

WinCon-8000 SDK

User Guide

for eMbedded Visual C++

(Version 1.2)

WinconSDK 、 UARTCE 、 I7000CE Library for WinCon-8000
Using I-7000/I-87000/I-8000 Series Modules

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

Copyright 1997-2006 by ICPDAS Inc., LTD. All rights reserved worldwide.

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. INTRODUCTION.....	12
2. WINCONSDK.LIB	14
2.1 System Information Functions	15
ChangeSlotTo87K	15
GetModuleType.....	16
GetNameOfModule.....	17
GetTimeTicks	18
GetSlotAddr	19
GetSystemSerialNumber.....	20
Select_IO_Speed	21
2.2 Software Information Functions	22
GetSDKversion	22
GetOSversion	23
2.3 Digital Input/Output Functions.....	24
DO_8.....	24
DO_16.....	25
DO_32.....	26
DO_8_BW.....	27
DO_16_BW.....	28
DO_32_BW.....	29
DI_8.....	30
DI_16.....	31
DI_32.....	32
DI_8_BW	33
DI_16_BW	34
DI_32_BW	35
DIO_DO_8.....	36

DIO_DO_16.....	37
DIO_DO_8_BW.....	38
DIO_DO_16_BW.....	39
DO_8_RB.....	40
DO_16_RB.....	41
DO_32_RB.....	42
DIO_DO_8_RB.....	43
DIO_DO_16_RB.....	44
2.4 Watch Dog Timer Functions.....	45
EnableWDT.....	45
DisableWDT.....	46
WatchDogSWEven.....	47
ClearWDTSWEven.....	48
2.5 EEPROM Read/Write Functions.....	49
ReadEEP.....	49
WriteEEP.....	50
2.6 Analog Input Functions.....	51
Init_8017H.....	51
Set_8017H_LED.....	52
Set_8017H_Channel_Gain_Mode.....	53
Get_AD_FValue.....	54
Get_AD_IValue.....	55
Get_AD_HValue.....	56
I8017H_AD_POLLING.....	57
ARRAY_HEX_TO_FLOAT_ALL.....	58
Read_8017HS_Mode.....	59
2.7 Analog Output Functions.....	60
I8024_Initial.....	60
I8024_VoltageOut.....	61
I8024_CurrentOut.....	62
I8024_VoltageHexOut.....	63
I8024_CurrentHexOut.....	64
I8024_VoltageOutReadBack.....	65
I8024_CurrentOutReadBack.....	66

i8024_VoltageHexOutReadBack	67
i8024_CurrentHexOutReadBack	68

2.8 3-axis Encoder Functions.....69

i8090_REGISTRATION	69
i8090_INIT_CARD.....	70
i8090_GET_ENCODER	71
i8090_RESET_ENCODER.....	72
i8090_GET_ENCODER32	73
i8090_RESET_ENCODER32.....	74
i8090_GET_INDEX.....	75
i8090_ENCODER32_ISR.....	76

2.9 2-axis Stepper/Servo Functions77

i8091_REGISTRATION	77
i8091_RESET_SYSTEM.....	78
i8091_SET_VAR.....	79
i8091_SET_DEFDIR	80
i8091_SET_MODE.....	81
i8091_SET_SERVO_ON	82
i8091_SET_NC	83
i8091_STOP_X	84
i8091_STOP_Y	85
i8091_STOP_ALL	86
i8091_EMG_STOP	87
i8091_LSP_ORG	88
i8091_HSP_ORG	89
i8091_LSP_PULSE_MOVE.....	90
i8091_HSP_PULSE_MOVE	91
i8091_LSP_MOVE	92
i8091_HSP_MOVE.....	93
i8091_CSP_MOVE.....	94
i8091_SLOW_DOWN	95
i8091_SLOW_STOP.....	96
i8091_INTP_PULSE.....	97
i8091_INTP_LINE.....	98
i8091_INTP_LINE02.....	99
i8091_INTP_CIRCLE02.....	100

i8091_INTP_ARC02.....	101
i8091_INTP_STOP	103
i8091_LIMIT_X.....	104
i8091_LIMIT_Y.....	105
i8091_WAIT_X.....	106
i8091_WAIT_Y.....	107
i8091_IS_X_STOP.....	108
i8091_IS_Y_STOP.....	109
i8091_IS_X_STOP_DELAY	110
i8091_IS_Y_STOP_DELAY	111

2.10 Counter/Frequency Functions..... 112

i8080_InitDriver.....	112
i8080_AutoScan	113
i8080_ReadDIXor	114
i8080_ReadDIXorLp.....	115
i8080_ReadXorRegister	116
i8080_SetXorRegister	117
i8080_EepWriteEnable.....	118
i8080_EepWriteDisable	119
i8080_EepWriteWord.....	120
i8080_EepReadWord.....	121
i8080_ReadChannelMode.....	122
i8080_SetChannelMode	124
i8080_ReadFreq	126
i8080_ReadCntUp.....	127
i8080_ReadCntUpDown	128
i8080_SetLowPassUs.....	129
i8080_ReadLowPassUs.....	130
i8080_ClrCnt.....	131
i8080_ReadFreqConfiguration.....	132
i8080_SetFreqConfiguration.....	134

2.11 FRnet Communication Functions 135

i8172_FRNET_IN.....	135
i8172_FRNET_OUT	136
i8172_FRNET_Status	137
i8172_FRNET_Reset	138

2.11 X SRAM Access Functions.....	139
XSRAM_Init	139
XSRAM_Set_Block(WORD Block);	140
XSRAM_Write_Byte	141
XSRAM_Read_Byte	142
XSRAM_Get_Type.....	143
XSRAM_Get_Max_Block	144
3. USING WINCON-8000 SERIAL PORTS.....	145
3.1 Serial Port	146
3.2 LIB Architecture of the Serial Port.....	149
3.3 WinCon Serial Port Applications in eMbedded VC⁺⁺	150
3.3.1 Example List for the Reference of User Program Design	152
4. UARTCE.LIB	157
Get_Uart_Version.....	157
Open_Com	158
Close_Com	159
Send_Receive_Cmd	160
Send_Receive_Cmd_WithChar.....	162
Send_Cmd	164
Send_Cmd_WithChar	166
Receive_Cmd	168
Receive_Cmd_WithChar.....	170
Send_Binary	172
Receive_Binary	174
Get_Com_Status.....	176
Change_BaudRate	177
Change_Config.....	178
ReadComn.....	179
DataSizeInCom	180
DataSizeOutCom.....	181
SetLineStyle.....	182

GetLineStatus	183
Set_FlowControl	184
Get_FlowControl.....	186
Set_Break	188
ModbusGetCRC16	189
Send_Receive_Binary	190
Send_Receive_UserCmd.....	191
Change_ParityErrorCheck	192

5. I7000CE.LIB..... 194

Get_Dll_Version.....	194
----------------------	-----

5.1 Analog Input Functions 195

5.1.1 I-7000 series modules 195

AnalogIn.....	195
AnalogInHex	197
AnalogInFsr.....	199
AnalogInAll.....	201
ThermocoupleOpen_7011	203
SetLedDisplay	205
GetLedDisplay	207

5.1.2 I-8000 series modules 209

AnalogIn_8K.....	209
AnalogInHex_8K	211
AnalogInFsr_8K.....	213
AnalogInAll_8K.....	215

5.1.3 I-87K series modules 217

AnalogIn_87K.....	217
AnalogInHex_87K	219
AnalogInFsr_87K.....	221
AnalogInAll_87K.....	223

5.2 Module Alarm Functions 225

5.2.1 I-7000 series modules 225

EnableAlarm.....	225
DisableAlarm	227
ClearLatchAlarm.....	229
SetAlarmLimitValue.....	231

ReadAlarmLimitValue	233
ReadOutputAlarmState	235
5.2.2 I-8000 series modules	237
SetAlarmMode_8K	237
SetAlarmConnect_8K	239
ClearLatchAlarm_8K	241
SetAlarmLimitValue_8K	243
ReadAlarmLimitValue_8K	245
ReadAlarmMode_8K	247
ReadAlarmStatus_8K	249
5.3 Strain Gauge Functions.....	251
SetupLinearMapping	251
EnableLinearMapping	253
DisableLinearMapping	255
ReadLinearMappingStatus	257
ReadSourceValueOfLM	259
ReadTargetValueOfLM	261
5.4 Analog Output Functions.....	263
5.4.1 I-7000 series modules	263
AnalogOut	263
AnalogOutReadBack	265
AnalogOutHex	267
AnalogOutFsr	269
AnalogOutReadBackHex	271
AnalogOutReadBackFsr	273
5.4.2 I-8000 series modules	275
AnalogOut_8K	275
AnalogOutReadBack_8K	277
ReadConfigurationStatus_8K	279
SetStartUpValue_8K	281
ReadStartUpValue_8K	283
5.4.3 I-87K series modules	285
AnalogOut_87K	285
AnalogOutReadBack_87K	287
AnalogOutHex_87K	289
AnalogOutFsr_87K	291

AnalogOutReadBackHex_87K	293
AnalogOutReadBackFsr_87K.....	295
ReadConfigurationStatus_87K.....	297
SetStartUpValue_87K	299
ReadStartUpValue_87K	301
5.5 Digital Input Functions	303
5.5.1 I-7000 series modules	303
DigitalIn	303
DigitalInLatch	305
ClearDigitalInLatch.....	307
DigitalInCounterRead	309
ClearDigitalInCounter.....	311
ReadEventCounter	313
ClearEventCounter	315
5.5.2 I-8000 series modules	317
DigitalIn_8K.....	317
DigitalInCounterRead_8K	319
ClearDigitalInCounter_8K.....	321
DigitalInLatch_8K	323
ClearDigitalInLatch_8K.....	325
5.5.3 I-87K series modules	327
DigitalIn_87K.....	327
DigitalInLatch_87K	329
ClearDigitalInLatch_87K.....	331
DigitalInCounterRead_87K	333
ClearDigitalInCounter_87K.....	335
5.6 Digital Output Functions	337
5.6.1 I-7000 series modules	337
DigitalOut.....	337
DigitalBitOut.....	339
DigitalOutReadBack	341
DigitalOut_7016.....	343
5.6.2 I-8000 series modules	345
DigitalOut_8K.....	345
DigitalBitOut_8K.....	347
5.6.3 I-87K series modules	349

DigitalOut_87K.....	349
DigitalOutReadBack_87K	351
DigitalBitOut_87K.....	353

5.7 Counter Functions.....355

CounterIn_7080.....	355
StartCounting_7080	357
ClearCounter_7080	359
ReadCounterMax Value _7080	361
SetCounterMax Value _7080	363
EnableCounterAlarm _7080.....	365
DisableCounterAlarm _7080.....	367
ReadInputSignalMode _7080.....	369
SetInputSignalMode _7080.....	371
PresetCounterValue_7080	373
ReadPresetCounterValue_7080.....	375
SetModuleMode_7080	377
ReadModuleMode_7080.....	379
SetLevelVolt_7080	381
ReadLevelVolt_7080.....	383
SetMinSignalWidth_7080	385
ReadMinSignalWidth_7080.....	387
SetGateMode_7080.....	389
ReadGateMode_7080.....	391
ReadOutputAlarmState_7080	393
ReadAlarmLimitValue_7080	395
SetAlarmLimitValue_7080.....	397
ReadCounterStatus_7080	399
SetConfiguration_7080	401
DataToLED_7080	403
ReadCounter.....	405
ReadCounter_All.....	407
ReadFreq	410
ReadFreq_All	412
ClearCounter	414
SetCounterStatus	416
ReadCounterStatus	418

5.8 Dual Watchdog Functions.....420

HostIsOK.....	420
ToSetupHostWatchdog	422
ToReadHostWatchdog.....	424
ReadModuleResetStatus.....	426
ReadModuleHostWahchdogStatus	428
ResetModuleHostWahchdogStatus	430
SetSafeValueForDo	432
SetPowerOnValueForDo	434
SetSafeValueForAo	436
SetPowerOnValueForAo	438
SetPowerOnSafeValue	440
ReadSafeValueForAo	442
ReadPowerOnValueForAo	444
ReadPowerOnValueForDo	446
ReadSafeValueForDo	448
ReadConfigStatus.....	450

APPENDIX A WATCHDOG.....452

Operation Principle.....452

Host WatchDog	453
Module WatchDog	456
Comparison of Host and Module Watchdog	458

APPENDIX B ERROR CODE459

Error Code459

PROBLEMS REPORT460

1. Introduction

Welcome to the **WinconSDK_LIB** user's manual. ICPDAS provides Library files, namely the **WinconSDK_LIB**, for the I-8000 series modules which are used in the Wincon-8000 Embedded Controller. The **WinconSDK_LIB** has all the essential Library functions designed for the I-8000 series modules for Microsoft WinCE.Net platform. It can be applied on eMbedded Visual C++ on WinCE 4.1.Net, and even on the newer platforms. Users can easily develop WinCE.NET applications on WinCon-8000 by using this toolkit. The various functions in **WinconSDK_LIB** are divided into the following sub-group functions for easy use in different applications. The main functions of the WinCon-8000 embedded controller are depicted in figure 1.2, which include **VGA, USB, mouse, keyboard, compact flash, series, Ethernet and an I/O interface.**

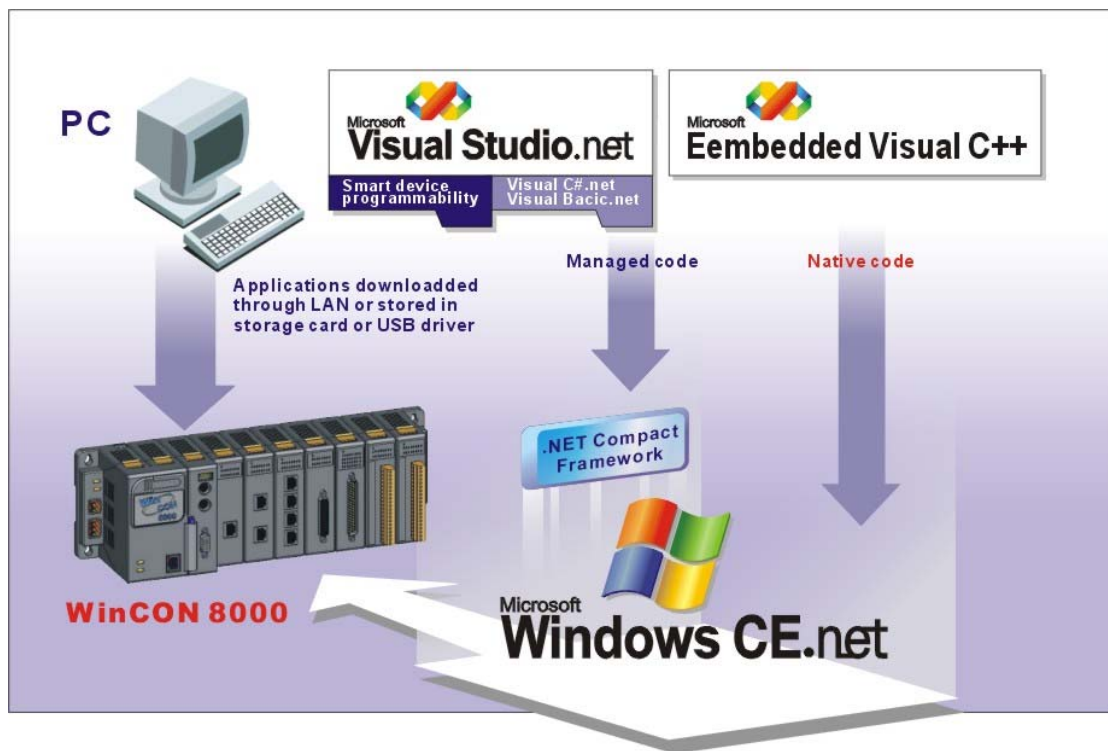


Fig. 1-1

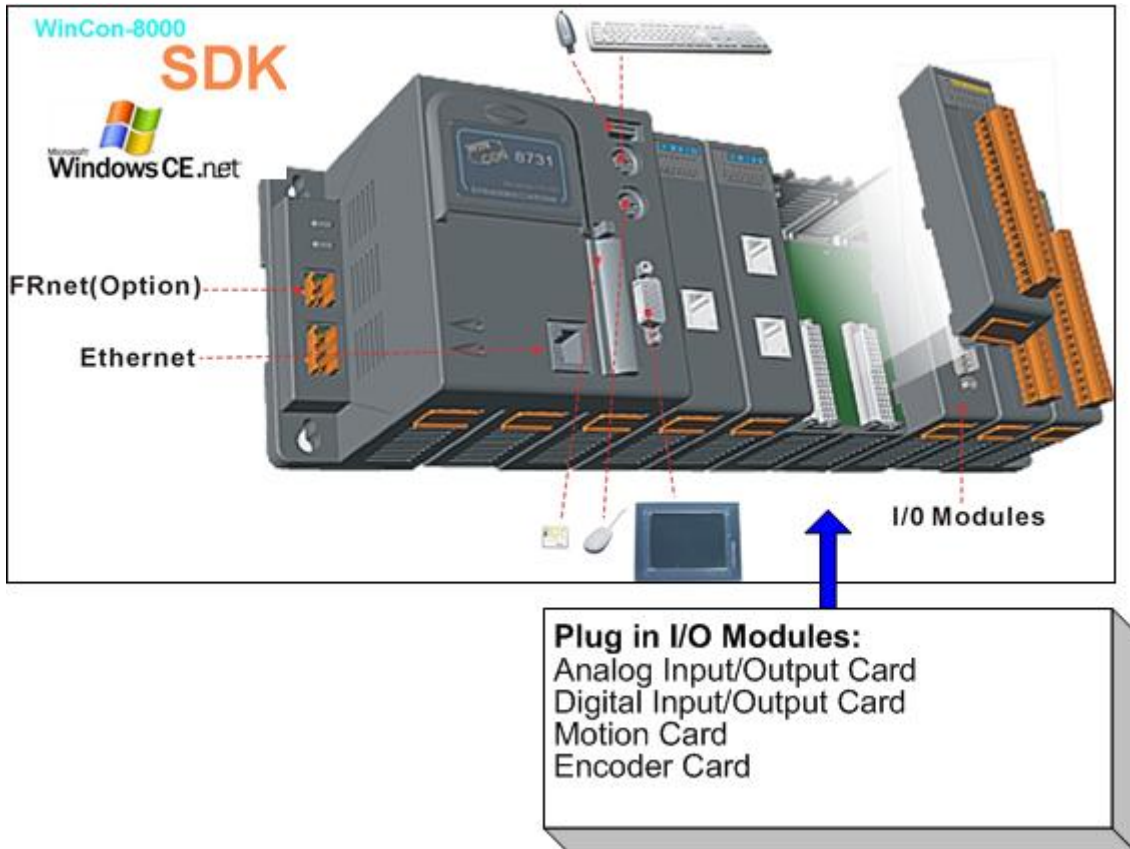


Fig. 1-2

1.1 What's new

2005/4

Added Support modules: I8037, i8069.

