MP-8x43
Function Reference

(Version 1.0)

ICP DAS CO., LTD.

Warranty

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

Warning

ICPDAS Inc. assumes no liability for damages consequent to the use of this product. ICPDAS Inc. reserves the right to change this manual at any time without notice. The information furnished by ICPDAS Inc. is believed to be accurate and reliable. However, no responsibility is assumed by ICPDAS Inc. for its use, or 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.

Technical Support

If you have problems about using the product, please contact

ICP DAS Product Support.

Email: Service@icpdas.com

TABLE OF CONTENTS

1. PREFACE

1.1 Introduction

Welcome to the EzPlatformSDK application library user's manual. ICPDAS provides Library files, namely the EzPlatformSDK application library, for the I-8000 series modules which are used in the MP-8x43 Embedded Controller. The EzPlatformSDK application library has all the essential Library functions designed for the I-8000 series modules for Microsoft WinCE 6.0 platform. User can use it to Developed the MP-8x43 application program. It can be applied on MP-8x43's WinCE 6.0 OS, and even on the newer platforms. Users can easily develop WinCE 6.0 applications for MP-8x43 by using this toolkit on PC. The various functions in the EzPlatformSDK application library are divided into the following sub-group functions for easy use in different applications. The main functions of the MP-8x43 embedded controller are depicted in figure 1.2, which include VGA, USB (mouse and keyboard), compact flash, series, Ethernet and an I/O interface.

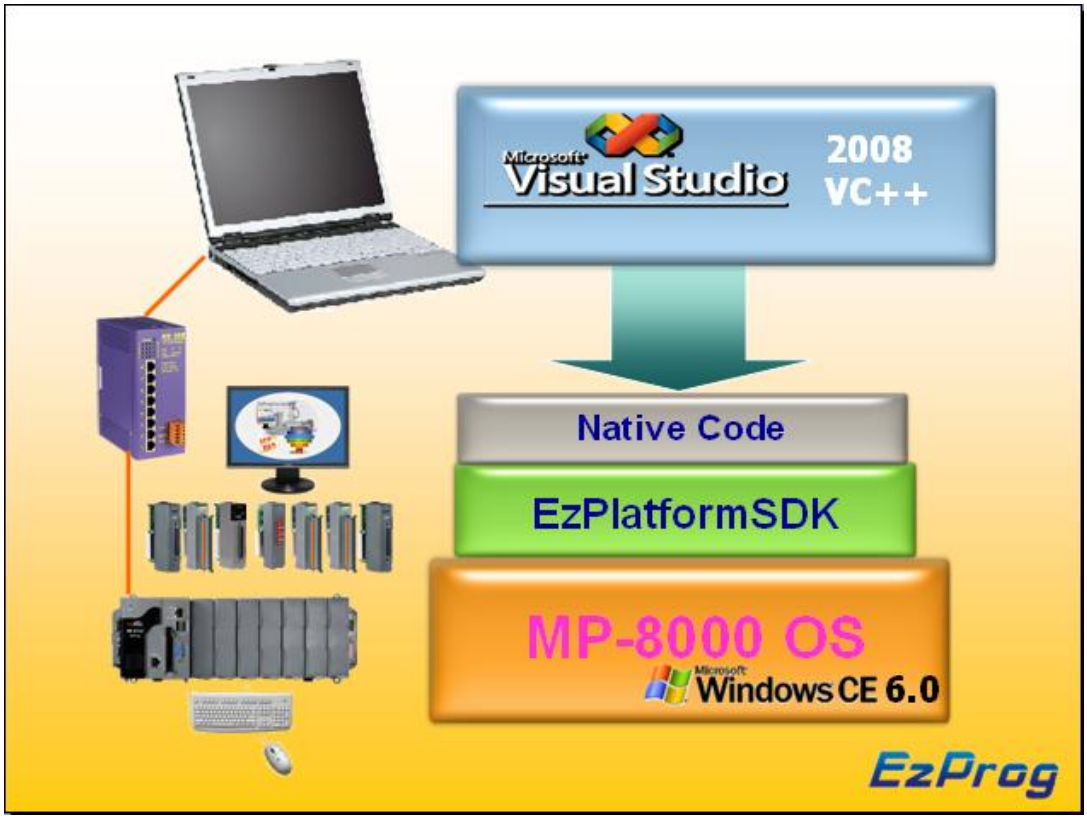

Microsoft Visual Studio 2008 VC++
Native Code
EzPlatformSDK
MP-8000 OS
Microsoft Windows CE 6.0
EzProg

Fig. 1-1

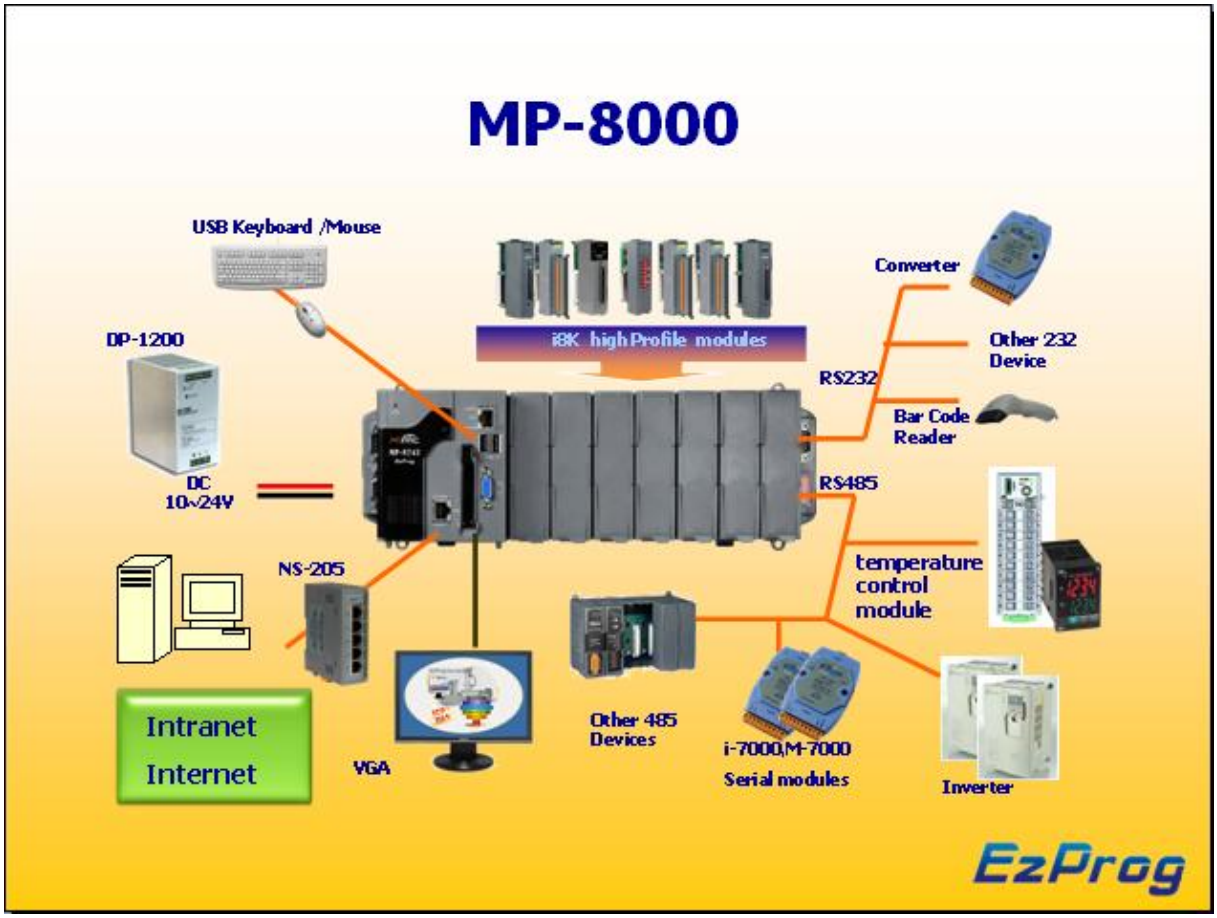

MP-8000
USB Keyboard /Mouse
i8K high Profile modules
Converter
Other 232 Device
RS232
Bar Code Reader
RS485
DP-1200
DC 10~24V
NS-205
temperature control module
Intranet
Internet
VGA
Other 485 Devices
i-7000,M-7000 Serial modules
Inverter
EzProg

Fig. 1-2

1.2 What's new Release

2010/01

New Release V 1.0

2. EzPlatformSDK API

In this section we will focus on the description and application example of EzPlatformSDK Library functions. The functions of EzPlatformSDK.LIB can be clarified into 10 groups which are listed below:

- **2.1 System Information Functions**
- **2.2 Software Information Functions**
- **2.3 Digital Input/Output Functions**
- **2.4 Watch Dog Timer Functions**
- **2.5 Analog Input Functions**
- **2.6 Analog Output Functions**
- **2.7 FRnet Communication Functions**
- **2.8 XSRAM Access Functions**
- **2.9 EEPROM Read/Write Functions**
- **2.10 Motion Control modules**

All the functions supplied for use with MP-8x43 which have been listed, come with more detailed information for each function and this is given in the following section. In the Syntax format, the function indicates the Visual C++ syntax, in order to make the description more simplified and clear, the attributes for the input and output parameters of a function are depicted as [input] and [output] respectively, as is shown in the following table.

Keyword	Users set parameters before calling this function?	Get the data from this parameter after calling this function?
[input]	Yes	No
[output]	No	Yes

When using the Visual C++ development tool to design an application, you must include the EzPlatformSDK.h file in the source program, and link it to EzPlatformSDK.lib when building user applications.

Applied on	WinCE Versions	Defined in	Include	Link to
MP-8x43	6.0 and later	EzPlatformSDK.h	EzPlatformSDK.h	EzPlatformSDK.lib

2.1 Software Information Functions

2.1.1 GetSDKversion

Description:

This function retrieves the version number of the linked EzPlatformSDK library files.

Syntax:

[C++]

```
void GetSDKversion(LPTSTR lpSDKversion)
```

Parameter:

lpSDKversion : [Output] the pointer to a string to receive the version number of EzPlatformSDK.DLL, The buffer size had better great then 50.

