None

Examples:

```
#define  COM_M2  "/dev/ttyO4"    // Defined the /dev/ttyO4 port
char fd[5];
 fd[0]=sio_open(COM_M2, B9600, DATA_BITS_8, NO_PARITY,ONE_STOP_BIT);
sio_close (fd[0]);
// The /dev/ttyO4 port will be closed.
```

Remark:

1) This function can be applied on COM port modules.
2) More detailed information about device node, user can refer to:
   C:\cygwin\LinPAC_AM335x_SDK\include\sio_52xx.h.

4.1.10 Read_SN

Description:

This function is used to retrieves the hardware serial identification number on the LP-52xx main controller. This function supports the control of hardware versions by reading the serial ID chip.

Syntax:

```
                                    [ C ]

void read_sn(char sn[])
```

Parameter:

sn :	[Output] Receive the serial ID number.

Return Value:

None

Examples:

```
int rs = 0;
char sn[16];
rs = read_sn(sn);
if(rs)
        printf("read sn fail!\n");
else
        printf("LP-52xx SN : %s\n", sn);
```

Remark:

4.1.11 rotary_switch_read

Description:

This function is used to retrieve the rotary ID number in the LP-52xx.

Syntax:

```
                                   [ C ]
   int rotary_switch_read (&value)
```

Parameter:

value: [Output] Rotary switch ID number.

Return Value:

0: The slot was successfully initialized.

Other: The initialization failed.

Examples:

```
int  result=0 ;
unsigned char value=0;
rotary_switch_read (&value);
if(result)
{
        printf("rsw(%d) : rotary switch read error\n",result);
        return FAILURE;
}
else
{
        printf("%d\n", value);     //Get the LP-52xx rotary id
}
```

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

Remark:

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

4.1.12 GetSDKversion

Description:

This function is used to retrieve the version of LP-52xx SDK.

Syntax:

```
[ C ]

float GetSDKversion(void)
```

Parameter:

None

Return Value:

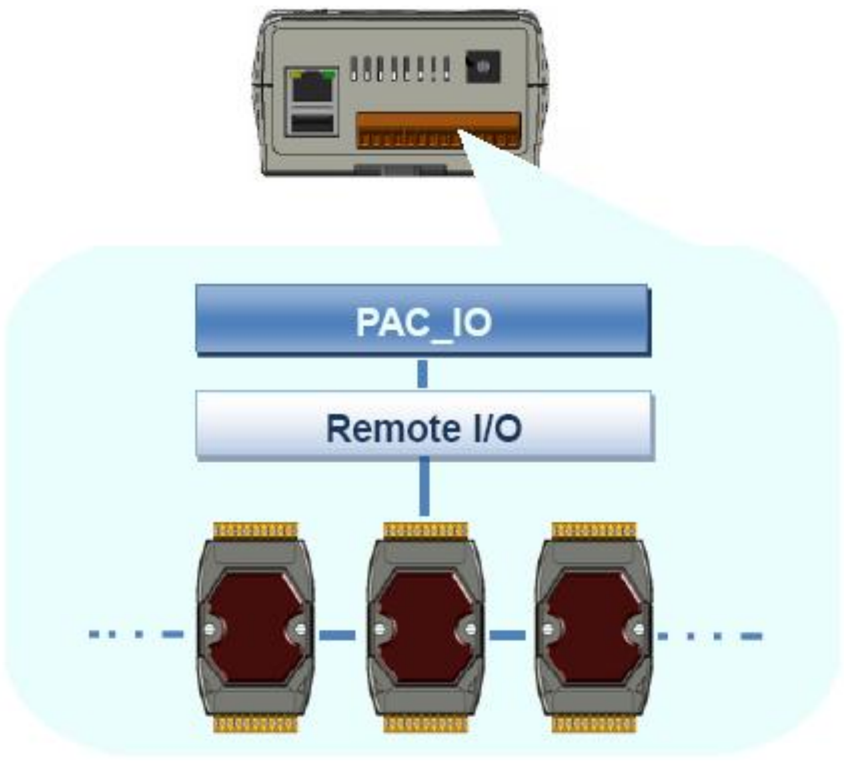
Version number.

Examples:

```
printf(" GetSDKversion = %4.2f \n ", GetSDKversion());
// Get the LP-52xx SDK version number.
// Returned Value: GetSDKversion = 1.
```

Remark:

4.2 Digital Input/Output Functions



4.2.1 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]:	[Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/12/14/42/43/44/50/60/63/65/66/67/80
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Input] 16-bit digital output data
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive

fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: “Error Code Definitions” for details of other returned values.

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=4;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(4, 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(COM4);
```

Remark:

4.2.2 DigitalBitOut

Description:

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

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument talbe
wBuf[0]:	[Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7042/43/44/50/60/63/65/66/67
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	Not used
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7]:	[Input] The digital output channel No.
wBuf[8]:	[Input] Logic value(0 or 1)
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: "Error Code Definitions" for details of other returned values.

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=3;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM3, 9600, Data8Bit, NonParity, OneStopBit);
wBuf[0] = m_port;
wBuf[1] = m_address;
wBuf[2] = 0x7065;
wBuf[3] = m_checksum;
wBuf[4] = m_timeout;
wBuf[6] = 0;
wBuf[7] = 0x08;          //RL4 relay On
wBuf[8] = 1;
DigitalBitOut (wBuf, fBuf, szSend, szReceive);
Close_Com(COM3);
```

Remark:

4.2.3 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]:	[Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7042/43/44/50/60/63/65/66/67/80
wBuf[3]:	[Input] 0=Checksum disable; 1=Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Output] 16-bit digital output data read back
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend & szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: “Error Code Definitions” for details of other returned values.

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD DO;
WORD wBuf[12];
WORD m_port=4;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
```

```
Open_Com(COM4, 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(COM4);
```

Remark:

4.2.4 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]:	[Input] COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7016
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Input] 2-bit digital output data in decimal format
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7]:	[Input] 0 : Bit0, Bit1 output 1 : Bit2, Bit3 output
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: “Error Code Definitions” for details of other returned values.

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=4;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM4, 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(COM4);
```

Remark:

4.2.5 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7041/44/50/52/53/55/58/60/63/65
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Output] 16-bit digital output data
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: “Error Code Definitions” for details of other returned values.