Fix :remove i8056 from DI_16 Function.

1.2 What's new

2006/7

Added Support modules: I8172 FRnet module.

Provided the Select_IO_Speed() for Original/Faster IO access

Included the functions to access S256/512 X SRAM

2. WinconSDK.LIB

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

1. System Information Functions;
2. Software Information Functions;
3. Digital Input/Output Functions;
4. Watch Dog Timer Functions;
5. EEPROM Read/Write Functions;
6. Analog Input Functions;
7. Analog Output Functions;
8. 3-axis Encoder Functions;
9. 2-axis Stepper/Servo Functions;
10. Counter/Frequency Functions.

All the functions supplied for use with Wincon-8000 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 eMbedded 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 eMbedded Visual C++ development tool to design an application, you must include the WinconSDK.h file in the source program, and link it to WinconSDK.lib when building user applications.

Applied on	WinCE Versions	Defined in	Include	Link to
WinCon-8000	4.1.0.01 and later	WinconSDK.h	WinconSDK.h	WinconSDK.lib

2.1 System Information Functions

■ 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 Wincon-8000 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);  
//The first slot is specified as COM2 port in Wincon-8000.
```

Remark:

■ GetModuleType

Description:

This function is used to retrieve the type of 8000 series I/O module plugged into a specific I/O slot in the WinCon 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	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-8057 card is plugged in slot 1 of Wincon-8000 and has a return Value of:0xC0
```

Remark:

■ GetNameOfModule

Description:

This function is used to retrieve the name of an 8000 series I/O module, which is plugged into a specific I/O slot in the Wincon-8000 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-8024 will return 24.

Example:

```
int slot=1,moduleID;  
char moduleName[5];  
moduleID=GetNameOfModule(slot, moduleName)  
//The I-8057 card plugged in slot 1 of Wincon-8000  
//Returned Value: moduleID=57  moduleName="8057"
```

Remark:

■ 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:

■ GetSlotAddr

Description:

This function retrieves the base memory address for a specific slot.

Syntax:

```
[C++]  
DWORD GetSlotAddr(int slot)
```

Parameter:

slot : [Input] Specify the slot number.

Return Value:

Slot base address. The WinCon-8000 system currently provides 7 slots (from 1 to 7). If the I/O Speed is not changed with the Select_IO_Speed(), the corresponding base addresses are:

Slot 1 -> 0xab00100

Slot 2 -> 0xab00200

Slot 3 -> 0xab00300

Slot 4 -> 0xab01800

Slot 5 -> 0xab01900

Slot 6 -> 0xab01a00

Slot 7 -> 0xab01b00

Otherwise, the GetSlotAddr() will return the mapped address.

Example:

```
DWORD slotAddr;  
slotAddr=GetSlotAddr( 1 );  
//Returned Value: slotAddr= 0xab00100
```

Remark:

■ GetSystemSerialNumber

Description:

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

Syntax:

```
[C++]  
int GetSystemSerialNumber(char *string1)
```

Parameter:

string1: [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:

■ Select_IO_Speed

Description:

Select the Access mode for Input/Output operations. It's recommended to select the IO speed before calling other functions.

Syntax:

```
[C++]  
  
bool Select_IO_Speed(unsigned char SpeedMode)
```

Parameter:

SpeedMode: [Input] The I/O Speed Mode.
 0: the Original I/O Speed
 1: the Fast I/O Speed

Return Value:

1: indicates success.
0: indicates failure.

Example:

```
bool bChanged;  
bChanged = unselect_IO_Speed(1);
```

Remark:

The high-speed buffer IC is required for faster I/O access. If your backplane is out-of-date, changing the access mode will fail to access the I/O modules. It's recommended to chose the newer backplane.

WB-831 Rev. 2.6 and above

WB-871 Rev. 2.9 and above.

2.2 Software Information Functions

■ GetSDKversion

Description:

This function retrieves the version number of the linked WinCon SDK library files.

Syntax:

```
[C++]  
void GetSDKversion(LPTSTR lpSDKversion)
```

Parameter:

lpSDKversion : [Output] the pointer to a string to receive the version number of WinConSDK.DLL.

Return Value:

None

Example:

```
TCHAR sdkVersion[20];  
GetSDKversion(sdkVersion);
```

Remark:

■ 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.

Return Value:

None

Example:

```
TCHAR osVersion[20];  
GetOSversion(osVersion);  
  
// Returned value: osVersion="Windows CE .NET 4.1 01-00-00"
```

Remark:

2.3 Digital Input/Output Functions

■ 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-8064 card is plugged in slot 1 of Wincon-8000 and can turn on channel 0 and 1.
```

Remark:

This function can be applied on modules: i8060, i8064, i8065, i8066, i8068, and i8069.

■ 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-8057 card is plugged in slot 1 of Wincon-8000 and can turn on channel 0 and 1.
```

Remark:

This function can be applied on modules: i8037, i8056,i8057.

■ 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-8041 card is plugged in slot 1 of Wincon-8000 and can turn on channel 0 and 1.
```

Remark:

This function can be applied on module: i8041.

■ 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-8060 card is plugged in slot 1 of Wincon-8000 turns on channel 3.
```

Remark:

This function can be applied on modules: i8060, i8064, i8065, i8066, i8068, and i8069.

■ 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-8057 card is plugged in slot 1 of Wincon-8000 turns on channel 3
```

Remark:

This function can be applied on modules: i8037, i8056,i8057.

■ 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-8041 card is plugged in slot 1 of Wincon-8000 and can turn on channel 3.
```

Remark:

This function can be applied on module: i8041.

■ 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++]
<code>unsigned char DI_8(int slot)</code>

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-8058 card is plugged in slot 1 of Wincon-8000 and has inputs in channel 0 and 1.
//Returned value: data=0xfC
```

Remark:

This function can be applied on modules: i8052, i8058.

■ 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-8053 card is plugged in slot 1 of Wincon-8000 and has inputs in //channel 0 and 1.  
//Returned value: data=0xffC
```

Remark:

This function can be applied on modules: i8051, i8053.

■ 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++]
<code>unsigned long DI_32(int slot)</code>

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-8040 card plugged is in slot 1 of Wincon-8000 and has inputs in channels 0 and 1.
//Returned value: data=0xfffffC
```

Remark:

This function can be applied on module: i8040.

■ 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-8058 card plugged is in slot 1 of Wincon-8000 and has inputs in channel 3.  
//Returned value: data=true
```

Remark:

This function can be applied on modules: i8052, i8058.

■ 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++]
<code>BOOL DI_16_BW(int slot, int channel)</code>

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-8051 card is plugged in slot 1 of Wincon-8000 and has inputs in channel 3.
//Returned value: data=true
```

Remark:

This function can be applied on modules: i8051, i8053.

■ 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-8040 card is plugged in slot 1 of Wincon-8000 and has inputs in channel 3.  
//Returned value: data=true
```

Remark:

This function can be applied on modules: i8040.

■ 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-8054 card is plugged in slot 1 of Wincon-8000 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: i8054, i8055, and i8063.

■ 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-8050 card is plugged in slot 1 of Wincon-8000 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: i8042, and i8050.

■ 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-8054 card is plugged in slot 1 of Wincon-8000 and can turn on channel 3.
```

Remark:

This function can be applied in these modules: i8054, i8055, and i8063.

■ 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-8042 card is plugged in slot 1 of Wincon-8000 and can turn on the channel 3.
```

Remark:

This function can be applied in these modules: i8042, and i8050.

■ 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:i8060, i8064, i8065, i8066, i8068, and i8069.

■ DO_16_RB

Description:

To read back the 16-channel digital output value on the I-8000 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: i8037,i8056, i8057.

■ DO_32_RB

Description:

To read back the 32-channel digital output value of I-8000 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: i8041.

■ DIO_DO_8_RB

Description:

To read back the 8-channel digital output value from the I-8000 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: i8054, i8055, and i8063.

■ DIO_DO_16_RB

Description:

To read back the 16-channel digital output value from I-8000 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: i8042, and i8050.

2.4 Watch Dog Timer Functions

■ EnableWDT

Description:

This function can be used to enable the watch dog timer.

Syntax:

```
[C++]  
Void EnableWDT (DWORD msecond)
```

Parameter:

msecond : [Input] watch dog timer interval, unit= millisecond

Return Value:

None

Example:

Remark:

■ DisableWDT

Description:

This function is used to disable the watch dog timer.

Syntax:

[C++]
<code>void DisableWDT(void)</code>

Parameter:

None

Return Value:

None

Example:

Remark:

■ WatchDogSWEven

Description:

This function is used to check whether the system is reset with the watch dog timer. The watch dog timer is started by the EnableWDT function, and stopped by calling the DisableWDT function and refreshed via the EnableWDT function.

Syntax:

[C++]
<code>int WatchDogSWEven(void)</code>

Parameter:

None

Return Value:

None

Example:

Remark:

■ ClearWDTSEven

Description:

This function is used to clear the flag that has been reset with the watch dog timer.

Syntax:

[C++]
<code>void ClearWDTSEven (void)</code>

Parameter:

None

Return Value:

None

Example:

Remark:

2.5 EEPROM Read/Write Functions

■ ReadEEP

Description:

Read one byte data from EEPROM. There is a 16K-byte EEPROM in the main control unit in the WinCon-8000 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.

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:

■ WriteEEP

Description:

To write one byte of data to the EEPROM. There is a 16K-byte EEPROM in the main control unit in the WinCon-8000 system. This EEPROM is divided into 256 blocks (0 to 255), and each block is 64 bytes in length from the offset of 0 to 63. This EEPROM with its accessing APIs, provides another mechanism for storing critical data inside non-volatile memory.

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:

2.6 Analog Input Functions

■ Init_8017H

Description:

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

Syntax:

```
[C++]  
  
void Init_8017H(int slot)
```

Parameter:

slot : [Input] specified slot of the Wincon-8000 system (Range: 1 to 7)

Return Value:

None

Example:

```
int slot=1;  
Init_8017H(slot);  
//The I-8017H card is plugged in slot 1 of Wincon-8000 and initializes the module I-8017H.
```

Remark:

This function can be applied on module: i8017H.

■ Set_8017H_LED

Description:

Turns the I-8017H 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 Wincon-8000 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  
Wincon-8000.
```

Remark:

This function can be applied on module: i8017H.

■ 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 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 Wincon-8000 system (Range: 1 to 7)
ch : [Input] Specify the I-8017H 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 Wincon-8000, 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.

■ 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 Wincon-8000 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.

■ 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 Wincon-8000, 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.

■ 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 Wincon-8000, 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.

■ 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 Wincon-8000 system (Range: 1 to 7)

ch : [Input] Specified channel for I-8017H (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:

■ 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 Wincon-8000 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.

■ 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 Wincon-8000 system (Range: 1 to 7)

Return Value:

- 1: Single-End Mode.
- 1: Differential Mode.

Example:

```
int slot=1, mode;  
mode = Read_8017H_Mode(slot);
```

Remark:

2.7 Analog Output Functions

■ 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 Wincon-8000 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 Wincon-8000 and initializes the I-8024 module.
```

Remark:

This function can be applied on module: i8024.

■ 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 Wincon-8000 system.

Syntax:

```
[C++]  
void I8024_VoltageOut(int slot, int ch, float data)
```

Parameter:

slot : [Input] Specified the Wincon-8000 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.

■ I8024_CurrentOut

Description:

This function is used to initialize the I-8024 module 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 Wincon-8000 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.

■ 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 Wincon-8000 system.

Syntax:

```
[C++]  
  
void I8024_VoltageHexOut(int slot, int ch, int data)
```

Parameter:

slot : [Input] Specify the Wincon-8000 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.

■ 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 Wincon-8000 system.

Syntax:

```
[C++]  
  
void I8024_CurrentHexOut(int slot, int ch, int data)
```

Parameter:

slot : [Input] Specify the Wincon-8000 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.

■ 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 Wincon-8000 system.

Syntax:

```
[C++]  
  
float I8024_VoltageOutReadBack(int slot, int ch)
```

Parameter:

slot : [Input] Specify the Wincon-8000 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.

■ 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 Wincon-8000 system.

Syntax:

```
[C++]  
float I8024_CurrentOutReadBack(int slot, int ch)
```

Parameter:

slot : [Input] Specify the Wincon-8000 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.

■ 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 Wincon-8000 system.

Syntax:

```
[C++]  
  
int I8024_VoltageHexOutReadBack(int slot, int ch)
```

Parameter:

slot : [Input] Specify the slot of the Wincon-8000 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.

■ 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 Wincon-8000 system.

Syntax:

```
[C++]  
  
int I8024_CurrentHexOutReadBack(int slot, int ch)
```

Parameter:

slot : [Input] Specify the slot of the Wincon-8000 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.

2.8 3-axis Encoder Functions

■ i8090_REGISTRATION

Description:

This function is used to assign a card number “cardNo” to the I-8090 module in the specific slot. In order to distinguish more than one I-8090 card in the WinCon-8000 platform, the I-8090 modules should be registered before applying it. If there is no I-8090 in WinCon-8000 with the given address, this function will return 0 which means a failure.

Syntax:

```
[C++]  
  
unsigned char i8090_REGISTRATION(unsigned char cardNo, int slot)
```

Parameter:

cardNO : [Input] 0~19, The assigned number to the i8090 card.
slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)

Return Value:

1: Success registration
0: Failure registration

Example:

```
#define CARD1 1  
  
int slot=1;  
  
i8090_REGISTRATION(CARD1, slot);  
//The I-8090 card plugged in slot 1 of Wincon-8000
```

Remark:

This function can be applied on module: i8090.

■ i8090_INIT_CARD

Description:

This function is applied to reset the counter values of three axes with a specific card number, and set the modes of three counters.

Syntax:

```
[C++]  
  
void i8090_INIT_CARD(unsigned char cardNo, unsigned char x_mode,  
                    unsigned char y_mode, unsigned char z_mode)
```

Parameter:

cardNO : [Input] 0~19, The specified card number.
x_mode : [Input] The X axis counter mode. Refer to the Remarks.
y_mode : [Input] The Y axis counter mode. Refer to the Remarks.
z_mode : [Input] The Z axis counter mode. Refer to the Remarks.

Return Value:

None

Example:

```
#define CARD1 1  
int slot=1;  
i8090_REGISTRATION(CARD1, slot);  
i8090_INIT_CARD(CARD1, ENC_QUADRANT, ENC_QUADRANT, ENC_QUADRANT);  
//The x, y, z axis encoder are ENC_QUADRANT mode.
```

Remark:

1. This function can be applied on module: i8090.
2. For more information about counting modes, refer to the Registers of i-8090 board chapter in the i-8090 user's manual.

■ i8090_GET_ENCODER

Description:

This function is used to obtain the counter value of the selected axis on the specified encoder card. This counter value is defined in the short (16-bit) format.

Syntax:

```
[C++]  
  
unsigned int i8090_GET_ENCODER(unsigned char cardNo, unsigned char axis)
```

Parameter:

cardNO : [Input] 0~19, The selected card number.
axis : [Input] The selected axis. 1: X-axis; 2: Y-axis; 3: Z-axis

Return Value:

A 16 bits unsigned integer value.

Example:

```
#define CARD1 1  
int slot=1;  
unsigned int data;  
i8090_REGISTRATION(CARD1, slot);  
i8090_INIT_CARD(CARD1, ENC_QUADRANT, ENC_QUADRANT,ENC_QUADRANT);  
data= i8090_GET_ENCODER(CARD1, X_axis);  
  
//The data value is the X-axis encoder value.
```

Remark:

This function can be applied on module: i8090.

■ i8090_RESET_ENCODER

Description:

This function is used to reset the counter value of the selected axis on the specified encoder card.

Syntax:

```
[C++]  
  
void i8090_RESET_ENCODER(unsigned char cardNo, unsigned char axis)
```

Parameter:

cardNO : [Input] 0~19, The selected card number.
axis : [Input] The selected axis. 1: X-axis; 2: Y-axis; 3: Z-axis

Return Value:

None

Example:

```
#define CARD1 1  
int slot=1;  
i8090_REGISTRATION(CARD1, slot);  
i8090_INIT_CARD(CARD1, ENC_QUADRANT, ENC_QUADRANT,ENC_QUADRANT);  
i8090_RESET_ENCODER(CARD1, X_axis);  
  
//The X-axis encoder value set zero.
```

Remark:

This function can be applied on module: i8090.

■ i8090_GET_ENCODER32

Description:

This function is used to obtain the counter value of the selected axis on the specific encoder card. The counter value is defined in the long (32-bit) format. Users must call the i8090_ENCODER32_ISR() function before using this function.

Syntax:

```
[C++]  
  
long i8090_GET_ENCODER32(unsigned char cardNo, unsigned char axis)
```

Parameter:

cardNO : [Input] 0~19, select which card.

axis : [Input] The selected axis. 1: X-axis; 2: Y-axis; 3: Z-axis

Return Value:

A 32 bits integer value.