Return Value:

None

Example:

```
TCHAR sdkVersion[100];
GetSDKversion(sdkVersion);
```

Remark:

2.1.2 GetOSversion

Description:

This function retrieves the version number of the embedded OS.

Syntax:

[C++]

```
void GetOSversion(LPTSTR lpOSversion)
```

Parameter:

lpOSversion : [Output] the pointer to a buffer to receive the OS version, The buffer size had better great then 50.

Return Value:

None

Example:

```
TCHAR osVersion[100];
GetOSversion(osVersion);
```

Remark:

2.2 System Information Functions

2.2.1 GetModuleType

Description:

This function is used to retrieve the type of i8000 series I/O module plugged into a specific I/O slot in the PAC system. This function performs a supporting task in the collection of information relating to system hardware configurations.

Syntax:

[C++]

```
int GetModuleType(int 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 following table.

Type	HEX Value
_PARALLEL	0x80
_SCAN	0x40
_32BIT	0x20
_DI32	0xE3
_DO32	0xE0
_DI16DO16	0xE2
_DI16	0xC3
_DO16	0xC0
_DI8DO8	0xC2

Example:

```
int slot=1,moduleType;
moduleType=GetModuleType(slot);
//The i-8057x card is plugged in slot 1 of PAC and has a return Value of:0xC0
```

Remark:

2.2.2 GetNameOfModule

Description:

This function is used to retrieve the name of an i8000 series I/O module, which is plugged into a specific I/O slot in the PAC system. This function supports the collection of system hardware configurations.

Syntax:

[C++]

```
int GetNameOfModule(int slot, char *string1)
```

Parameter:

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

string1: [Output] the pointer to a buffer to receive the name of the I/O module.

Return Value:

I/O module ID. For Example, the I-8024x will return 24.

Example:

```
int slot=1,moduleID;
char moduleName[5];
moduleID=GetNameOfModule(slot, moduleName) ;
//The I-8057x card plugged in slot 1 of MP-8x43
//Returned Value: moduleID=57     moduleName=”8057”
```

Remark:

2.2.3 GetSystemSerialNumber

Description:

This function retrieves the hardware serial identification number on the PAC main controller. This function supports the control of hardware versions by reading the 64-bit serial ID chip.

Syntax:

[C++]

```
int GetSystemSerialNumber(char *SN)
```

Parameter:

SN: **[Output] the pointer to a buffer to receive the serial ID number.**

Return Value:

0: indicates success.

1: indicates failure.

Example:

```
char serialNo[8];
GetSystemSerialNumber(serialNo);
//Returned value: serialNo=0x9 EF EF EB BA EA BE AF
```

Remark:

2.2.4 GetTimeTicks

Description:

This function is used to retrieve the number of milliseconds that have elapsed since Windows CE started. The elapsed time is stored as a DWORD value. Therefore, the time will wrap around to zero if the system is run continuously for 49.7 days.

Syntax:

[C++]

```
DWORD GetTimeTicks(void)
```

Parameter:

None

Return Value:

The number of milliseconds that have elapsed since the system was started and it indicates success.

Example:

```
DWORD time;
time=GetTimeTicks();
//Returned Value: time=94367
```

Remark:

2.2.5 Get_Slot_No

Description:

This function retrieves the IO slot number on the PAC backplane (hardware IO Slot number).

Syntax:

[C++]

```
unsigned char Get_Slot_No (void)
```

Parameter:

None

Return Value:

The slot numbers of backplane.

Example:

```
unsigned char slot_no;
slot_no = Get_Slot_No();
```

Remark:

2.2.6 Get_RotarySW

Description:

This function retrieves the user selected rotary switch number on front of PAC.

Syntax:

[C++]

```
unsigned char Get_RotarySW (void)
```

Parameter:

None

Return Value:

The rotary switch selected number.

Example:

```
unsigned char select_no;
select_no = Get_RotarySW ();
```

Remark:

2.2.7 ChangeSlotTo87K

Description:

This function is used to dedicate serial control to the specified slot for the control of 87K series. The serial bus in the PAC backplane is for mapping through to COM1. For example, if you want to send or receive data from a specified slot, you need to call this function first. Then you can use the other series functions.

Syntax:

[C++]

```
void ChangeSlotTo87K(unsigned char slotNo)
```

Parameter:

slotNo : **[Input] Specify the slot number.**

Return Value:

None

Example:

```
unsigned char slot=1;
ChangeSlotTo87K(slot);
```

Remark:

2.2.8 Get_DIP_Switch

Description:

This function retrieves the user selected DIP switch selected on side of PAC.

Syntax:

[C++]

```
int Get_DIP_Switch(void)
```

Parameter:

None

Return Value:

The DIP switch selected number.

Example:

```
int select_no;
select_no = Get_DIP_Switch ();
```

Remark:

2.3 Digital Input/Output Functions

API support Modules table:

Modules	Module Type	DO API	DI API
i8060x, i8064x, i8065x, i8066x, i8068x, i8069x	8 DO module	DO_8 DO_8_BW DO_8_RB	~
i8037x, i8056x, i8057x	16 DO module	DO_16 DO_16_BW DO_16_RB	~
i8041x	32 DO module	DO_32 DO_32_BW DO_32_RB	~
i8052x, i8058x	8 DI module	~	**DI_8** **DI_8_BW**
i8051x, i8053x	16 DI module	~	**DI_16** **DI_16_BW**
i8040x	32 DI module	~	**DI_32** **DI_32_BW**
i8054x, i8055x, i8063x	8 DI /8 DO module	**DIO_DO_8** **DIO_DO_8_BW** **DIO_DO_8_RB**	**DI_8** **DI_8_BW**
i8042x	16 DI /16 DO module	**DIO_DO_16** **DIO_DO_16_BW** **DIO_DO_16_RB**	**DI_16** **DI_16_BW**

2.3.1 8 channels DO modules

2.3.1.1 DO_8

Description:

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

Syntax:

[C++]

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

Parameter:

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

cdata : **[Input] output data.**

Return Value:

None

Example:

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

Remark:

This function can be applied on modules: i8060x, i8064x, i8065x, i8066x, i8068x, and i8069x.

2.3.1.2 DO_8_BW

Description:

Set the digital output value of the channel No. in the 8-channel Digital Output series modules. The output Value is “true” or “false”.

Syntax:

[C++]

```
void DO_8_BW(int slot, int channel, BOOL data)
```

Parameter:

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

channel : [Input] the digital output channel No.(0~7).

data : [Input] output data “true” or “false”.

Return Value:

None

Example:

```
int slot=1;
int channel=3;
BOOL data=true;
DO_8_BW(slot, channel, data);
//The I-8060x card is plugged in slot 1 of PAC turns on channel 3.
```

Remark:

This function can be applied on modules: i8060x, i8064x, i8065x, i8066x, i8068x, and i8069x.

2.3.1.3 DO_8_RB

Description:

Read back the 8-channel digital output value for the I-8000 series modules.

Syntax:

[C++]

```
unsigned char DO_8_RB(int slot)
```

Parameter:

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

Return Value:

8-bit digital output data read back value

Example:

```
int slot=1;
unsigned char outData=0x03;
DO_8(slot, outData);
unsigned char data;
data=DO_8_RB(slot);
//The data read back has a digital output value=0x03.
```

Remark:

This function can be applied on modules: i8060x, i8064x, i8065x, i8066x, i8068x, and i8069x.

2.3.2 16 channels DO modules

2.3.2.1 DO_16

Description:

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

Syntax:

[C++]

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

Parameter:

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

cdata : [Input] output data.

Return Value:

None

Example:

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

Remark:

This function can be applied on modules: i8037x, i8056x, i8057x.

2.3.2.2 DO_16_BW

Description:

Set the digital output value of the channel No. of the 16-channel Digital Output series modules. The output Value is “true” or “false”.

Syntax:

[C++]

```
void DO_16_BW(int slot, int channel, BOOL data)
```

Parameter:

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

channel : [Input] the digital output channel No.(0~15)

data : [Input] output data “true” or “false”.

Return Value:

None

Example:

```
int slot=1;
int channel=3;
BOOL data=true;
DO_16_BW(slot, channel, data);
//The I-8057x card is plugged in slot 1 of PAC turns on channel 3
```

Remark:

This function can be applied on modules: i8037x, i8056x, i8057x.

2.3.2.3 DO_16_RB

Description:

To read back the 16-channel digital output value on the i8000 series modules.

Syntax:

[C++]

```
unsigned int DO_16_RB(int slot)
```

Parameter:

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

Return Value:

16-bit digital output data read back value

Example:

```
int slot=1;
unsigned int outData=0x03;
DO_16(slot, outData);
unsigned int data;
data=DO_16_RB(slot);
//The data read back has a digital output value=0x03
```

Remark:

This function can be applied on modules: i8037x, i8056x, i8057x.

2.3.3 32 channels DO modules

2.3.3.1 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 long cdata)
```

Parameter:

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

cdata : [Input] output data.

Return Value:

None

Example:

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

Remark:

This function can be applied on module: i8041x.

2.3.3.2 DO_32_BW

Description:

Set the digital output value of the channel No. on the 32-channel Digital Output series modules. The output Value is “true” or “false”.

Syntax:

[C++]

```
void DO_32_BW(int slot, int channel, BOOL data)
```

Parameter:

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

channel: **[Input] the digital output channel No.(0~31)**

data : **[Input] output data “true” or “false”.**

Return Value:

None

Example:

```
int slot=1;
int channel=3;
BOOL data=true;
DO_32_BW(slot, channel, data);
//The I-8041x card is plugged in slot 1 of PAC and can turn on channel 3.
```

Remark:

This function can be applied on module: i8041x.

2.3.3.3 DO_32_RB

Description:

To read back the 32-channel digital output value of i8000 series modules.

Syntax:

[C++]

```
unsigned long DO_32_RB(int slot)
```

Parameter:

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

Return Value:

32-bit digital output data read back value

Example:

```
int slot=1;
unsigned long outData=0x03;
DO_32(slot, outData);
unsigned long data;
data=DO_32_RB(slot);
//The data read back has a digital output value=0x03
```

Remark:

This function can be applied on modules: i8041x.