Example:

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

```
Open_Com(COM2, 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(COM2);
```

Remark:

4.2.6 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7041/44/50/52/53/55/58/60/63/65/66
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Input] 0: low Latch mode ; 1:high Latch mode
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7]:	[Output] Latch value
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: "Error Code Definitions" for details of other returned values.

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=2;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM2, 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(COM2);
```

Remark:

4.2.7 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/12/14/42/43/44/50/55/58/60/63/65/66/67
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	Not used.
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules .

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: "Error Code Definitions" for details of other returned values.

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=5;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
```

```
Open_Com(COM5, 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(COM5);
```

Remark:

4.2.8 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7041/44/50/51/52/53/55/58/60/63/65
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Input] The digital input Channel No.
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7]:	[Output] Counter value of the digital input channel No.
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: "Error Code Definitions" for details of other returned values.

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD DI_counter;
WORD wBuf[12];
WORD m_port=5;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM5, 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(COM5);
```

Remark:

4.2.9 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7041/44/50/51/52/53/55/58/60/63/65
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Input] The digital input channel No.
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: "Error Code Definitions" for details of other returned values.

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=2;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
```

```
Open_Com(COM2, 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(COM2);
```

Remark:

4.2.10 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/12/14/16
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	Not used
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7]:	[Output] The value of event counter
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: "Error Code Definitions" for details of other returned values.

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD Counter;
WORD wBuf[12];
WORD m_port=2;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM2, 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(COM2);
```

Remark:

4.2.11 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/12/14/16
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	Not used
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules .

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: "Error Code Definitions" for details of other returned values.

Example:

```
char szSend[80];
char szReceive[80];
float fBuf[12];
WORD wBuf[12];
WORD m_port=2;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
```

```
Open_Com(COM2, 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(COM2);
```

Remark:

4.3 Analog Input Functions

4.3.1 AnalogIn

Description:

This function is used to obtain input value from 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7005/11/12/13/14/15/16/17/18/19/33
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Input] Channel number for multi-channel
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf:	Float Input/Ouput argument table.
fBuf[0]:	[Output] Analog input value return
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

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

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: "Error Code Definitions" for details of other returned values.

Example:

```
float AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=2;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM2, 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(COM2);
```

Remark:

4.3.2 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7005/11/12/13/14/15/16/17/18/19/33
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Input] Channel number for multi-channel
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
wBuf[7]:	[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 4.5: "Error Code Definitions" for details of other returned values.

Example:

```
float AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=2;
```

```
    WORD m_address=1;
    WORD m_timeout=100;
    WORD m_checksum=0;
    Open_Com(COM2, 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(COM2);
```

Remark:

4.3.3 AnalogInFsr

Description:

This function is used to obtain the analog input value in “FSR” format from 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7005/11/12/13/14/15/16/17/18/19/33
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Input] Channel number for multi-channel
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend &szReceive
fBuf:	Float Input/Output argument table.
fBuf[0]:	[Output] Analog input value return
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

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

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: “Error Code Definitions” for details of other returned values.

Example:

```
float AI;
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=2;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM2, 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(COM2);
```

Remark:

4.3.4 AnalogInAll

Description:

This function is used to obtain the analog input value of all channels from 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7005/15/16/17/18/19/33
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf:	Float Input/Output argument table.
fBuf[0]:	[Output] Analog input value return of channel_0
fBuf[1]:	[Output] Analog input value return of channel_1
fBuf[2]:	[Output] Analog input value return of channel_2
fBuf[3]:	[Output] Analog input value return of channel_3
fBuf[4]:	[Output] Analog input value return of channel_4
fBuf[5]:	[Output] Analog input value return of channel_5
fBuf[6]:	[Output] Analog input value return of channel_6
fBuf[7]:	[Output] Analog input value return of channel_7
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

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

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 4.5: “Error Code Definitions” for details of other returned values.

Example:

```
    float AI[12];
    float fBuf[12];
    char szSend[80];
    char szReceive[80];
    WORD wBuf[12];
    WORD m_port=2;
    WORD m_address=1;
    WORD m_timeout=100;
    WORD m_checksum=0;
    Open_Com(COM2, 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(COM2);
```

Remark:

4.3.5 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Output] response value 0 → the thermocouple is close response value 1 → the thermocouple is open
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: "Error Code Definitions" for details of other returned values.

Example:

```
WORD state;
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=2;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM2, 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(COM2);
```

Remark:

4.3.6 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7013/16/33
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Input] Set display channel
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: “Error Code Definitions” for details of other returned values.

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=2;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
```

```
Open_Com(COM2, 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(COM2);
```

Remark:

4.3.7 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7013/16/33
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Output] Current channel for LED display 0 = channel_0 1 = channel_1
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: "Error Code Definitions" for details of other returned values.

Example:

```
WORD led;
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=2;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM2, 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(COM2);
```

Remark:

4.4 Analog Output Functions

4.4.1 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7016/21/22/24
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Input] The analog output channel number
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf:	Float Input/Ouput argument table.
fBuf[0]:	[Input] Analog output value
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: "Error Code Definitions" for details of other returned values.

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=2;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM2, 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(COM2);
```

Remark:

4.4.2 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 reading back functions, as described in the following:

1. Last value is read back by $AA6 command.
2. The 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7016/21/22/24
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Input] 0: command $AA6 read back 1: command $AA8 read back

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

wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7]:	[Input] The analog output channel No. (0 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 4.5: “Error Code Definitions” for details of other returned values.

Example:

```
Float Volt;
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=2;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM2, 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(COM2);
```

Remark:

4.4.3 AnalogOutHex

Description:

This function is used to obtain analog value of the 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7021/21P/22
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Input] The analog output channel number (No used for single analog output module)
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7]:	[Input] Analog output value in Hexadecimal data format
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 4.5: “Error Code Definitions” for details of other returned values.

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=2;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM2, 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                 // Analog output value in Hexadecimal data format
AnalogOutHex (wBuf, fBuf, szSend, szReceive);
Close_Com(COM2);
```

Remark:

4.4.4 AnalogOutFsr

Description:

This function is used to obtain the 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7021/21P/22
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Input] The analog output channel number (No used for single analog output module)
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
fBuf:	Float Input/Output argument table.
FBuf[0]:	[Input] Analog output value in % of Span data format.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: “Error Code Definitions” for details of other returned values.