Example:

```
#define CARD1 1  
int slot=1;  
long data;  
i8090_REGISTRATION(CARD1, slot);  
i8090_INIT_CARD(CARD1, ENC_QUADRANT, ENC_QUADRANT,ENC_QUADRANT);  
i8090_ENCODER32_ISR(CARD1);  
data=i8090_GET_ENCODER32(CARD1, X_axis);  
//The data value is the X-axis encoder value.
```

Remark:

This function can be applied on module: i8090.

■ i8090_RESET_ENCODER32

Description:

This function is applied to reset the counter variable of the function i8090_Get_Encoder32.

Syntax:

```
[C++]  
  
void i8090_RESET_ENCODER32(unsigned char cardNo, unsigned char axis)
```

Parameter:

cardNO : [Input] 0~19, The selected card number.
axis : [Input] The selected axis. 1: X-axis; 2: Y-axis; 3: Z-axis

Return Value:

None

Example:

```
#define CARD1 1  
  
int slot=1;  
  
i8090_REGISTRATION(CARD1, slot);  
  
i8090_INIT_CARD32(CARD1, ENC_QUADRANT, ENC_QUADRANT,ENC_QUADRANT);  
  
i8090_RESET_ENCODER(CARD1, X_axis);  
  
//The X-axis encoder value set zero.
```

Remark:

This function can be applied on module: i8090.

■ i8090_GET_INDEX

Description:

This function is used to get the value of the “INDEX” register on the specific card.

Syntax:

```
[C++]  
  
unsigned char i8090_GET_INDEX(unsigned char cardNo)
```

Parameter:

cardNO : [Input] 0~19, The specific card number.

Return Value:

Register	Add.	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INDEX	0x08	R						ZI	YI	XI

Example:

```
#define CARD1 1  
unsigned char data;  
data=i8090_GET_INDEX(CARD1);  
  
//Returned value: data=0x07
```

Remark:

This function can be applied on module: i8090. The index input C+/C- can read out from this register. These bits are highly active.

XI : Indicate the index of X-axis.

YI : Indicate the index of Y-axis.

ZI : Indicate the index of Z-axis.

■ i8090_ENCODER32_ISR

Description:

This function is used to calculate the pulse value between present and last time with a "long type" format. According to this idea, the i8090_ENCODER32_ISR() function should be executed periodically (2~10ms) using the timer interrupt or manual method.

Syntax:

```
[C++]  
  
void i8090_ENCODER32_ISR(unsigned char cardNo)
```

Parameter:

cardNO : [Input] 0~19, The selected card number.

Return Value:

None

Example:

```
#define CARD1 1  
  
long data;  
  
i8090_ENCODER32_ISR(CARD1);  
  
data=i8090_GET_ENCODER32(CARD1, X_axis);  
//The i8090_ENCODER32_ISR() should be called in 2~20ms.
```

Remark:

This function can be applied on module: i8090.

2.9 2-axis Stepper/Servo Functions

■ i8091_REGISTRATION

Description:

This function is used to assign a card number “cardNo” to the I-8091 card in the specific slot. In order to distinguish more than one of the I-8091 cards in WinCon-8000 platform, the I-8091 cards should be registered before using them. If there is no I-8091 at the given address, this command will return 0 which is a failure value.

Syntax:

```
[C++]  
  
unsigned char i8091_REGISTRATION(unsigned char cardNo, int slot)
```

Parameter:

cardNO : [Input] The board number (0~19)
slot : [Input] The specific slot which i8091 card is plugged in (1~7)

Return Value:

1: card exist
0: card not exist

Example:

```
#define CARD1 1  
int slot=1;  
i8091_REGISTRATION(CARD1, slot);  
//The I-8091 card is plugged into slot 1 of Wincon-8000.
```

Remark:

This function can be applied on module: i8091.

■ i8091_RESET_SYSTEM

Description:

This function is used to reset and terminate the running command in the 8091 module. Users can apply this command in software emergencies as a stop function. It can also clear all the card settings. After calling this function, users need to configure all the parameters in the I-8091 card.

Syntax:

```
[C++]  
  
void i8091_RESET_SYSTEM(unsigned char cardNo)
```

Parameter:

cardNO : [Input] 0~19, The selected card number.

Return Value:

None

Example:

```
#define CARD1 1  
int slot=1;  
i8091_REGISTRATION(CARD1, slot);  
i8091_RESET_SYSTEM(CARD1);  
//The I-8091 card plugged in slot 1 of Wincon-8000
```

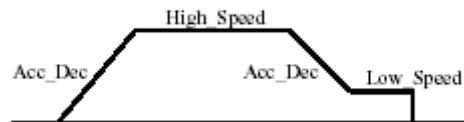
Remark:

This function can be applied on module: i8091.

■ i8091_SET_VAR

Description:

This function is used to set the DDA cycle, plus accelerating/decelerating speeds, low-speed and the high-speed values in the specified I-8091 card.



Syntax:

[C++]

```
void i8091_SET_VAR(unsigned char cardNo, unsigned char DDA_cycle,  
                  unsigned char Acc_Dec, unsigned int Low_Speed, unsigned int High_Speed)
```

Parameter:

cardNO : [Input] 0~19, The selected card number.
DDA_cycle : [Input] 1<=DDA_cycle<=254.
Acc_Dec : [Input] 1<=Acc_Dec<=200.
Low_Speed : [Input] 1<=Low_Speed<=200.
High_Speed : [Input] Low_Speed<=High_Speed<=2047.

Return Value:

None

Example:

```
#define CARD1 1  
int slot=1;  
i8091_REGISTRATION(CARD1, slot);  
i8091_SET_VAR(CARD1, 5, 2, 10, 150);
```

Remark:

This function can be applied on module: i8091.

■ i8091_SET_DEFDIR

Description:

This function is used to define the rotating directions of X and Y axes for controlling motors. Sometimes, the output direction of X-axis or Y-axis is in an undesired direction because of the wire connection to the motor or gear train mechanism. In order to unify the output direction, the CW/FW directions of X/Y axis are defined as an outside going motion through the motor control, and similarly the CCW/BW directions are defined as the inward motion through the motor control.

Syntax:

```
[C++]  
  
void i8091_SET_DEFDIR(unsigned char cardNo, unsigned char defdirX,  
                     unsigned char defdirY)
```

Parameter:

cardNO : [Input] The board number (0~19)
defdirX : [Input] X axis direction definition
 (0:NORMAL_DIR, 1:REVERSE_DIR)
defdirY : [Input] Y axis direction definition
 (0:NORMAL_DIR, 1:REVERSE_DIR)

Return Value:

None

Example:

```
#define CARD1 1  
int slot=1;  
i8091_REGISTRATION(CARD1, slot);  
i8091_SET_DEFDIR(CARD1, 0, 0);
```

Remark:

This function can be applied on module: i8091.

■ i8091_SET_MODE

Description:

This function is used to set the motor control modes of X and Y axes in the specific I-8091 card.

Syntax:

```
[C++]  
void i8091_SET_MODE(unsigned char cardNo, unsigned char modeX,  
                   unsigned char modeY)
```

Parameter:

cardNO : [Input] The board number (0~19)
modeX : [Input] X axis output mode
(0:CW/CCW mode, 1:Pulse/Direction mode)
modeY : [Input] Y axis output mode
(0:CW/CCW mode, 1:Pulse/Direction mode)

Return Value:

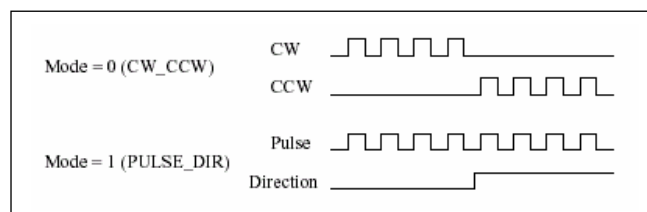
None

Example:

```
#define CARD1 1  
int slot=1;  
i8091_REGISTRATION(CARD1, slot);  
i8091_SET_MODE(CARD1, 0, 0);
```

Remark:

This function can be applied on module: i8091.



■ i8091_SET_SERVO_ON

Description:

This function is used to turn the servo function on/off to get the motor driver ready or to stop motor control.

Syntax:

```
[C++]  
  
void i8091_SET_SERVO_ON(unsigned char cardNo, unsigned char sonX,  
                        unsigned char sonY)
```

Parameter:

cardNO : [Input] The board number (0~19)
modeX : [Input] X-axis servo/hold on switch (1:ON , 0:OFF)
modeY : [Input] X-axis servo/hold on switch (1:ON , 0:OFF)

Return Value:

None

Example:

```
#define CARD1 1  
int slot=1;  
i8091_REGISTRATION(CARD1, slot);  
i8091_SET_SERVO_ON(CARD1, 1, 1);
```

Remark:

This function can be applied on module: i8091.

■ i8091_SET_NC

Description:

This function is used to set all of the following limit switches to N.C.(normal close) or N.O.(normal open). If users set the “sw” parameter as N.O, then those limit switches are active low. If users set the value as N.C, those limit switches are then “active high”. The auto-protection system will automatically change the judgments, whatever it is, to N.O. or N.C.

Limit switches: ORG1, LS11, LS14, ORG2, LS21, LS24, EMG.

Syntax:

```
[C++]  
  
void i8091_SET_NC(unsigned char cardNo, unsigned char sw)
```

Parameter:

cardNO : [Input] The board number (0~19)

sw : [Input] 0(NO) normal open (default), 1(YES) normal close.

Return Value:

None

Example:

```
#define CARD1 1  
int slot=1;  
i8091_REGISTRATION(CARD1, slot);  
i8091_SET_NC(CARD1, 1, 1);
```

Remark:

This function can be applied on module: i8091.

■ i8091_STOP_X

Description:

This function is used to stop the X-axis from action immediately.

Syntax:

```
[C++]  
void i8091_STOP_X(unsigned char cardNo)
```

Parameter:

cardNO : [Input] The board number (0~19)

Return Value:

None

Example:

```
#define CARD1 1  
int slot=1;  
i8091_REGISTRATION(CARD1, slot);  
i8091_STOP_X(CARD1);  
//The X-axis is stopped.
```

Remark:

This function can be applied on module: i8091.

This command would stop the X axis immediately.

■ i8091_STOP_Y

Description:

This function is used to stop the Y-axis from action immediately.

Syntax:

```
[C++]  
void i8091_STOP_Y(unsigned char cardNo)
```

Parameter:

cardNO : [Input] The board number (0~19)

Return Value:

None

Example:

```
#define CARD1 1  
int slot=1;  
i8091_REGISTRATION(CARD1, slot);  
i8091_STOP_Y(CARD1);  
//The Y-axis is stopped.
```

Remark:

This function can be applied on module: i8091.

This command would stop the Y axis immediately.

■ i8091_STOP_ALL

Description:

This function is used to stop both the X and Y axis immediately. It will clear all commands that are pending in the FIFO.

Syntax:

```
[C++]  
  
void i8091_STOP_ALL(unsigned char cardNo)
```

Parameter:

cardNO : [Input] The board number (0~19)

Return Value:

None

Example:

```
#define CARD1 1  
int slot=1;  
i8091_REGISTRATION(CARD1, slot);  
i8091_STOP_ALL(CARD1);  
//The X-axis and Y-axis are stopped.
```

Remark:

This function can be applied on module: i8091.

■ i8091_EMG_STOP

Description:

This function is the same as the i8091_STOP_ALL function, but can only be used in the interrupt routine. It can clear all the commands that are pending in the FIFO.

Syntax:

```
[C++]  
  
void i8091_EMG_STOP(unsigned char cardNo)
```

Parameter:

cardNO : [Input] The board number (0~19)

Return Value:

None

Example:

```
#define CARD1 1  
int slot=1;  
i8091_REGISTRATION(CARD1, slot);  
i8091_EMG_STOP(CARD1);  
//The X-axis and Y-axis are stopped.
```

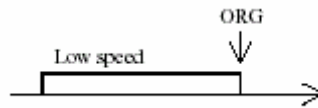
Remark:

This function can be applied on module: i8091.

■ i8091_LSP_ORG

Description:

This function is used to stop the specific (X/Y) axis in low-speed movement when the ORG1/ORG2 limit switch is in contact.



Syntax:

```
[C++]  
void i8091_LSP_ORG(unsigned char cardNo, unsigned char DIR,  
                  unsigned char AXIS)
```

Parameter:

cardNO : [Input] The board number (0~19)
DIR : [Input] The moving direction. (0: CW, 1: CCW)
AXIS : [Input] The selected axis. (1: X_axis, 2: Y_axis)

Return Value:

None

Example:

```
#define CARD1 1  
int slot=1;  
i8091_REGISTRATION(CARD1, slot);  
i8091_LSP_ORG(CARD1, CCW, X_axis);  
i8091_LSP_ORG(CARD1, CCW, Y_axis);
```

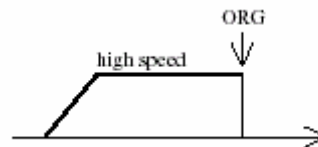
Remark:

This function can be applied on module: i8091.

■ i8091_HSP_ORG

Description:

This function drives the specific (X/Y) axis to search for home position (ORG1/ORG2) in the high-speed mode motion and stops the motion when the ORG1/ORG2 limit switch is touched.



Syntax:

```
[C++]  
void i8091_HSP_ORG(unsigned char cardNo, unsigned char DIR,  
                  unsigned char AXIS)
```

Parameter:

cardNO : [Input] The board number (0~19)
DIR : [Input] The moving direction. (0: CW, 1: CCW)
AXIS : [Input] The selected axis. (1: X_axis, 2: Y_axis)

Return Value:

None

Example:

```
#define CARD1 1  
int slot=1;  
i8091_REGISTRATION(CARD1, slot);  
i8091_HSP_ORG(CARD1, CCW, X_axis);  
i8091_HSP_ORG(CARD1, CCW, Y_axis);
```

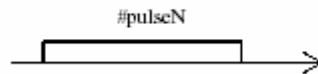
Remark:

This function can be applied on module: i8091.

■ i8091_LSP_PULSE_MOVE

Description:

This function drives the specified axis into motion toward the desired position from the given pulse number in low-speed mode.



Syntax:

```
[C++]  
void i8091_LSP_PULSE_MOVE(unsigned char cardNo, unsigned char AXIS,  
                           long pulseN)
```

Parameter:

cardNO : [Input] The board number (0~19)
AXIS : [Input] The selected axis (1: X_axis, 2: Y_axis)
pulseN : [Input] The moving number of pulse.

Return Value:

None

Example:

```
i8091_LSP_PULSE_MOVE(CARD1, X_axis, 20000);  
i8091_LSP_PULSE_MOVE(CARD1, X_axis, -2000);  
i8091_LSP_PULSE_MOVE(CARD1, Y_axis, 20000);  
i8091_LSP_PULSE_MOVE(CARD1, Y_axis, -2000);  
  
//where  
//when pulseN>0, move toward CW/FW direction  
//when pulseN<0, move toward CCW/BW direction
```

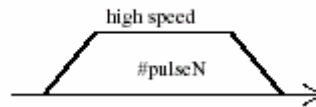
Remark:

This function can be applied on module: i8091.

■ i8091_HSP_PULSE_MOVE

Description:

This function drives the specified axis into motion toward the desired position from the given pulse number in high-speed mode.



Syntax:

```
[C++]  
void i8091_HSP_PULSE_MOVE(unsigned char cardNo, unsigned char AXIS,  
                           long pulseN)
```

Parameter:

cardNO : [Input] The board number (0~19)
AXIS : [Input] The selected axis (1: X_axis, 2: Y_axis)
pulseN : [Input] The moving number of pulse.

Return Value:

None

Example:

```
i8091_HSP_PULSE_MOVE(CARD1, X_axis, 20000);  
i8091_HSP_PULSE_MOVE(CARD1, X_axis, -2000);  
i8091_HSP_PULSE_MOVE(CARD1, Y_axis, 20000);  
i8091_HSP_PULSE_MOVE(CARD1, Y_axis, -2000);  
//where  
//when pulseN>0, move toward CW/FW direction  
//when pulseN<0, move toward CCW/BW direction
```

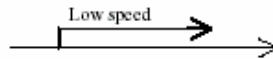
Remark:

This function can be applied on module: i8091.

■ i8091_LSP_MOVE

Description:

This function drives the specified axis into motion toward the selected direction in low-speed mode. It can be stopped by the i8091_STOP_X function, or the i8091_STOP_Y function, or the i8091_STOP_ALL function.



Syntax:

```
[C++]  
void i8091_LSP_MOVE(unsigned char cardNo, unsigned char DIR,  
                    unsigned char AXIS)
```

Parameter:

cardNO : [Input] The board number (0~19)
DIR : [Input] The moving direction. (0: CW, 1: CCW)
AXIS : [Input] The selected axis (1: X_axis, 2: Y_axis)

Return Value:

None

Example:

```
i8091_LSP_MOVE(CARD1, CW, X_axis);  
i8091_LSP_MOVE(CARD1, CW, Y_axis);
```

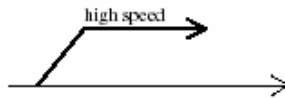
Remark:

This function can be applied on module: i8091.

■ i8091_HSP_MOVE

Description:

This function drives the specified axis into motion toward the selected direction in high-speed mode. It can be stopped by the i8091_STOP_X, or i8091_STOP_Y, or i8091_STOP_ALL functions.



Syntax:

```
[C++]  
void i8091_HSP_MOVE(unsigned char cardNo, unsigned char DIR,  
                    unsigned char AXIS)
```

Parameter:

cardNO : [Input] The board number (0~19)
DIR : [Input] The moving direction. (0: CW, 1: CCW)
AXIS : [Input] The selected axis (1: X_axis, 2: Y_axis)

Return Value:

None

Example:

```
i8091_HSP_MOVE(CARD1, CW, X_axis);  
i8091_HSP_MOVE(CARD1, CW, Y_axis);
```

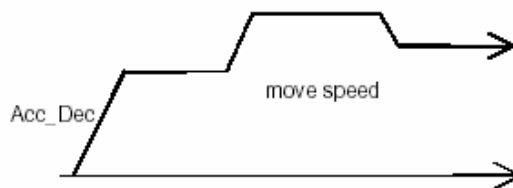
Remark:

This function can be applied on module: i8091.