2.3.4 8 channels DI modules

2.3.4.1 DI_8

Description:

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

Syntax:

[C++]

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

Parameter:

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

Return Value:

Input data

Example:

```
int slot=1;
unsigned char data;
data=DI_8(slot);
//The I-8058x card is plugged in slot 1 of PAC and has inputs in channel 0 and 1.
//Returned value: data=0xfC
```

Remark:

This function can be applied on modules: i8052x, i8058x, i8054x, i8055x, i8063x.

2.3.4.2 DI_8_BW

Description:

Obtains channel input data from an 8-channel digital input series module. The Input Value is “true” or “false”.

Syntax:

[C++]

```
BOOL DI_8_BW(int slot, int channel)
```

Parameter:

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

channel : **[Input] the digital output channel No.(0~7)**

Return Value:

Input data

Example:

```
int slot=1;
int channel=3;
BOOL data;
data=DI_8_BW(slot, channel);
//The I-8058x card plugged is in slot 1 of PAC and has inputs in channel 3.
//Returned value: data=true
```

Remark:

This function can be applied on modules: i8052x, i8058x, i8054x, i8055x, i8063x.

2.3.5 16 channels DI modules

2.3.5.1 DI_16

Description:

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

Syntax:

[C++]

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

Parameter:

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

Return Value:

Input data

Example:

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

Remark:

This function can be applied on modules: i8051x, i8053x, i8042x.

2.3.5.2 DI_16_BW

Description:

Obtains channel input data from a 16-channel digital input series module. The Input Value is “true” or “false”.

Syntax:

[C++]

```
BOOL DI_16_BW(int slot, int channel)
```

Parameter:

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

channel : **[Input] the digital output channel No.(0~15)**

Return Value:

Input data

Example:

```
int slot=1;
int channel=3;
BOOL data;
data=DI_16_BW(slot, channel);
//The I-8051x card is plugged in slot 1 of PAC and has inputs in channel 3.
//Returned value: data=true
```

Remark:

This function can be applied on modules: i8051x, i8053x, i8042x.

2.3.6 32 channels DI modules

2.3.6.1 DI_32

Description:

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

Syntax:

[C++]

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

Parameter:

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

Return Value:

Input data

Example:

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

Remark:

This function can be applied on module: i8040x.

2.3.6.2 DI_32_BW

Description:

Obtains channel input data from a 32-channel digital input series module. The Input Value is “true” or “false”.

Syntax:

[C++]

```
BOOL DI_32_BW(int slot, int channel)
```

Parameter:

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

channel : **[Input] the digital output channel No.(0~31)**

Return Value:

Input data

Example:

```
int slot=1;
int channel=3;
BOOL data;
data=DI_32_BW(slot, channel);
//The I-8040x card is plugged in slot 1 of PAC and has inputs in channel 3.
//Returned value: data=true
```

Remark:

This function can be applied on modules: i8040x.

2.3.7 8 channels DI and 8 channels DO modules

2.3.7.1 DIO_DO_8

Description:

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

Syntax:

[C++]

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

Parameter:

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

data : **[Input] output data.**

Return Value:

None

Example:

```
int slot=1;
unsigned char data=3;
DIO_DO_8(slot, data);
//The I-8054x card is plugged in slot 1 of PAC and can turn on channels 0 and 1.
```

Remark:

This function can be applied in modules: i8054x, i8055x, and i8063x.

2.3.7.2 DIO_DO_8_BW

Description:

Set the digital output value of the channel No. for the 8-channel Digital I/O series modules. The output Value is “true” or “false”.

Syntax:

[C++]

```
void DIO_DO_8_BW(int slot, int channel, BOOL data)
```

Parameter:

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

channel : [Input] the digital output channel No.(0~7)

data : [Input] output data “true” or “false”.

Return Value:

None

Example:

```
int slot=1;
int channel=3;
BOOL data=true;
DIO_DO_8_BW(slot, channel, data);
//The I-8054x card is plugged in slot 1 of PAC and can turn on channel 3.
```

Remark:

This function can be applied in these modules: i8054x, i8055x, and i8063x.

2.3.7.3 DIO_DO_8_RB

Description:

To read back the 8-channel digital output value from the i8000 digital I/O series modules.

Syntax:

[C++]

```
unsigned char DIO_DO_8_RB(int slot)
```

Parameter:

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

Return Value:

8-bit digital output data read back value

Example:

```
int slot=1;
unsigned char outData=0x03;
DIO_DO_8(slot, outData);
unsigned char data;
data=DIO_DO_8_RB(slot);
//The data read back has a digital output value=0x03.
```

Remark:

This function can be applied on modules: i8054x, i8055x, and i8063x.

2.3.8 16 channels DI and 16 channels DO modules

2.3.8.1 DIO_DO_16

Description:

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

Syntax:

[C++]

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

Parameter:

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

data : [Input] output data.

Return Value:

None

Example:

```
int slot=1;
unsigned int data=3;
DIO_DO_16(slot, data);
//The I-8042x card is plugged in slot 1 of PAC and can turn on the channels 0 and 1.
```

Remark:

This function can be applied on modules: i8042x.

2.3.8.2 DIO_DO_16_BW

Description:

Set the digital output value on the channel No. for the 16-channel Digital I/O series modules. The output Value is “true” or “false”.

Syntax:

[C++]

```
void DIO_DO_16_BW(int slot, int channel, BOOL data)
```

Parameter:

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

channel : [Input] the digital output channel No.(0~15)

data : [Input] output data true” or “false”.

Return Value:

None

Example:

```
int slot=1;
int channel=3;
BOOL data=true;
DIO_DO_16_BW(slot, channel, data);
//The I-8042x card is plugged in slot 1 of PAC and can turn on the channel 3.
```

Remark:

This function can be applied in these modules: i8042x.

2.3.8.3 DIO_DO_16_RB

Description:

To read back the 16-channel digital output value from i8000 digital I/O series modules.

Syntax:

[C++]

```
unsigned int DIO_DO_16_RB(int slot)
```

Parameter:

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

Return Value:

16-bit digital output data read back value

Example:

```
int slot=1;
unsigned int outData=0x03;
DIO_DO_16(slot, outData);
unsigned int data;
data=DIO_DO_16_RB(slot);
//The data read back has a digital output value=0x03.
```

Remark:

This function can be applied on modules: i8042x.

2.4 Watch Dog Timer Functions

2.4.1 EnableWDT

Description:

This function can be used to enable and reset the watch dog timer. You must to call this API in the watch dog timer period, until call the DisableWDT(), or the system will reboot by watch dog timer.

Syntax:

[C++]

Void EnableWDT (DWORD msecond)

Parameter:

msecond : [Input] watch dog timer interval, unit= millisecond, the msecond value must above 50 ms.

Return Value:

None

Example:

Remark:

2.4.2 DisableWDT

Description:

This function is used to disable the watch dog timer.

Syntax:

[C++]

```
void DisableWDT(void)
```

Parameter:

None

Return Value:

None

Example:

Remark:

2.5 Analog Input Functions

2.5.1 Init_8017H

Description:

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

Syntax:

[C++]

```
int Init_8017H(int slot)
```

Parameter:

slot : **[Input] specified slot of the MP-8x43 system (Range: 1 to 7)**

Return Value:

8017HS_Mode:

1: Single-End Mode, You will have 16 AI channels (8017H ‘s API channel Parameter Range can be 0 to 15).

0: Differential Mode, You will have 8 AI channels (8017H ‘s API channel Parameter Range can be 0 to 7).

Example:

```
int slot=1;
Init_8017H(slot);
//The I-8017H card is plugged in slot 1 of MP-8x43 and initializes the module I-8017H.
```

Remark:

This function can be applied on module: i8017H, i8017HS.

2.5.2 Set_8017H_LED

Description:

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

Syntax:

[C++]

```
void Set_8017H_LED(int slot, unsigned int led)
```

Parameter:

slot : **[Input] specified slot of the MP-8x43 system (Range: 1 to 7)**

led : **[Input] range from 0 to 0xffff**

Return Value:

None

Example:

```
int slot=1;
unsigned int led=0x0001;
Set_8017H_LED(slot, led);
//The LED will have a L-LED light on channel 0 on the I-8017H card which is plugged in slot 1 on the MP-8x43.
```

Remark:

This function can be applied on module: i8017H, i8017HS.

2.5.3 Set_8017H_Channel_Gain_Mode

Description:

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

Syntax:

[C++]

```
void Set_8017H_Channel_Gain_Mode(int slot, int ch, int gain, int mode)
```

Parameter:

slot : [Input] Specify the slot in the MP-8x43 system (Range: 1 to 7)

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

gain : [Input] input range:

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

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

Return Value:

None

Example:

```
int slot=1,ch=0,gain=0;
float data;
Set_8017H_Channel_Gain_Mode(slot, ch, gain,0);
data=Get_AD_FValue(gain);
//The I-8017H card is plugged in slot 1 of MP-8x43, and the data value from channel 0 for I-8017H, and the data range is: -10 ~ +10 V.
```

Remark:

This function can be applied on module: i8017H, i8017HS.

2.5.4 Get_AD_FValue

Description:

Obtain the analog input voltage value from the analog input module in the float format according to the configuration of function Set_8017H_Channel_Gain_Mode.

Syntax:

[C++]

```
float Get_AD_FValue(int gain)
```

Parameter:

gain : [Input] input range

0: +/- 10.0V,

1: +/- 5.0V,

2: +/- 2.5V,

3: +/- 1.25V,

4: +/- 20mA.

Return Value:

Return (float): The analog input value.

Example:

```
int slot=1,ch=0,gain=1;
float data;
Set_8017H_Channel_Gain_Mode(slot, ch, gain,0);
data=Get_AD_FValue(gain);
// The I-8017H card is plugged into slot 1 of MP-8x43 and the data value from channel 0 for I-8017H, and the data range is: -5 ~ +5 V.
```

Remark:

This function can be applied on module: i8017H, i8017HS.

2.5.5 Get_AD_IValue

Description:

This function is used to obtain the analog input current value from an analog input module in the float format according to the configuration in function Set_8017H_Channel_Gain_Mode.

Syntax:

[C++]

```
float Get_AD_IValue(void)
```

Parameter:

None

Return Value:

Return (float): The analog input current value (mA).

