Software Guide

ICP DAS LX-8000/9000 Series SDK

Implement industry control with Linux Technique

Warranty

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

Warning

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

Copyright

Trademark

The names used for identification only maybe registered trademarks of their respective companies.

License

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

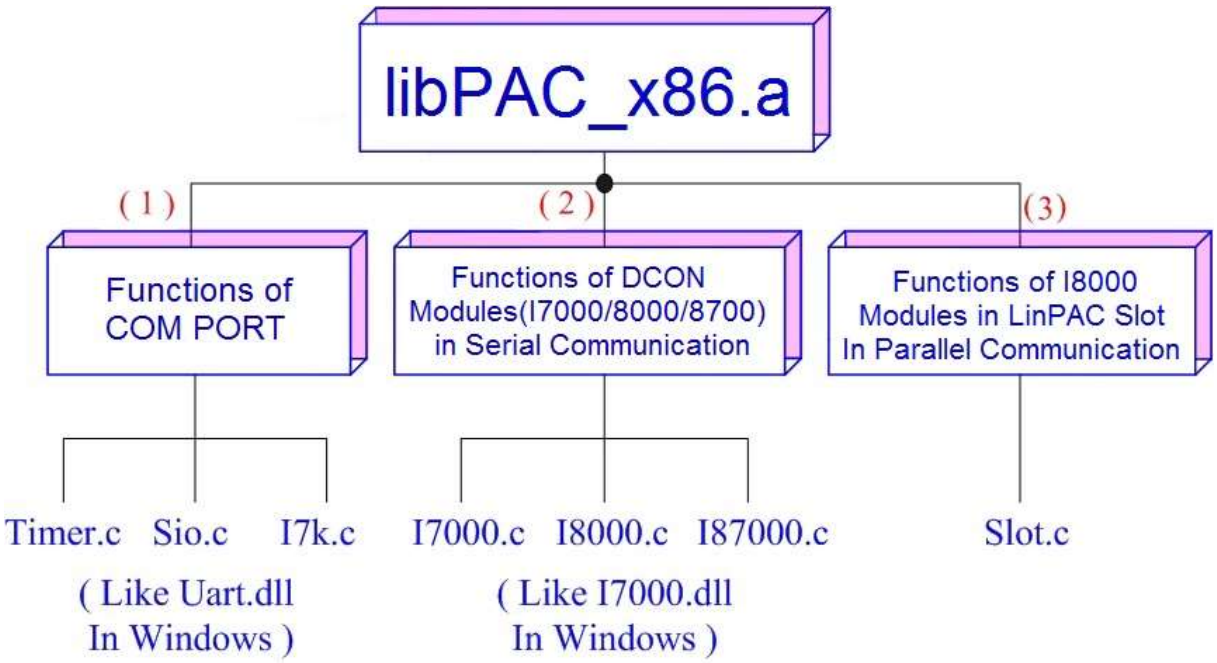
Contents

1. libPAC_x86.a

You can download LX-8000/9000 Series SDK in below link

http://ftp.icpdas.com.tw/pub/cd/linpac/napdos/lx-series/sdk/linpac_x86_sdk.tgz

In this section, we will focus on examples for the description of and application of the functions found in the libPAC_x86.a. The libPAC_x86.a functions can be clarified into 3 groups which are listed in Fig. 1-1



Structure of libPAC_x86.a

Fig. 1-1

Functions (1) and (2) in the libPAC_x86.a are the same as with the DCON.DLL Driver (including Uart.dll and I7000.dll) as used in the DCON modules (I-7000 / I-8000 / I-87000 in serial communication). You can refer to the DCON.DLL Driver manual which includes the functions on how to use DCON modules. The DCON.DLL

Driver has already been wrapped into the libPAC_x86.a. Functions (3) of the libPAC_x86.a consist of the most important functions as they are especially designed for I-8000 modules in the LX-8000/9000 Series slots. They are different from functions (1) and (2) because the communication of I-8000 modules in the LX-8000/9000 Series slots are parallel and not serial. Therefore ICP DAS rewrote I8000.c to “slot.c” especially for I-8000 modules in the LX-8000/9000 Series slots. Here we will introduce all the functions for “slot.c” and they can be divided into multiple parts for ease of use.

1. System Information Functions;
2. Digital Input Functions;
3. Digital Output Functions;
4. Watch Dog Timer Functions;
5. EEPROM Read/Write Functions;
6. Analog Input Functions;
7. Analog Output Functions;
8. I-8026W Module Functions;
9. I-8048W Module Functions;
10. I-8014W Module Functions;
11. I-8084W Module Functions;
12. I-8088W Module Functions;

When using the development tools to develop applications, the “**msw.h”** file must be included in front of the source program, and when building applications, **libPAC_x86.a** must be linked. If you want to control ICP DAS I/O remote modules like i7k, i8k and i87k **through ttyS0 or ttyS1 or ttyS34 of the LinPAC**, the functions are all the same with DCON DLL. And if you want to control **i8k modules** that are plugged in the slots of the LinPAC, then the functions are different and they are described as below:

LX-8X31 each slot device file name.

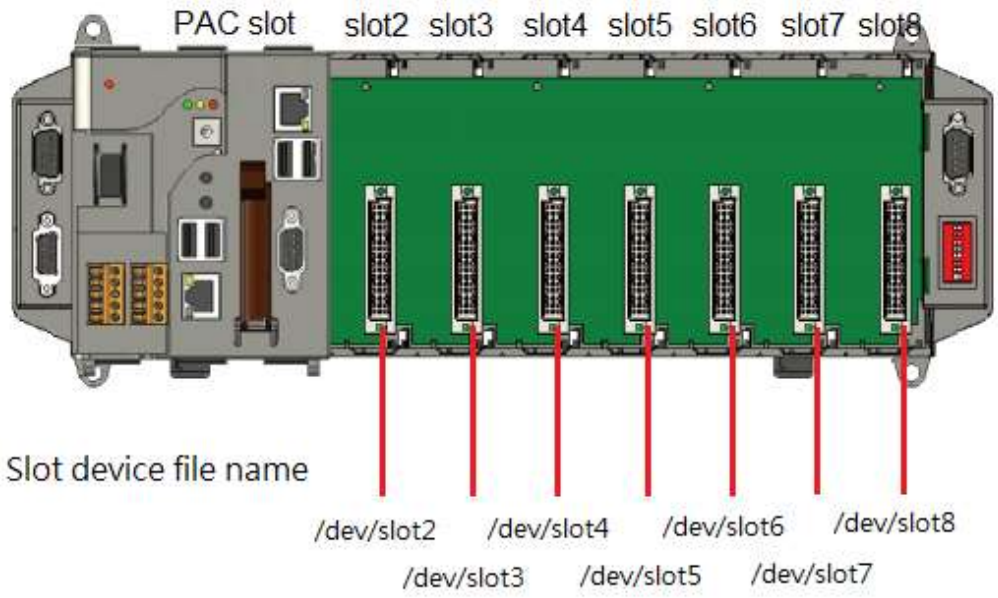


Fig-1.2

LX-9X31, LX-9X71, LX-9X81 each slot device file name

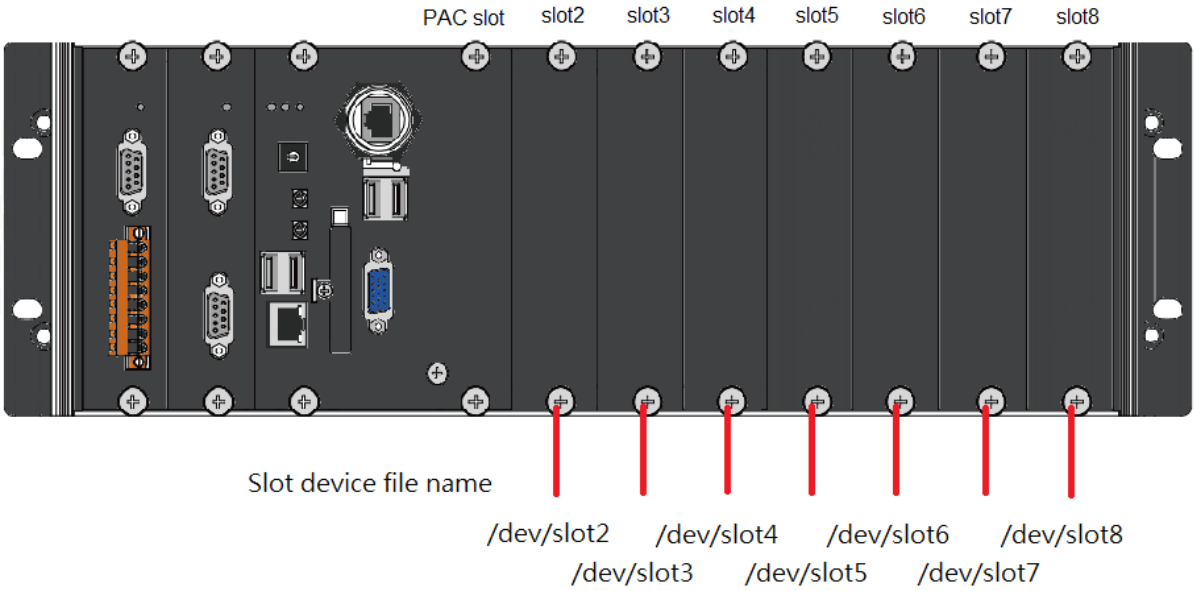


Fig-1.3

In LX-8000/9000 Series PAC, If you want control I8000 or I87000 module which is plug in slot2, you have to open “/dev/slot2” device file name to control slot2 module in your program.

1.1 System Information Functions

1.1.1 ChangeToSlot

Description:

This function is used to control multiple I-87K modules in PAC device. The serial bus in the LX-8000/9000 Series backplane is for mapping through to ttySA0.

For example, if you want to send or receive data from a specified slot, you need to call this function first.

Syntax:

[C]

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

Parameter:

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

Return Value:

None

Example:

```
char slot=2;
ChangeToSlot (slot);
// The second slot is specified as ttySA0 port in LX-8000/9000 Series.
```

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i87k/ i87kaio.c

1.1.2 Open_Slot

Description:

This function is used to open and initiate a specified slot in LX-8000/9000 Series. The I-8k or I-87k modules in the LX-8000/9000 Series will use this function.

For example, if you want to send or receive data from a specified slot, this function must be called first. Then the other functions can be used later.

Syntax:

[C]

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

Parameter:

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

Return Value:

0 is for Success
Not 0 is for Failure

Example:

Int slot=2;

Open_Slot(slot);

// The second slot in LX-8000/9000 Series will be open and initiated.

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/ rotary_sw.c

1.1.3 Close_Slot

Description:

If you have used the function of Open_Slot() to open the specified slot in the LX-8000/9000 Series, you need to use the Close_Slot() function to close the specified slot in the LX-8000/9000 Series.

The I-8k or I-87k modules in the LX-8000/9000 Series will use this function. For example, once you have finished sending or receiving data from a specified slot, this function would then need to be called.

Syntax:

[C]

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

Parameter:

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

Return Value:

None

Example:

Int slot=2;

Close_Slot(slot);

// The second slot in LX-8000/9000 Series will be closed.

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/ rotary_sw.c

1.1.4 Open_SlotAll

Description:

This function is used to open and initiate **all slots** in LX-8000/9000 Series.

For example, if you want to send or receive data from multiple slots, you can call this function to simplify your program. Then you can use the other functions later.

Syntax:

[C]

```
int Open_SlotAll(void);
```

Parameter:

None

Return Value:

0 is for Success

Not 0 is for Failure

Example:

Open_SlotAll();

// All slots in the LX-8000/9000 Series will be open and initiated.

Remark:

1.1.5 Close_SlotAll

Description:

If you have used the function Open_SlotAll() to open all the slots in LX-8000/9000 Series, you can use the Close_SlotAll() function to close all the slots in LX-8000/9000 Series.

For example, once you are finish sending or receiving data from many slots, this function can be called to close all the slots rapidly.