■ i8091_CSP_MOVE

Description:

This function is used to accelerate/decelerate the motor on the selected axis into motion up/down to the desired speed. If commands are continuously applied to the I-8091, then this function can dynamically change the motor speed. The rotating motor can be stopped by using the following commands: i8091_STOP_X(), i8091_STOP_Y(), i8091_STOP_ALL(), or i8091_SLOW_STOP().



Syntax:

```
[C++]  
void i8091_CSP_MOVE(unsigned char cardNo, unsigned char dir,  
                   unsigned char axis, unsigned int move_speed)
```

Parameter:

cardNO : [Input] The board number (0~19)
dir : [Input] The moving direction. (0: CW, 1: CCW)
axis : [Input] The selected axis. (1: X_axis, 2: Y_axis)
move_speed : [Input] 0 < move_speed <= 2040

Return Value:

None

Example:

```
i8091_CSP_MOVE(CARD1, CW, X_axis, 10); //Delay(10000);  
i8091_CSP_MOVE(CARD1, CW, X_axis, 20);
```

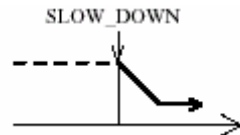
Remark:

This function can be applied on module: i8091.

■ i8091_SLOW_DOWN

Description:

This function is used to decelerate the motor motion to a specific low speed in order to be able to call either the i8091_STOP_X(), or i8091_STOP_Y(), or i8091_STOP_ALL functions so that the motor's motion can be stopped.



Syntax:

[C++]

```
void i8091_SLOW_DOWN(unsigned char cardNo, unsigned char AXIS)
```

Parameter:

cardNO : [Input] The board number (0~19)

AXIS : [Input] The selected axis (1: X_axis, 2: Y_axis)

Return Value:

None

Example:

```
i8091_HSP_MOVE(CARD1, CW, X_axis);
```

```
i8091_SLOW_DOWN(CARD1, X_axis);
```

```
i8091_STOP_X(CARD1);
```

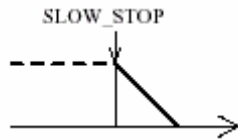
Remark:

This function can be applied on module: i8091.

■ i8091_SLOW_STOP

Description:

This function is used to decelerate the speed of a specified axis and then stop it (as shown in following figure).



Syntax:

[C++]

```
void i8091_SLOW_STOP(unsigned char cardNo, unsigned char AXIS)
```

Parameter:

cardNO : [Input] The board number (0~19)
AXIS : [Input] The selected axis (1: X_axis, 2: Y_axis)

Return Value:

None

Example:

```
i8091_HSP_MOVE(CARD1, CW, Y_axis);  
i8091_SLOW_STOP(CARD1, Y_axis);
```

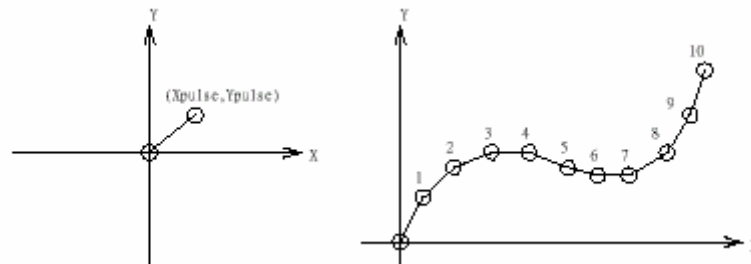
Remark:

This function can be applied on module: i8091.

■ i8091_INTP_PULSE

Description:

This function is used to move a short distance (interpolation short line) in the X-Y plane. This command provides a method for users to generate an arbitrary curve in a X-Y plane.



Syntax:

[C++]

```
void i8091_INTP_PULSE(unsigned char cardNo, int Xpulse, int Ypulse)
```

Parameter:

cardNO : [Input] The board number (0~19)
Xpulse : [Input] $-2047 \leq \# \text{ Xpulse} \leq 2047$
Ypulse : [Input] $-2047 \leq \# \text{ Ypulse} \leq 2047$

Return Value:

None

Example:

```
i8091_INTP_PULSE(CARD1, 20, 20);  
i8091_INTP_PULSE(CARD1, 20, 13);  
i8091_INTP_PULSE(CARD1, 20, 7);  
i8091_INTP_PULSE(CARD1, 20, 0);  
i8091_INTP_PULSE(CARD1, 15, -5);
```

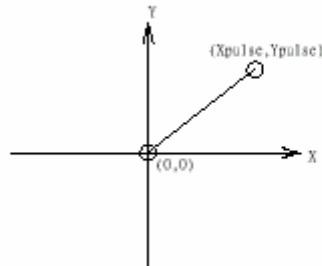
Remark:

This function can be applied on module: i8091.

■ i8091_INTP_LINE

Description:

This command will move a long distance (interpolation line) in the X-Y plane. The CPU on the I-8091 card will generate a trapezoidal speed profile of the X-axis and Y-axis, and then execute interpolation by way of a DDA chip.



Syntax:

[C++]

```
void i8091_INTP_LINE(unsigned char cardNo, int Xpulse, int Ypulse)
```

Parameter:

cardNO : [Input] The board number (0~19)
Xpulse : [Input] $-524287 \leq \# \text{ Xpulse} \leq 524287$
Ypulse : [Input] $-524287 \leq \# \text{ Ypulse} \leq 524287$

Return Value:

None

Example:

```
i8091_INTP_LINE(CARD1, 2000, -3000);  
i8091_INTP_LINE(CARD1, -500, 200);
```

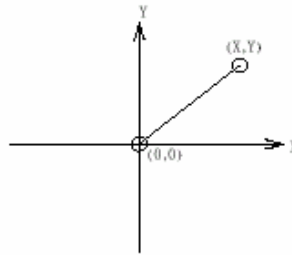
Remark:

This function can be applied on module: i8091.

■ i8091_INTP_LINE02

Description:

This function is used to move a long interpolation line in the X-Y plane. The host will automatically generate a trapezoidal speed profile of the X-axis and Y-axis via the state-machine-type calculation method. The i8091_INTP_LINE02 function only sets parameters into the driver. Users can directly utilize the `do { } while (i8091_INTP_STOP() !=READY)` method to apply the computing entity.



Syntax:

[C++]

```
void i8091_INTP_LINE02(unsigned char cardNo, long x, long y,  
                      unsigned int speed, unsigned char acc_mode)
```

Parameter:

cardNO : [Input] The board number (0~19)
x : [Input] The end point of the line relates to the present position
y : [Input] The end point of the line relates to the present position
speed : [Input] 0~2040
acc_mode : [Input] 0: enables the acceleration and deceleration profiles
 1: disables the acceleration and deceleration profiles

Return Value:

None

Example:

```
i8091_INTP_LINE02(CARD1, 1000, 1000, 100, 0);  
do { } while(i8091_INTP_STOP() !=READY);  
//call state machine
```

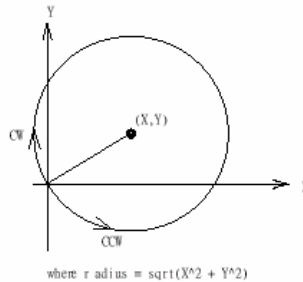
Remark:

This function can be applied on module: i8091.

■ i8091_INTP_CIRCLE02

Description:

This function is used to generate an interpolation circle in the X-Y plane. The host will automatically generate a trapezoidal speed profile of the X-axis and Y-axis via the state-machine-type calculation method. The i8091_INTP_CIRCLE02 function only sets parameters into the driver. Users can directly utilize the `do { } while (i8091_INTP_STOP() !=READY)` method to execute the computing entity.



Syntax:

[C++]

```
void i8091_INTP_CIRCLE02(unsigned char cardNo, long x, long y,  
    unsigned chat dir, unsigned int speed, unsigned char acc_mode)
```

Parameter:

cardNO : [Input] The board number (0~19)
x : [Input] The center point of the circle relates to the present position
y : [Input] The center point of the circle relates to the present position
dir : [Input] The moving direction. (0: CW, 1: CCW)
speed : [Input] 0~2040
acc_mode : [Input] 0: enable acceleration and deceleration profile
 1: disable acceleration and deceleration profile

Return Value:

None

Example:

```
i8091_INTP_CIRCLE02(CARD1, 2000, 2000, 100, 0);  
do { } while(i8091_INTP_STOP() !=READY);  
//call state machine
```

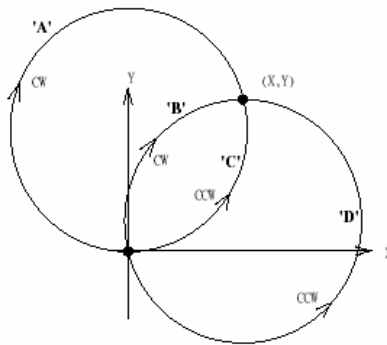
Remark:

This function can be applied on module: i8091.

■ i8091_INTP_ARC02

Description:

This command generates an interpolation arc in the X-Y plane. The host will automatically generate a trapezoidal speed profile of the X-axis and Y-axis via the state-machine-type calculation method. The i8091_INTP_ARC02() only sets parameters into the driver. Users can directly call the do { } while (i8091_INTP_STOP() !=READY) to execute the computing entity.



Syntax:

```

                                [C++]
void i8091_INTP_ARC02(unsigned char cardNo, long x, long y, long R,
                      unsigned char dir, unsigned int speed, unsigned char acc_mode)

```

Parameter:

- cardNO : [Input] The board number (0~19)
- x : [Input] The center point of the circle relates to the present position
- y : [Input] The center point of the circle relates to the present position
- R : [Input] The radius of arc
- dir : [Input] The moving direction (0: CW, 1: CCW)

R	dir	path of curve
R>0	CW	'B'
R>0	CCW	'C'
R<0	CW	'A'
R<0	CCW	'D'

- speed : [Input] 0~2040
- acc_mode : [Input] 0: enables the acceleration and deceleration profiles
1: disables the acceleration and deceleration profiles

Return Value:

None

Example:

```
i8091_INTP_ARC02(CARD1, 2000, -2000, 2000, CW, 100, 0);  
do { } while(i8091_INTP_STOP() != READY);  
//call state machine
```

Remark:

This function can be applied on module: i8091.

Restriction:

$$-2^{32} + 1 \leq x \leq 2^{32} - 1$$

$$-2^{32} + 1 \leq y \leq 2^{32} - 1$$

$$-2^{32} + 1 \leq R \leq 2^{32} - 1$$

$$R \geq \frac{\sqrt{x^2 + y^2}}{2}$$

■ i8091_INTP_STOP

Description:

This function must be applied when stopping the motor motion control for the above described 3 state-machine-type interpolation functions, to be precise the i8091_INTP_LINE02, i8091_INTP_CIRCLE02 and i8091_INTP_ARC02 functions. The state-machine-type interpolation functions only set parameters into the driver. The computing entity is in the i8091_INTP_STOP function. This function computes the interpolation profile. It will return READY(0) for interpolation command completion, and return BUSY(1) for when it is not yet completed.

Syntax:

```
[C++]  
  
unsigned char i8091_INTP_STOP(void)
```

Parameter:

None

Return Value:

0: READY

1: BUSY

Example:

```
i8091_INTP_CIRCLE02(CARD1,2000,2000,100,0);  
do { } while(i8091_INTP_STOP()!=READY);  
//call state machine
```

Remark:

This function can be applied on module: i8091

■ i8091_LIMIT_X

Description:

This function is used to request the condition of the X-axis limit switches.

Syntax:

```
[C++]  
  
unsigned char i8091_LIMIT_X(unsigned char cardNo)
```

Parameter:

cardNO : [Input] The board number (0~19)

Return Value:

a unsigned char value

MSB 7	6	5	4	3	2	1	0
/EMG	/FFFF	/FFEF	/LS14	xx	xx	/LS11	/ORG1

/ORG1 : original point switch of X-axis, low active.

/LS11, /LS14 : limit switches of X-axis, low active, which must be configured as Fig.(5). (Refer to I-8091 User's Manual)

/EMG : emergency switch, low active.

/FFEF : active low, FIFO is empty

/FFFF : active low, FIFO is full

Example:

```
unsigned char limit1;  
limit1 = i8091_LIMIT_X(CARD1);
```

Remark:

This function can be applied on module: i8091.

■ i8091_LIMIT_Y

Description:

This function is used to request the condition of the Y-axis limit switches.

Syntax:

```
[C++]  
  
unsigned char i8091_LIMIT_Y(unsigned char cardNo)
```

Parameter:

cardNO : [Input] The board number (0~19)

Return Value:

a unsigned char value

MSB 7	6	5	4	3	2	1	0
Ystop	Xstop	xx	/LS24	xx	xx	/LS21	/ORG2

/ORG2: original point switch of Y-axis, low active.

/LS21, /LS24: limit switches of Y-axis, low active, which must be configured as Fig.(6). (Refer to I-8091 User's Manual)

Xstop: 1:indicate X-axis is stop.

Ystop: 1:indicate Y-axis is stop.

Example:

```
unsigned char limit2;  
limit2 = i8091_LIMIT_Y(CARD1);
```

Remark:

This function can be applied on module: i8091.

■ i8091_WAIT_X

Description:

This function is used to make the X-axis wait before going to the STOP state.

Syntax:

```
[C++]  
void i8091_WAIT_X(unsigned char cardNo)
```

Parameter:

cardNO : [Input] The board number (0~19)

Return Value:

None

Example:

Remark:

This function can be applied on module: I8091.

■ i8091_WAIT_Y

Description:

This function is used to make the Y-axis wait before going to the STOP state.

Syntax:

```
[C++]  
void i8091_WAIT_Y(unsigned char cardNo)
```

Parameter:

cardNO : [Input] The board number (0~19)

Return Value:

None

Example:

Remark:

This function can be applied on module: i8091.

■ i8091_IS_X_STOP

Description:

This function is used to check whether the X-axis is in the stop state or not.

Syntax:

```
[C++]  
  
unsigned char i8091_IS_X_STOP(unsigned char cardNo)
```

Parameter:

cardNO : [Input] The board number (0~19)

Return Value:

0 (NO) : not yet stop
1 (YES) : stop

Example:

```
unsigned char data;  
data= i8091_IS_X_STOP(CARD1);
```

Remark:

This function can be applied on module: i8091.

■ i8091_IS_Y_STOP

Description:

This function is used to check whether the Y-axis is in the stop state or not.

Syntax:

```
[C++]  
unsigned char i8091_IS_Y_STOP(unsigned char cardNo)
```

Parameter:

cardNO : [Input] The board number (0~19)

Return Value:

0 (NO) : not yet stop
1 (YES) : stop

Example:

```
unsigned char data;  
data= i8091_IS_Y_STOP(CARD1);
```

Remark:

This function can be applied on module: i8091.

■ i8091_IS_X_STOP_DELAY

Description:

Except for the user-defined delay, this function is very similar to i8091_IS_X_STOP()

Syntax:

```
[C++]  
  
unsigned char i8091_IS_X_STOP_DELAY(unsigned char cardNo, unsigned char  
wT)
```

Parameter:

cardNO : [Input] The board number (0~19)
wT: [Input] The user-defined delay

Return Value:

0 (NO) : not yet stop
1 (YES) : stop

Example:

```
unsigned char data;  
data= i8091_IS_X_STOP_DELAY(CARD1, 20);
```

Remark:

This function can be applied on module: i8091.

■ i8091_IS_Y_STOP_DELAY

Description:

Except for the user-defined delay, this function is very similar to i8091_IS_Y_STOP()

Syntax:

```
[C++]  
  
unsigned char i8091_IS_Y_STOP_DELAY(unsigned char cardNo, unsigned char  
wT)
```

Parameter:

cardNO : [Input] The board number (0~19)
wT: [Input] The user-defined delay

Return Value:

0 (NO) : not yet stop
1 (YES) : stop

Example:

```
unsigned char data;  
data= i8091_IS_Y_STOP_DELAY(CARD1, 20);
```

Remark:

This function can be applied on module: i8091.

2.10 Counter/Frequency Functions

■ i8080_InitDriver

Description:

The 'i8080_InitDriver ()' is used to initialize the 8080 module.

Syntax:

```
[C++]  
  
int i8080_InitDriver(int Slot)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)

Return Value:

0: OK

The others: Error codes, Refer to I8080.h

Example:

```
int slot=1;  
int ErrorCode;  
ErrorCode=i8080_InitDriver(slot);  
//The I-8080 card is plugged in slot 1 of Wincon-8000  
//If the ErrorCode=0 expresses OK.
```

Remark:

This function can be applied on module: i8080.

■ i8080_AutoScan

Description:

This function is used to update counter values and to correctly transmit from the low 16-bit hardware counter to the high 16-bit software counter. The user's program must call this function from within their loop function or timer event. Refer to the "I-8080 Software User's Manual" Sec. 2.2 for more details.

Syntax:

[C++]
<code>void i8080_AutoScan(void)</code>

Parameter:

None

Return Value:

None

Example:

```
void CUpCount1Dlg::OnTimer(UINT nIDEvent)
{
    // TODO: Add your message handler code here and/or call default
    i8080_AutoScan();
}
```

Remark:

This function can be applied on module: i8080.

■ i8080_ReadDiXor

Description:

This function is used to read the signal status for the 8 channels in the i8080 after Xor Control processing is accomplished.

Syntax:

```
[C++]  
  
int i8080_ReadDiXor(int Slot, short *Di)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)
Di: [Output] An integer point, which allows the user to read the 8 channels of signal status after Xor control.
Bit0 of Di = A0 after Xor Control
Bit7 of Di = B3 after Xor Control

Return Value:

0: OK
The others: Error codes

Example:

```
int slot=1;  
short j;  
i8080_ReadDiXor(slot, &j);  
//j=DiXor Value
```

Remark:

This function can be applied on module: i8080.

■ i8080_ReadDIXorLp

Description:

This function is used to read the signal status for the 8 channels in the i8080 after Xor Control and Low Pass Filter processing is accomplished.