Example:

```
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=2;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM2, 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(COM2);
```

Remark:

4.4.5 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 reading back functions, as described in the following:

1. Last value is read back by $AA6 command.
2. The 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7021/21P/22
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Input] 0: command $AA6 read back 1: command $AA8 read back
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7]:	[Input] The analog output channel No. No used for single analog output module
wBuf[9]:	[Output] Analog output value in Hexadecimal data format.
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.
Other: The processing failed.
Refer to Chapter 4.5: “Error Code Definitions” for details of other returned values.

Example:

```
WORD Volt;
```

```
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=2;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM2, 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(COM2);
```

Remark:

4.4.6 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 reading back functions, as described in the following:

1. Last value is read back by $AA6 command.
2. The 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]:	[Input] COM port number: COM1, COM2, COM4, COM5 (1=COM1, 2=COM2=/dev/ttyO2, 4=COM4=/dev/ttyO4, 5=COM5=/dev/ttyO5)
wBuf[1]:	[Input] Module address, form 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7021/21P/22
wBuf[3]:	[Input] 0= Checksum disable; 1= Checksum enable
wBuf[4]:	[Input] Timeout setting , normal=100 msecond
wBuf[5]:	[Input] 0: command $AA6 read back 1: command $AA8 read back
wBuf[6]:	[Input] 0 → no save to szSend &szReceive 1 → Save to szSend &szReceive
wBuf[7]:	[Input] The analog output channel No. No used for single analog output module
fBuf:	Float input/output argument table.
fBuf[0]:	[Output] Analog output value in % Span data format.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

0: The function was successfully processed.

Other: The processing failed.

Refer to Chapter 4.5: “Error Code Definitions” for details of other returned values.

Example:

```
float Volt;
float fBuf[12];
char szSend[80];
char szReceive[80];
WORD wBuf[12];
WORD m_port=2;
WORD m_address=1;
WORD m_timeout=100;
WORD m_checksum=0;
Open_Com(COM2, 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(COM2);
```

Remark:

4.5 Error Code Explanation

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

5. System Settings

5.1 WDT

To **Enable WDT** working status, the process can be divided into two steps, which are described below:

(1) Refresh WDT source.

```
#echo timer > /sys/class/leds/beaglebone::wdt/trigger   //Refresh WDT
```

(2) Enable WDT.

```
#echo 0 > /proc/hmistat/radiopower                        //Enable WDT
```

To **Disable WDT** working status, the process can be divided into four steps, which are described below:

(1) Disable WDT

```
#echo 1 > /proc/hmistat/radiopower                        //Disable WDT
```

(2) Clear WDT refresh source.

```
#echo none > /sys/class/leds/beaglebone::wdt/trigger   //Clear WDT Refresh Source
```

5.2 Execute Demo at Boot Time

User can refer to below steps to auto-execute demo “helloworld” at boot time.

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

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

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

```
root@ LP-5231:~# vi /etc/init.d/hello
```

```
#!/bin/sh

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

helloworld > /tmp/test.log
```

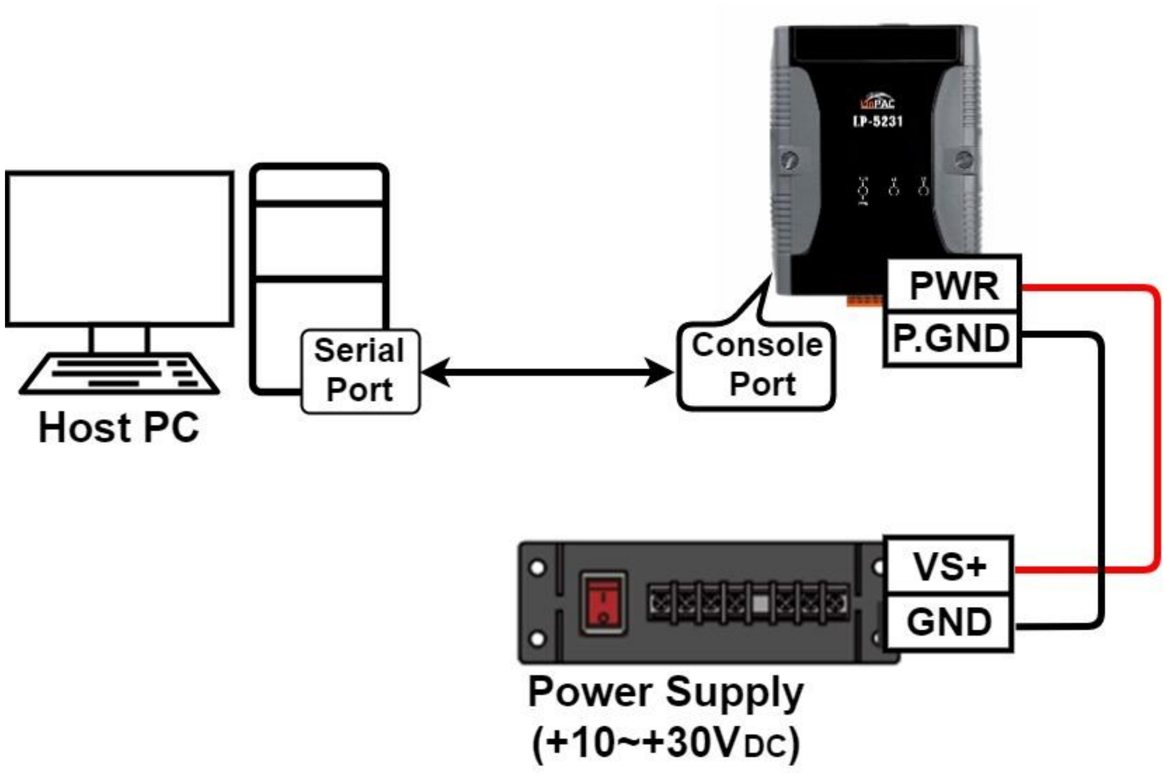
3. Use “**update-rc.d**” command to add the script “hello” automatically.

```
root@ LP-5231:~# chmod +x /etc/init.d/hello
root@ LP-5231:~# update-rc.d hello defaults
```