Syntax:

[C]

```
void Close_SlotAll(void);
```

Parameter:

None

Return Value:

None

Example:

```
Close_SlotAll();
// All slots in the LX-8000/9000 series will be closed.
```

Remark:

1.1.6 GetModuleType

Description:

This function is used to retrieve which type of 8000 series I/O module is plugged into a specific I/O slot in the LX-8000/9000 Series.

This function performs a supporting task in the collection of information related to the system's hardware configurations.

Syntax:

[C]

```
int GetModuleType(char slot);
```

Parameter:

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

Return Value:

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

Example:

```
char slot=2;
int moduleType;
Open_Slot(slot);
moduleType=GetModuleType(slot);
Close_Slot(Slot);
// I-8048 card is plugged in slot 2 of LX-8000/9000 Series and has a return value: 176
```

Remark:

1.1.7 SetLED

Description:

This function is used to set led(Run, L1, L2) status in LX-8000/9000 Series, You can check SDK path: “LinPAC_X86_SDK/examples/lx-series/common/led.c” that how to operate led.

Syntax:

[C]

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

Parameter:

bFlag : [Input] select one number to control led status.

There are eight options in below.

1. Led “RUN” ON

2. Led “L1” ON

3. Led “RUN” and “L1” ON

4. Led “L2” ON

5. Led “RUN” and “L2” ON

6. Led ”L1” and “L2” ON

7. All led ON

8. All led OFF

Return Value:

None

Example:

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

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/led.c

1.1.8 GetNameOfModule

Description:

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

Syntax:

[C]

```
int GetNameOfModule(char slot);
```

Parameter:

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

Return Value:

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

Example:

```
char slot=2;
int moduleName;
Open_Slot(slot);
moduleID=GetNameOfModule(slot);
Close_Slot(Slot);
// The I-8017 card plugged in slot 2 of LX-8000/9000 Series
// Returned Value: moduleName=” 8017”
```

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/getlist.c

1.1.9 GetSlotCount

Description:

This function is used to get number of your LX-8000/9000 Series PAC slots.

Syntax:

[C]

```
int GetSlotCount();
```

Parameter:

None

Return Value:

Your slot count.

Notice, your return value will more than one with your PAC, you should decrease 1 to get real count.

Example:

int count;

count = GetSlotCount();

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/getlist.c

1.1.10 GetDIPswitch

Description:

This function is used to get Dip switch number in LX-9000 Series PAC.

Syntax:

[C]

```
int GetDIPswitch();
```

Parameter:

None

Return Value:

Dip switch number.

Example:

Int count;

Open_Slot (9);

count = GetDIPswitch();

Close_Slot(9);

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/dip_switch.c

1.1.11 GetRotaryID

Description:

This function is used to get rotary id in LX-8000/9000 Series PAC.

Syntax:

[C]

```
int GetRotaryID();
```

Parameter:

None

Return Value:

Rotary id.

Example:

Int count;

Open_Slot (9);

count = GetRotaryID(9);

Close_Slot(9);

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/rotary_sw.c

fdisk -l | grep /dev/sda | tail -1 | awk -F" " '{printf $1}' | tail -c 1

1.1.12 Read_SN

Description:

This function is used to get hardware serial number in LX-8000/9000 Series PAC.

Syntax:

[C]

```
void Read_SN (unsigned char serial_num[8]);
```

Parameter:

serial_num[8]: [output] Get 8 byte serial number.

Return Value:

None

Example:

```
unsigned char serial_num[8];
Open_Slot (0);
Read_SN(serial_num);
Close_Slot(0);
```

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/ read_sn.c

1.1.13 Read_SRAM

Description:

This function is used to read mram data in LX-8000/9000 Series PAC.

Syntax:

[C]

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

Parameter:

offset: [input] Get mram offset address value.

Return Value:

Mram value of offset address.

Example:

```
int offset = 0;        //read mram address 0 value
Open_Slot (0);
Read_SRAM(offset);
```

Close_Slot(0);

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/ mram.c

1.1.14 Write_SRAM

Description:

This function is used to write mram data to LX-8000/9000 Series PAC.

Syntax:

[C]

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

Parameter:

offset: [input] mram offset address to write.

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

Return Value:

None

Example:

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

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/ mram.c

1.2 Digital Input/Output Functions

1.2.1 DO_8

Description:

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

Syntax:

[C]

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Example:

```
int slot=2;
unsigned char data=3;
DO_8(slot, data);
// The I-8064 card is plugged in slot 2 of LX8000/9000 Series and can turn on channel 0 and 1.
```

Remark:

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

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/i8kdio.c

1.2.2 DO_16

Description:

This function is used to output 16-bit data to a digital output module. The 0~15

bits of output data are mapped into the 0~15 channels of digital output modules respectively.

Syntax:

[C]

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Example:

```
int slot=2;
unsigned int data=3;
DO_16(slot, data);
// The I-8057 card is plugged in slot 2 of Lx-8000/9000 Series and can turn on channel 0 and 1.
```

Remark:

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

1.2.3 DO_32

Description:

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

Syntax:

[C]

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Example:

```
int slot=2;
unsigned int data=3;
DO_32(slot, data);
// The I-8041 card is plugged in slot 2 of LX-8000/9000 Series and can turn on channel 0 and 1.
```

Remark:

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

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/setdo.c

1.2.4 DI_8

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

Input data

Example:

```
int slot=2;
```

unsigned char data;

data=DI_8(slot);

// The I-8058 card is plugged in slot 2 of LX-8000/9000 Series and has inputs in

// channel 0 and 1.

// Returned value: data=0xfC

Remark:

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

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/getdi.c

1.2.5 DI_16

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

Input data

Example:

int slot=2;

unsigned int data;

data=DI_16(slot);

// The I-8053 card is plugged in slot 2 of LX-8000/9000 Series and has inputs in

// channel 0 and 1.

// Returned value: data=0xfffC

Remark:

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

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/getdi.c

1.2.6 DI_32

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

Input data

Example:

```
int slot=2;
unsigned long data;
data=DI_32(slot);
// The I-8040 card plugged is in slot 2 of LX-8000/9000 Series and has inputs in
// channels 0 and 1.
// Returned value: data=0xfffffffC
```

Remark:

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

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/getdi.c

1.2.7 DIO_DO_8

Description:

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

Syntax:

[C]

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Example:

```
int slot=2;
unsigned char data=3;
DIO_DO_8(slot, data);
// The I-8054 card is plugged in slot 2 of LX-8000/9000 Series and can turn on channels 0 and 1.
// It not only outputs a value, but also shows 16LEDs.
```

Remark:

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

1.2.8 DIO_DO_16

Description:

This function is used to output 16-bits of data to DIO modules, which have 16 digital input and 16 digital output channels running simultaneously. The 0~15 bits of

output data are mapped onto the 0~15 output channels for their specific DIO modules respectively.

Syntax:

[C]

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Example:

```
int slot=2;
unsigned int data=3;
DIO_DO_16(slot, data);
// The I-8042 card is plugged in slot 2 of LX-8000/9000 Series and can turn on the
// channels 0 and 1.
// It not only outputs a value, but also shows 32LEDs.
```

Remark:

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

1.2.9 DIO_DI_8

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

Input data

Example:

```
int slot=2;
unsigned char data;
data=DIO_DI_8(slot);
// The I-8054 card is plugged in slot 2 of LX-8000/9000 Series and has inputs in
// channel 0 and 1.
// Returned value: data=0xfC
```

Remark:

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

1.2.10 DIO_DI_16

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

Input data

Example:

int slot=2;

unsigned char data;

data=DIO_DI_16(slot);

// The I-8042 card is plugged in slot 2 of LX-8000/9000 Series and has inputs in

// channel 0 and 1.

// Returned value: data=0xfffC

Remark:

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

1.2.11 DO_8_RB、DO_16_RB、DO_32_RB DIO_DO_8_RB、DIO_DO_16_RB

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

all DO channel status

Example:

int slot=2;

```
Open_Slot(slot);
printf("%u",DO_32_RB(slot));
Close_Slot(slot);
```

// The I-8041 module is plugged in slot 2 of LX-8000/9000 Series and return all DO channel status. (Return value：From 0 ~ 4294967295 for 32 channel)

Remark:

These functions can be applied on modules:

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

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

DO 32 channel：I-8041

1.2.12 DO_8_BW、DO_16_ BW、DO_32_ BW DIO_DO_8_ BW、DIO_DO_16_ BW

Description:

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

Syntax:

[C]

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

Parameter:

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

bit : [Input] channel of module.

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

Return Value:

None

Example:

int slot=1, bit=0, data=1;

Open_Slot(slot);

DO_32_BW(slot, bit, data);

Close_Slot(slot);

// The I-8041 module is plugged in slot 2 of LX-8000/9000 Series and just turn on channel 0 of I-8041.

Remark:

These functions can be applied on modules:

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

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

DO 32 channel：I-8041

1.2.13 DI_8_BW、DI_16_ BW、DI_32_ BW

Description:

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

Syntax:

[C]

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

Parameter:

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

bit : [Input] channel of module.

Return Value:

None

Example:

```
int slot=2, bit=0;
Open_Slot(slot);
printf("DI channel %d = %d\n", bit, DI_32_BW(slot, bit));
Close_Slot(slot);
// The I-8040 module is plugged in slot 2 of LX-8000/9000 Series and return channel 0
// status. ( 0：ON；1：OFF ).
```

Remark:

These functions can be applied on modules:

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

DI 16 channel：I-8051 and I-8053

DI 32 channel：I-8040

1.2.14 UDIO_WriteConfig_16

Description:

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

Syntax:

[C]

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

Parameter:

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

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

Return Value:

None

Example:

```
int slot=2;
unsigned short config=0xffff;
```

```
UDIO_WriteConfig_16(slot, config);
// The I-8050 card is plugged in slot 2 of LX-8000/9000 Series.
// WriteConfig: 0xffff (ch 0~ch15 is DO mode)
```

Remark:

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

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/demo8050.c

1.2.15 UDIO_ReadConfig_16

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

None

Example:

```
int slot=2;
unsigned int ret;
unsigned short config=0x0000;
UDIO_WriteConfig_16(slot, config);
ret=UDIO_ReadConfig_16(slot);
printf("Read the I/O Type is : 0x%04lx \n\r",ret);
// The I-8050 card is plugged in slot 2 of LX-8000/9000 Series.
// config: 0x0000 (ch 0~ch15 is DI mode)
// Read the I/O Type is: 0x0000
```

Remark:

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

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/demo8050.c

1.2.16 UDIO_DO16

Description:

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

Syntax:

[C]

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

Parameter:

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

config : [Input] output data.

Return Value:

None

Example:

```
int slot=2;
unsigned int data;
unsigned short config =0x00ff;
UDIO_WriteConfig_16(slot, config);
scanf("%d:",&data);
UDIO_DO16(slot, data);
printf("DO(Ch0~Ch7) of I-8050 in Slot %d = 0x%x\n\r",slot, data);
// The I-8050 card is plugged in slot 2 of LX-8000/9000 Series
// config: 0x00ff (ch 0~ch7 is DO mode and ch8~ch15 is DI mode)
```

// Input DO value: 255

// DO(Ch0~Ch7) of I-8050 in Slot 1 = 0xff

Remark:

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

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/demo8050.c

1.2.17 UDIO_DI16

Description:

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

Syntax:

[C]

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

Parameter:

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

Return Value:

None

Example:

```
int slot=2;
unsigned int data;
unsigned short config =0xff00;
UDIO_WriteConfig_16(slot, config);
data=UDIO_DI16(slot);
printf("DI(Ch0~Ch7) of I-8055 in Slot %d = 0x%x\n\r",slot, data);
scanf("%d:",&data);
UDIO_DO16(slot, data);
printf("DO(Ch8~Ch15) of I-8050 in Slot %d = 0x%x\n\r",slot, data);
// The I-8050 card is plugged in slot 2 of LX-8000/9000 Series.
```

// config: 0x00ff (ch 0~ch7 is DI mode and ch8~ch15 is DO mode)

// DI(Ch0~Ch7) of I-8055 in Slot 1 = 0xfbff

// Input DO value: 255

// DO(Ch8~Ch15) of I-8050 in Slot 1 = 0xff

Remark:

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

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/demo8050.c

1.3 Watch Dog Timer Functions

1.3.1 EnableWDT

Description:

This function can be used to enable the watch dog timer (WDT) and users need to reset WDT in the assigned time set by users. Or LinPAC will reset automatically in LX-8000/9000 PAC.