Syntax:

```
[C++]  
  
int i8080_ReadDIXorLp(int Slot, short *Di)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)
Di: [Output] An integer point, which allows the user to read the 8 channels of signal status after the Xor Controls and Low Pass Filtering are accomplished.
Bit0 of Di = A0 after Xor Control & Low Pass Filter
Bit7 of Di = B3 after Xor Control & Low Pass Filter

Return Value:

0: OK
The others: Error codes

Example:

```
int slot=1;  
short Di;  
i8080_ReadDIXorLp(slot, &Di);  
// Di=DiXorLp Value
```

Remark:

This function can be applied on module: i8080.

■ i8080_ReadXorRegister

Description:

The function is used to read the 8 channel Xor registers in the i8080.

Syntax:

```
[C++]  
int i8080_ReadXorRegister(int Slot, short *XorVal)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)
XorVal: [Output] An integer point, which allows the user to save the 8 channels of Xor Control Registers.
Bit0 = A0's Xor Control Register
Bit7 = B3's Xor Control Register

Return Value:

0: OK
The others: Error codes

Example:

```
int slot=1;  
short XorV;  
i8080_ReadXorRegister(slot, &XorV);  
// XorV=XorRegister Value
```

Remark:

This function can be applied on module: i8080.

■ i8080_SetXorRegister

Description:

This function is used to set the 8 channel Xor registers in the i8080 to 0 or 1. The setting value isn't saved to EEPROM, so all of the Xor registers will be default (0) after rebooting the i8080.

Syntax:

```
[C++]  
  
int i8080_SetXorRegister(int Slot, short XorVal)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7).
XorVal: [Input] Allows the user to set the 8 channels in the Xor Control Registers.
Bit0 = A0's Xor Control Register
Bit7 = B3's Xor Control Register

Return Value:

0: OK
The others: Error codes

Example:

```
int slot=1;  
short XorV;  
XorV=0;  
Set8080XorRegister(slot,XorV);
```

Remark:

This function can be applied on module: i8080.

■ i8080_EepWriteEnable

Description:

The function is used to set the EEPROM to the write-enable mode. Then the EEPROM will allow users to write information to the EEPROM.

Syntax:

```
[C++]  
  
int i8080_EepWriteEnable(int Slot)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)

Return Value:

0: OK

The others: Error codes

Example:

```
int slot=1;  
i8080_EepWriteEnable(slot);
```

Remark:

This function can be applied on module: i8080.

■ i8080_EepWriteDisable

Description:

Set's the EEPROM to the write-protected mode. The EEPROM will then not allow users to write to the EEPROM.

Syntax:

```
[C++]  
  
int i8080_EepWriteDisable(int Slot)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)

Return Value:

0: OK

The others: Error codes

Example:

```
int slot=1;  
i8080_EepWriteDisable (slot);
```

Remark:

This function can be applied on module: i8080.

■ i8080_EepWriteWord

Description:

This function is used to write 16-bit data to the EEPROM on the i8080. This 16-bit data is divided into high-byte (8 bits) and low-byte (8-bits).

Syntax:

```
[C++]  
  
int i8080_EepWriteWord(int Slot, short Addr, short Hi, short Lo)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)
Addr : [Input] 0 ~ 63 = Address Number
Hi : [Input] High Byte Value
Lo : [Input] Low Byte Value

Return Value:

0: OK
The others: Error codes

Example:

```
int slot=1, k;  
TCHAR temp[80];  
i8080_EepWriteEnable(slot);  
for (int i=0; i<64; i++)  
{  
    k=EepWriteWord(slot,i,i+2,i+1);  
    swprintf(temp, _T("Write_%x Error=%x"), i, k );  
    if (k) MessageBox(NULL, temp, _T("Information"), MB_OK);  
}
```

Remark:

This function can be applied on module: i8080.

■ i8080_EepReadWord

Description:

This function is used to read 16-bit data from the i8080 EEPROM. This 16-bit data is divided into high-bytes and low-bytes.

Syntax:

```
[C++]  
  
int i8080_EepReadWord(int Slot, short *Addr, short *Hi, short *Lo)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)
Addr : [Output] 0 ~ 63 = Address Number
Hi : [Output] High Byte Value
Lo : [Output] Low Byte Value

Return Value:

0: OK
The others: Error codes

Example:

```
int slot=1, i,j,k;  
TCHAR temp[80];  
for (i=0; i<64; i++)  
{  
    i8080_EepReadWord(slot,i,&j,&k);  
    swprintf(temp, _T("Read ADDR=%x --> HIGH=%x, LOW=%x"),i,j,k);  
    MessageBox(NULL, temp, _T("Information"), MB_OK);  
}
```

Remark:

This function can be applied on module: i8080.

■ i8080_ReadChannelMode

Description:

This function is used to read the operational mode, which includes the measured algorithm of the Frequency mode, Low Pass Filter status and the Xor Control from each channel.

Syntax:

```
[C++]  
  
int i8080_ReadChannelMode(int Slot, int Channel, short *Mode)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)
Channel : [Input] 0 ~ 7 = Channel Number(0=A0, 1=B0 ..., 6=A3, 7=B3)
Mode : [Output] Mode is defined as follows:
Bit1 and bit0 are used to select the operational mode.
Bit1 and bit0 = 00 --> Dir/Pulse Mode
 = 01 --> Up/Down Mode
 = 10 --> Frequency Mode
 = 11 --> Up Count Mode
Bit3 and bit2 are valid for Frequency Mode Only.
Bit3 and bit2 = 00 --> Auto Select Frequency.
 = 01 --> Select Low Frequency
 = 10 --> Select High Frequency
 = 11 --> Stop this channel (For all 4 operational mode)
Bit4 = 1/0 --> Enable/Disable Low Alarm (The function is reserved)
Bit5 = 1/0 --> Enable/Disable High Alarm (The function is reserved)
Bit6 = 1/0 --> Enable/Disable Low Pass filter
Bit7 = 0 --> Xor =0 --> non-invert this channel
Bit7 = 1 --> Xor =1 --> invert this channel

Return Value:

0: OK
The others: Error codes

Example:

```
int slot=1, j, k;  
TCHAR temp[80];  
for (j=0; j<8; j++)  
{ //for human's double check  
    i8080_ReadChannelMode(slot,j,&k);  
    swprintf(temp, _T(" %02x"), k);  
    MessageBox(NULL, temp, _T("Information"), MB_OK);  
}
```

Remark:

This function can be applied on module: i8080.

■ i8080_SetChannelMode

Description:

Set's the operation mode, the measured algorithm of the Frequency mode, Low Pass Filter status and the Xor Control for each channel. The setting value isn't saved to the EEPROM, so all the values will be set at default after rebooting the i8080.

Syntax:

```
[C++]  
  
int i8080_SetChannelMode(int Slot, int Channel, short Mode)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)
Channel : [Input] 0 ~ 7 = Channel Number(0=A0, 1=B0 ..., 6=A3, 7=B3)
Mode : [Input] Mode is defined as follows:
Bit1 and bit0 are used to select the operational mode.
Bit1 and bit0 = 00 --> Dir/Pulse Mode
 = 01 --> Up/Down Mode
 = 10 --> Frequency Mode
 = 11 --> Up Count Mode
Bit3 and bit2 are valid for Frequency Mode Only
Bit3 and bit2 =00 -->Auto Select Frequency
 =01 -->Select Low Frequency
 =10 -->Select High Frequency
 =11 -->Stop this channel (For all 4 operational modes)
Bit4 = 1/0 --> Enable/Disable Low Alarm (The function is reserved)
Bit5 = 1/0 --> Enable/Disable High Alarm (The function is reserved)
Bit6 = 1/0 --> Enable/Disable Low Pass filter
Bit7 = 0 --> Xor=0 --> non-invert this channel
Bit7 = 1 --> Xor=1 --> invert this channel

Return Value:

0: OK

The others: Error codes

Example:

```
int slot=1;
i8080_SetChannelMode(slot,0,0x01); //A0, Up_counting_0
i8080_SetChannelMode(slot,1,0x01); //B0, Down_counting_0
//A0 & B0 must be same
i8080_SetChannelMode(slot,6,0x03); //A3, Up_counting
i8080_Set8080ChannelMode(slot,7,0x03); //B3, Up_counting
//Xor=0, isolated+active_Low
```

Remark:

This function can be applied on module: i8080.

■ i8080_ReadFreq

Description:

This function is used to read the Measured Frequency value in the Frequency mode.

Syntax:

```
[C++]  
  
int i8080_ReadFreq(int Slot, int Channel, float *f)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)

Channel : [Input] 0 ~ 7 = Channel Number(0=A0, 1=B0 ..., 6=A3, 7=B3)

f : [Output] The frequency Measured value. The variable must be a float point.

Return Value:

0: OK

The others: Error codes

Example:

```
int slot=1;  
float f;  
TCHAR temp[80];  
i8080_ReadFreq(slot,0,&f);  
swprintf(temp, _T("Channel_A0 : Frequency=%.1f"), f);  
MessageBox(NULL, temp, _T("Information"), MB_OK);
```

Remark:

This function can be applied on module: i8080.

■ i8080_ReadCntUp

Description:

This function is used for the Up Counter mode to read the Up Counter value.

Syntax:

```
[C++]  
  
int i8080_ReadCntUp(int Slot, int Channel, unsigned long *Cnt32U,  
                   unsigned int *Overflow)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)
Channel : [Input] 0 ~ 7 = Channel Number(0=A0, 1=B0 ..., 6=A3, 7=B3)
Cnt32U : [Output] 32-bit Up Counter (High 16-bit software Counter and Low
16-bit hardware Counter)
Overflow : [Output] number of Overflow (16-bit)

Return Value:

0: OK
The others: Error codes

Example:

```
int slot=1;  
unsigned long cnt32U;  
unsigned short overflow;  
TCHAR temp[80];  
i8080_ReadCntUp(slot,0,&cnt32U,&overflow);  
swprintf(temp, _T("Channel_A0 : overflow=%x, cnt32=%lx"),overflow,cnt32U);  
MessageBox(NULL, temp, _T("Information"), MB_OK);
```

Remark:

This function can be applied on module: i8080.

■ i8080_ReadCntUpDown

Description:

This function is used for the Up/Down counter and for the Dir/Pulse mode in order to read the up-counting and down-counting values.

Syntax:

```
[C++]  
  
int i8080_ReadCntUpDown(int Slot, int Channel, unsigned long *Cnt32U,  
                        unsigned int *Overflow)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)
Channel : [Input] 0 ~ 7 = Channel Number(0=A0, 1=B0 ..., 6=A3, 7=B3)
Cnt32U : [Output] 32-bit Up/Down Counter
 MSB=0 --> Up Count
 MSB=1 --> Down Count
Overflow : [Output] number of overflow (16 bit)

Return Value:

0: OK
The others: Error codes

Example:

```
//These variable as forward  
i8080_ReadCntUpDown(slot,0,&cnt32U,&overflow); // (A0,B0)  
swprintf(temp,_T("Channel_A0_B0:overflow=%x,cnt32=%lx"),overflow,cnt32U);  
MessageBox(NULL, temp, _T("Information"), MB_OK);
```

Remark:

This function can be applied on module: i8080.

■ i8080_SetLowPassUs

Description:

This function is used to set the parameters of Low Pass Filter in a specific channel in the specified slot. For more details relating to Low Pass Filter information, please refer to the I-8080 hardware manual.

Syntax:

```
[C++]  
  
int i8080_SetLowPassUs(int Slot, int Channel, unsigned short Us)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)
Channel : [Input] Low Pass filter channel number
 Channel0 is valid for A0, B0
 Channel1 is valid for A1, B1
 Channel2 is valid for A2, B2, A3 and B3
Us : [Input] 1 to 0x7fff (1~32767), unit=0.000001 second (μS)

Return Value:

0: OK
The others: Error codes

Example:

```
int slot=1;  
i8080_SetLowPassUs(slot,0,1000U); //valid for A0 & B0
```

Remark:

This function can be applied on module: i8080.

■ i8080_ReadLowPassUs

Description:

To read Low-Pass Filter information. For more details relating to Low Pass Filter information, please refer to the I-8080 hardware manual.

Syntax:

```
[C++]  
  
int i8080_ReadLowPassUs(int Slot, int Channel, unsigned short *Us)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)
Channel : [Input] Low Pass filter channel number
 Channel0 is valid for A0, B0
 Channel1 is valid for A1, B1
 Channel2 is valid for A2, B2, A3 and B3
Us : [Output] 1 to 0x7fff (1~32767), unit=0.000001 second (μS)

Return Value:

0: OK
The others: Error codes

Example:

```
int slot=1;  
unsigned short MinWidth;  
i8080_SetLowPassUs(slot, 0,1000U); //valid for A0 & B0  
i8080_ReadLowPassUs(slot, 0, &MinWidth);
```

Remark:

This function can be applied on module: i8080.

■ i8080_ClrCnt

Description:

This function is used to clear the specific channel counter value in a specific slot. All four-operation modes are supported in this function. (Dir/ Pulse, Up/Down, Up Counter, and Frequency mode)

Syntax:

```
[C++]  
  
int i8080_ClrCnt(int Slot, int Channel)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)
Channel : [Input] 0 ~ 7 = Channel Number(0=A0, 1=B0 ..., 6=A3, 7=B3)

Return Value:

0: OK
The others: Error codes

Example:

```
int slot=1;  
i8080_ClrCnt(slot,0); //clear counter A0  
i8080_ClrCnt(slot,1); //clear counter B0  
i8080_ClrCnt(slot,6); //clear counter A3  
i8080_ClrCnt(slot,7); //clear counter B3
```

Remark:

This function can be applied on module: i8080.

■ i8080_ReadFreqConfiguration

Description:

This function is used to read the configuration data in the frequency mode.

Syntax:

```
[C++]  
int i8080_ReadFreqConfiguration(int Slot, short *AutoTT , short *AutoVV,  
    short *LowTT, short *LowVV, short *HighTT, short *HighVV)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)
AutoTT: [Output] Sampling time period for Auto mode, unit= millisecond
AutoVV: [Output] Samples to start frequency update (Auto mode)
LowTT: [Output] Sampling time period for Low Frequency mode, unit=
millisecond
LowVV: [Output] Samples to start frequency update (Low Frequency Mode)
HighTT: [Output] Sampling time period for High Frequency mode, unit=
millisecond
HighVV: [Output] Samples to start frequency update (High Frequency Mode)

Return Value:

0: OK
The others: Error codes

Example:

```
int slot=1;  
short AutoTT, AutoVV, LowTT, LowVV, HighTT, HighVV;  
TCHAR temp[80];  
i8080_ReadFreqConfiguration(slot,&AutoTT,&AutoVV,&LowTT,&LowVV,&HighTT,&HighVV);  
//Current Configuration Data for Frequency Measurement Algorithm  
swprintf(temp, _T("AutoTT=%d mS, AutoVV=%d counts"),AutoTT,AutoVV);  
MessageBox(NULL, temp, _T("Information"), MB_OK);  
swprintf(temp, _T("LowTT=%d mS, LowVV=%d counts"),LowTT,LowVV);  
MessageBox(NULL, temp, _T("Information"), MB_OK);  
swprintf(temp, _T("\n\rHighTT=%d mS, HighVV=%d counts"),HighTT,HighVV);  
MessageBox(NULL, temp, _T("Information"), MB_OK);
```

Remark:

This function can be applied on module: i8080.

■ i8080_SetFreqConfiguration

Description:

This function is used to set the configuration data used by frequency measurement algorithms. The setting value isn't saved to the EEPROM, so the all values will be set to default after rebooting the i8080.

Syntax:

```
[C++]  
  
int i8080_SetFreqConfiguration(int Slot, short AutoTT , short AutoVV,  
                               short LowTT, short LowVV, short HighTT, short HighVV)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)
AutoTT: [Input] Sampling time period for Auto mode, unit= millisecond
AutoVV: [Input] Samples to start frequency update (Auto mode)
LowTT: [Input] Sampling time period for Low Frequency mode, unit=
 millisecond
LowVV: [Input] Samples to start frequency update (Low Frequency Mode)
HighTT: [Input] Sampling time period for High Frequency mode, unit=
 millisecond
HighVV: [Input] Samples to start frequency update (High Frequency Mode)

Return Value:

0: OK
The others: Error codes

Example:

```
int slot=1;  
LowTT=10000; //set time period to 10 seconds for more low frequency  
HighTT=50; //set time period to 50 mS for more high update rate  
i8080_SetFreqConfiguration(slot,AutoTT,AutoVV,LowTT,LowVV,HighTT,HighVV);
```

Remark:

This function can be applied on module: i8080.

2.11 FRnet Communication Functions

■ 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 Wincon-8000 system slot (Range: 1 to 7)
Port: [Input] Specify the FRnet port of i8172 (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: i8172.

■ 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 Wincon-8000 system slot (Range: 1 to 7)
Port: [Input] Specify the FRnet port of i8172 (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: i8172.

■ i8172_FRNET_Status

Description:

Read FRnet digital input status.

Syntax:

```
[C++]  
BYTE i8172_FRNET_Status(int Slot, BYTE Port)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)

Port: [Input] Specify the FRnet port of i8172 (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: i8172.

■ i8172_FRNET_Reset

Description:

Reset FRnet digital output status.

Syntax:

```
[C++]  
void i8172_FRNET_Reset(int Slot, BYTE Port)
```

Parameter:

Slot : [Input] Specify the Wincon-8000 system slot (Range: 1 to 7)
Port: [Input] Specify the FRnet port of i8172 (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: i8172.

2.11 X SRAM Access Functions

SRAM function for WinConSDK support the S-256/S-512 on the WinCon backplane Xsocket. The S-256/S-512 have the data block and offset address. The S-256 have 2048 blocks(0~2047) and S-512 have 4096 blocks(0~4095). Every block have 128 offset addresses. The S-256 have 2048*128=256KB SRAM and S-256 have 4096*128=1024KB SRAM. Before you use the function in this section, you must call the XSRAM_Init() function first.