Example:

```
int slot=1,ch=0,gain=4;
float data;
Set_8017H_Channel_Gain_Mode(slot, ch, gain,0);
data=Get_AD_IValue(gain);
// The I-8017H card is plugged into slot 1 of MP-8x43, and the data value from channel 0 in I-8017H, and the data range is: 0 ~ 20 mA.
```

Remark:

This function can be applied on module: i8017H, i8017HS.

2.5.6 Get_AD_HValue

Description:

This function is used to obtain the voltage analog input value from the analog input module in the HEX format according to the configuration in function Set_8017H_Channel_Gain_Mode.

Syntax:

[C++]

```
short Get_AD_HValue(void)
```

Parameter:

None

Return Value:

Return (Hex): The voltage analog input value.

Example:

```
int slot=1,ch=0,gain=4;
short data;
Set_8017H_Channel_Gain_Mode(slot, ch, gain,0);
data=Get_AD_HValue(gain);
//The I-8017H card is plugged in slot 1 of MP-8x43, and the data value from channel 0 in I-8017H, and the data range is: 0x0000 ~ 0x3fff.
```

Remark:

This function can be applied on module: i8017H, i8017HS.

2.5.7 I8017H_AD_POLLING

Description:

This function is used to get the analog input values of the specific channel from an analog input module and convert the value in HEX format according to the configuration of the slot, the gain and the data number.

Syntax:

[C++]

```
int I8017H_AD_POLLING(int slot, int ch, int gain, int datacount, int *DataPtr)
```

Parameter:

slot : [Input] Specified slot in the MP-8x43 system (Range: 1 to 7)

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

gain : [Input] Input range:

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

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

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

Return Value:

0: indicates success.

1: indicates failure.

Example:

```
int slot=1, ch=0, gain=0, count=10, data[10];
I8017H_AD_POLLING(slot, ch, gain, datacount, data);
//You gain ten record data values via channel 0 in the i-8017H module.
```

Remark:

This function can be applied on module: i8017H, i8017HS.

2.5.8 ARRAY_HEX_TO_FLOAT_ALL

Description:

This function is used to convert the data from hex to float values based on the configuration of the slot, gain and data length. (Voltage or current)

Syntax:

[C++]

```
void ARRAY_HEX_TO_FLOAT_ALL(int *HexValue, float *Ivalue, int slot,
                            int gain, int len)
```

Parameter:

*HexValue : [Input] data array in integer type before converting.

*Ivalue : [Output] Converted data array in float type (voltage or current).

slot : [Input] Specify the slot in the MP-8x43 system (Range: 1 to 7)

gain : [Input] input range:

len : [Input] ADC data length.

Return Value:

None

Example:

```
int slot=1, ch=0, gain=0, count=10, data[10];
float fdata[10];
I8017H_AD_POLLING(slot, ch, gain, datacount, data);
ARRAY_HEX_TO_FLOAT_ALL(data, fdata, slot, gain, int len);
//You gain ten record HEX values to change ten record float values.
```

Remark:

This function can be applied on module: i8017H, i8017HS.

2.5.9 Read_8017HS_Mode

Description:

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

Syntax:

[C++]

```
int Read_8017HS_Mode(int slot)
```

Parameter:

slot : **[Input] Specify the slot in the MP-8x43 system (Range: 1 to 7)**

Return Value:

1: Single-End Mode, You will have 16 AI channels (8017H ‘s API channel Parameter Range can be 0 to 15).

0: Differential Mode, You will have 8 AI channels (8017H ‘s API channel Parameter Range can be 0 to 7).

Example:

```
int slot=1, mode;
mode = Read_8017HS_Mode(slot);
```

Remark:

This function can be applied on module: i8017HS.

2.6 Analog Output Functions

2.6.1 I8024_Initial

Description:

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

Syntax:

[C++]

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

Parameter:

slot : **[Input] Specify the MP-8x43 system slot (Range: 1 to 7)**

Return Value:

None

Example:

```
int slot=1;
I8024_Initial(slot);
//The I-8024 card is plugged into slot 1 of MP-8x43 and initializes the I-8024 module.
```

Remark:

This function can be applied on module: i8024.

2.6.2 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 MP-8x43 system.

Syntax:

[C++]

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

Parameter:

slot : [Input] Specified the MP-8x43 system slot (Range: 1 to 7)

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

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

Return Value:

None

Example:

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

Remark:

This function can be applied on module: i8024.

2.6.3 I8024_CurrentOut

Description:

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

Syntax:

[C++]

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

Parameter:

slot : **[Input] Specify the MP-8x43 system slot (Range: 1 to 7)**

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

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

Return Value:

None

Example:

```
int slot=1, ch=0;
float data=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: i8024.

2.6.4 I8024_VoltageHexOut

Description:

This function is used to send the voltage value in hex format to the specified channel in the I-8024 module, which is plugged into the slot in the MP-8x43 system.

Syntax:

[C++]

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

Parameter:

slot : **[Input] Specify the MP-8x43 system slot (Range: 1 to 7)**

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

data : **[Input] Output data with hexadecimal**

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

Return Value:

None

Example:

```
int slot=1, ch=0; data=0x3000;
I8024_VoltageHexOut(slot, ch, data);
//The I-8024 module output the 5.0V voltage from the channel 0.
```

Remark:

This function can be applied on module: i8024.

2.6.5 I8024_CurrentHexOut

Description:

This function is used to send the current value in Hex format to the specified channel in the analog output module I-8024, which is plugged into the slot in the MP-8x43 system.

Syntax:

[C++]

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

Parameter:

slot : [Input] Specify the MP-8x43 system slot (Range: 1 to 7)

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

data : [Input] Output data with hexadecimal

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

Return Value:

None

Example:

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

Remark:

This function can be applied on module: i8024.

2.6.6 I8024_VoltageOutReadBack

Description:

This function is used to read back the output data in float format from the specified channel on the I-8024 module in the MP-8x43 system.

Syntax:

[C++]

```
float I8024_VoltageOutReadBack(int slot, int ch)
```

Parameter:

slot : **[Input] Specify the MP-8x43 system slot (Range: 1 to 7)**

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

Return Value:

a float value.

Example:

```
int slot=1, ch=0;
float data;
data=I8024_VoltageOutReadBack(slot, ch);
//Users can read back channel 0 on the I-8024 module outputted voltage value for the last outputted value.
```

Remark:

This function can be applied on module: i8024.

2.6.7 I8024_CurrentOutReadBack

Description:

This function is used to read back the current output value in float format from the specified channel on I-8024 module in the specific slot of the MP-8x43 system.

Syntax:

[C++]

```
float I8024_CurrentOutReadBack(int slot, int ch)
```

Parameter:

slot : **[Input] Specify the MP-8x43 system slot (Range: 1 to 7)**

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

Return Value:

a float value.

Example:

```
int slot=1, ch=0;
float data;
data= I8024_CurrentOutReadBack(slot, ch);
//You can read back channel 0 on the I-8024 module outputted current value at last time.
```

Remark:

This function can be applied on module: i8024.

2.7.8 I8024_VoltageHexOutReadBack

Description:

This function is used to read back the voltage output value in hex format from the specified channel on the analog output module I-8024 in the specific slot of the MP-8x43 system.

Syntax:

[C++]

```
int I8024_VoltageHexOutReadBack(int slot, int ch)
```

Parameter:

slot : **[Input] Specify the slot of the MP-8x43 system (Range: 1 to 7)**

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

Return Value:

return (hex): a 32 bits integer value.

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

Example:

```
int slot=1, ch=0, data;
data=I8024_VoltageHexOutReadBack(slot, ch);
//You can read back channel 0 on the I-8024 modules outputted HEX voltage //value at last time.
```

Remark:

This function can be applied on module: i8024.

2.7.9 I8024_CurrentHexOutReadBack

Description:

This function is used to read back the current value in Hex format from the specified channel of I-8024 module in the slot of the MP-8x43 system.

Syntax:

[C++]

```
int I8024_CurrentHexOutReadBack(int slot, int ch)
```

Parameter:

slot : **[Input] Specify the slot of the MP-8x43 system (Range: 1 to 7)**

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

Return Value:

return (hex): a 32 bits integer value.

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

Example:

```
int slot=1, ch=0, data;
data=I8024_CurrentOutReadBack(slot, ch);
//Users can read back the channel 0 of the I-8024 module outputted HEX current value at last time.
```

Remark:

This function can be applied on module: i8024.

This function can be applied on module: i8080.

2.7 FRnet Communication Functions

2.7.1 i8172_FRNET_IN

Description:

Read FRnet digital input data.

Syntax:

[C++]

```
WORD i8172_FRNET_IN(int Slot, BYTE Port, BYTE wSA)
```

Parameter:

Slot : [Input] Specify the PAC system slot (Range: 1 to 7)

Port: [Input] Specify the FRnet port of i8172x (0 or 1)

wSA: [Input] Specify the SA (sender address) of remote FRnet module

(Range: 8 to 15)

Return Value:

The 16-bit input data

Example:

```
int slot=1;
BYTE port=0;
BYTE wSA=8;
WORD wInput;
wInput =i8172_FRNET_IN(slot, port, wSA);
```

Remark:

This function can be applied on module: i8172x.

2.7.2 i8172_FRNET_OUT

Description:

Read FRnet digital input data.

Syntax:

[C++]

```
void i8172_FRNET_OUT(int Slot, BYTE Port, BYTE wRA, WORD Data)
```

Parameter:

Slot : **[Input] Specify the PACsystem slot (Range: 1 to 7)**

Port: **[Input] Specify the FRnet port of i8172x (0 or 1)**

wRA: **[Input] Specify the RA (receiver address) of remote FRnet module**

(Range: 0 to 7)

Data: **[Input] The output data for the remote specific FRnet module**

Return Value:

NONE

Example:

```
int slot=1;
BYTE port=0;
BYTE wRA=1;
WORD wOutData = 0x5555;
wInput =i8172_FRNET_OUT(slot, port, wRA, wOutData);
```

Remark:

This function can be applied on module: i8172x.

2.7.3 i8172_FRNET_Status

Description:

Read FRnet digital input status.

Syntax:

[C++]

```
BYTE i8172_FRNET_Status(int Slot, BYTE Port)
```

Parameter:

Slot : [Input] Specify the PAC system slot (Range: 1 to 7)

Port: [Input] Specify the FRnet port of i8172x (0 or 1)

Return Value:

The status of input-group.

Bit0 stand for Group-0

Bit1 stand for Group-1

:

Bit7 stand for Group-7

Example:

```
int slot=1;
BYTE port=0;
BYTE status;
status =i8172_FRNET_Status(slot, port);
```

Remark:

This function can be applied on module: i8172x.

2.7.4 i8172_FRNET_Reset

Description:

Reset FRnet digital output status.

Syntax:

[C++]

```
void i8172_FRNET_Reset(int Slot, BYTE Port)
```

Parameter:

Slot : **[Input] Specify the PAC system slot (Range: 1 to 7)**

Port: **[Input] Specify the FRnet port of i8172x (0 or 1)**

Return Value:

NONE

Example:

```
int slot=1;
BYTE port=0;
i8172_FRNET_Reset(slot, port);
```

Remark:

This function can be applied on module: i8172x.

2.8 XSRAM Access Functions

XSRAM function for EzPlatformSDK support the Battery Backup SRAM (BBSRAM) on the PAC backplane. The BBSRAM have the data block and offset address. The BBSRAM have 4096 blocks (0~4095), and every block has 128 offset addresses. The BBSRAM have 4096*128=512KB SRAM. Before you use the function in this session, you must call the XSRAM_Init() function first.

2.8.1 XSRAM_Init

Description:

This function is used to initial the BBSRAM on the PAC Backplane.

Syntax:

```
[C++]
WORD XSRAM_Init(void);
```

Parameter:

None :

Return Value:

512	**512KB on PAC Backplane.**
0	**None BBSRAM on PAC Backplane.**

Example:

```
WORD Ret;
Ret= XSRAM_Init();
```

Remark:

This function can be applied BBSRAM on PAC Backplane.

2.8.2 XSRAM_Get_Max_Block

Description:

This function is used to get the BBSRAM Max_Block on PAC Backplane.

Syntax:

[C++]

```
WORD XSRAM_Get_Max_Block ();
```

Parameter:

None

Return Value:

4096	**There have 4096 Blocks on BBSRAM.**
0	**There don't have BBSRAM on PAC Backplane.**

Example:

```
unsigned char Data;
WORD Ret;
XSRAM_Init();
Ret= XSRAM_Get_Type();
Ret= XSRAM_Get_Max_Block();
```

Remark:

This function can be applied BBSRAM on PAC Backplane.

2.8.3 XSRAM_Set_Block

Description:

This function is used to set the current Block number of BBSRAM on the PAC Backplane.

Syntax:

[C++]

```
bool XSRAM_Set_Block(WORD Block);
```

Parameter:

Block : **[Input] current Block number of BBSRAM.**
For BBSRAM Block is 0~4095

Return Value:

True **Set ok**

False **Set NG**

Example:

```
bool Ret;
XSRAM_Init();
Ret= XSRAM_Set_Block(0);
```

Remark:

This function can be applied BBSRAM on PAC Backplane.

2.8.4 XSRAM_Write_Byte

Description:

This function is used to write a byte to BBSRAM' offset address on Current Block.

Syntax:

```
[C++]
void XSRAM_Write_Byte(unsigned char address,unsigned char data);
```

Parameter:

address : **[Input] the offset address on Current Block.**

data : **[Input] the byte data will be write.**

Return Value:

None

Example:

```
bool Ret;
XSRAM_Init();
Ret= XSRAM_Set_Block(0);
XSRAM_Write_Byte(0,100);
```

Remark:

This function can be applied BBSRAM on PAC Backplane.

2.8.5 XSRAM_Read_Byte

Description:

This function is used to read a byte from BBSRAM' offset address on Current Block.

Syntax:

[C++]

```
unsigned char XSRAM_Read_Byte(unsigned char address);
```

Parameter:

address : **[Input] the offset address on Current Block.**

data : **[Input] the byte data will be write.**

Return Value:

Return the byte data that was read.

Example:

```
unsigned char Data;
bool Ret;
XSRAM_Init();
Ret= XSRAM_Set_Block(0);
XSRAM_Write_Byte(0,100);
Data =XSRAM_Read_Byte(0);
```

Remark:

This function can be applied BBSRAM on PAC Backplane.

2.8.6 XSRAM_Get_Type

Description:

This function is used to get the BBSRAM type on PAC Backplane.

Syntax:

[C++]

```
WORD XSRAM_Get_Type();
```

Parameter:

None

Return Value:

512: There have BBSRAM on PAC Backplane.

0 : There don't have BBSRAM on PAC Backplane.

Example:

```
unsigned char Data;
WORD Ret;
XSRAM_Init();
Ret= XSRAM_Get_Type();
```

Remark:

This function can be applied BBSRAM on PAC Backplane.

2.8.7 XSRAM_Get_Battery_Status

Description:

This function is used to read status of battery level. If the level of battery is low voltage please change that battery, so the BBSRAM can work normally. Otherwise the BBSRAM will lose the data.

Syntax:

```
[C++]
long XSRAM_Get_Battery_Status (int BatteryNo);
```

Parameter:

BatteryNo : [Input] the number of battery that will be read.
1: battery 1
2: battery 2

Return Value:

1: the battery's level is good.
2: the battery's level is low voltage, please change the battery.

Example:

```
long Battery1, Battery2;
Battery1= XSRAM_Get_Battery_Status(1);
Battery2= XSRAM_Get_Battery_Status(2);
```

Remark:

This function can be applied BBSRAM on PAC Backplane.

2.9 EEPROM Read/Write Functions

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

2.9.1 ReadEEP

Description:

This function is used to Read one byte data from EEPROM.

Syntax:

[C++]

```
unsigned char ReadEEP(int block, int offset);
```

Parameter:

block: [Input] the block number of EEPROM.

offset: [Input] the offset within the block.

Return Value:

Data read from the EEPROM.

Example:

```
int block, offset;
unsigned char data;
data= ReadEEP(block, offset);
//Returned value: data= read an 8-bit value from the EEPROM (block & offset)
```

Remark:

2.9.2 WriteEEP

Description:

This function is used to write one byte of data to the EEPROM.

Syntax:

[C++]

```
void WriteEEP(int block, int offset, unsigned char ucData)
```

Parameter:

block : [Input] the block number of EEPROM.

offset: [Input] the offset within the block.

ucData: [Input] data to write to EEPROM.

Return Value:

None

Example:

```
int block, offset;
unsigned char data=10;
WriteEEP(block, offset, data);
//Writes a 10 value output to the EEPROM (block & offset) location
```

Remark:

2.10 Motion Control modules

PAC support the Motion Control modules, which was list below:

module	description
i8092F	High Speed 2-Axes Motion Control Module with FRNET
i8094	High Speed 4-Axes Motion Control Module
i8094F	High Speed 4-Axes Motion Control Module with FRNET
i8094A	High Speed 4-Axes Motion Control Module with CPU
i8094H	High Speed 4-Axes Motion Control Module with CPU and FRNET

For more detail information please reference to the relate manuals.

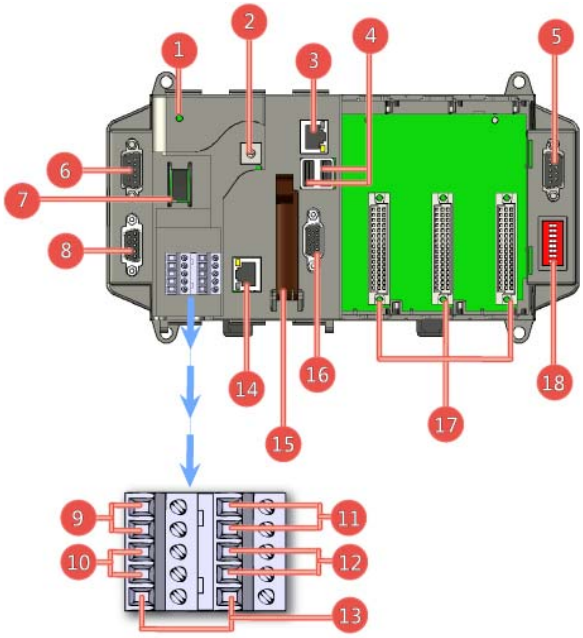
3. Using MP-8x43 Serial Ports

This section describes how to use the three serial ports (RS-232/RS-485 interface) on the PAC embedded controller. The information in this section is organized as follows:

COM1 Port
COM2 Port
COM3 Port
COM4 Port
COM5 Port

3.1 Serial port

The serial ports on the PAC are stander serial port device for WinCE 6.0, so user can use the WinCE stander API to access the serial port directly. The ICPDAS developed a number of useful API for convenient to use the serial ports. The user can easy communicate with the serial port by use these APIs.



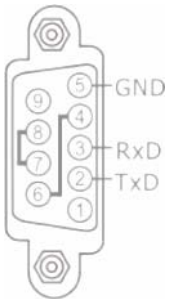
Overview Item Descriptions

1. Power LED Indicator
2. Rotary Switch
3. Ethernet Port 1
4. USB 2.0 Ports
5. COM Port 5 (RS-232)
6. COM Port 4 (RS-232/RS-485)
7. Power Switch
8. COM Port 2 (RS-232)
9. Power and Ground
10. COM Port 3 (RS-485)
11. Redundant Power and round
12. Relay Output R.COM and .NO
13. Frame Ground
14. Ethernet Port 2
15. CF (Compact Flash) Card Slot
16. VGA Port
17. Slot 1 ~ Slot 3
18. DIP Switch

3.1.1 COM1 Port

This COM1 port is located on the PAC backplane. It is a standard RS-232 serial port, and it serial control to the specified slot for the control of 87K series, and the Baud Rate is fixed to 115.2K bps.

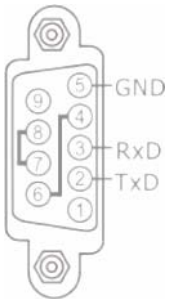
RS-232, Non-isolation (TxD, RxD and GND)
Data Bits: 8
Parity: None, Even, Odd
Stop Bits: 1
Note: CPU built-in UART



3.1.2 COM2 Port

This COM2 port is located on the right-upper corner on the PAC. It is a standard RS-232 serial port, and it provides TXD, RXD, GND, non-isolated and a maximum speed of 115.2K bps.

RS-232, Non-isolation (TxD, RxD and GND)
(used to update firmware)
Baud Rate: 115200, 57600, 38400, 19200, 9600, 4800, 2400, 1200 bps
Data Bits: 7, 8
Parity: None, Even, Odd
Stop Bits: 1
Note: CPU built-in UART



3.1.3 COM3 Port

This COM3 port provides RS-485 serial communication (DATA+ and DATA-) port on the PAC. You can connect to the RS-485 device with modules like the I-7000, M-7000, I-87K and I-8000 serial modules, via this port. That is, you can control the ICP DAS I-7000/I-87K/I-8000 series modules directly from this port with any converter.

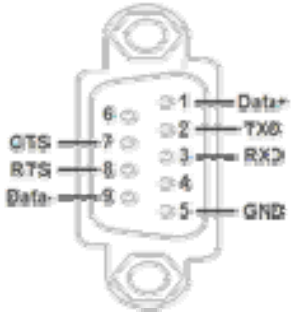
RS-485
Baud Rate: 115200, 57600, 38400, 19200, 9600, 4800, 2400, 1200 bps
Data Bits: 5, 6, 7, 8
Parity: None, Even, Odd, Mark (Always 1), Space (Always 0)
Stop Bits: 1, 2
FIFO: 16 bytes

Note: 16C550 compatible

3.1.4 COM4 Port

This COM4 port provides RS-232/485 serial communication port on the PAC.

RS-232(TxD, RxD and GND) or 485 (DATA+ and DATA-)
Baud Rate: 115200, 57600, 38400, 19200, 9600, 4800, 2400, 1200 bps
Data Bits: 5, 6, 7, 8
Parity: None, Even, Odd,
Mark (Always 1),
Space (Always 0)
Stop Bits: 1, 2
FIFO: 16 bytes
Note: 16C550 compatible



COM4 can be configured as either RS-232 or RS-485, and the configuration depends on the pin connections as follows:
RS-232 (RxD, TxD, CTS, RTS and GND)
RS-485 (Data+ and Data-)

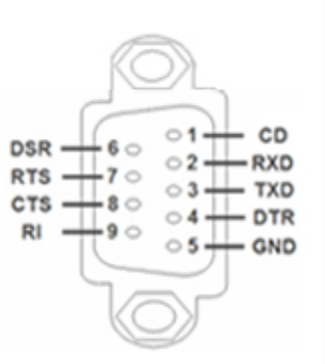
There is no software configuration or hardware jumper needed.

3.1.5 COM5 Port

This COM5 port provides RS-232 serial communication port on the PAC.

RS-232 (TxD, RxD, CD, CTS, RTS, DSR, DTR, GND, RI)
Baud Rate: 115200, 57600, 38400, 19200, 9600, 4800, 2400, 1200 bps
Data Bits: 5, 6, 7, 8
Parity: None, Even, Odd,
Mark (Always 1),
Space (Always 0)
Stop Bits: 1, 2
FIFO: 16 bytes

Note: 16C550 compatible



3.1.6 Library Architecture of the Serial Port

The SerialCE are library files that are designed for applications that run on the PAC main controller unit and can control the remote modules (I-7000/I-8000/I-87K series modules) through a serial port using Windows CE 6.0. The user can apply them to develop their own applications with many development tools such as Microsoft VS 2008 Visual C++. For your convenience, there are many demo programs are provided for VC++. Based on the demo programs provided, users can easily understand how to use the functions and develop their own applications quickly. For more information please reference to the Section 4 SerialCE API of this manual.

SerialCE API:

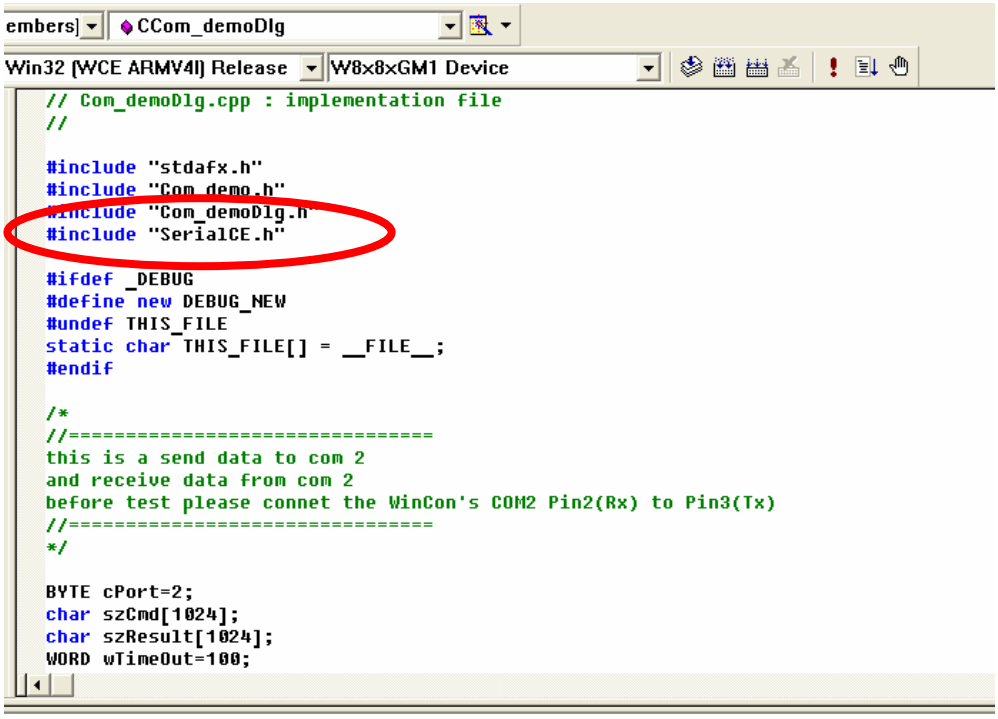
4.1 Serial port basic function
4.2 Send and Receive binary data
4.3 Send and Receive with CR end character
4.4 Other Serial port API

3.2 Serial port applications demo in VC++

The SerialCE is powerful library tools designed specifically for the development of user applications within PAC embedded controllers. The directory settings found in the options for Visual C++.

The user program is use in the following:

First create a new Visual C++ project. And user must include the "SerialCE.h" file into the Visual C++ user source code in header. (Show as following figure).



There for you can use the SerialCE API to program the COM port input and output functions.

4. SerialCE API

4.1 Serial port basic function

4.1.1 Get_Version

Description:

The function is used to obtain the version information.

Syntax:

[C++]

```
long Get_Version(void)
```

Parameter:

None

Return Value:

Version number ==> 0x1000

Example:

```
long Version;
Version= Get_Version ();      //Version= 0x1000
//For example version 1.0.0.0
```

Remark:

4.1.2 Open_Com

Description:

This function is used to configure and open the COM port. It must be called once before sending/receiving command through COM port.

Syntax:

[C++]

```
long Open_Com(unsigned char cPort, DWORD dwBaudrate,char cData,
              char cParity, char cStop)
```

Parameter:

cPort:	[Input] 1=COM1, 2=COM2 …. , MAXPORT
dwBaudRate:	[Input] 1200/1800/2400/4800/7200/9600/ 19200/38400/57600/115200
cData:	[Input] 5/6/7/8 data bit
cParity:	[Input] 0= NonParity, 1= Odd Parity, 2= Even Parity 3= Mark Parity, 4= Space Parity
cStop:	[Input] 0= 1 Stop Bit, 1= 1.5 Stop Bit, 2= 2 Stop Bit

NOTE cData=8, cParity=0, cStop=0 is the default for DCON series modules.

Return Value:

NoError :	OK
Others :	Error code, please refer to “Appendix A Error Code”.

Example:

```
unsigned char   m_ComPort=3;   //Com_port of PAC
DWORD m_baudrate=9600;
char   m_databit=8;
char   m_parity =0;
char   m_stopbit =0;
long RET=Open_Com(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
```

Remark:

4.1.3 Close_Com

Description:

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

Syntax:

[C++]

```
long Close_Com(unsigned char cPort)
```

Parameter:

cPort : **[Input] 1=COM1, 2=COM2 …. , MAXPORT**

Return Value:

NoError : **OK**

Others : **Error code, please refer to “Appendix A Error Code”.**

Example:

```
unsigned char m_ComPort= 3;          //Com_port of PAC
long RET=Close_Com(m_ComPort);
```

Remark:

4.1.4 DataSizeInCom

Description:

Get the how many data existed on the input buffer of COM port.

Syntax:

[C++]

```
long DataSizeInCom (unsigned char cPort)
```

Parameter:

port:	[Input] 1=COM1, 2=COM2 …. , MAXPORT

Return Value:

>=0 :	Data size in the input buffer.
<0 :	Error code, please refer to “Appendix A Error Code”.

Example:

```
int port=3;  //Com_port of PAC
long Size;
Size= DataSizeInCom(port);
```

Remark:

4.1.5 DataSizeOutCom

Description:

Get the how many data existed on the output buffer of COM port.

Syntax:

[C++]

```
long DataSizeOutCom (unsigned char cPort)
```

Parameter:

port:	[Input] 1=COM1, 2=COM2 …. , MAXPORT

Return Value:

>=0 :	Data size in the output buffer.
<0 :	Error code, please refer to “Appendix A Error Code”.

Example:

```
int port=3;  //Com_port of PAC
long Size;
Size= DataSizeOutCom(port);
```

Remark:

4.1.6 ReadComn

Description:

This function is applied to receive the multi-bytes data from COM port input buffer.

Syntax:

[C++]

```
long ReadComn (unsigned char cPort, char szResult[], WORD wmaxLen,
               WORD *wT)
```

Parameter:

Cport::	[Input] 1=COM1, 2=COM2 …. , MAXPORT
szResult:	[Output] Pointer to the buffer that receives the data read
wmaxLen::	[Input] Number of bytes to be read COM port
*wT:	[Output] return a reference number for identify the performance

Return Value:

>=0 :	The number of bytes actually be read.
<0 :	Error code, please refer to “Appendix A Error Code”.

Example:

```
unsigned char   m_ComPort=3;   //Com_port of PAC
DWORD m_baudrate=9600;
char   m_databit=8;   char   m_parity=0;
char   m_stopbit=0;   char   m_szSend[100];
char   m_szReceive[100];
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wmaxLen=10;
WORD m_wT;
long m_Number;
Open_Com(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
m_Number= ReadComn(m_ ComPort, m_szReceiv, m_wmaxLen , &m_wT);
Close_Com(m_ComPort);
```

Remark:

4.2 Send and Receive binary data

4.2.1 Send_Binary

Description:

This function is used to send binary data to the serial port with the fixed length.

Syntax:

[C++]