Syntax:

[C]

```
void EnableWDT (unsigned int msecond);
```

Parameter:

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

The unit is mini-second.

Return Value:

None

Example:

```
Open_Slot(SLOT0);
EnableWDT(10000);
Close_Slot(SLOT0);
```

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/ wdt.c

1.3.2 DisableWDT

Description:

This function is used to disable WDT in LX-8000/9000 PAC.

Syntax:

[C]

```
void DisableWDT(void)
```

Parameter:

None

Return Value:

None

Example:

```
Open_Slot(SLOT0);
DisableWDT();
Close_Slot(SLOT0);
```

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/ wdt.c

1.3.3 LX8x_9x31_WDT

Description:

This function is used to set watch dog timeout in LX-8x31 & LX-9x31 series PAC.

Syntax:

[C]

```
void LX8x_9x31_WDT(unsigned int second);
```

Parameter:

second: [input] watch dog timeout limit.

Return Value:

None

Example:

```
int second=5;
LX8x_9x31_WDT(second);
```

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/ lx8x_9x31_wdt.c

1.3.4 LX8x_9x31_WDT_rlod

Description:

This function is used to reload watch dog timeout in LX-8x31 & LX-9x31 series PAC.

Syntax:

[C]

```
void LX8x_9x31_WDT_rlod(void);
```

Parameter:

None

Return Value:

None

Example:

```
LX8x_9x31_WDT_rlod();
```

//Use this function to reload wdt.

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/ lx8x_9x31_wdt.c

1.3.5 LX8x_9x31_WDT_dis

Description:

This function is used to disable watch dog in LX-8x31 & LX-9x31 series PAC.

Syntax:

[C]

```
void LX8x_9x31_WDT_dis(void);
```

Parameter:

None

Return Value:

None

Example:

LX8x_9x31_WDT_dis();

//Use this function to disable wdt.

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/ lx8x_9x31_wdt.c

1.4 EEPROM Read/Write Functions

1.4.1 Enable_EEP

Description:

This function is used to make EEPROM able to read or write. It must be used

before using Read_EEP or Write_EEP.

Syntax:

[C]

```
void Enable_EEP(void)
```

Parameter:

None

Return Value:

None

Example:

```
Enable_EEP();

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

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/ eeprom.c

1.4.2 Disable_EEP

Description:

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

Syntax:

[C]

```
void Disable_EEP(void)
```

Parameter:

None

Return Value:

None

Example:

```
Disable_EEP();

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

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/ eeprom.c

1.4.3 Read_EEP

Description:

This function will read one byte data from the EEPROM. There is a 16K-byte EEPROM in the main control unit in LX-8000/9000 PAC. This EEPROM with its accessing APIs provides another mechanism for storing critical data inside non-volatile memory.

Syntax:

[C]

```
unsigned char Read_EEP(int block, int offset)
```

Parameter:

block : [Input] the block number of EEPROM.

offset : [Input] the offset within the block.

Return Value:

Data read from the EEPROM.

Example:

```
int block, offset;
unsigned char data;
```

data= Read_EEP(block, offset);

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

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/ eeprom.c

1.4.4 Write_EEP

Description:

Write one byte of data to the EEPROM. There is a 16K-byte EEPROM in the main control unit of the LX-8000/9000 PAC. This EEPROM with its accessing APIs, provides another mechanism for storing critical data inside non-volatile memory.

Syntax:

[C]

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

Parameter:

block : [Input] the block number of EEPROM.

offset : [Input] the offset within the block.

data : [Input] data to write to EEPROM.

Return Value:

None

Example:

int block, offset;

unsigned char data=10;

Write_EEP(block, offset, data);

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

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/common/ eeprom.c

1.5 Analog Input Functions

I8017_Init

Description:

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

Syntax:

[C]

```
int I8017_Init(int slot)
```

Parameter:

slot : [Input] specified slot of the LX-8000/9000 Series system (Range: 2 to 8)

Return Value:

The version of library

Example:

```
int slot=1,ver;
ver=I8017_Init(slot);
// The I-8017H card is plugged in slot 1 of LX-8000/9000 Series and initializes the module I-8017H.
```

Remark:

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

I8017_SetLed

Description:

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

Syntax:

[C]

```
void I8017_SetLed(int slot, unsigned int led)
```

Parameter:

slot : [Input] specified slot of the LX-8000/9000 Series system (Range: 2 to 8)

led : [Input] range from 0 to 0xffff

Return Value:

None

Example:

```
int slot=2;
unsigned int led=0x0001;
I8017_SetLed (slot, led);
// There will be a L/A-LED light on channel 0 of the I-8017H card which is plugged in slot 2 on the LX-8000/9000 Series.
```

Remark:

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

I8017_SetChannelGainMode

Description:

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

Syntax:

[C]

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

Parameter:

slot : [Input] Specify the slot in the LX-8000/9000 Series system (Range: 2 to 8)

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

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

gain : [Input] input range:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

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

Return Value:

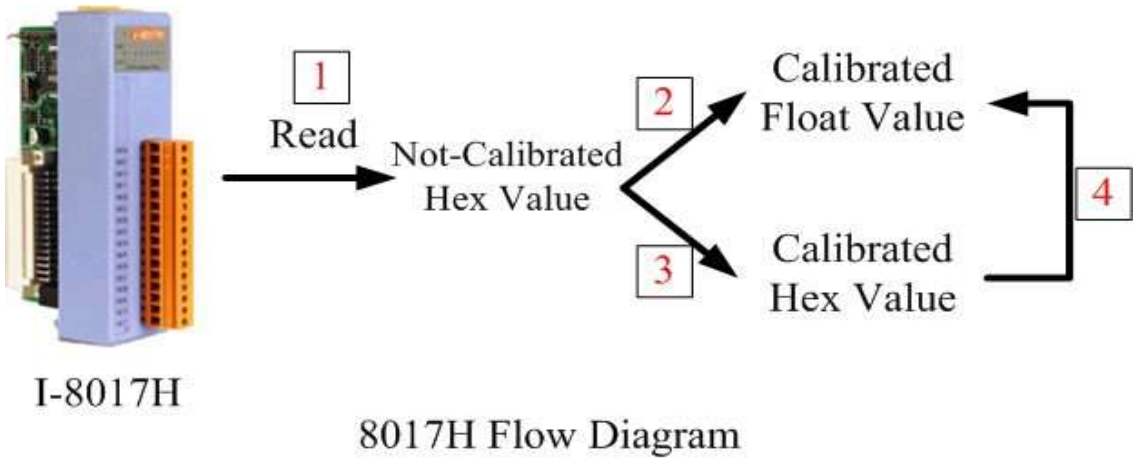
None

Example:

```
int slot=2,ch=0,gain=0;
I8017_SetChannelGainMode (slot, ch, gain,0);
// The I-8017H card is plugged in slot 2 of LX-8000/9000Series, and the range of the
    data value from channel 0 for I-8017H will be -10 ~ +10 V.
```

Remark:

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



8017H Flow Diagram

Fig.1-2

Function of [1]

I8017_GetCurAdChannel_Hex

Description:

Obtains the non-calibrated analog input value in the Hex format from the analog

input I-8017H modules. Please refer to Fig. 1-2

Syntax:

[C]

```
int I8017_GetCurAdChannel_Hex (int slot)
```

Parameter:

slot : [Input] specified slot of the LX-8000/9000 Series system (Range: 2 to 8)

Return Value:

The analog input value in Hex format.

Example:

```
int slot=2,ch=0,gain=4;
int data;
I8017_SetChannelGainMode (slot, ch, gain,0);
data= I8017_GetCurAdChannel_Hex (slot);
// The I-8017H card is plugged into slot 2 of LX-8000/9000 Series and the range of the data value from channel 0 in I-8017H is +/- 20mA
```

I8017_AD_POLLING

Description:

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

Syntax:

[C]

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

Parameter:

slot : [Input] Specified slot in the LX-8000/9000 Series system (Range: 2 to 8)

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

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

gain : [Input] Input range:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

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

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

Return Value:

0 : indicates success.

Not 0 : indicates failure.

Example:

```
int slot=2, ch=0, gain=0, data[10];
unsigned int datacount = 10;
I8017_AD_POLLING(slot, ch, gain, datacount, data);
// You gain ten not-calibrated hex values via channel 0 in the I-8017H module.
```

Function of [2]

HEX_TO_FLOAT_Cal

Description:

This function is used to convert the data from not-calibrated hex to calibrated float values based on the configuration of the slot, gain. (Voltage or current). Please refer to the Fig. 1-2.

Syntax:

[C]

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

Parameter:

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

slot : [Input] specified slot of the LX-8000/9000 Series system (Range: 2 to 8)

gain : [Input] Input range:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

Return Value:

The Calibrated Float Value.

Example:

```
int slot=2, ch=0, gain=0, hdata;
float fdata;
I8017_SetChannelGainMode (slot, ch, gain,0);
hdata = I8017_GetCurAdChannel_Hex (slot);
fdata = HEX_TO_FLOAT_Cal(hdata, slot, gain);
// You can convert not-calibrated Hex Value to calibrated Float Value
```

Remark:

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

ARRAY_HEX_TO_FLOAT_Cal

Description:

This function is used to convert the data from non-calibrated hex values to calibrated float values in the array mode based on the slot's configuration. (Voltage or current). Please refer to Fig. 1-2.

Syntax:

[C]

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

Parameter:

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

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

slot : [Input] specified slot of the LX-8000/9000 Series system (Range: 2 to 8)

gain : [Input] Input range:

len : [input] ADC data length

Return Value:

None

Example:

```
int slot=2, ch=0, gain=0, datacount=10, hdata[10];
float fdata[10];
I8017_SetChannelGainMode (slot, ch, gain,0);
I8017_AD_POLLING(slot, ch, gain, datacount, data);
ARRAY_HEX_TO_FLOAT_Cal(data, fdata, slot, gain, len);
// You can convert ten not-calibrated Hex values to ten calibrated Float values
```

Remark:

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

Function of [3]

I8017_Hex_Cal

Description:

This function is used to convert the data from non-calibrated hex values to calibrated hex values. (Voltage or current). Please refer to Fig. 1-2.

Syntax:

[C]

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

Parameter:

data : [Input] specified not-calibrated hex value

Return Value:

The Calibrated Hex Value.

Example:

```
int slot=2, ch=0, gain=0, hdata;
int hdata_cal;
I8017_SetChannelGainMode (slot, ch, gain,0);
hdata = I8017_GetCurAdChannel_Hex (slot);
hdata_cal = I8017_Hex_Cal (hdata);
// You can convert not-calibrated Hex Value to calibrated Hex Value
```

Remark:

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

I8017_Hex_Cal_Slot_Gain

Description:

This function is used to convert the data from non-calibrated hex values to calibrated hex values based on the configuration of the slot, gain. (Voltage or current).. (Voltage or current). Please refer to the Fig. 1-2.

Syntax:

[C]

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

Parameter:

slot : [Input] specified slot of the LX-8000/9000 Series system (Range: 2 to 8)

gain : [Input] Input range:

data : [Input] specified not-calibrated hex value

Return Value:

The Calibrated Hex Value.

Example:

```
int slot=2, ch=0, gain=0, hdata;
```

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

Remark:

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

Function of [4]

CalHEX_TO_FLOAT

Description:

This function is used to convert the data from calibrated hex values to calibrated float values based on the configuration of the gain. (Voltage or current). Please refer to Fig. 1-2.

Syntax:

[C]

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

Parameter:

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

gain : [Input] Input range:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

Return Value:

The Calibrated Float Value.