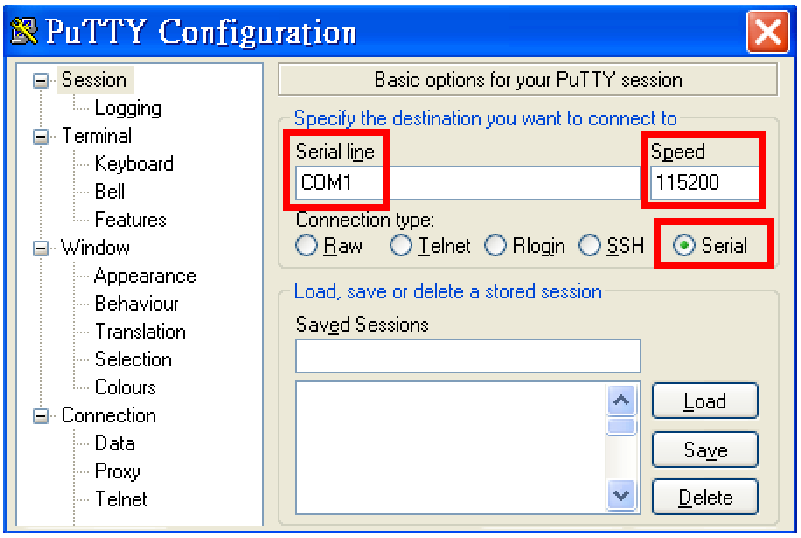
4. After setting the file, the LP-52xx/9x21 will execute binary “helloworld” at boot time

5.3 Remote Connection

5.3.1 Console Connection



1. Connect both the LP-5231 and your computer through the **"Console Port"**, and power the LP-5231 on.
2. Using the serial terminal software (ex Putty or others) and set the baud rate **"115200"** to connect to the device.
3. Type default ID- **"root"** and password- "**icpdas**" to login.



Serial line: COM1
Speed: 115200
Connection type: Raw Telnet Rlogin SSH [✓ Serial]

Fig 5-1 Console Connection

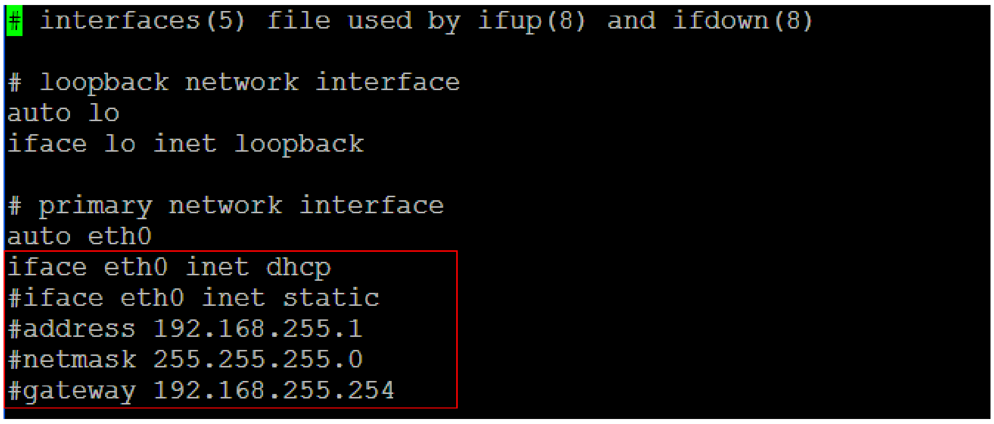
5.3.2 Network Connection

1. After user follow step 5 “Console Connection” to connect to the device bash terminal, user can change the default network setting:

IP	**192.168.255.1**
Netmask	**255.255.255.0**
User ID	**root**
Password	**icpdas**

2. If user want to use DHCP to set the network configuration, user can use the command “**vi**” to modify the configuration file “**/etc/network/interfaces**”.

3. Using the ‘**#**’ to mark the default configuration and saving the file. Please refer to below Fig 5-2:



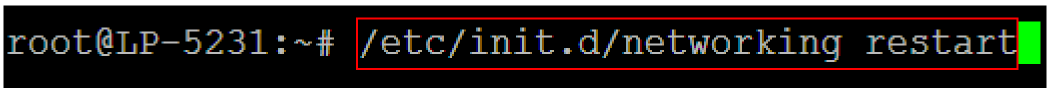
```
# interfaces(5) file used by ifup(8) and ifdown(8)

# loopback network interface
auto lo
iface lo inet loopback

# primary network interface
auto eth0
iface eth0 inet dhcp
#iface eth0 inet static
#address 192.168.255.1
#netmask 255.255.255.0
#gateway 192.168.255.254
```

Fig. 5-2 DCHP/Static IP Setting

4. After saving the new network setting, user can use the command “**/etc/init.d/networking**” to enable the setting.



```
root@LP-5231:~# /etc/init.d/networking restart
```

Fig. 5-3 Enable New Network Setting

5. User can use the new network setting to connect to the device.

5.4 Basic Linux Instructions

User can use below basic Linux command (Fig. 5-4) to control LP-5231:

Instruction	Function Description
ls	list the file information
cd	Change directory
mkdir	create the subdirectory
rm	delete file or directory
cp	copy file
mv	move or rename file or directory
pwd	show the current path
who	show the on-line users
chmod	change authority of file
uname	show the version of linux
ps	show the procedures that execute now
date	show date and time
netstat	show the state of network
ifconfig	show the ip and network mask
wget	get the file from the web link
ping	check to see if the host in the network is alive
passwd	change the password
vi	a programmers text editor
reboot	reboot the LP-5231