```
long Send_Binary (unsigned char cPort, char szCmd[], int iLen ,WORD
                  wSTimeOut=0)
```

Parameter:

cPort:	**[Input] 1=COM1, 2=COM2 …. , MAXPORT**
szCmd:	**[Input] pointer of Sending binary data.**
iLen :	**[Input] The length of binary data.**
[wSTimeOut]:	**Optional [Input] for the sending timeout in ms Unit. Before use this, you must enable the flow control first.**

Return Value:

NoError :	**OK**
Others :	**Error code, please refer to "Appendix A Error Code".**

Example:

```
unsigned char   m_ComPort=3;   //Com_port of PAC
DWORD m_baudrate=9600;
char   m_databit=8;
char   m_parity=0;
char   m_stopbit=0;
char   m_szSend[100];
int   m_length=10;
long RET;
RET =Open_Com(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
memset(m_szSend, '\n', sizeof(m_szSend));
sprintf(m_szSend, "%s", "0123456789");                    //Remote other device
RET =Send_Binary(m_ComPort , m_szSend, m_length); //Send 10 character
RET =Close_Com(m_ComPort);
```

Remark:

4.2.2 Receive_Binary

Description:

This function is used to receive binary data from the serial port with the fixed length.

Syntax:

[C++]

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

Input Parameter:

cPort:	**[Input] 1=COM1, 2=COM2 …. , MAXPORT**
szResult:	**[Output] The receiving binary data from the module**
wTimeOut:	**[Input] Communicating timeout setting in ms Unit**
wLen:	**[Input] The length of result binary data.**
***wT:**	**[Output] Total time of receiving interval in ms Unit.**

Return Value:

NoError :	**OK**
Others :	**Error code, please refer to “Appendix A Error Code”.**

Example:

```
unsigned char   m_ComPort=3;   //Com_port of PAC
DWORD m_baudrate=9600;
char   m_databit=8;
char   m_parity=0;
char   m_stopbit=0;
char   m_szSend[100];
char   m_szReceive[100];
int   m_length=10;
WORD m_TimeOut=100;
WORD m_wlength=10;
WORD m_wT;
long RET;
RET =Open_Com(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
```

```
memset(m_szSend, '\n', sizeof(m_szSend));
sprintf(m_szSend, "%s", "0123456789");                    //Remote other device
Send_Binary(m_ComPort , m_szSend, m_length);  //Send 10 character
RET =Receive_Binary(m_ComPort , m_szReceive, m_TimeOut, m_wlength, &m_wT);
RET =Close_Com(m_ComPort);                                //Receive 10 character
```

Remark:

4.2.3 Send_Receive_Binary

Description:

This function is used to Send binary data and receive binary data with the fixed length.

Syntax:

[C++]

```
long Send_Receive_Binary(unsigned char cPort, char ByteCmd[],WORD in_Len,char ByteResult[], WORD out_Len, WORD wTimeOut, WORD wSTimeOut=0)
```

Parameter:

cPort :	[Input] 1=COM1, 2=COM2 …. , MAXPORT
ByteCmd[]:	Pointer to binary data to be send.
in_Len:	Number of binary data bytes to send.
ByteResult[]:	Buffer into which received data will be stored
out_Len:	Number of bytes binary data to be received.
wTimeOut:	Timeout for receiving result string in ms Unit
[wSTimeOut]:	Optional [Input] for the sending timeout in ms Unit. Before use this, you must enable the flow control first.

Return Value:

0:	NO_ERROR
Others:	Error code, please refer to “Appendix A Error Code”.

Example:

```
unsigned char   m_ComPort=3;   //Com_port of PAC
DWORD m_baudrate=9600;
char   m_databit=8;
char   m_parity=0;
char   m_stopbit=0;
char   m_ByteCmd[100],ByteResult[100];
int   in_Len=10,out_Len=10;
Word   wTimeout=10;
long RET;
RET =Open_Com(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
```

```
memset(m_ByteCmd, '\n', sizeof(m_ByteCmd));
sprintf(m_ByteCmd, "%s", "0123456789");                    //Remote other device
RET =Send_Receive_Binary(m_ComPort, ByteCmd,in_Len,ByteResult, out_Len,wTimeOut)       //Send
10 character
RET =Close_Com(m_ComPort);
```

Remark:

4.3 Send and Receive with CR end character

4.3.1 Send_Cmd

Description:

This function only sends a command string to DCON series modules. If the wCheckSum=1, it automatically adds the two checksum bytes to the command string. And then the end of sending string is further added [0x0D] to mean the termination of the command (szCmd). Note that the function Send_Cmd is a multi-task and multi-thread DLL. And this command string cannot include space char within the command string. Otherwise, the command string will be stoped by space character. For example:"$01M 02 03" is user's command string. However, the actual command sent out is "$01M".

Syntax:

[C++]

```
long Send_Cmd (unsigned char cPort, char szCmd[], WORD wTimeOut,
               WORD wCheckSum,WORD wSTimeOut=0)
```

Parameter:

cPort:	**[Input] 1=COM1, 2=COM2 …. , MAXPORT**
szCmd:	**[Input] Sending command string(Terminated with "0")**
wTimeOut:	**[Input] Communicating timeout setting, time in ms Unit.**
wCheckSum:	**[Input] 0=DISABLE, 1=ENABLE**
[wSTimeOut]:	**Optional [Input] for the sending timeout in ms Unit. Before use this, you must enable the flow control first.**

Return Value:

NoError :	**OK**
Others :	**Error code, please refer to "Appendix A Error Code".**

Example:

```
unsigned char   m_ComPort=3;   //Com_port of PAC
DWORD m_baudrate=9600;
char   m_databit=8;
```

```
char    m_parity=0;
char    m_stopbit=0;
char    m_szSend[100];
WORD m_TimeOut=100;
WORD m_Checksum=0;
long RET;
RET =Open_Com(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
memset(m_szSend, '\n', sizeof(m_szSend));
sprintf(m_szSend, "%s", "$01M");//Remote module address “1” ,I7016D
RET=Send_Cmd(m_ComPort, m_szSend, m_TimeOut, m_Checksum);
RET=Close_Com(m_ComPort);
```

Remark:

4.3.2 Receive_Cmd

Description:

User can utilize this function to obtain the response string from the modules in RS-485 Network. And this function provides a response string without the last byte [0x0D].

Syntax:

[C++]

```
long Receive_Cmd (unsigned char cPort, char szResult[], WORD wTimeOut,
                  WORD wChksum, WORD *wT)
```

Parameter:

Cport::	**[Input] 1=COM1, 2=COM2 …. , MAXPORT**
szResult:	**[Output] Receiving the response string from the modules**
wTimeOut::	**[Input] Communicating timeout setting, time in ms Unit**
wCheckSum:	**[Input] 0=DISABLE, 1=ENABLE**
***wT:**	**[Output] Total time of receiving interval in ms Unit.**

Return Value:

NoError :	**OK**
Others :	**Error code, please refer to “Appendix A Error Code”.**

Example:

```
unsigned char   m_ComPort=3;   //Com_port of PAC
DWORD m_baudrate=9600;
char   m_databit=8;
char   m_parity=0;
char   m_stopbit=0;
char   m_szSend[100];
char   m_szReceive[100];
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wT;
long RET;
RET=Open_Com(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
memset(m_szSend, '\n', sizeof(m_szSend));
```

```
sprintf(m_szSend, "%s", "$01M");//Remote module address “1” ,I7016D
RET=Send_Cmd(m_ComPort, m_szSend, m_Checksum);
RET=Receive_Cmd(m_ComPort, m_szReceive, m_TimeOut, m_Checksum, &m_wT);
RET=Close_Com(m_ComPort);            //m_szReceive <- “!017016D”
```

Remark:

4.3.3 Send_Receive_Cmd

Description:

This function sends a command string to RS485 Network and receives the response from RS485 Network. If the wCheckSum =1, this function automatically adds the two checksum bytes into the command string and also check the checksum status when receiving response from the modules. Note that the end of sending string is added [0x0D] to mean the termination of every command. This Send_Receive_Cmd is not a multi-task DLL.

Syntax:

[C++]

```
long Send_Receive_Cmd (unsigned char cPort, char szCmd[], char
                       szResult[], WORD wTimeOut, WORD wCheckSum, WORD *wT,
                       WORD wSTimeOut=0)
```

Parameter:

cPort:	**[Input] 1=COM1, 2=COM2 …. , MAXPORT**
szCmd:	**[Input] 1023(no check sum, 1021 with check sum) Bytes maximum, without zero (0x0D) terminal character.**
szResult:	**[Input] 1023(no check sum, 1021 with check sum) Bytes maximum, with one zero or 0x0D terminal character.**
wTimeOut:	**[Input] Communicating timeout setting, time in ms Unit**
wCheckSum:	**[Input] 0 : add one 0x0D byte to the end of the szCmd >0 : add two check sum bytes and one 0x0D byte to the end of the szCmd**
***wT:**	**[Output] return a reference number for identify the performance. *wT --> 0 :good**

[wSTimeOut]: Optional [Input] for the sending timeout in ms Unit.
Before use this, you must enable the flow control first.

Return Value:

NoError : **OK**

Others : **Error code, please refer to “Appendix A Error Code”.**

Example:

```
unsigned char   m_ComPort=3;   //Com_port of PAC
DWORD m_baudrate=9600;
char   m_databit=8;
char   m_parity=0;
char   m_stopbit=0;
char   m_szSend[100];
char   m_szReceive[100];
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wT;
long RET;
RET=Open_Com(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
memset(m_szSend, '\n', sizeof(m_szSend));
sprintf(m_szSend, "%s", "$01M");//Remote module address “1” ,I7016D
RET=Send_Receive_Cmd(m_ComPort, m_szSend, m_szReceive, m_TimeOut, m_Checksum,
&m_wT);
RET=Close_Com(m_ComPort);                    //m_szReceive <- “!017016D”
```

Remark:

4.4 Other Serial port API

4.4.1 Get_Com_Status

Description:

The function can obtain COM Port status.

Syntax:

[C++]