Example:

```
int slot=1, ch=0, gain=0, hdata, hdata_cal;
float fdata;
```

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

Remark:

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

ARRAY_CalHEX_TO_FLOAT

Description:

This function is used to convert the data from calibrated hex values to calibrated float values in the array mode based on the configuration of the gain. (Voltage or current). Please refer to the Fig. 1-2.

Syntax:

[C]

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

Parameter:

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

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

gain : [Input] Input range:

len : [input] ADC data length

Return Value:

The Calibrated Float Value.

Example:

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

Remark:

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

Function of [1] + [3]

I8017_GetCurAdChannel_Hex_Cal

Description:

Obtain the calibrated analog input values in the Hex format directly from the analog input modules, I-8017H. This function is a combination of the "I8017_GetCurAdChannel_Hex" function and the "I8017_Hex_Cal". Please refer to Fig 1-2

Syntax:

[C]

```
int I8017_GetCurAdChannel_Hex_Cal(int slot)
```

Parameter:

slot : [Input] specified slot of the LX-8000/9000 Series system (Range: 2 to 8)

Return Value:

The analog input value in Calibrated Hex format.

Example:

```
int slot=1,ch=0,gain=0, data;
I8017_SetChannelGainMode (slot, ch, gain,0);
data = I8017_GetCurAdChannel_Hex_Cal (slot);
// The I-8017H card is plugged into slot 1 of LP-8X81 Series and the range of the
// data value from channel 0 in I-8017H is 0x0000 ~ 0x3fff.
```

Remark:

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

I8017_AD_POLLING_Cal

Description:

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

Syntax:

[C]

int I8017_AD_POLLING_Cal(int slot, int ch, int gain, unsigned int datacount, int *DataPtr)

Parameter:

slot : [Input] Specified slot in the LP-8X81 Series system (Range: 2 to 8)

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

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

gain : [Input] Input range:

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

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

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

Return Value:

0 : indicates success.

Not 0 : indicates failure.

Example:

```
int slot=2, ch=0, gain=0, data[10];
unsigned int datacount = 10;
I8017_AD_POLLING_Cal(slot, ch, gain, datacount, data);
// You gain ten calibrated hex values via channel 0 in the I-8017H module.
```

Function of [1]+[2]

I8017_GetCurAdChannel_Float_Cal

Description:

Obtains the calibrated analog input value in the Float format directly from the analog i8017H input modules. This function is a combination of the "I8017_GetCurAdChannel_Hex" function and the "Hex_TO_FLOAT_Cal" function. Please refer to Fig 1-2

Syntax:

[C]

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

Parameter:

slot : [Input] specified slot of the LX-8000/9000 Series system (Range: 2 to 8)

Return Value:

The analog input value in Calibrated Float format.

Example:

```
int slot=2,ch=0,gain=0;
float data;
I8017_SetChannelGainMode (slot, ch, gain,0);
data = I8017_GetCurAdChannel_Float_Cal (slot);
// The I-8017H card is plugged into slot 1 of LP-8X81 Series and the range of the
// data value from channel 0 in I-8017H is –10V ~ +10V.
```

Remark:

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

1.6 Analog Output Functions

I8024_Initial

Description:

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

Syntax:

[C]

```
void I8024_Initial(int slot)
```

Parameter:

slot : [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Return Value:

None

Example:

```
int slot=2;
I8024_Initial(slot);
// The I-8024 card is plugged into slot 2 of LX-8000/9000 Series and initializes the I-8024 module.
```

Remark:

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

I8024_VoltageOut

Description:

This function is used to send the voltage float value to the I-8024 module with the specified channel and slot in the LX-8000/9000 Series system.

Syntax:

[C]

```
void I8024_VoltageOut(int slot, int ch, float data)
```

Parameter:

slot : [Input] Specified the LX-8000/9000 Series system slot (Range: 2 to 8)

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

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

Return Value:

None

Example:

```
int slot=2, ch=0;
float data=3.0f;
I8024_VoltageOut(slot, ch, data);
//The I-8024 module output the 3.0V voltage from the channel 0.
```

Remark:

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

I8024_CurrentOut

Description:

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

Syntax:

[C]

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

Parameter:

slot : [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

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

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

Return Value:

None

Example:

```
int slot=2, ch=0;
```

float cdata=10.0f;

I8024_CurrentOut(slot, ch, data);

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

Remark:

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

I8024_VoltageHexOut

Description:

This function is used to send the voltage value in the Hex format to the specified channel in the I-8024 module, which is plugged into the slot in the LX-8000/9000 Series system.

Syntax:

[C]

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

Parameter:

slot : [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

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

hdata : [Input] Output data with hexadecimal

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

Return Value:

None

Example:

int slot=2, ch=0; data=0x3000;

I8024_VoltageHexOut(slot, ch, data);

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

Remark:

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

I8024_CurrentHexOut

Description:

This function is used to send the current value in the Hex format to the specified channel in the analog output module I-8024, which is plugged into the slot in the LX-8000/9000 Series system.

Syntax:

[C]

```
void I8024_CurrentHexOut(int slot, int ch, int hdata)
```

Parameter:

slot : [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

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

hdata : [Input] Output data with hexadecimal

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

Return Value:

None

Example:

```
int slot=2, ch=0; data=0x2000;
I8024_CurrentHexOut(slot, ch, data);
// Output the 10.0mA current from the channel 0 of I-8024 module.
```

Remark:

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

1.7 I-8026W Module Functions

1.7.1 i8026W_Init

Description:

This function is used to initialize the driver and confirm the hardware ID information.

Syntax:

[C]

```
short i8026W_Init (int slot);
```

Parameter:

slot : [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Return Value:

0 = the module inserted in the slot is an I-8026W.

-1 = there are no I-8026W modules inserted in this slot.

Example:

```
int slot=2;
i8026W_Init (slot);
// The I-8026 card is plugged into slot 2 of LX-8000/9000 Series and initializes the I-8026 module.
```

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/ demo8026W.c

1.7.2 i8026W_GetFirmwareVer

Description:

This function is used to retrieve the version information for the FPGA firmware. The function is only used for troubleshooting or recording purposes.

Syntax:

[C]

```
short i8026W_GetFirmwareVer (int slot);
```

Parameter:

slot : [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Return Value:

Firmware version.

Example:

int slot=2;

short firmware;

firmware = i8026W_GetFirmwareVer(slot);

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/ demo8026W.c

1.7.3 i8026W_WriteAO

Description:

This function is used to write the output value to a single specified Analog Output channel in floating point format.

Syntax:

[C]

```
short i8026W_WriteAO(int slot ,int chIndex, short gain, float aoData);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

chIndex: [Input] specifies the Analog Output channel number (0 - 2)

gain: [Input] specifies the input type (0 - 4),

where: 0: +/-10 V, 1: +/-5 V, 2: +/-2.5 V, 3: +/-1.25 V, 4: +/-20 mA

aoData: [Input] the Analog Output data in floating point format

Return Value:

0 = No Error

Example:

int slot=2, ch=0, gain=0, fVal=3;

i8026W_WriteAO(slot, ch, gain, fVal);

// The I-8026 module send value 3 to channel 0 at gain 0 mode at slot 2 of LX-8000/9000 Series PAC.

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/ demo8026W.c

1.7.4 i8026W_WriteAOHex

Description:

This function is used to write the output value to a single specified Analog Output channel in hexadecimal format.

Syntax:

[C]

short i8026W_WriteAOHex (int slot ,int chIndex, short gain, short hexData);

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

chIndex: [Input] specifies the Analog Output channel number (0 - 2)

gain: [Input] specifies the input type (0 - 4),

where: 0: +/-10 V, 1: +/-5 V, 2: +/-2.5 V, 3: +/-1.25 V, 4: +/-20 mA

hexData: [Input] the Analog Output data in hexadecimal format.

Return Value:

0 = No Error

Example:

```
int slot=2, ch=0, gain=0;
short hVal=0x3fff;
i8026W_WriteAOHex(slot, ch, gain, hVal);
// The I-8026 module send hex value 0x3fff to channel 0 at gain 0 mode at slot 2
of LX-8000/9000 Series PAC.
```

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/ demo8026W.c

1.7.5 i8026W_WriteDO

Description:

This function is used to write the Digital Output value to the i-8026W module.

Syntax:

[C]

```
short i8026W_WriteDO (int slot , short doVal);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

doVal: [Input] the Digital Output value (0 - 3), as per the table below.

Output Value	CH0	CH1
0	OFF	OFF
1	ON	OFF
2	OFF	ON
3	ON	ON

Return Value:

0 = No Error

Example:

```
int slot=2, doVal=3;
i8026W_WriteDO(slot, doVal);
```

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/ demo8026W.c

1.7.6 i8026W_WriteDOBit

Description:

This function is used to set a specific Digital Output channel on the i-8026W module ON or OFF.

Syntax:

[C]

```
short i8026W_WriteDOBit(int slot , int chIndex, short bitVal);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

chIndex: [Input] specifies the Analog Output channel number (0 - 2)

bitVal: specifies the status of the digital output, where:

0: OFF 1: ON

Return Value:

0 = No Error

Example:

```
int slot=2, chIndex =0, doVal=1;
i8026W_WriteDOBit(slot, chIndex, doVal);
```

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/ demo8026W.c

1.7.7 i8026W_ReadDIO

Description:

This function is used to read the Digital Input and Digital Output values from the i-8026W module.

Syntax:

[C]

```
short i8026W_ReadDIO (int slot, short* diVal, short* doVal, unsigned char
diBitArr[], unsigned char doBitArr[]);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

*diVal: [Output] the Digital Input data

*doVal: [Output] the Digital Output data

diBitArr: [Output] the bit status of the Digital Input data

doBitArr: [Output] the bit status of the Digital Output data

Return Value:

0 = No Error

Example:

int slot=2;

short diVal=0, doVal=0;

unsigned char diBitArr[2], doBitArr[2];

i8026W_ReadDIO(slot, &diVal, &doVal, diBitArr, doBitArr);

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/ demo8026W.c

1.7.8 i8026W_ReadAI

Description:

This function is used to read the calibrated input value from a single specified Analog Input channel in floating point format.

Syntax:

[C]

```
short i8026W_ReadAI (int slot, int ch, short gain, float* fVal);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

ch: specifies the Analog Input channel number (0 ~ 5)

gain: specifies the input type (0 - 4), where: 0: +/-10 V, 1: +/-5 V, 2: +/-2.5 V, 3: +/-1.25 V, 4: +/-20 mA

fVal: [Output] the input data in float format

Return Value:

0 = No Error

Example:

int slot=2, ch=0, gain=0;

float fVal=0.0;

i8026W_ReadAI(slot, ch, gain, &fVal);

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/ demo8026W.c

1.7.9 i8026W_ReadAIHex

Description:

This function is used to read the calibrated input value from a single specified Analog Input channel in hexadecimal format.

Syntax:

[C]

short i8026W_ReadAIHex (int slot, int ch, short gain, short* hVal);

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

ch: specifies the Analog Input channel number (0 ~ 5)

gain: specifies the input type (0 - 4), where: 0: +/-10 V, 1: +/-5 V, 2: +/-2.5 V, 3: +/-1.25 V, 4: +/-20 mA

hVal: [Output] the input data in hexadecimal format.

Return Value:

0 = No Error

Example:

int slot=2, ch=0, gain=0;

short fVal=0;

i8026W_ReadAI(slot, ch, gain, &hVal);

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/ demo8026W.c

1.8 I-8048W Module Functions

1.8.1 I8048W_Init

Description:

This function is used to initialize the driver and confirm the hardware ID information.

Syntax:

[C]

```
int I8048W_Init (int slot);
```

Parameter:

slot : [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Return Value:

0 = the module inserted in the slot is an I-8048W.

-1 = there are no I-8048W modules inserted in this slot.

Example:

```
int slot=2;
i8048W_Init (slot);
// The I-8048 card is plugged into slot 2 of LX-8000/9000 Series and initializes the I-8048 module.
```

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/8048W/8048W_Int.c

1.8.2 I8048W_UnFreezeINT

Description:

This function is used to clear interrupted status of an i-8048w channel.

Syntax:

[C]

```
void I8048W_UnFreezeINT (int slot, int Channel);
```

Parameter:

slot : [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Channel: [Input] Specify the I-8048W channel (Range: 0 to 7)

Return Value:

None

Example:

slot=2, ch=0;

```
void ISRFUN (int slot)
{
        int Ret;
        if(I8048W_Read_RisingEvent(slot,ch) != 0)    //rising INT status enable
                        I8048W_Read_RisingEventCount(slot,ch);
        if(I8048W_Read_FallingEvent(slot,ch) != 0)    //falling INT status enable
                        I8048W_Read_FallingEventCount(slot,ch);
        I8048W_UnFreezeINT(slot,ch);
}
// clear channel 0 interrupted status.
```

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/8048W/8048W_Int.c
Note:
If the interrupt of an i-8048w channel does not clear to LOW,
then this channel interrupts will be blocked and the CPU will not
be able to receive any further interrupts.

1.8.3 I8048W_InstallISR

Description:

This function is used to install a slot interrupt service route for i-8048W.

Syntax:

[C]

```
short I8048W_InstallISR(int slot, void (*isr)(int slot));
```

Parameter:

slot : [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

(*isr)(int slot): a callback function.

Return Value:

0 = No error

1 = Error

Example:

slot=2, ch=0;

I8048W_InstallISR(slot, ISRFUN);

```
void ISRFUN (int slot)
{
        int Ret;
        if(I8048W_Read_RisingEvent(slot,ch) != 0)    //rising INT status enable
                        I8048W_Read_RisingEventCount(slot,ch);
        if(I8048W_Read_FallingEvent(slot,ch) != 0)    //falling INT status enable
                        I8048W_Read_FallingEventCount(slot,ch);
        I8048W_UnFreezeINT(slot,ch);
}
```

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/8048W/8048W_Int.c

1.8.4 I8048W_UnInstallISR

Description:

This function is used to uninstall a slot interrupt service route and disables a hardware interrupt for i-8048W.

Syntax:

[C]

```
void I8048W_UnInstallISR (int slot);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Return Value:

None

Example:

slot=2;

I8048W_UnInstallISR(slot);

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/8048W/8048W_Int.c

1.8.5 I8048W_Read_RisingEvent

Description:

This function is used to read rising interrupt status of an i-8048W channel.

Syntax:

[C]

```
int I8048W_Read_RisingEvent (int slot, int Channel);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Channel: [Input] Specify the I-8048W channel (Range: 0 to 7)

Return Value:

1: Rising interrupt occur

0: No rising interrupt occur

Example:

slot=2, ch=0, ret;

ret = I8048W_Read_RisingEvent(slot,ch); //rising INT status enable

printf("%d\n", ret);

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/8048W/8048W_Int.c

1.8.6 I8048W_Read_FallingEvent

Description:

This function is used to read falling interrupt status of an i-8048W channel.

Syntax:

[C]

```
int I8048W_Read_FallingEvent (int slot, int Channel);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Channel: [Input] Specify the I-8048W channel (Range: 0 to 7)

Return Value:

1: Falling interrupt occur

0: No falling interrupt occur

Example:

slot=2, ch=0, ret;

ret = I8048W_Read_FallingEvent (slot,ch); //falling INT status enable

printf("%d\n", ret);

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/8048W/8048W_Int.c

1.8.7 I8048W_Read_RisingEventCount

Description:

This function is used to read total count values of the rising interrupt occurred on an i-8048W channel.

Syntax:

[C]

```
DWORD I8048W_Read_RisingEventCount (int slot, int Channel);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Channel: [Input] Specify the I-8048W channel (Range: 0 to 7)

Return Value:

Rising interrupt occurred total count.

Example:

slot=2, ch=0, ret;

ret = I8048W_Read_RisingEventCount (slot,ch);

printf("%d\n", ret);

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/8048W/8048W_Int.c

1.8.8 I8048W_Read_FallingEventCount

Description:

This function is used to read total count values of the falling interrupt occurred on an i-8048W channel.

Syntax:

[C]

```
DWORD I8048W_Read_FallingEventCount (int slot, int Channel);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Channel: [Input] Specify the I-8048W channel (Range: 0 to 7)

Return Value:

Falling interrupt occurred total count.

Example:

slot=2, ch=0, ret;

ret = I8048W_Read_FallingEventCount (slot,ch);

printf("%d\n", ret);

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/8048W/8048W_Int.c

1.8.9 I8048W_Clear_RisingEventCount

Description:

This function is used to clear total count values of the rising interrupt occurred on an i-8048W channel.

Syntax:

[C]

```
void I8048W_Clear_RisingEventCount (int slot, int Channel);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Channel: [Input] Specify the I-8048W channel (Range: 0 to 7)

Return Value:

None

Example:

slot=2, ch=0;

I8048W_Clear_RisingEventCount(slot, ch); //clear channel 0 rising event count.

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/8048W/8048W_Int.c

1.8.10 I8048W_Clear_FallingEventCount

Description:

This function is used to clear total count values of the falling interrupt occurred on an i-8048W channel.

Syntax:

[C]

```
void I8048W_Clear_FallingEventCount (int slot, int Channel);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Channel: [Input] Specify the I-8048W channel (Range: 0 to 7)

Return Value:

None

Example:

slot=2, ch=0;

I8048W_Clear_FallingEventCount(slot, ch); //clear channel 0 falling event count.

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/8048W/8048W_Int.c

1.8.11 I8048W_Set_RisingReg

Description:

This function is used to Enable/Disable the rising interrupt of an i-8048W channel.

Syntax:

[C]

void I8048W_Set_RisingReg (int slot, int Channel, int Enable);

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Channel: [Input] Specify the I-8048W channel (Range: 0 to 7)

Enable: [Input] 1: enable, 0: disable

Return Value:

None

Example:

slot=2, ch=0, rising=1;

I8048W_Set_RisingReg(slot, ch, rising); //set rising edge register enable

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/8048W/8048W_Int.c

1.8.12 I8048W_Set_FallingReg

Description:

This function is used to Enable/Disable the falling interrupt of an i-8048W channel.

Syntax:

[C]

void I8048W_Set_FallingReg (int slot, int Channel, int Enable);

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Channel: [Input] Specify the I-8048W channel (Range: 0 to 7)

Enable: [Input] 1: enable, 0: disable

Return Value:

None

Example:

slot=2, ch=0, falling=1;

I8048W_Set_FallngReg(slot, ch, falling); //set falling edge register enable

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/8048W/8048W_Int.c

1.8.13 I8048W_Read_RisingReg

Description:

This function is used to read the rising interrupt setting status of an i-8048W channel.

Syntax:

[C]

```
int I8048W_Read_RisingReg (int slot, int Channel);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Channel: [Input] Specify the I-8048W channel (Range: 0 to 7)

Return Value:

1: Enable

0: Disable

Example:

slot=2, ch=0, ret;

I8048W_Read_RisingReg(slot, ch); //read channel 0 rising edge register.

printf("%d\n", ret);

Remark:

1.8.14 I8048W_Read_FallingReg

Description:

This function is used to read the falling interrupt setting status of an i-8048W channel.

Syntax:

[C]

```
int I8048W_Read_FallingReg (int slot, int Channel);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Channel: [Input] Specify the I-8048W channel (Range: 0 to 7)

Return Value:

1: Enable

0: Disable

Example:

slot=2, ch=0, ret;

I8048W_Read_FallingReg(slot, ch); //read channel 0 falling edge register.

printf("%d\n", ret);

Remark:

1.8.15 I8048W_DI_ALL

Description:

This function is used to reads the DI values of all channels of an i-8048W.

Syntax:

[C]

```
int I8048W_DI_ALL (int slot);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Return Value:

Return value can analysis each channel status.

0~7 bit corresponding 0~7 channel.

Example:

slot=2, ret;

ret = I8048W_DI_ALL(slot); //read 0~7 channel status

printf("%d\n", ret);

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/8048W/ getdi-8048W.c

1.8.16 I8048W_DI_Ch

Description:

This function is used to reads single DI channel status with i-8048W.

Syntax:

[C]

```
int I8048W_DI_Ch (int slot, int Channel);
```

Parameter:

slot: [Input] Specify the LX-8000/9000 Series system slot (Range: 2 to 8)

Channel: [Input] Specify the I-8048W channel (Range: 0 to 7)

Return Value:

1: DI ON

0: DI OFF

Example:

slot=2, ch=0, ret;

ret = I8048W_DI_Ch(slot, ch);; //read DI channel 0 status

printf(“%d\n”, ret); //0 OFF, 1 ON

Remark:

Refer to LinPAC_X86_SDK/examples/lx-series/i8k/8048W/ getdi-8048W.c

1.9 I-8014W Module Functions

In this section you can link below URL to get I-8014W module introduction.
http://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x4x/user_manual/i-8014w_api_reference.pdf

1.10 I-8084W Module Functions

In this section you can link below URL to get I-8084W module introduction.
http://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x4x/user_manual/i-8084w_api_reference.pdf

1.11 I-8088W Module Functions

In this section you can link below URL to get I-8088W module introduction.
http://ftp.icpdas.com/pub/cd/linpac/napdos/lp-8x4x/user_manual/i-8088w_api_reference.pdf

1.12 The Software Develop Toolkit Error Code

Error Code	Error ID
0	NoError
1	FunctionError
2	PortError
3	BaudRateError
4	DataError
5	StopError
6	ParityError

7	CheckSumError
8	ComPortNotOpen
9	SendThreadCreateError
10	SendCmdError
11	ReadComStatusError
12	StrCheck Error
13	CmdError
14	X
15	TimeOut
16	X
17	ModuleId Error
18	AdChannelError
19	UnderRange
20	ExceedRange
21	InvalidateCounterNo
22	InvalidateCounterValue
23	InvalidateGateMode
24	InvalidateChannelNo
25	ComPortInUse

2. Demo of LX-8000/9000 Series Modules With C Language

In this section, we will focus on examples for the description and application of the control sections on the I-7000/I-8000/I-87K series modules for use in the LX-8000/9000 Series. After user downloading the latest version of LX-8000/9000 Series SDK “LinPAC_X86_SDK.tgz”,(refer to chapter 1) user could use command “**tar zxvf LinPAC_X86_SDK.tgz**” to unzip SDK in the LX-8000/9000 Series, user can find all demo programs (include demo source code and executes file) in the “examples” directory.

2.1 I-7k Modules DIO Control Demo

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

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

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

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

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

    // Check Open ttyS1
```

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

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

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

Follow the below steps to achieve the desired results：

STEP 1：(To transmit “LinPAC_X86_SDK.tgz” SDK to LX-8000/9000 Series PAC)

User could use WinSCP software or linux “scp” command to transmit **“LinPAC_X86_SDK.tgz”** from Windows or Linux PC to LX-8000/9000 Series, or you can use command “wget http://ftp.icpdas.com.tw/pub/cd/linpac/napdos/lx-series/sdk/linpac_x86_sdk.tgz to download linpac_x86_sdk.tgz in LX-8000/9000 Series.

STEP 2：(To connect to LX-8000/9000 Series to execute i7kdio)

User could use putty software to connect to LX-8000/9000 Series.

STEP 3：(To execute i7kdio in the "LinPAC_X86_SDK/examples/lx-series/i7k)

After users connect to LX-8000/9000 Series, user could find the demo "i7kdio" in the directory **"LinPAC_X86_SDK/examples/lx-series/i7k**. The result of execution refers to Fig. 2-1.