Fig. 5-4 Basic Command

5.5 i-Talk Utility

User can use below the i-Talk utility (Fig. 5-5) to control LP-5231 or ICP DAS XV-Board and can be found in the path —**/usr/sbin/iTalk**. An overview of the i-Talk utility functions is given below:

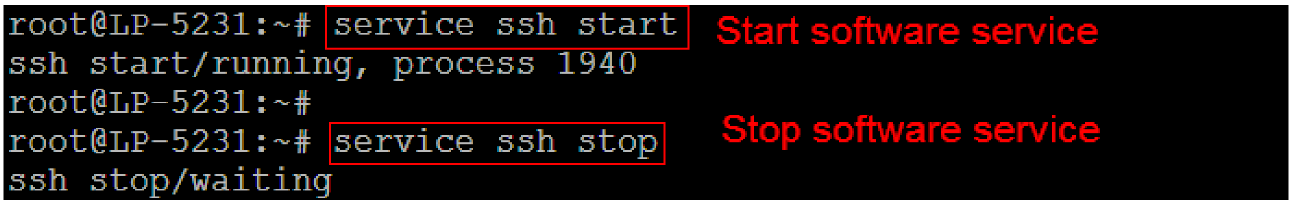
Instruction	Function Description
setxvdo	Set digital output value to XV-Board
setxvao	Set analog output value to XV-Board
getxvdi	Get digital input value from XV-Board
getxvai	Get analog input value from XV-Board
getxvdo	Get digital output value from XV-Board
getxvao	Get analog output value from XV-Board
setmodbus	Set the modbus device
getmodbus	Get the status of modbus device
rsw	Get the rotary switch ID
led	Set LED(L1~L2)

Fig. 5-5 i-Talk Utility

5.6 SysVinit Support

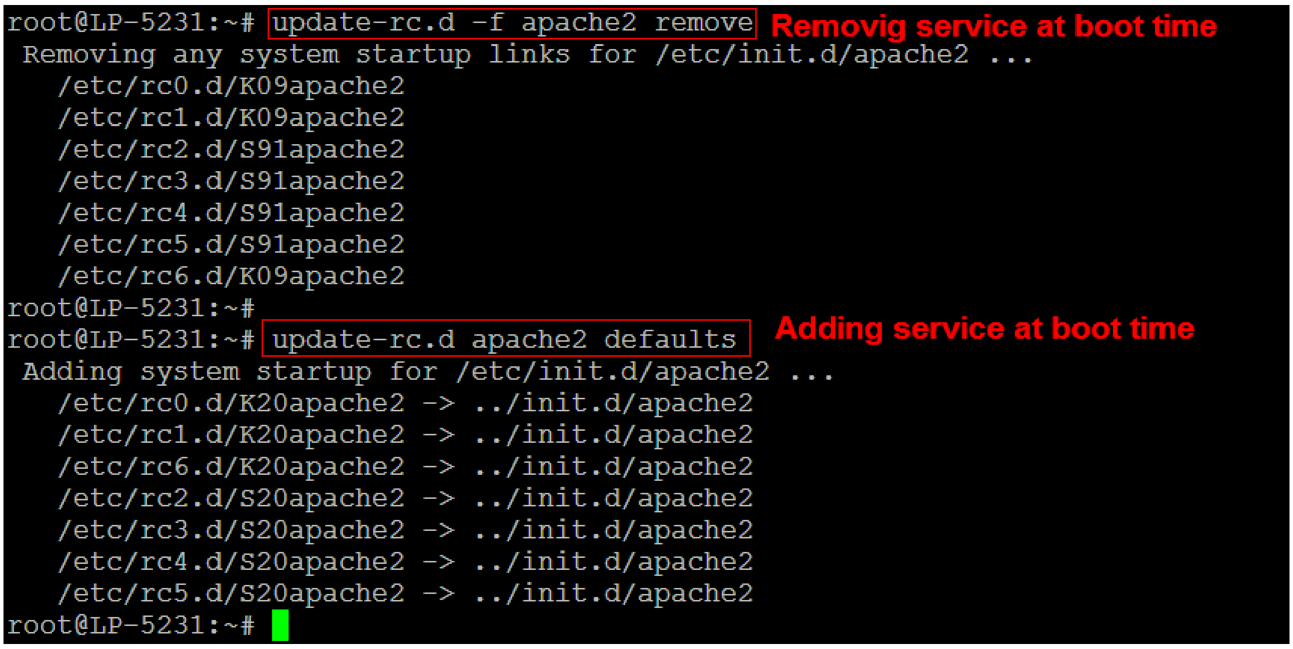
SysVinit is a system and service manager for Linux operating systems.

User can **start/stop/enable/disable** software service by using linux command "**service**" and "**update-rc.d**". Please refer to below steps to start/stop/enable/disable software.



```
root@LP-5231:~# service ssh start
ssh start/running, process 1940
root@LP-5231:~#
root@LP-5231:~# service ssh stop
ssh stop/waiting
```