XSRAM_Init

Description:

This function is used to initial the S256/S512 on the WinCon BackPlane Xsloket.

Syntax:

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

Parameter:

None :

Return Value:

256 S-256 on WinCon BackPlane Xsloket.
512 S-512 on WinCon BackPlane Xsloket.
0 None S-256/ S-512 on WinCon BackPlane Xsloket.

Example:

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

Remark:

This function can be applied S-256/ S-512 on WinCon BackPlane Xsloket.

XSRAM_Set_Block(WORD Block);

Description:

This function is used to set the current Block number of S-256/S512 on the WinCon BackPlane Xsloket.

Syntax:

```
[C++]  
  
bool XSRAM_Set_Block(WORD Block);
```

Parameter:

Block : [Input] current Block number of S-256/S512.
For S-256 Block is 0~2047
For S-512 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 S-256/ S-512 on WinCon BackPlane Xsloket.

XSRAM_Write_Byte

Description:

This function is used to write a byte to S-256/S512' 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 S-256/ S-512 on WinCon BackPlane Xsocket.

XSRAM_Read_Byte

Description:

This function is used to read a byte from S-256/S512' offset address on Current Block.

Syntax:

```
[C++]  
void 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:

Byte 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 S-256/ S-512 on WinCon BackPlane Xsloket.

XSRAM_Get_Type

Description:

This function is used to get the S-256/S512 type on WinCon BackPlane Xsloket.

Syntax:

```
[C++]  
WORD XSRAM_Get_Type();
```

Parameter:

None

Return Value:

256	There have S-256 on WinCon BackPlane Xsloket.
512	There have S-512 on WinCon BackPlane Xsloket.
0	There don't have S-256/512 on WinCon BackPlane Xsloket.

Example:

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

Remark:

This function can be applied S-256/ S-512 on WinCon BackPlane Xsloket.

XSRAM_Get_Max_Block

Description:

This function is used to get the S-256/S512 Max_Block on WinCon BackPlane Xsloket.

Syntax:

```
[C++]  
WORD XSRAM_Get_Max_Block ();
```

Parameter:

None

Return Value:

2048 There have 2048 Blocks on S-256.
4096 There have 4096 Blocks on S-512.
0 There don't have S-256/512 on WinCon BackPlane Xsloket.

Example:

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

Remark:

This function can be applied S-256/ S-512 on WinCon BackPlane Xsloket.

3. Using Wincon-8000 Serial Ports

This section describes how to use the three serial ports (RS-232/RS-485 interface) on the Wincon-8000 embedded controller (see figure). The information in this section is organized as follows:

- COM1 Port
- COM2 Port
- COM3 Port

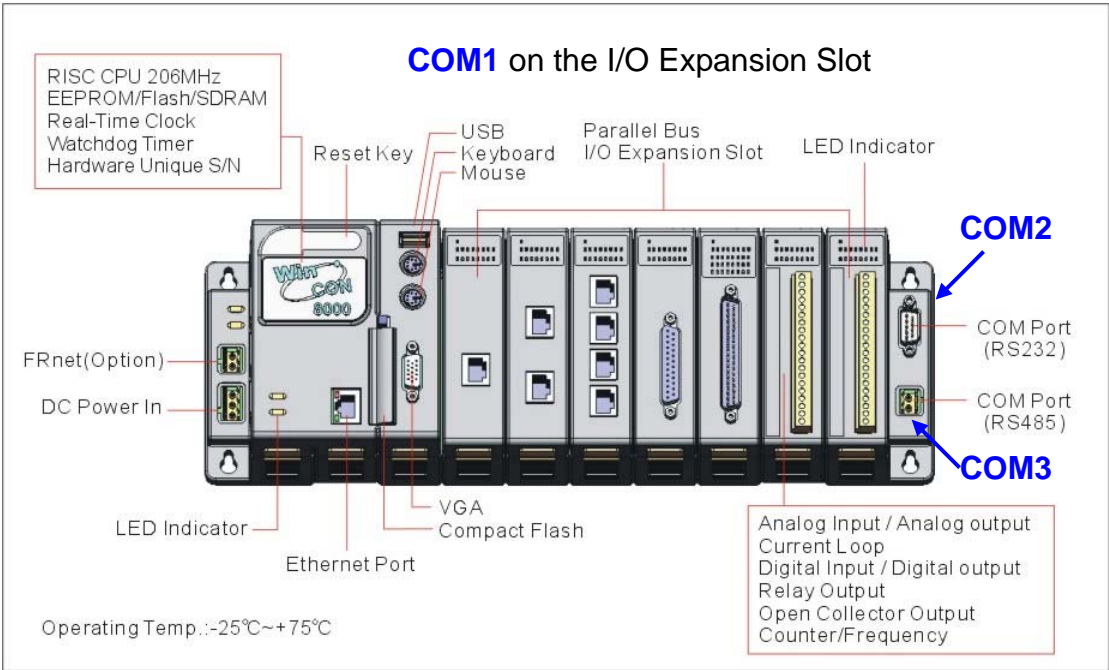


Fig. 7-1

3.1 Serial Port

COM1 Port

The COM1 port is located on the Wincon-8000 I/O expansion slot. This port is used to connect the I-87K series module plugged into the Wincon-8000 embedded controller. Users must use the serial command to control the I-87K series module. For controlling the I-87K module, you must input the Com-port parameters and call the Open_Com function to open the com1 port based on the appropriate settings. Finally, the user can call the ChangeSlotTo87K(slot) function in order to change the control slot. This is like the serial address, you can then send out the control commands to the I/O module, which is plugged into this slot, so that the module's serial address for its slot is 0. A detailed example is provided below:

For Example:

```
int slot=1;
unsigned char port=1;
DWORD baudrate=115200; //for all modules
char data=8;
char parity=0;
char stopbit=1;
Open_Com(port, baudrate, data, parity, stopbit);
ChangeSlotTo87K(slot);
//send command...
```

COM2 Port

This COM2 port is located on the right-upper corner on the Wincon-8000. It is a standard RS-232 serial port, and it provides TXD, RXD, RTS, CTS, GND, non-isolated and a maximum speed of 115.2K bps. It can also connect to the I-7520 module in order to provide a general RS-485 communication. The COM2 port can also connect to a wireless modem so that it can be controlled from a remote device. The application example and code is demonstrated below.

```
int slot=1;
unsigned char port=2;
DWORD baudrate=9600;
char data=8;
```

```

char parity=0;
char stopbit=1;
Open_Com(port, baudrate, data, parity, stopbit);
//send command...

```



Fig. 7-2

COM3 Port

This COM3 port provides RS-485 serial communication (DATA+ and DATA-) and is located on bottom-right corner on the Wincon-8000. You can connect to the RS-485 device with modules like the I-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. ICP DAS will provide very easy to use functions with a LIB driver and then you can easily handle the I-7000/I-87K/I-8000 series modules. Below is an example of the program code demo.

```

int slot=1;
unsigned char port=3;
DWORD baudrate=9600;
char data=8;
char parity=0;

```

```
char stopbit=1;  
Open_Com(port, baudrate, data, parity, stopbit);  
  