```
long Get_Com_Status (unsigned char cPort)
```

Parameter:

cPort:	[Input] 1=COM1, 2=COM2 …. , MAXPORT

Return Value:

0 :	COM port is not in used.
1 :	COM port is in used.
<0:	Error code, please refer to “Appendix A Error Code”.

Example:

```
unsigned char   m_ComPort=3;   //Com_port of PAC
DWORD m_baudrate=9600;
char   m_databit=8;
char   m_parity=0;
char   m_stopbit=0;
WORD t1,t2;
long RET ;
RET= Get_Com_Status(m_ComPort);          //RET= 0
RET=Open_Com(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
RET= Get_Com_Status(m_ComPort);          //RET= 1
RET=Close_Com(m_ComPort);
```

Remark:

4.4.2 Change_BaudRate

Description:

This function only can be applied to change the Baudrate setting of serial communication after COM port was opened.

Syntax:

[C++]

```
long Change_BaudRate (unsigned char cPort, DWORD dwBaudrate)
```

Parameter:

cPort :	**[Input] 1=COM1, 2=COM2 …. , MAXPORT**
dwBaudrate:	**[Input] 1200/1800/2400/4800/7200/9600/ 19200/38400/57600/115200**

Return Value:

NoError :	**OK**
Others :	**Error code, please refer to “Appendix A Error Code”.**

Example:

```
unsigned char   m_ComPort=3;   //Com_port of PAC
DWORD m_baudrate=9600;
long RET=Change_BaudRate(m_ComPort, m_baudrate);
```

Remark:

4.4.3 Change_Config

Description:

This function only can be used to change the configuration of the COM port after COM port was opened.

Syntax:

[C++]

```
long Change_Config (unsigned char cPort, DWORD dwBaudrate,char
cData, char cParity, char cStop)
```

Parameter:

cPort:	**[Input] 1=COM1, 2=COM2 …. , MAXPORT**
dwBaudRate:	**[Input] 150/300/600/1200/1800/2400/4800/ 7200/9600/19200/38400/57600/115200**
cData:	**[Input] 5/6/7/8 data bit**
cParity:	**[Input] 0=NonParity, 1=OddParity, 2=EvenParity**
cStop:	**[Input] 0=1-stop, 1=1.5-stp, 2=2-stop**

Return Value:

NoError :	**OK**
Others :	**Error code, please refer to “Appendix A Error Code”.**

Example:

```
unsigned char   m_ComPort=3;   //Com_port of PAC
DWORD m_baudrate=9600;
char   m_databit=8;
char   m_parity=0;
char   m_stopbit=0;
long RET=Change_Config(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
```

Remark:

4.4.4 SetLineStatus

Description:

Set the Line Status (DTR, RTS).

Syntax:

[C++]

```
long SetLineStatus (int port, int pin, int mode)
```

Parameter:

port:	[Input] 1=COM1, 2=COM2 …. , MAXPORT
pin:	[Input] 1: DTR, 2: RTS, 3: DTR+RTS
mode:	[Input] 0: Disable, 1: Enable, 2: HandShake

Return Value:

NoError :	OK
Others :	Error code, please refer to “Appendix A Error Code”.

Example:

```
int port=3;  //Com_port of PAC
int pin=3;
int mode=1;
long RET =SetLineStatus(port, pin, mode);
```

Remark:

4.4.5 GetLineStatus

Description:

Get the Line Status (CTS, DSR, RI, CD).

Syntax:

[C++]

```
long GetLineStatus (int port, int pin)
```

Parameter:

port:	[Input] 1=COM1, 2=COM2 …. , MAXPORT
pin:	[Input] 0: CTS, 1: DSR, 2: RI, 3: CD

Return Value:

1 :	ON
0 :	OFF
Others :	Error code, please refer to “Appendix A Error Code”.

Example:

```
int port=3;  //Com_port of PAC
int pin=0;
WORD Status;
Status= GetLineStatus(port, pin);
```

Remark:

4.4.6 Set_FlowControl

Description:

This function is used to set DCB settings which are related to the flow control (Software and Hardware).

Syntax:

[C++]

```
long Set_FlowControl (unsigned char cPort, int DCB_Member, int mode)
```

Parameter:

cPort : [Input] 1=COM1, 2=COM2 …. , MAXPORT

DCB_Member:
- 0 CtS (fOutxCtsFlow)
- 1 Rts (fRtsControl)
- 2 Dsr (fOutxDsrFlow)
- 3 Dtr (fDtrControl)
- 4 Tx (fOutx :Tx XON/XOFF flow control)
- 5 Rx (fInx :Rx XON/XOFF flow control)

Mode:

CTS:
- 1: Enable CTS to monitored for output flow control
- 0: Disable CTS flow control

RTS:
- 0: Disable RTS flow control
- 1: Enable RTS flow control
- 2: RTS_CONTROL_HANDSHAKE
- 3: RTS_CONTROL_TOGGLE

DSR:
- 0: Disable DSR flow control
- 1: Enable DSR flow control

DTR:
- 0: Disable DTR flow control
- 1: Enable DTR flow control
- 2: DTR_CONTROL_HANDSHAKE

TX:
- 0: Disable TX XON/XOFF
- 1: Enable TX XON/XOFF

RX:
- 0: Disable RX XON/XOFF
- 1: Enable RX XON/XOFF

Return Value:

0: NO_ERROR

Others: Error code, please refer to “Appendix A Error Code”.

Example:

```
unsigned char   m_ComPort=3;    //Com_port of PAC
DWORD m_baudrate=9600;
char    m_databit=8;
char    m_parity=0;
char    m_stopbit=0;
char    m_szSend[100];
char    m_szReceive[100];
long RET;
RET =Open_Com(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
RET =Set_FlowControl(m_ComPort, 3, 1);
RET =Close_Com(m_ComPort);
```

Remark:

Software flow control:

Use XOFF and XON characters to control the transmission and reception of data.

Hareware flow control:

Use control lines of the serial cable to control whether sending or receiving is enabled.

4.4.7 Get_FlowControl

Description:

This function is used to get DCB settings which are related to the flow control (Software and Hardware).

Syntax:

[C++]

```
long Get_FlowControl (unsigned char cPort, int DCB_Member)
```

Parameter:

cPort :	**[Input] 1=COM1, 2=COM2 …. , MAXPORT**
DCB_Member:	**0 CtS (fOutxCtsFlow)**
	1 Rts (fRtsControl)
	2 Dsr (fOutxDsrFlow)
	3 Dtr (fDtrControl)
	4 Tx (fOutx :Tx XON/XOFF flow control)
	5 Rx (fInx :Rx XON/XOFF flow control)

Return Value:

CTS:	**0: Disable CTS flow control**
	1: Enable CTS to monitored for output flow control
RTS:	**0: Disable RTS flow control**
	1: Enable RTS flow control
	2: RTS_CONTROL_HANDSHAKE
	3: RTS_CONTROL_TOGGLE
DSR:	**0:Disable DSR flow control**
	1: Enable DSR flow control
DTR:	**0:Disable DTR flow control**
	1: Enable DTR flow control
	2:DTR_CONTROL_HANDSHAKE
TX:	**0: Disable TX XON/XOFF**
	1: Enable TX XON/XOFF
RX:	**0: Disable RX XON/XOFF**
	1: Enable RX XON/XOFF
Others:	**Error code, please refer to “Appendix A Error Code”.**

Example:

```
unsigned char   m_ComPort=3;   //Com_port of PAC
DWORD m_baudrate=9600;
char   m_databit=8;
char   m_parity=0;
char   m_stopbit=0;
char   m_szSend[100];
char   m_szReceive[100];
long RET;
RET=Open_Com(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
RET =Get_FlowControl(m_ComPort, 3);
RET=Close_Com(m_ComPort);
```

Remark:

Software flow control:

Use XOFF and XON characters to control the transmission and reception of data.

Hareware flow control:

Use control lines of the serial cable to control whether sending or receiving is enabled.

4.4.8 Clear_Buffer

Description:

This function is used to clear the buffer of input (output) for the serial port.

Syntax:

[C++]

```
long Clear_Buffer (unsigned char cPort, unsigned char cAction)
```

Parameter:

cPort :	[Input] 1=COM1, 2=COM2 …. , MAXPORT
cAction:	[Input]
	0: clear Tx,
	1: clear Rx
	2: clear Tx and Rx

Return Value:

0:	NO_ERROR
Others:	Error code, please refer to “Appendix A Error Code”.

Example:

```
unsigned char m_ComPort= 3;          //Com_port of PAC
long RET=Clear_Buffer(m_ComPort,3);
```

Remark:

APPENDIX A SerialCE API Error Code

NoError	0	Functions work normally.
Open_Com_Error	-101	Open Com port error.
Close_Com_Error	-102	Cloes Com port error.
SetCommState_Error	-103	Set COM port status error.
GetCommState_Error	-104	Read COM port status error.
SetTimeOut_Error	-105	Set COM port TimeOut error.
Win32API_Error	-109	Call wrong function error.
BaudRate_Error	-110	Wrong baud rate error.
Bad_Paramater_Error	-111	Invalidate paramater error.
ComPort_No_Error	-120	Use wrong COM Port error.
ComPort_InUse_Error	-121	COM port is in use error.
ComPort_NotOpen_Error	-122	COM is not open error.
TimeOut_Error	-140	TimeOut error.
CheckSum_Error	-141	CheckSum mechanism error.

PROBLEMS REPORT

Technical support is available at no charge. The best way to report problems is send electronic mail to

service@icpdas.com

When reporting problems, please include the following information:

1. **Is the problem reproducible? If so, how?**
2. **What kind and version of Platform are you using? For example, Windows 2000, Windows for Workgroups, Windows XP, etc.**
3. **What kinds of our products are you using? Please see the product's manual.**
4. **If a dialog box with an error message was displayed, please include the full text of the dialog box, including the text in the title bar.**
5. **If the problem involves other programs or hardware devices, what devices or version of the failing programs do you use?**
6. **Other comments relative to this problem or any suggestions will be welcomed.**

After we had received your comments, we will take about two business days to test the problems that you have reported. And then We will reply it as soon as possible to you. Please check that we had received your comments? And please keep in contact with us.

E-mail: service@icpdas.com
Web site: http://www.icpdas.com.tw