Fig. 5-6 start/stop software

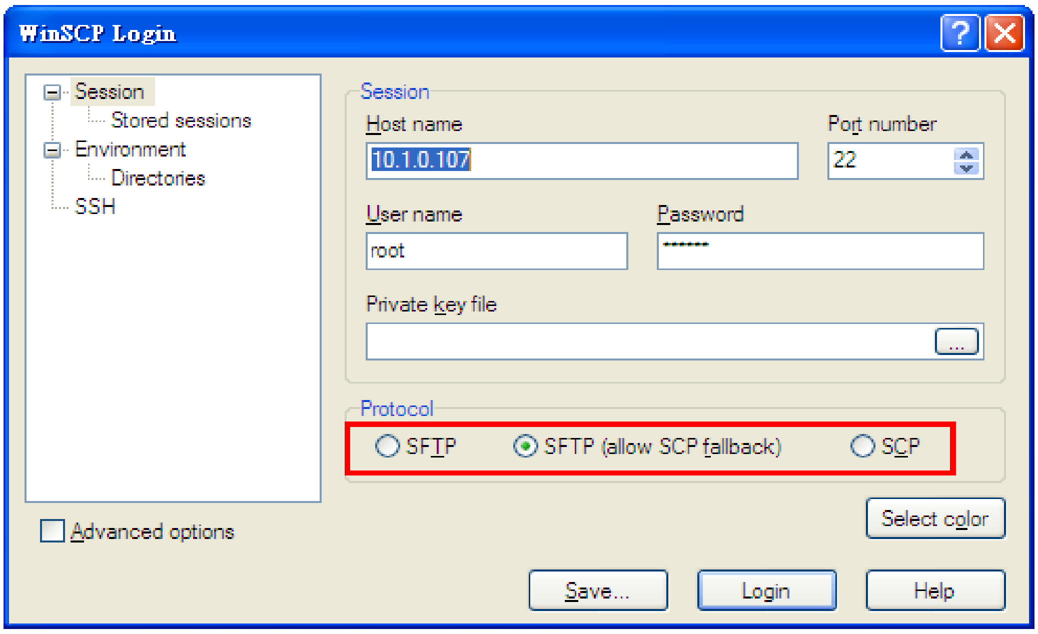


```
root@LP-5231:~# update-rc.d -f apache2 remove
 Removing any system startup links for /etc/init.d/apache2 ...
   /etc/rc0.d/K09apache2
   /etc/rc1.d/K09apache2
   /etc/rc2.d/S91apache2
   /etc/rc3.d/S91apache2
   /etc/rc4.d/S91apache2
   /etc/rc5.d/S91apache2
   /etc/rc6.d/K09apache2
root@LP-5231:~#
root@LP-5231:~# update-rc.d apache2 defaults
 Adding system startup for /etc/init.d/apache2 ...
   /etc/rc0.d/K20apache2 -> ../init.d/apache2
   /etc/rc1.d/K20apache2 -> ../init.d/apache2
   /etc/rc6.d/K20apache2 -> ../init.d/apache2
   /etc/rc2.d/S20apache2 -> ../init.d/apache2
   /etc/rc3.d/S20apache2 -> ../init.d/apache2
   /etc/rc4.d/S20apache2 -> ../init.d/apache2
   /etc/rc5.d/S20apache2 -> ../init.d/apache2
root@LP-5231:~#
```

Fig. 5-7 Enable/Disable software

5.7 SFTP

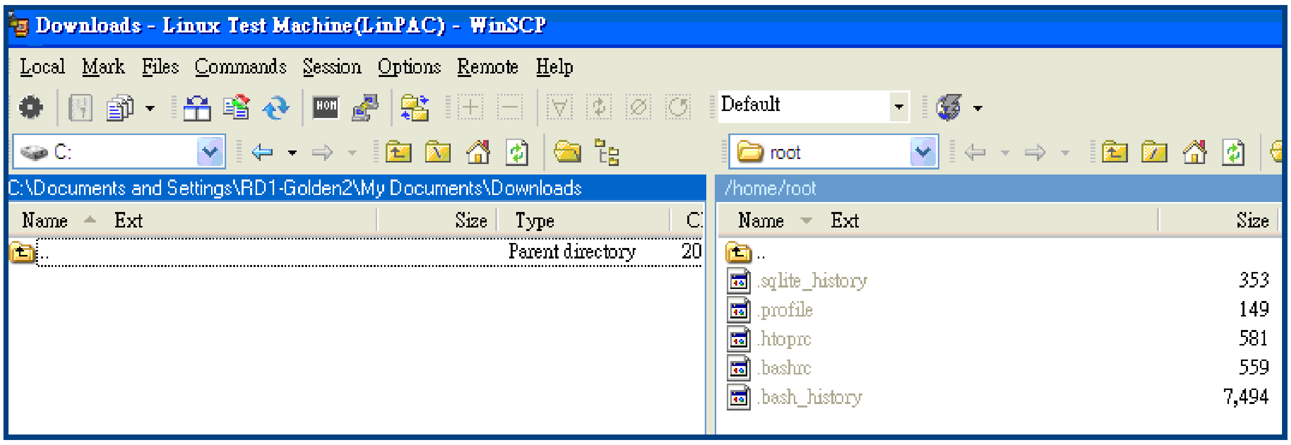
The LP-5231 series had supported SFTP (or SCP), user can transfer the file from Windows (or Linux). For examples, using Windows Program "WinSCP" to access the device over network (please refer to Fig. 5-8 and Fig. 5-9). For free download "WinSCP" software, user can visit https://winscp.net/eng/download.php


WinSCP Login

Session
Host name: 10.1.0.107
Port number: 22
User name: root
Password: *******
Private key file:

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

Fig. 5-8 WinSCP Login


Downloads - Linux Test Machine(LinPAC) - WinSCP

C:\Documents and Settings\RD1-Golden2\My Documents\Downloads

Name	Ext	Size	Type	C
..			Parent directory	20

/home/root

Name	Ext	Size
..		
.sqlite_history		353
.profile		149
.htoprc		581
.bashrc		559
.bash_history		7,494