//send command
```

3.2 LIB Architecture of the Serial Port

The UARTCE.LIB, I7000CE.LIB are library files that are designed for applications that run on the Wincon-8000 main controller unit and can control the remote modules (I-7000/I-8000/I-87K series modules) through a serial port using Windows CE.NET. The user can apply them to develop their own applications with many development tools such as eMbedded Visual C++. For your convenience, there are many demo programs are provided for EVC++. Based on the demo programs provided, users can easily understand how to use the functions and develop their own applications quickly.

3.3 WinCon Serial Port Applications in eMbedded VC++

The UARTCE_LIB and I7000CE_LIB are very powerful library tools designed specifically for the development of user applications within Wincon-8000 embedded controllers. The directory settings found in the options for eMbedded Visual C++, specify search paths for files in your projects.

The paths are set in the following steps:

1. Open the “**Options**” dialog in the “**Tools**” menu.
2. In the “**Directories**” tab, select the “**STANDARDSDK**” in the “**Platform**” item, the “**Win32 [WCE ARMV4]**” in the “**CPUs**” item , and the “**Show directories**” in the “**include files**” item.
3. To add in the include files path, double-click the blank line (indicated by an empty rectangle) on the bottom line under “**Directories**” and set the full path for the included files as shown in the following figure. This should be defined as the include files path that the user has installed the driver into.

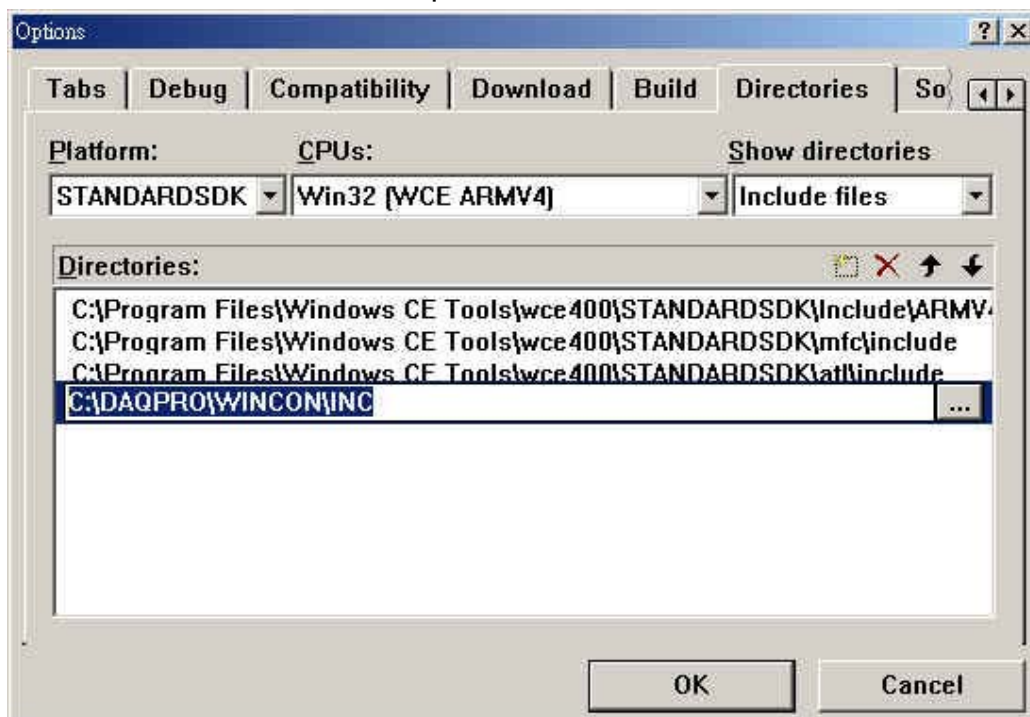


Fig. 7-3

4. Then, select “**Library files**” in the “**Show directories**” item.
5. To add in the library files path, double-click the blank line (indicated by an empty rectangle) on the bottom line under “**Directories**” and type the full path

in as is shown in following figure. This should be defined as the path for the Lib files that the user has installed the driver into.

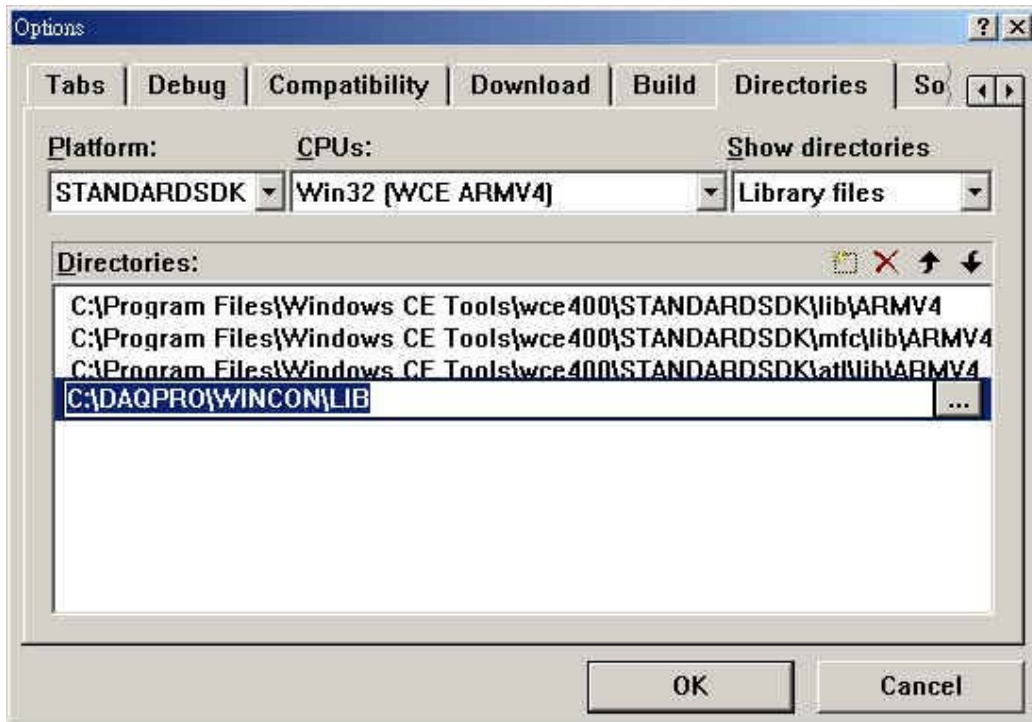


Fig. 7-4

Users must include the “UARTCE.H” and “I7000CE.H” files into the eMbedded Visual C++ user source code development environment. When administering the project settings, users must add the “UARTCE.lib” and “I7000CE.lib” into the object/library modules EditBox found on the Link tab (as following figure).

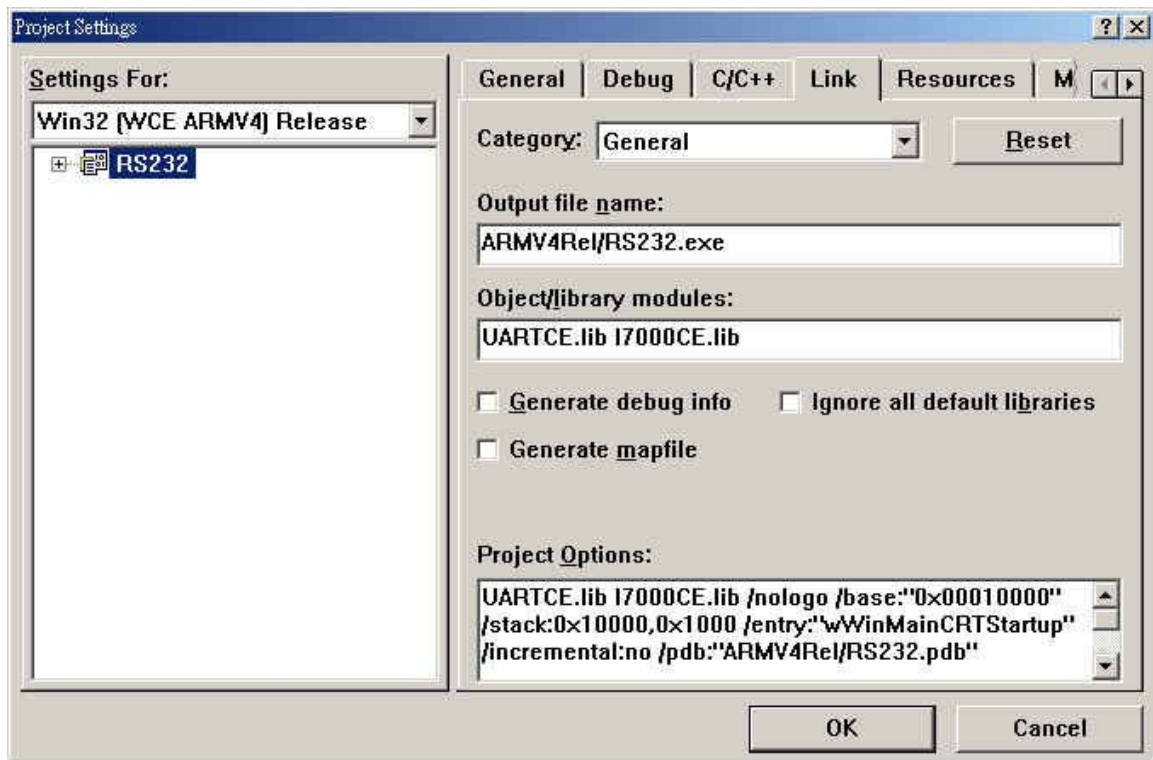


Fig. 7-5

Refer to “getting started” of this manual for explanations on how to create a new project for EVC++. For more detailed information on the UARTCE.lib and I7000CE.lib all functions.

3.3.1 Example List for the Reference of User Program Design

The main idea on how to implement the first demo program is depicted in the above section. For the easy design of different applications, several demo programs are provided for the users reference in the eMbedded Visual C++ and Wincon-8000 application platforms. These demo programs are listed below.

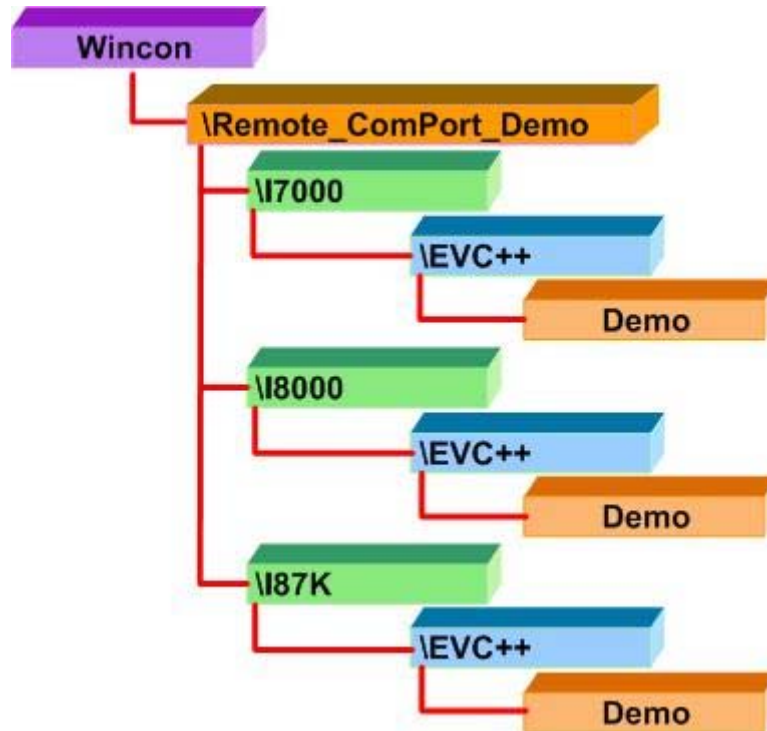


Fig. 7-6

Remote I7000 Demo for EVC++:

This demo program can be applied to the following remote modules: I-7060, I-7012 and I-7021 modules. They can be applied through the RS-485 network to connect to the Wincon-8000 embedded controller (see figure 7-7). Users need to set their COM port configurations to connect to their remote modules, and they need to set the RS-485 address for each module. Click on the “Open COM” button to open the WinCon-8000 embedded controller COM port. Check the DO0~DO3 checkbox to output a digital value for the I-7060 module. Type in a 0~10 float value and click the “Analog Output I-7021” button to output an analog value for the I-7021 module. Click on the “Auto Scan DI/AI Enable” button to read the digital input value for the I-7060 module and to simultaneously get the analog input value for the I-7012 module.

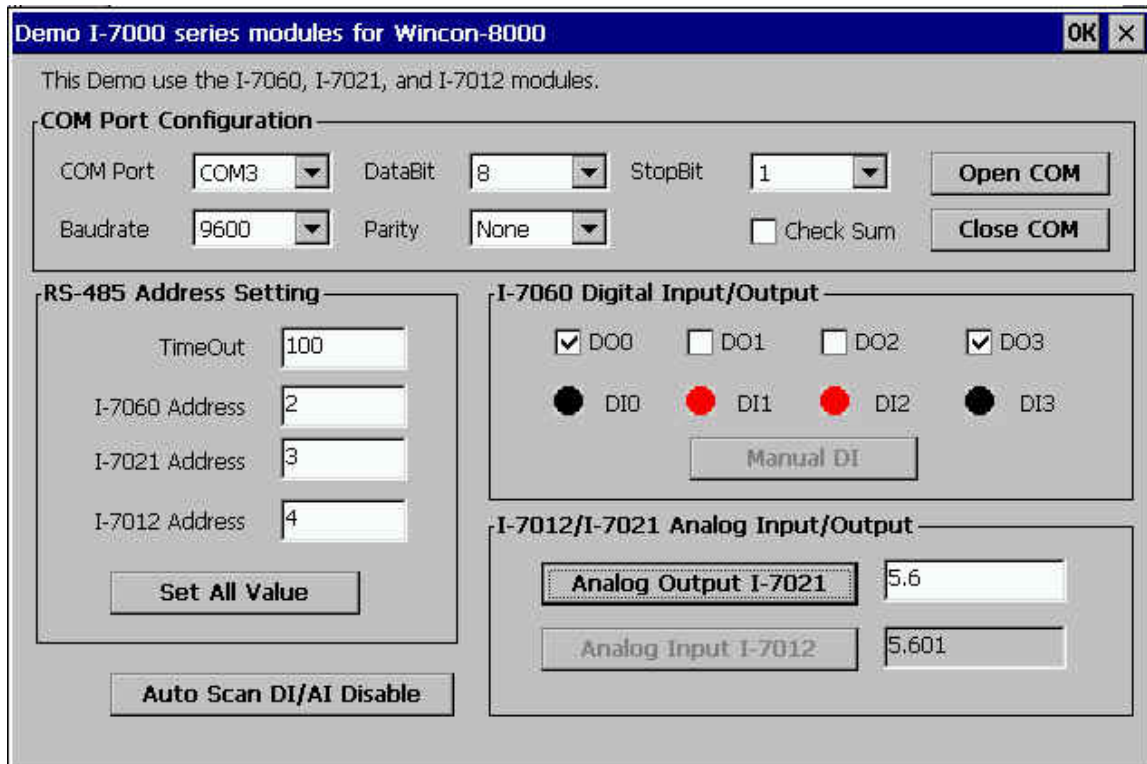


Fig. 7-7

Remote I8000 Demo for EVC++:

This demo program can be applied to either of these remote I-8053, I-8057, I-8017H, I-8024 modules within an I-8000 embedded controller. This demo can be used through the RS-485 network to connect to a Wincon-8000 embedded controller (see figure 7-8). Users need to set their COM port configurations for connections to their remote modules, and to set the RS-485 Net ID for their I-8000 embedded controller. Select a slot for the module to plug into in the I-8000 embedded controller. Click the “Open COM” button to open the WinCon-8000 embedded controller COM port. Check the DO0~DO15 checkbox to output a digital value from the I-8057 module. Type in a 0~10 float value into the channel0~4 Edit Box , then click the “Analog Output” button to output an analog value from the I-8024 module. Click the “Manual DI” button to read the digital input value from the I-8053 module, then click the “Manual AI” to read the input analog value from the I-8017H module.

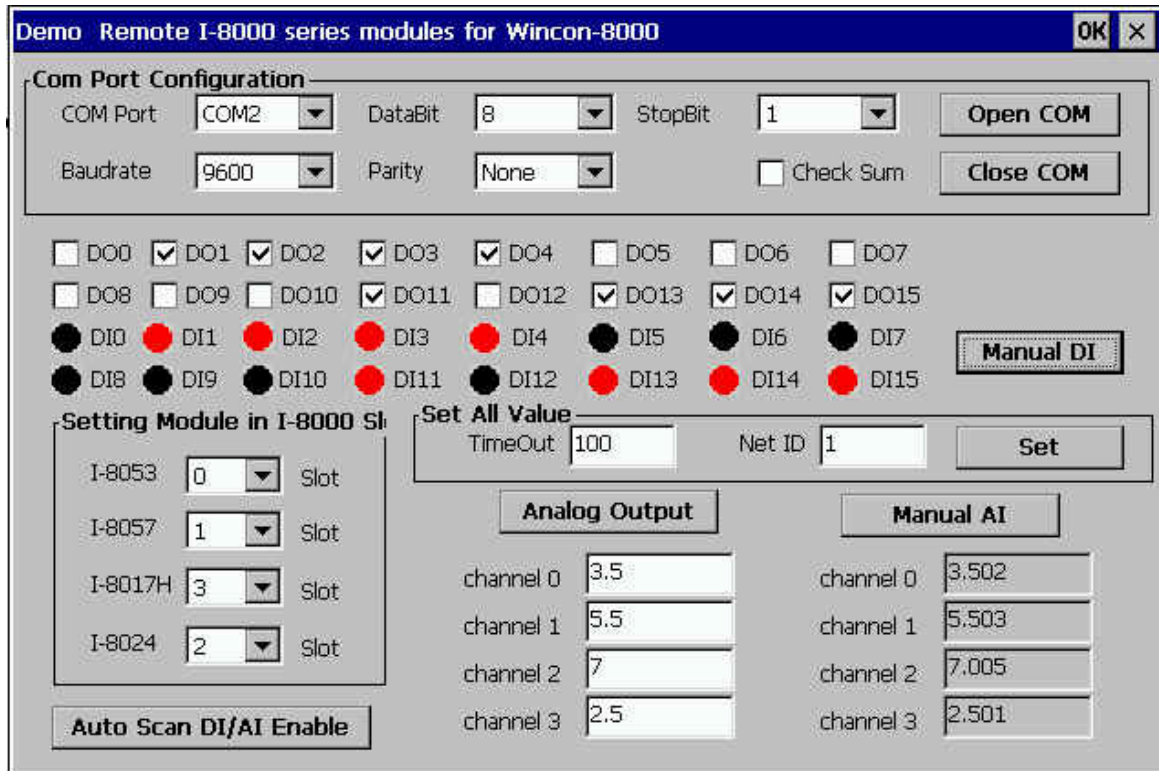


Fig. 7-8

Remote I87K Demo for EVC++:

This demo program can be applied to either of these remote I-87055, I-87017 and I-87024 modules within the I-87K I/O expansion units. This demo can be used through the RS-485 network to connect to the Wincon-8000 embedded controller (see figure 7-9). Users need to set the COM port configurations for their connections to their remote modules, and set the RS-485 address for each module. Click the “Open COM” button to open the WinCon-8000 embedded controller COM port. Check the DO0~DO7 checkbox to output a digital value from the I-87055 module. Type in a 0~10 float value and click the “Analog Output” button to output an analog value for the I-87024 module. Click the “Auto Scan DI/AI Enable” button to read the digital input value for the I-87055 module and to get an analog input value for the I-87017 module.

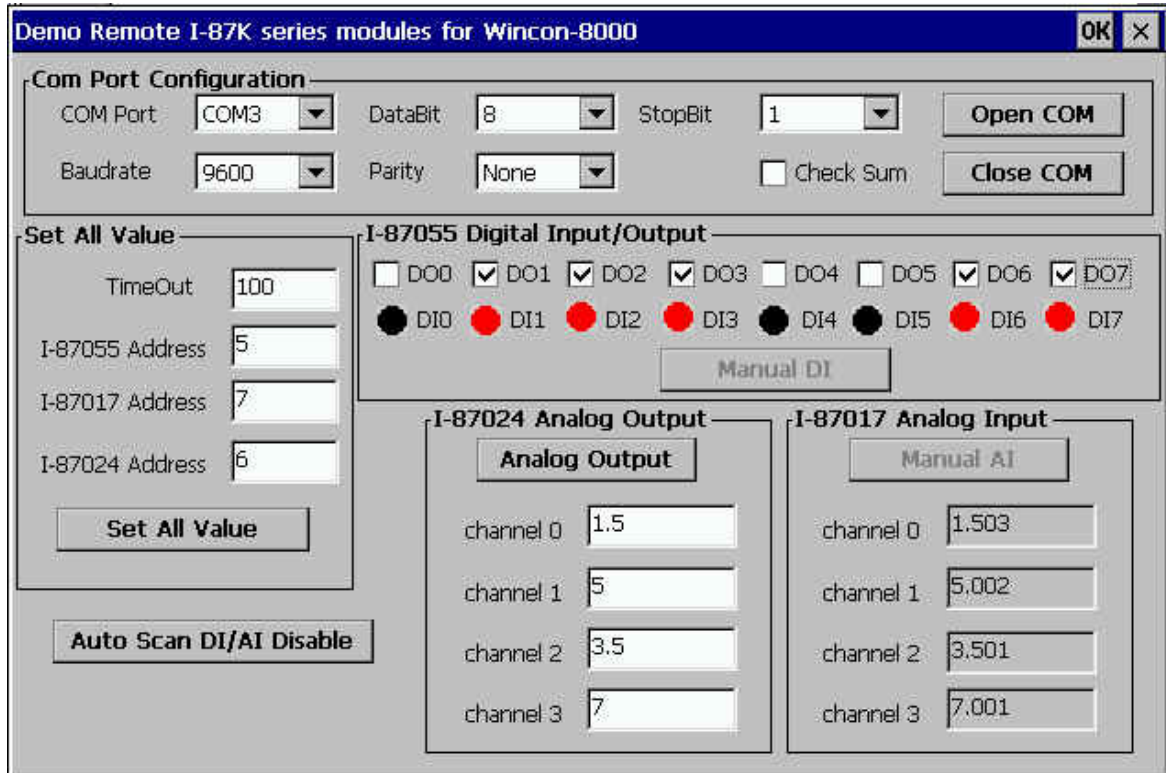


Fig. 7-9

4. UARTCE.LIB

■ Get_Uart_Version

Description:

The function is used to obtain the version information.

Syntax:

```
WORD Get_Uart_Version(void) [C++]
```

Parameter:

None

Return Value:

Version number ==> 0x0205

Example:

```
WORD Version;  
Version= Get_Uart_Version ();//Version= 0x0205  
//For example version 2.0.5
```

Remark:

■ 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++]  
WORD Open_Com(unsigned char cPort, DWORD dwBaudrate, char cData,  
              char cParity, char cStop)
```

Parameter:

cPort: [Input] 1=COM1, 2=COM2 , 255=COM255
dwBaudRate:[Input]150/300/600/1200/1800/2400/4800/7200/9600/
19200/38400/57600/115200/230400/460800/921600
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

NOTE About the “Error Code” please refer to “Appendix B Error Code”.

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000 slot is 1  
DWORD m_baudrate=9600;  
char m_databit=8;  
char m_parity =0;  
char m_stopbit =0;  
Open_Com(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
```

Remark:

■ 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++]  
BOOL Close_Com(unsigned char cPort)
```

Parameter:

cPort : [Input] 1=COM1, 2=COM2 , 255=COM25

Return Value:

NoError : OK

Example:

```
unsigned char m_ComPort= 3;           //Com_port of Wincon-8000  
Close_Com(m_ComPort);
```

Remark:

■ 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++]

```
WORD Send_Receive_Cmd (unsigned char cPort, char szCmd[], char szResult[],  
WORD wTimeOut, WORD wChecksum, WORD *wT)
```

Parameter:

cPort:	[Input] 1=COM1,2=COM2,3=COM3,4=COM4...255=COM255
szCmd:	[Input] 1024Bytes maximum, without zero (0x0D) character
szResult:	[Input] 1024Bytes maximum, with one zero or 0x0D terminal character
wTimeOut:	[Input] Communicating timeout setting, time unit = 1ms
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

Return Value:

NoError :	OK
Others :	Error code

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000  
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;
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
Send_Receive_Cmd(m_ComPort, m_szSend, m_szReceive, m_TimeOut, m_Checksum, &m_wT);
Close_Com(m_ComPort); //m_szReceive <- "!017016D"
```

Remark:

■ Send_Receive_Cmd_WithChar

Description:

It is similar to the Send_Receive_Cmd except cEndChar parameter

Syntax:

```
[C++]  
WORD Send_Receive_Cmd_WithChar (unsigned char cPort, char szCmd[], char  
szResult[], WORD wTimeout, WORD wChecksum, WORD *wT,  
unsigned char cEndChar)
```

Parameter:

cPort:	[Input] 1=COM1,2=COM2,3=COM3,4=COM4...255=COM255
szCmd:	[Input] 1024Bytes maximum
szResult:	[Input] 1024Bytes maximum
wTimeout:	[Input] Communicating timeout setting, time unit = 1ms
wChecksum:	[Input] 0 ==> None <>0 ==> add two check sum bytes
*wT:	[Output] return a reference number for identify the performance. *wT --> 0 :good
cEndChar:	[Input] the last character

Return Value:

NoError :	OK
Others :	Error code

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000  
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;
```

```
unsigned char m_cEndChar=0x0D;
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
Send_Receive_Cmd_WithChar(m_ComPort, m_szSend, m_szReceive, m_TimeOut, m_Checksum,
&m_wT,m_cEndChar);
Close_Com(m_ComPort); //m_szReceive <- "!017016D"
```

Remark:

■ 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 stopped by space character. For example: "\$01M 02 03" is user's command string. However, the actual command sent out is "\$01M".

Syntax:

```
[C++]  
WORD Send_Cmd (unsigned char cPort, char szCmd[], WORD wTimeOut,  
WORD wChecksum)
```

Parameter:

cPort:	[Input] 1=COM1,2=COM2,3=COM3,4=COM4...255=COM255
szCmd:	[Input] Sending command string(Terminated with "0")
wTimeOut:	[Input] Communicating timeout setting, time unit = 1ms
wChecksum:	[Input] 0=DISABLE, 1=ENABLE

Return Value:

NoError :	OK
Others :	Error code

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000  
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;  
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" ,!7016D  
Send_Cmd(m_ComPort, m_szSend, m_TimeOut, m_Checksum);  
Close_Com(m_ComPort);
```

Remark:

■ Send_Cmd_WithChar

Description:

It is similar to the Send_Cmd except cEndChar parameter

Syntax:

```
[C++]  
WORD Send_Cmd_WithChar (unsigned char cPort, char szCmd[], WORD  
wTimeOut, WORD wCheckSum, unsigned char cEndChar)
```

Parameter:

cPort: [Input] 1=COM1,2=COM2,3=COM3,4=COM4...255=COM255
szCmd: [Input] Sending command string(Terminated with "0")
wTimeOut: [Input] Communicating timeout setting, time unit = 1ms
wCheckSum: [Input] 0=DISABLE, 1=ENABLE
cEndChar: [Input] the last character

Return Value:

NoError : OK
Others : Error code

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000  
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;  
unsigned char m_cEndChar=0x0D;  
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",!7016D  
Send_Cmd_WithChar(m_ComPort, m_szSend, m_TimeOut, m_Checksum, m_cEndChar);  
Close_Com(m_ComPort);
```

Remark:

■ Receive_Cmd

Description:

Users 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++]

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

Parameter:

Cport::	[Input] 1=COM1,2=COM2,3=COM3,4=COM4...255=COM255
szResult:	[Output] Receiving the response string from the modules
wTimeOut::	[Input] Communicating timeout setting, time unit = 1ms
wCheckSum:	[Input] 0=DISABLE, 1=ENABLE
*wT:	[Output] Total time of receiving interval, unit = 1 ms.