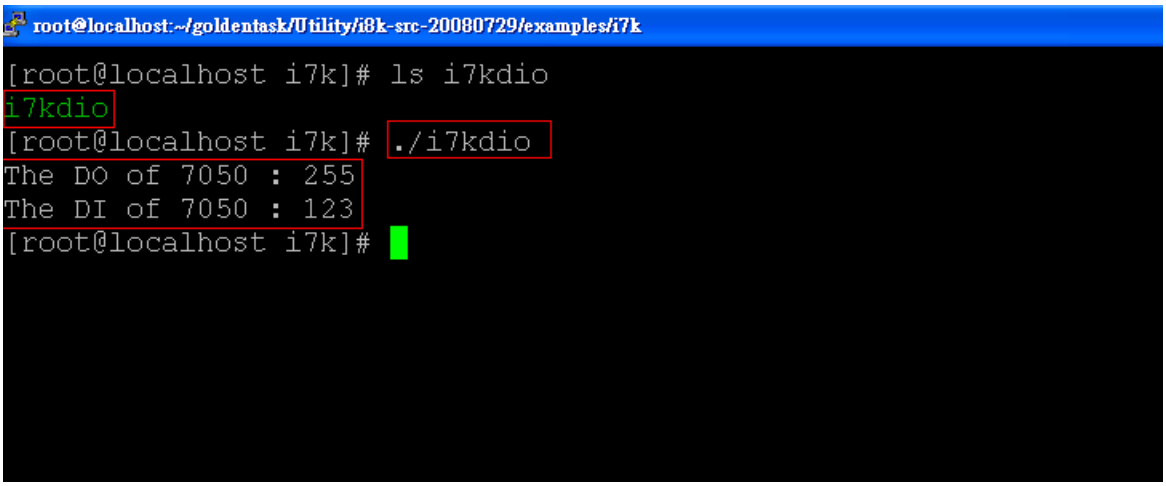


Fig. 2-1

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

2.2 I-7k Modules AIO Control Demo

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

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

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

```
#include "msw.h"

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

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

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

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

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

	//--- Analog Input ----    ****     7017 -- AI    ****
```

```
    j = 1;
    wBuf[0] = ttyS1;            // COM Port
    wBuf[1] = 0x03;             // Address
    wBuf[2] = 0x7017;           // ID
    wBuf[3] = 0;                // CheckSum disable
    wBuf[4] = 100;              // TimeOut , 100 msecond
    wBuf[5] = j;                // Channel of AI
    wBuf[6] = 0;                // string debug

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

    Close_Com(ttyS1);

    return 0;
}
```

All the steps from programming to execution are the same as those in the section 2.1. The result of execution refers to Fig. 2-2.

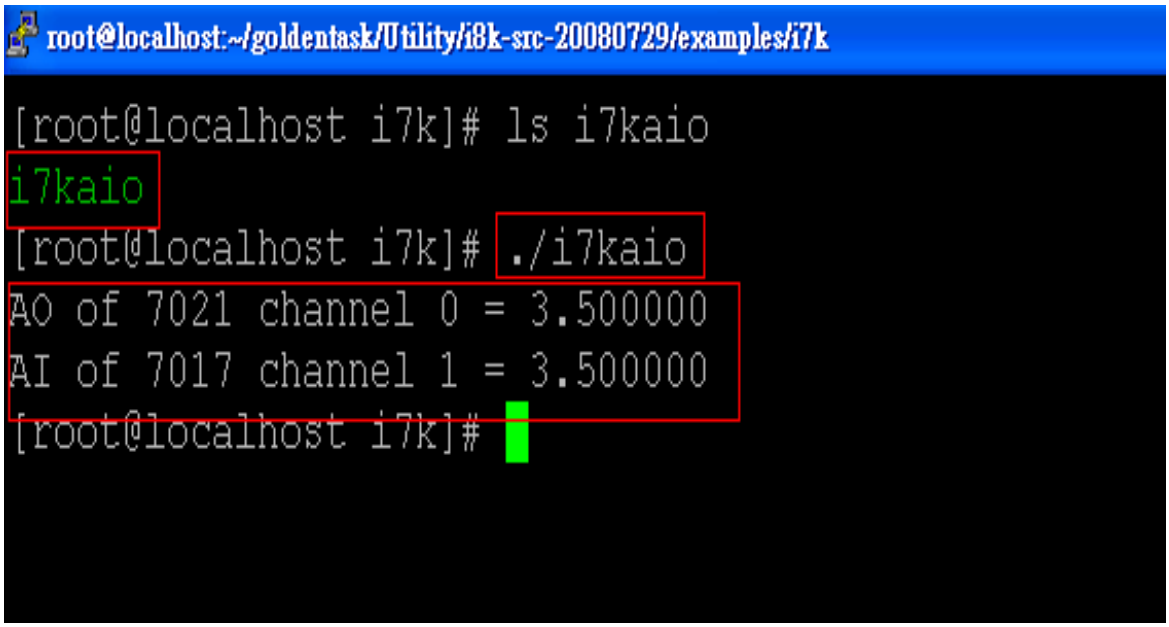


Fig. 2-2

2.3 I-87k Modules DIO Control Demo

When using I-87k modules for I/O control of the LX-8000/9000 Series, the program will be a little different, according to the location of I-87k modules. There are three conditions for the location of the I-87k modules:

(1) When I-87k DIO modules are **in the LX-8000/9000 Series slots**, the two functions " Open_Slot " and " ChangeToSlot ", must be added before using other functions for the I-87k modules and the function of "Close_Slot() " also needs to be added to the end of the program. Please refer to demo in section 2.3.1.

(2) When I-87K DIO modules are **in the I-87k I/O expansion unit slots**, then please refer to the demo in section 2.3.2.

(3) When the I-87k DIO modules are **in the I-8000 controller slots**, then the I-87k modules will be regarded as I-8k modules and so please refer to I/O control of I-8k modules in section 2.5.2

2.3.1 I-87k Modules in slots of LX-8000/9000 Series

This demo – **i87kdio.c** will illustrate how to control the DI/DO with the I-87054 module (8 DO channels and 8 DI channels). The I-87054 module is in slot 3 of the LX-8000/9000 Series. The address and baudrate in the LX-8000/9000 Series are constant and they are 00 and 115200 respectively. The result of this demo let's DO channel 0 ~ 7 of I-87054 output and DI channel 1 of I-87054 input. The source code of this demo program is as follows：

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

char szSend[80], szReceive[80];
DWORD dwBuf[12];
```

```
float fBuf[12];

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

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

    //Check Open ttySA0
    wRetVal = Open_Com(ttySA0, 115200, Data8Bit, NonParity,
OneStopBit);
    if (wRetVal > 0) {
        printf("open port failed!\n");
        return (-1);
    }

    //Choose Slot3
    ChangeToSlot(3);
    //--- digital output ----   **(DigitalOut_87k()**)
    dwBuf[0] = ttySA0;              // COM Port
    dwBuf[1] = 00;                  // Address
    dwBuf[2] = 0x87054;             // ID
    dwBuf[3] = 0;                   // CheckSum disable
    dwBuf[4] = 100;                 // TimeOut , 100 msecond
    dwBuf[5] = 0xff;                // digital output
    dwBuf[6] = 0;                   // string debug
  wRetVal = DigitalOut_87k(dwBuf, fBuf, szSend, szReceive); // DO Output
  printf("DO Value= %u", dwBuf[5]);

  //--- digital Input ----     **(DigitalIn_87k()**)
    dwBuf[0] = ttySA0;              // COM Port
```

```
    dwBuf[1] = 00;                  // Address
    dwBuf[2] = 0x87054;             // ID
    dwBuf[3] = 0;                   // CheckSum disable
    dwBuf[4] = 100;                 // TimeOut , 100 msecond

    dwBuf[6] = 0;                   // string debug
    getch();
    DigitalIn_87k(dwBuf, fBuf, szSend, szReceive);     // DI Input
   printf("DI= %u",dwBuf[5])
    //--- digital output ----     ** Close DO **
    dwBuf[0] = ttySA0;              // COM Port
    dwBuf[1] = 00;                  // Address
    dwBuf[2] = 0x87054;             // ID
    dwBuf[3] = 0;                   // CheckSum disable
    dwBuf[4] = 100;                 // TimeOut , 100 msecond
    dwBuf[5] = 0x00;                // digital output
    dwBuf[6] = 0;                   // string debug
    getch();                        // push any key to continue
   wRetVal = DigitalOut_87k(dwBuf, fBuf, szSend, szReceive);

    Close_Com(ttySA0);
    Close_SlotAll();
    return 0;
}
```

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

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

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

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

(3) **Open ttySA0**(internal serial port of LX-8000/9000 Series) will be modified to **open ttyS1**(RS-485 port of LX-8000/9000 Series)

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

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

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

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

  //--- digital Input ----    **(DigitalIn_87k()**)
    dwBuf[0] = ttyS1;           // COM Port
    dwBuf[1] = 06;              // Address
    dwBuf[2] = 0x87054;         // ID
```

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

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

All the steps from programming to execution are the same as those in the section 2.1. The result of execution refers to Fig. 2-3.

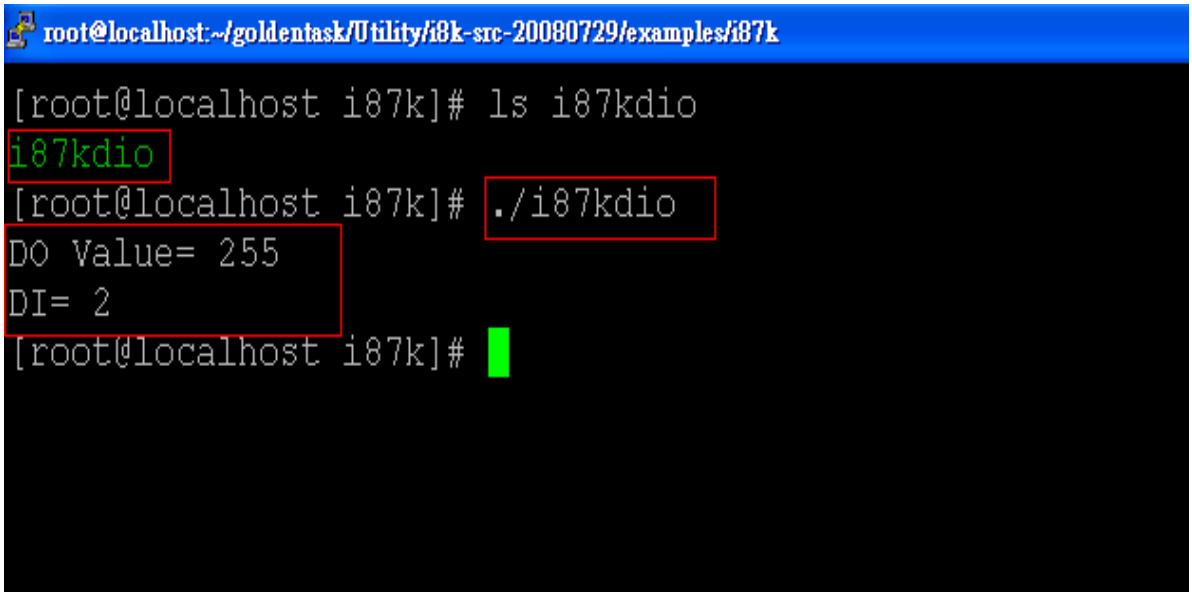


Fig. 2-3

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

If the I-87k DIO modules are in the I-8000 controller slots, I-87k modules will be regarded as I-8k modules and so please refer to DI/DO control of I-8k modules in the section 2.5.

2.4 I-87k Modules AIO Control Demo

When using I-87k modules for I/O control of the LX-8000/9000 Series, according to the location of the I-87k modules, the program will be a little different. There are three conditions for the location of the I-87k modules：

(1) When the I-87k AIO modules are **in the LX-8000/9000 Series slots**, the two functions “ Open_Slot ” and “ ChangeToSlot ” must be added before using the other functions of the I-87k modules and the function “ Close_Slot() “ also needs to be added to the end of the program. Please refer to the demo in section 2.4.1.

(2) When I-87K AIO modules are **in the I-87k I/O expansion unit slots**, please refer to the demo in section 2.4.2.

(3) When the I-87k AIO modules are **in the I-8000 controller slots**, the I-87k modules will be regarded as I-8k modules and so please refer to I/O control of I-8k modules in section 2.4.3.