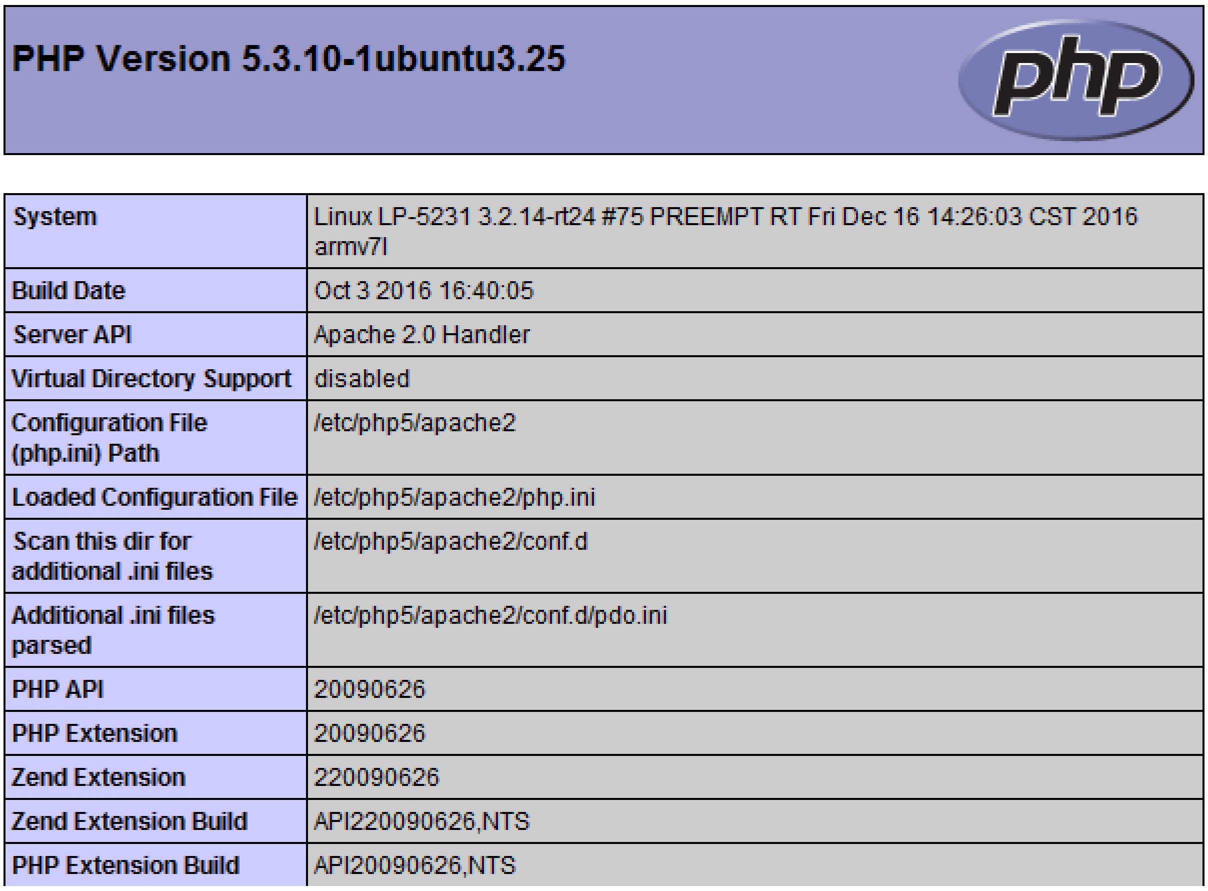
Fig. 5-9 WinSCP

5.8 LAMP Server

The LAMP (Apache2 + PHP5 + MySQL) server has been built in the LP-5231 and it will be started automatically at boot time. The default path of web page in the **"/var/www"**.

If user want to change the web page's path, user can use command "vi" to modify the configuration file **"/etc/apache2/sites-enabled/000-default"** of daemon "apache2".

User can use the web browser and input the device IP to connect to default index page "index.php" to get detail information.



PHP Version 5.3.10-1ubuntu3.25

System	Linux LP-5231 3.2.14-rt24 #75 PREEMPT RT Fri Dec 16 14:26:03 CST 2016 armv7l
Build Date	Oct 3 2016 16:40:05
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/pdo.ini
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626
Zend Extension Build	API220090626,NTS
PHP Extension Build	API20090626,NTS

Fig. 5-10 index.php

5.9 XFCE GUI Desktop

Xfce is a lightweight desktop environment for UNIX-like operating systems. It aims to be fast and low on system resources, while still being visually appealing and user friendly. Now the LP-5231 series Linux provides the XFCE package, after user type “**root**” and password “**icpdas**” to login, the local terminal would execute the XFCE Desktop.

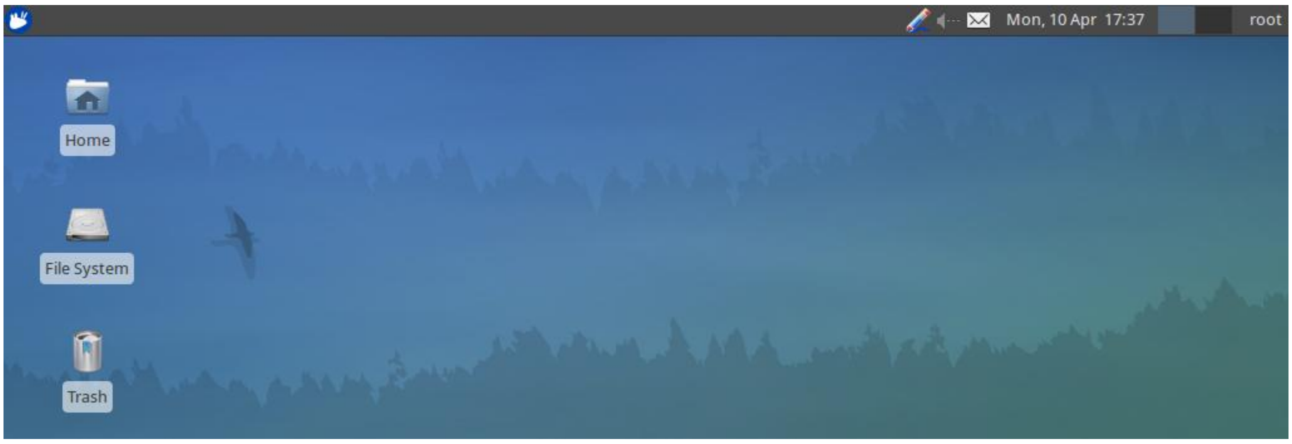
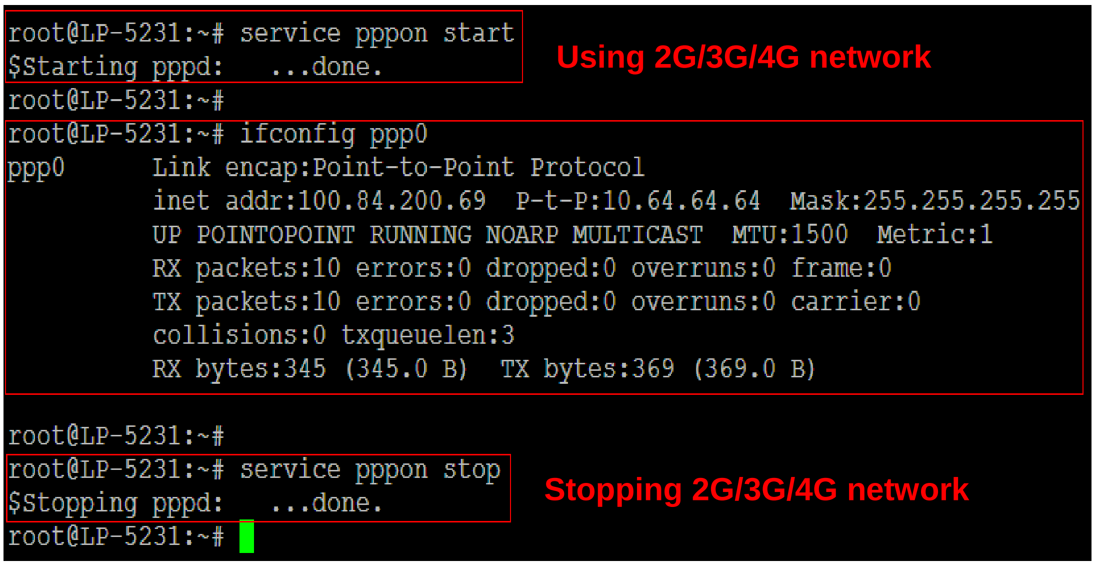


Fig. 5-11 XFCE Desktop

5.10 2G/3G/4G System

LP-5231PM-3GWM support the 2G/3G system and LP-5231PM-4GE/4GC support the 2G/3G/4G system. User must modify the "etc/ppp/ppp-on-dialer" for user's ISR first. After setting the network configuration for the user's ISP, user can use the command **"service pppon start"** to start 2G/3G or the command "**service pppon stop**" to stop 2G/3G. After checking for an IP address from the network provider, look for whether the "**ppp0**" network interface is active.



```
root@LP-5231:~# service pppon start
$Starting pppd:   ...done.
root@LP-5231:~#
root@LP-5231:~# ifconfig ppp0
ppp0      Link encap:Point-to-Point Protocol
          inet addr:100.84.200.69  P-t-P:10.64.64.64  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:10 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:345 (345.0 B)  TX bytes:369 (369.0 B)

root@LP-5231:~#
root@LP-5231:~# service pppon stop
$Stopping pppd:   ...done.
root@LP-5231:~#
```

Fig. 5-12 pppon service

5.11 Software Package Manager

The "**apt-get**" utility is the Ubuntu package manager used to download and install software packages from local package repositories or ones located in the Internet.

- To install a package run the following commands:

```
root@ LP-5231:~# apt-get update
root@ LP-5231:~# apt-get install <package>
```

- To search available package run the following commands:

```
root@ LP-5231:~# apt-cache search <package name>
```

5.12 EEPROM

To **Enable EEPROM** working status, the process can be divided into five steps, which are described below:

(1) Startup EEPROM GPIO function.

```
# echo  64 > /sys/class/gpio/export
```

(2) The EEPROM is write protected by default, user need to modify default value of EEPROM.

```
# echo  out > /sys/class/gpio/gpio64/direction
```

(3) Change to writable of EEPROM

```
#  echo  0 > /sys/class/gpio/gpio64/value
```

(4) To write a data to EEPROM.

```
# echo hello > /sys/bus/i2c/devices/1-0050/eeprom
```

More detailed information, please refer to the demo code: SDK\example\common\eeprom.c

Appendix A. XV-Board Modules

The XV-board series are for LP-5000, WP-5000-CE7. One PAC can only plug only one XV-board. The XV-board series have following common specification:

● DI channel is dry contact, sink type.

● DO channel is open collector, sink type.

DIO Expansion

Model	DI			DO	
	Channel	Type	Sink/Source	Channel	Sink/Source
XV107	8	Wet	Source	8	Sink
XV107A			Sink		Source
XV110	16	Dry/Wet	Sink/Source	-	-
XV111	-			16	Sink
XV111A					Source

Relay Output Expansion

Model	DI			Relay Output	
	Channel	Type	Sink/Source	Channel	Type
XV116	5	Wet	Sink/Source	2	Signal Relay
				4	Power Relay

Multi-Function Expansion

Model	AI	AO	DI			DO	
	Channel			Type	Sink/Source	Channel	Sink/Source
XV308	8	-	DI+DO=8	Dry/Wet	Source	DI+DO=8	Sink
XV310	4	5	4		Sink		Source

For more detailed information about these support modules, please refer to
http://www.icpdas.com/root/product/solutions/hmi_touch_monitor/touchpad/xv-board_selection.html

Appendix B. Service Information Software Introduction

■ **LinPAC-5231 Series Product Page:**

http://www.icpdas.com/root/product/solutions/pac/linpac/lp-52xx.html
http://www.icpdas.com/root/product/solutions/pac/linpac/lp-5231pm-3gwa.html

■ **LinPAC-5231 Series Document Download:**

http://ftp.icpdas.com.tw/pub/cd/linpac/napdos/lp-5000/lp-52xx/lp-5231/user_manual/

■ **LinPAC-5231 Series Software Download:**

http://www.icpdas.com/root/product/solutions/pac/linpac/linpac-5000_download.html

■ **NS-205 and DP-665 Product Page (optional):**

http://www.icpdas.com/products/Switch/industrial/ns-205.htm
http://www.icpdas.com/products/Accessories/power_supply/dp-665.htm