2.4.1 I-87k Modules in slots of LX-8000/9000 Series

This demo – **i87kaio.c** will illustrate how to control the AI/AO with the I-87022 module (2 AO channels) and the I-87017 module (8 AI channels).The I-87022 and I-87017 modules are plugged into slot 2 and slot 3 of the LX-8000/9000 Series separately. The address and baudrate in the LX-8000/9000 Series are constant and they are 00 and 115200 separately. The result of this demo lets AO channel 0 of I-87022 output 2.5V and AI channel 1 of I-87017 input. The source code of this demo program is as follows：

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

char szSend[80], szReceive[80];
DWORD wBuf[12];
DWORD    wBuf7[12];
float fBuf[12];
/* ---------------------------------------------------------------- */
int main()
{
      int i,j, wRetVal;
      DWORD temp;

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

      //Check Open ttySA0
      wRetVal = Open_Com(ttySA0, 115200, Data8Bit, NonParity, OneStopBit);
      if (wRetVal > 0) {
            printf("open port failed!\n");
            return (-1);
      }

      ChangeToSlot(2);
      //--- Analog output ----    ****     87022 -- AO    ****
      i=0;
      wBuf[0] = ttySA0;             // COM Port
      wBuf[1] = 0x00;               // Address
      wBuf[2] = 0x87022;            // ID
      wBuf[3] = 0;                  // CheckSum disable
      wBuf[4] = 100;                // TimeOut , 100 msecond
      wBuf[5] = i;                  // Channel Number of AO
      wBuf[6] = 0;                  // string debug
```

```
    fBuf[0] = 2.5;                  // AO Value

    wRetVal = AnalogOut_87k(wBuf, fBuf, szSend, szReceive);
    if (wRetVal)
        printf("AO of 87022 Error !, Error Code=%d\n", wRetVal);
    else
        printf("AO of 87022 channel %d = %f \n",i,fBuf[0]);

    ChangeToSlot(3);
    //--- Analog Input ----   ****     87017 -- AI    ****
    j=1;
    wBuf7[0] = ttySA0;          // COM Port
    wBuf7[1] = 0x00;            // Address
    wBuf7[2] = 0x87017;         // ID
    wBuf7[3] = 0;               // CheckSum disable
    wBuf7[4] = 100;             // TimeOut , 100 msecond
    wBuf7[5] = j;               //Channel Number of AI
    wBuf7[6] = 0;               // string debug

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

    Close_Com(ttySA0);
    Close_SlotAll();
    return 0;
}
```

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

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

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

(2) The **address** and **baudrate** of I-87k modules in the network of RS-485 need

to be set by DCON Utility

(3) **Open ttySA0** (internal serial port of LX-8000/9000 Series) will be modified to **open ttyS1** (RS-485 port of LX-8000/9000 Series)

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

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

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

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

```
            wRetVal = AnalogOut_87k(wBuf, fBuf, szSend, szReceive);
        if (wRetVal)
            printf("AO of 87022 Error !, Error Code=%d\n", wRetVal);
        else
            printf("AO of 87022 channel %d = %f \n",i,fBuf[0]);
    //--- Analog Input ----    ****     87017 -- AI    ****
    j=1;
    wBuf7[0] = ttyS1;               // COM Port
    wBuf7[1] = 0x02;                // Address
    wBuf7[2] = 0x87017;             // ID
    wBuf7[3] = 0;                   // CheckSum disable
    wBuf7[4] = 100;                 // TimeOut , 100 msecond
    wBuf7[5] = j;                   //Channel Number of AI
    wBuf7[6] = 0;                   // string debug
                wRetVal = AnalogIn_87k(wBuf7, fBuf, szSend, szReceive);
        if (wRetVal)
            printf("AI of 87017 Error !, Error Code=%d\n", wRetVal);
        else
            printf("AI of 87017 channel %d = %f \n",j,fBuf[0]);

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

All the steps from programming to execution are the same as those in the section 2.1. The result of execution refers to Fig. 2-4.

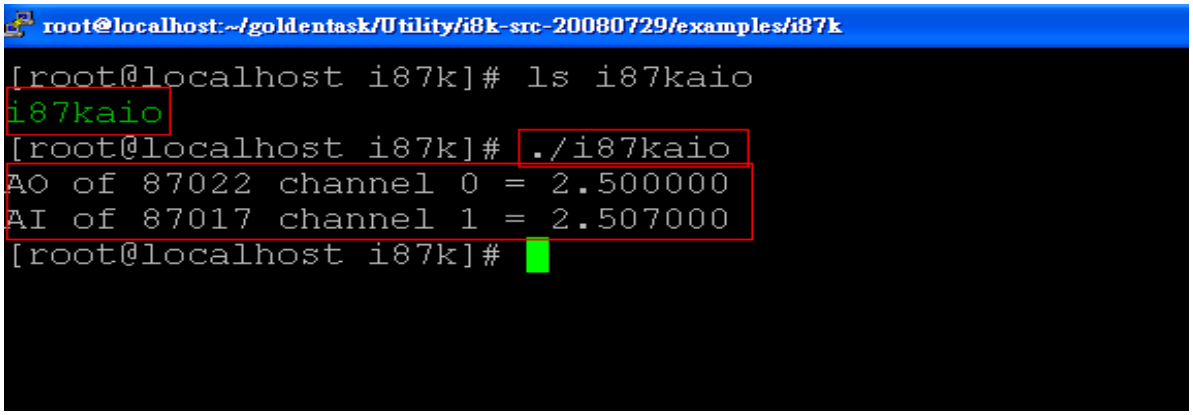


Fig. 2-4

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

If the I-87k AIO modules are in slots of I-8000 controller, I-87k modules will be regarded as I-8k modules and refer to AI/AO control of I-8k modules in the section 2.6.

2.5 I-8k Modules DIO Control Demo

I8000.c of libPAC_x86.a is the source file for i8k modules in slots of I-8000 controller. **slot.c** of libPAC_x86.a is the source file for i8k modules in slots of LX-8000/9000 Series. Therefore the functions for i8k modules in slots of LX-8000/9000 Series and in slots of I-8000 controller are different completely. There are two conditions for the location of the I-8k modules：

(1) When I-8K DIO modules are **in the LX-8000/9000 Series**, then please refer to the demo in section 2.5.1.

(2) When I-8K DIO modules are **in the I-8000 controller**, then please refer to the demo in section 2.5.2.

2.5.1 I-8k Modules in slots of LP-8X81 Series

In this section, this demo program – **i8kdio.c** will introduce how to control the DI/DO with the I-8055 (8 DO channels and 8 DI channels) module and it is plugged into slot 3 of the LX-8000/9000 Series.

The address and baudrate in the LX-8000/9000 Series are constant and they are 00 and 115200 separately. The result of this demo let's DO channel 0 ~7 of I-8055 output and DI channel 0 of I-8055 input. The source code of this demo program is as follows：

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

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

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

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

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


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

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

All the steps from programming to execution are the same as those in the section 2.1. The result of execution refers to Fig. 2-5.

Fig. 2-5

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

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

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

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

(3) Connect the **ttyS1** of LX-8000/9000 Series to the com1 of I-8000 controller with the RS-232 cable.

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

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

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

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

    //Check Open ttyS1
    wRetVal = Open_Com(ttyS1, 115200, Data8Bit, NonParity, OneStopBit);
    if (wRetVal > 0) {
        printf("open port failed!\n");
        return (-1);
    }
```

```
   //--- digital output ----    **(DigitalOut_8K()**)
   dwBuf[0] = ttyS1;              // COM Port
   dwBuf[1] = 01;                 // Address
   dwBuf[2] = 0x8055;             // ID
   dwBuf[3] = 0;                  // CheckSum disable
   dwBuf[4] = 100;                // TimeOut , 100 msecond
   dwBuf[5] = 0xff;               // digital output
   dwBuf[6] = 0;                  // string debug
   dwBuf[7] = 1;                  // slot number

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

//--- digital Input ----     **(DigitalIn_8K()**)
   dwBuf[0] = ttyS1;              // COM Port
   dwBuf[1] = 01;                 // Address
   dwBuf[2] = 0x8055;             // ID
   dwBuf[3] = 0;                  // CheckSum disable
   dwBuf[4] = 100;                // TimeOut , 100 msecond
   dwBuf[6] = 0;                  // string debug
   dwBuf[7] = 1;                  // slot number

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

//--- digital output ----     ** Close DO **
   dwBuf[0] = ttyS1;              // COM Port
   dwBuf[1] = 01;                 // Address
   dwBuf[2] = 0x8055;             // ID
   dwBuf[3] = 0;                  // CheckSum disable
   dwBuf[4] = 100;                // TimeOut , 100 msecond
   dwBuf[5] = 0x00;               // digital output
   dwBuf[6] = 0;                  // string debug
   dwBuf[7] = 1;                  // slot number
```

```
    getch();                        // push any key to continue
    wRetVal = DigitalOut_8K(dwBuf, fBuf, szSend, szReceive);

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

All the steps from programming to execution are the same as those in the section 2.1. The result of execution refers to Fig. 2-6.

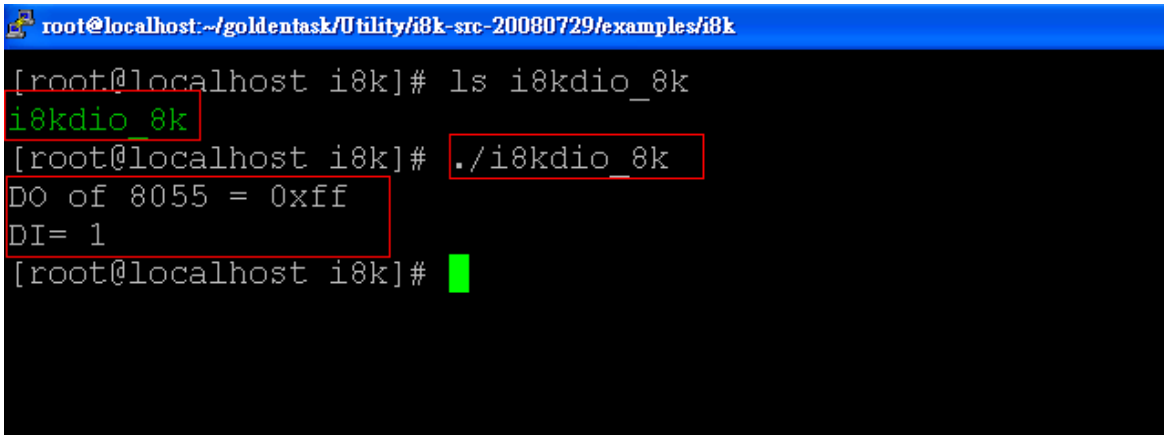


Fig. 2-6

2.6 I-8k Modules AIO Control Demo

I8000.c of libPAC_x86.a is the source file for i8k modules in slots of I-8000 controller. **slot.c** of libPAC_x86.a is the source file for i8k modules in slots of the LX-8000/9000 Series. Therefore the functions for the i8k modules in LX-8000/9000 Series slots and in the I-8000 controller slots are completely different. There are two conditions for the location of the I-8k modules：

(1) When I-8K AIO modules are **in the LX-8000/9000 Series**, then please refer to the demo in section 2.6.1.

(2) When I-8K AIO modules are **in the I-8000 controller**, then please refer to the demo in section 2.6.2.

2.6.1 I-8k Modules in slots of LP-8X81 Series

In this section, this demo program – **i8kaio.c** will illustrate how to control the AI/AO with the I-8024 (4 AO channels) and I-8017 (8 AI channels) module and they

are in slot 2 and slot 3 of the LX-8000/9000 Series separately.

The address and baudrate in the LX-8000/9000 Series are constant and they are 00 and 115200 separately. The result of this demo lets AO voltage channel 0 of I-8024 output 5.5V and AI channel 2 of I-8017H input. The source code of this demo program is as follows：

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

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

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

    //I-8024
    wRetVal = Open_Slot(2);
    if (wRetVal > 0) {
        printf("open Slot failed!\n");
        return (-1);
    }

    //I8024 Initial
    I8024_Initial(2);

    //I8024_AO Output
    I8024_VoltageOut(2,0,5.5);
    Close_Slot(2);

    /*************************************/
```

```
	//I-8017H
	wRetVal = Open_Slot(3);
	if (wRetVal > 0) {
		printf("open Slot failed!\n");
		return (-1);
	}

	//I8017H Initial
	I8017_Init(3);
	//I8017H _Channel Setup
	I8017_SetChannelGainMode(3,2,0,0);

	// First Method：Get AI Value
	hAi = I8017_GetCurAdChannel_Hex(3);	//Get Not-calibrated AI Hex Value
	printf("8017_AI_not_Cal_Hex =%x\n",hAi);
	fAi = HEX_TO_FLOAT_Cal(hAi,3,0);		//Not-calibrated AI Hex Value modify
to calibrated AI Float Value
	printf("8017_AI_Cal_Float =%f\n\n",fAi);

	// Second Method：Get AI Value
	hAi = I8017_GetCurAdChannel_Hex_Cal(3);	//Get Calibrated AI Hex Value
	printf("8017_AI_Cal_Hex =%x\n",hAi);
	fAi = CalHex_TO_FLOAT(hAi,0);			//Calibrated AI Hex Value modify
to Calibrated AI Float Value
	printf("8017_AI_Cal_Float =%f\n\n",fAi);

	// Third Method：Get AI Value
	fAi = I8017_GetCurAdChannel_Float_Cal(3);	//Get Calibrated AI Float Value
	printf("8017_AI_Cal_Float =%f\n\n\n",fAi);

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

All the steps from programming to execution are the same as those in the section 2.1. The result of execution refers to Fig. 2-7.

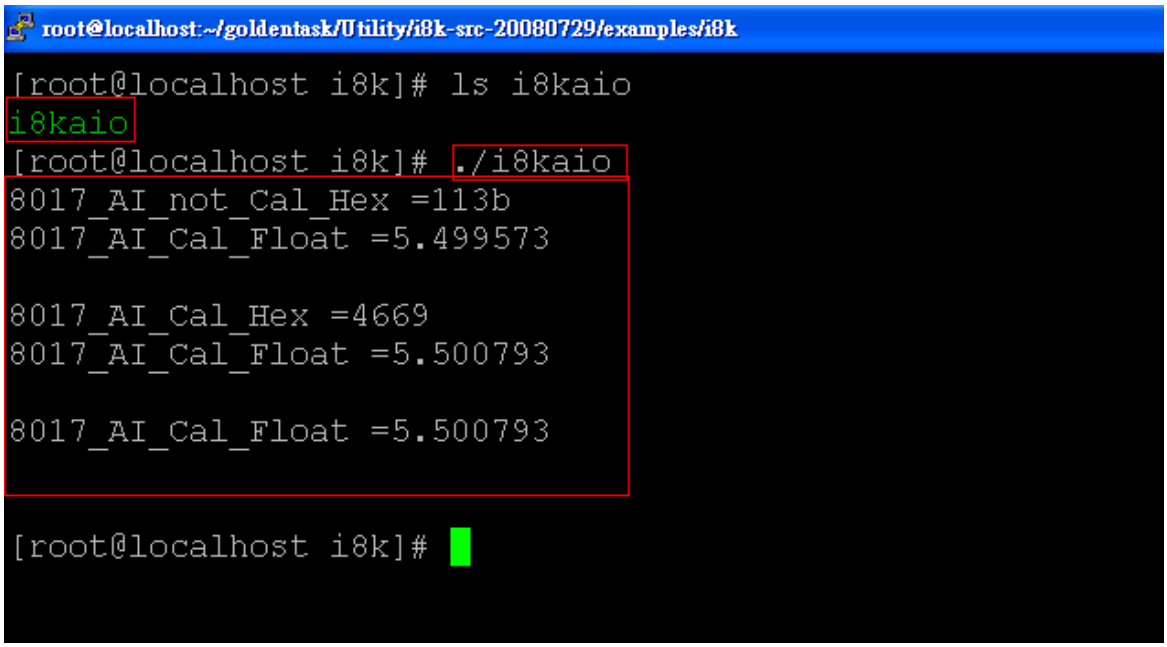


Fig. 2-7

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

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

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

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

(3) Connect **ttyS1** of LX-8000/9000 Series to com1 of I-8000 controller with RS-232 cable.

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

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

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

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

    wRetVal = Open_Com(ttyS1, 115200, Data8Bit, NonParity, OneStopBit);
    if (wRetVal > 0) {
        printf("open port failed!\n");
        return (-1);
    }

    //--- Analog output ----   ****    8024 -- AO   ****
    i = 0;
    wBuf[0] = ttyS1;            // COM Port
    wBuf[1] = 0x01;             // Address
    wBuf[2] = 0x8024;           // ID
    wBuf[3] = 0;                // CheckSum disable
    wBuf[4] = 100;              // TimeOut , 100 msecond
    wBuf[5] = i;                // Channel No. of AO
    wBuf[6] = 0;                // string debug
    wBuf[7] = 0;                // Slot Number
    fBuf[0] = 3.5;

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

    //--- Analog Input ----   ****    8017H -- AI   ****
    j = 2;
    wBuf[0] = ttyS1;            // COM Port
```

```
    wBuf[1] = 0x01;             // Address
    wBuf[2] = 0x8017;           // ID
    wBuf[3] = 0;                // CheckSum disable
    wBuf[4] = 100;              // TimeOut , 100 msecond
    wBuf[5] = j;                // Channel of AI
    wBuf[6] = 0;                // string debug
    wBuf[7] = 1;                // Slot Number

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

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

All the steps from programming to execution are the same as those in the section 2.1. The result of execution refers to Fig. 2-8

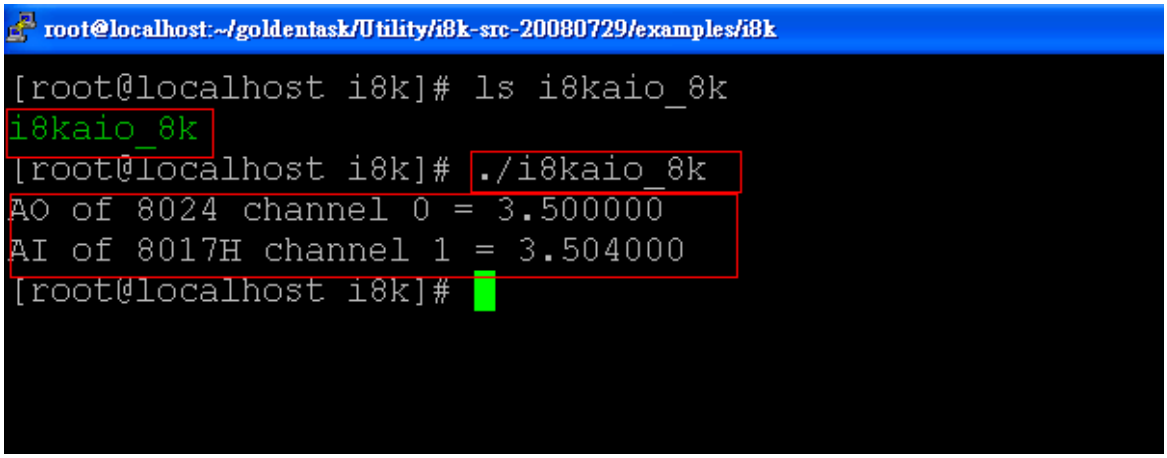


Fig. 2-8

2.7 Conclusion of Module Control Demo

Fig. 2-9 is the table of communication functions for the I-7000/I-8000/I-87000 modules in different locations. When using the ICP DAS modules in the LX-8000/9000 Series, this table will be helpful to let users understand which functions of communication should be used.

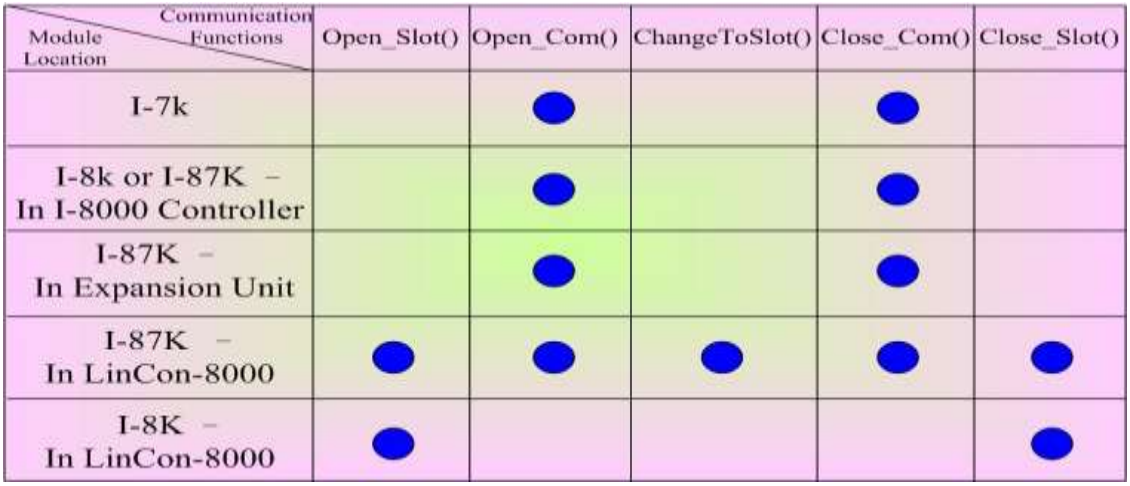


Fig 2-9

Fig. 2-10 is the table of source files for the I-7000/I-8000/I-87000 modules in different locations. When using ICP DAS modules in the LX-8000/9000 Series, this table will be helpful to let users understand which source files of the libPAC_x86.a should be called.

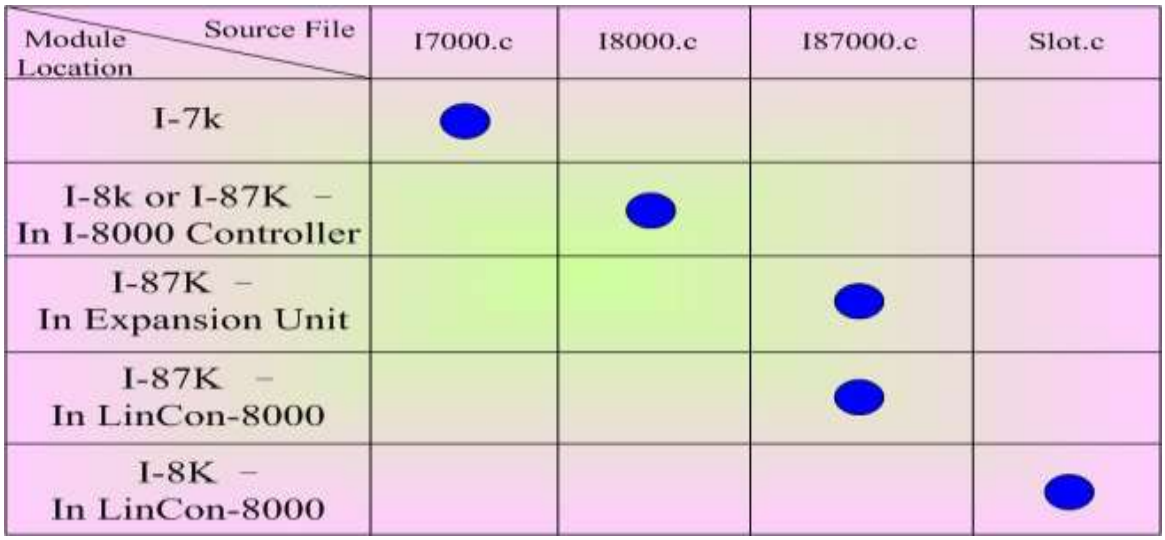


Fig. 2-10