Return Value:

NoError :	OK
Others :	Error code

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000
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;
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
```

```
Send_Cmd(m_ComPort, m_szSend, m_Checksum);  
Receive_Cmd(m_ComPort, m_szReceive, m_TimeOut, m_Checksum, &m_wT);  
Close_Com(m_ComPort);           //m_szReceive <- "!017016D"
```

Remark:

■ Receive_Cmd_WithChar

Description:

It is similar to the Receive_Cmd except cEndChar parameter

Syntax:

```
[C++]  
WORD Receive_Cmd_WithChar (unsigned char cPort, char szResult[], WORD  
wTimeOut, WORD wChksum, WORD *wT, unsigned char  
cEndChar)
```

Parameter:

Cport::	[Input] 1=COM1,2=COM2,3=COM3,4=COM4...255=COM255
szResult:	[Output] Receiving the response string from the modules
wTimeOut::	[Input] Communicating timeout setting, time unit = 1ms
wCheckSum:	[Input] 0=DISABLE, 1=ENABLE
*wT:	[Output] Total time of receiving interval, unit = 1 ms.
cEndChar:	[Input] the last character

Return Value:

NoError :	OK
Others :	Error code

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000  
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;  
unsigned char m_cEndChar=0x0D;  
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
```

```
Send_Cmd(m_ComPort, m_szSend, m_Checksum);  
Receive_Cmd_WithChar(m_ComPort, m_szReceive, m_TimeOut, m_Checksum, &m_wT, m_cEndChar);  
Close_Com(m_ComPort);           //m_szReceive <- "!017016D"
```

Remark:

■ Send_Binary

Description:

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

Syntax:

```
                                [C++]  
WORD Send_Binary (unsigned char cPort, char szCmd[], int iLen)  
;
```

Parameter:

cPort:	[Input] 1=COM1,2=COM2,3=COM3,4=COM4 ...255=COM255
szCmd:	[Input] Sending command string(Terminated with "0")
iLen :	[Input] The length of command string.

Return Value:

NoError :	OK
Others :	Error code

Example:

```
unsigned char m_ComPort=3;    //Com_port of Wincon-8000  
DWORD m_baudrate=9600;  
char m_databit=8;  
char m_parity=0;  
char m_stopbit=0;  
char m_szSend[100];  
int m_length=10;  
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  
Close_Com(m_ComPort);
```

Remark:

■ Receive_Binary

Description:

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

Syntax:

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

Input Parameter:

cPort:	[Input] 1=COM1,2=COM2,3=COM3,4=COM4 ...255=COM255
szResult:	[Output] The receiving string from the module
wTimeOut:	[Input] Communicating timeout setting, time unit = 1ms
wLen:	[Input] The length of result string.
*wT:	[Output] Total time of receiving interval, unit = 1 ms.

Return Value:

NoError :	OK
Others :	Error code

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000  
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;
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
Receive_Binary(m_ComPort , m_szReceive, m_TimeOut, m_wlength, &m_wT);
Close_Com(m_ComPort); //Receive 10 character
```

Remark:

■ Get_Com_Status

Description:

The function can obtain COM Port status. If return value is “0” (false), it means “The COM Port is not in used!”. Otherwise, if the return value is “1” (true), it means “The COM Port is in used!”

Syntax:

```
[C++]  
WORD Get_Com_Status (unsigned char cPort)
```

Parameter:

cPort: [Input] 1=COM1,2=COM2,3=COM3,4=COM4... 255=COM255

Return Value:

0 : COM port is not in used.
1 : COM port is in used.

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000  
DWORD m_baudrate=9600;  
char m_databit=8;  
char m_parity=0;  
char m_stopbit=0;  
WORD t1,t2;  
t1= Get_Com_Status(m_ComPort); //t1= 0  
Open_Com(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);  
t2= Get_Com_Status(m_ComPort); //t2= 1  
Close_Com(m_ComPort);
```

Remark:

■ Change_BaudRate

Description:

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

Syntax:

```
[C++]  
WORD Change_BaudRate (unsigned char cPort, DWORD dwBaudrate)
```

Parameter:

cPort :	[Input] 1=COM1,2=COM2,3=COM3,4=COM4... 255=COM255
dwBaudrate:	[Input]150/300/600/1200/1800/2400/4800/7200/9600/ 19200/38400/57600/115200/230400/460800/921600

Return Value:

NoError :	OK
Others :	Error code

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000  
DWORD m_baudrate=9600;  
Change_BaudRate(m_ComPort, m_baudrate);
```

Remark:

■ Change_Config

Description:

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

Syntax:

[C++]

WORD Change_Config (**unsigned char** cPort, DWORD dwBaudrate, **char** cData, **char** cParity, **char** cStop)

Parameter:

cPort:	[Input] 1=COM1, 2=COM2 , 255=COM255
dwBaudRate:	[Input] 50/75/110/134.5/150/300/600/1200/1800/2400/4800/ 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

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000
DWORD m_baudrate=9600;
char m_databit=8;
char m_parity=0;
char m_stopbit=0;
Change_Config(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
```

Remark:

■ ReadComn

Description:

This function is applied to receive the multi-bytes response from COM port

Syntax:

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

Parameter:

Cport::	[Input] 1=COM1,2=COM2,3=COM3,4=COM4...255=COM255
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:

The number of bytes actually read

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000  
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;  
WORD 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:

■ DataSizeInCom

Description:

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

Syntax:

[C++]
DWORD DataSizeInCom (int port)

Parameter:

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

Return Value:

Com Port : buffer size

Example:

```
int port=3; //Com_port of Wincon-8000
DWORD Size;
Size= DataSizeInCom(port);
```

Remark:

■ DataSizeOutCom

Description:

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

Syntax:

[C++] DWORD DataSizeOutCom (int port)
--

Parameter:

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

Return Value:

Com Port : buffer size

Example:

```
int port=3; //Com_port of Wincon-8000  
DWORD Size;  
Size= DataSizeOutCom(port);
```

Remark:

■ SetLineStatus

Description:

Set the Line Status (DTR,RTS).

Syntax:

[C++]
WORD SetLineStatus (int port, int pin, int mode)

Parameter:

port:	[Input] 1=COM1, 2=COM2 , 255=COM255
pin:	[Input] 1: DTR, 2: RTS, 3: DTR+RTS
mode:	[Input] 0: Disable, 1: Enable, 2: HandShake

Return Value:

NoError :	0: OK
Others :	< 0: Error

Example:

```
int port=3; //Com_port of Wincon-8000
int pin=3;
int mode=1;
SetLineStatus(port, pin, mode);
```

Remark:

■ GetLineStatus

Description:

Get the Line Status (DTR,RTS).

Syntax:

[C++]
WORD GetLineStatus (int port, int pin)

Parameter:

port:	[Input] 1=COM1, 2=COM2 , 255=COM255
pin:	[Input] 0: CTS, 1: DSR, 2: RI, 3: CD

Return Value:

1 :	ON
0 :	OFF

Example:

```
int port=3; //Com_port of Wincon-8000
int pin=0;
WORD Status;
Status= GetLineStatus(port, pin);
```

Remark:

■ Set_FlowControl

Description:

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

Syntax:

[C++]

WORD Set_FlowControl (unsigned char cPort, int DCB_Member, int mode)

Parameter:

cPort : [Input] 2=COM2, 3=COM3

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

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000
DWORD m_baudrate=9600;
char m_databit=8;
char m_parity=0;
char m_stopbit=0;
char m_szSend[100];
char m_szReceive[100];
Open_Com(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
Set_FlowControl(m_ComPort, 3, 1);
Close_Com(m_ComPort);
```

Remark:

Software flow control:

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

Hardware flow control:

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

■ Get_FlowControl

Description:

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

Syntax:

```
[C++]  
WORD Get_FlowControl (unsigned char cPort, int DCB_Member)
```

Parameter:

cPort : [Input] 2=COM2, 3=COM3
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

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000  
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 Mode;
Open_Com(m_ComPort, m_baudrate, m_databit, m_parity, m_stopbit);
Mode =Get_FlowControl(m_ComPort, 3);
Close_Com(m_ComPort);
```

Remark:

Software flow control:

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

Hardware flow control:

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

■ Set_Break

Description:

This function is used to send the Break signal for the COM port.

Syntax:

```
[C++]  
WORD Set_Break (unsigned char cPort, unsigned char cAction)
```

Parameter:

cPort : [Input] 2=COM2, 3=COM3
cAction: [Input] 0=clear break, 1=set break

Return Value:

0: NO_ERROR
Others: Error code

Example:

```
unsigned char m_ComPort= 3;           //Com_port of Wincon-8000  
Set_Break(m_ComPort);
```

Remark:

■ ModbusGetCRC16

Description:

This function is used to Get the 16-bit CRC from the CRC table.

Syntax:

```
[C++]  
WORD ModbusGetCRC16(unsigned char puchMsg[], unsigned char CRC[], int  
DataLen)
```

Parameter:

puchMsg[] :

CRC[]:

DataLen: data length

Return Value:

0: NO_ERROR

Others: Error code

Example:

Remark:

■ Send_Receive_Binary

Description:

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

Syntax:

```
[C++]  
  
WORD Send_Receive_Binary(unsigned char cPort, char ByteCmd[],WORD  
in_Len,char ByteResult[], WORD out_Len, WORD wTimeOut)
```

Parameter:

cPort : [Input] 2=COM2, 3=COM3
ByteCmd[]: Pointer to data to be send
in_Len: Maximum number of bytes to send
ByteResult[]: Buffer into which received data will be stored
out_Len: Maximum number of bytes to be received
wTimeOut: Timeout for receiving result string Unit: ms

Return Value:

0: NO_ERROR
Others: Error code

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000  
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;  
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  
Send_Receive_Binary(m_ComPort, ByteCmd,in_Len,ByteResult, out_Len,wTimeout) //Send 10 character  
Close_Com(m_ComPort);
```

Remark:

■ Send_Receive_UserCmd

Description:

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

Syntax:

[C++]

```
WORD Send_Receive_UserCmd(unsigned char cPort, char szCmd[], char szResult[], WORD wTimeOut, WORD wChksum, WORD *wT, char sUserString[], int iUserLen)
```

Parameter:

cPort : [Input] 2=COM2, 3=COM3

szCmd[]: 1024 Bytes maximum, without zero (0x0D) character

szResult[]: 1024 Bytes maximum, with one zero or 0x0D terminal character

wTimeOut: Timeout for receiving result string

wChksum: 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: return a reference number for identify the performance
*wT --> 0 :good

sUserString: The user-defined terminal string

iUserLen: The length of the user-defined terminal string

Return Value:

0: NO_ERROR

8: CompPortNotOpen

-1: Vxc_ERR_Port_NO

-3: vxc_ERR_OPEN_COM

-4: vxc_ERR_BAD_PARAM

Example:

```
unsigned char m_ComPort=3; //Com_port of Wincon-8000
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 wTimeOut=100;
WORD m_wlength=10;
WORD m_wT;
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_Receive_UserCmd(m_ComPort, m_szSend ,m_szReceive, wTimeOut, wChecksum, &
m_wT,sUserString[],m_wlength)
Close_Com(m_ComPort); //Receive 10 character

```

Remark:

The length of szCmd and sUserString string cannot over 1024 bytes

■ Change_ParityErrorCheck

Description:

This functions is used to change the parity error check status.

Syntax:

[C++]

```
int Change_ParityErrorCheck(int port, int mode, char cErrorChar)
```

Parameter:

port : [Input] 2=COM2, 3=COM3
mode: [Input] 0=disable, 1=enable, default value=1
cErrorChar: Specifies the value of the character used to replace bytes received with a parity error.

Return Value:

0: vxc_NO_ERROR
-1: vxc_ERR_PORT_NO // Port Number Error
-2: vxc_ERR_WIN32_API // WIN32 API Call Error
-3: vxc_ERR_OPEN_COM

Example:

Remark:

5. I7000CE.LIB

The functions of I7000CE.LIB can be clarified as 8 groups as depicted as below:

1. Analog Input Functions;
2. Module Alarm Functions ;
3. Stain Gauge module Functions;
4. Analog Output Functions;
5. Digital Input Functions;
6. Digital Output Functions;
7. Counter module Functions;
8. Dual WatchDog and safety Functions.

The following sections will explore the more detail description for all functions.

■ Get_Dll_Version

Description:

Obtain the version information of I7000.LIB.

Syntax:

[C++]
WORD Get_Dll_Version (void)

Parameter:

Users don't need to provide any parameter.

Return Value:

Return the version message by hexadecimal format.

Example:

```
int version ;           //define "ver" is a integer Variable
version= Get_DLL_Version();
//If the return value is 0X508, it means "the version of I7000.LIB" is 5.0.8
```

Remark:

5.1 Analog Input Functions

5.1.1 I-7000 series modules

■ AnalogIn

Description:

Obtain the analog input value from DCON series modules.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/7012/7013/7014/7016/7017/ 7018/7033
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] Channel number for multi-channel
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive string
fBuf:	Float Input/Output argument table
fBuf[0]:	[Output] Analog input value return
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

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

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
```

```
WORD m_7016Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float AI;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]=m_7016Addr;
m_wBuf[2]=0x7016;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=0;
m_wBuf[6]=1; //watch NOTE
AnalogIn(m_wBuf, m_fBuf, m_szSend, m_szReceive);//m_szSend= "#02", m_szReceive= ">+001.97"
AI=m_fBuf[0]; //AI= 1.97
Close_Com(3);
```

Remark:

■ AnalogInHex

Description:

Obtain the analog input value in “Hexadecimal” format from I-7000 series modules.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/7012/7013/7014/7017/7018/7033
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] Channel number for multi-channel
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
wBuf[7]:	[Output] The analog input value in “Hexadecimal” format.
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

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

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7012Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];
float AI;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]=m_7012Addr;
m_wBuf[2]=0x7012;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=0;
m_wBuf[6]=1;
AnalogInHex(m_wBuf, m_fBuf, m_szSend, m_szReceive);
AI=m_wBuf[7]; //Hex format
Close_Com(3);
```

Remark:

■ AnalogInFsr

Description:

Obtain the analog input value in “FSR” format from I-7000 series modules. The “FSR” means “Percent” format.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/7012/7013/7014/7017/7018/7033
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] Channel number for multi-channel
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive string
fBuf:	Float Input/Output argument table
fBuf[0]:	[Output] Analog input value return
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

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

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7012Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];  
float AI;  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]=m_7012Addr;  
m_wBuf[2]=0x7012;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[5]=0;  
m_wBuf[6]=1;  
AnalogInFsr(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
AI=m_fBuf[0];  
Close_Com(3);
```

Remark:

■ AnalogInAll

Description:

Obtain analog input values of all channels from I-7017 or I-7018 or I-7033.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]	[Input] COM port number: 1/2/3/4.../255
wBuf[1]:	[Input] Module address: 0x00 to 0xFF
wBuf[2]	[Input] Module ID: 0x7017/18/33
wBuf[3]	[Input] Checksum: 0=disable, 1=enable
wBuf[4]	[Input] Time out setting, normal=100, unit=ms.
wBuf[6]	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0]	[Output] Analog input value of channel_0
fBuf[1]	[Output] Analog input value of channel_1
fBuf[2]	[Output] Analog input value of channel_2
fBuf[3]	[Output] Analog input value of channel_3
fBuf[4]	[Output] Analog input value of channel_4
fBuf[5]	[Output] Analog input value of channel_5
fBuf[6]	[Output] Analog input value of channel_6
fBuf[7]	[Output] Analog input value of channel_7
szSend:	[Input] Command string to be sent to I-7000 series modules
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7017Addr=1;
WORD m_TimeOut=100;
```

```
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float AI[12];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]=m_7017Addr;
m_wBuf[2]=0x7017;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=0;
m_wBuf[6]=1;
AnalogInAll(m_wBuf, m_fBuf, m_szSend, m_szReceive);
AI[0]=m_fBuf[0];
AI[1]=m_fBuf[1];
AI[2]=m_fBuf[2];
AI[3]=m_fBuf[3];
AI[4]=m_fBuf[4];
AI[5]=m_fBuf[5];
AI[6]=m_fBuf[6];
AI[7]=m_fBuf[7];
Close_Com(3);
```

Remark:

■ ThermocoupleOpen_7011

Description:

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

Syntax:

[C++]

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

Parameter:

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

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7011Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];
```

```
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
WORD state;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]=m_7011Addr;
m_wBuf[2]=0x7011;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[6]=1;
ThermocoupleOpen_7011(m_wBuf, m_fBuf, m_szSend, m_szReceive);
state=m_wBuf[5]; //7011 Thermocouple state
Close_Com(3);
```

Remark:

■ SetLedDisplay

Description:

Configure LED Display for specified channel of I-7033 or I-7016.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]	[Input] Module address: 0x00 to 0xFF
wBuf[2]:	[Input] Module ID; 0X7013, 0X7033, 0X7016;
wBuf[3]:	[Input] Checksum: 0=disable, 1=enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] Set display channel for 7013, 7033 or 7016
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7033Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
```

```
m_wBuf[1]=m_7033Addr;
m_wBuf[2]=0x7033;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=1;           //Set channel 1 display
m_wBuf[6]=1;
SetLedDisplay(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ GetLedDisplay

Description:

Get the current setting of the specified channel for LED Display channel in I-7033 or I-7016.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address: 0x00 to 0xFF
wBuf[2]:	[Input] Module ID 0x7033, 0x7016.
wBuf[3]:	[Input] Checksum: 0=disable, 1=enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Output] Current channel for LED display. 0=channel_0; 1=channel_1
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7033Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
```

```
WORD Led;  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]=m_7033Addr;  
m_wBuf[2]=0x7033;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[6]=1;  
GetLedDisplay(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Led=m_wBuf[5];           //Which channel display  
Close_Com(3);
```

Remark:

5.1.2 I-8000 series modules

■ AnalogIn_8K

Description:

Obtain the analog input value in float format from I-8000 series modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x8017h/0x8018/0x8013/0x8033
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The channel number of analog input module for 8013/17/18
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit.
fBuf:	Float Input/Output argument table
fBuf[0]:	[Output] Analog input value
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_NetID=1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;
```

```
DWORD m_8017hSlot=1;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float AI;
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]=m_NetID;
m_dwBuf[2]=0x8017h;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1; //Analog In ch1
m_dwBuf[6]=1;
m_dwBuf[7]=m_8017hSlot;
AnalogIn_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
AI=m_fBuf[0];
Close_Com(3);
```

Remark:

■ AnalogInHex_8K

Description:

Obtain the analog input value in “Hexadecimal” format from I-8000 series modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x8013/0x8017h/0x8018
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The channel number of analog input module for 8013/17/18
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive.
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit.
dwBuf[8]:	[Output] The analog input value in Hex format.
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_NetID=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_8017hSlot=1;
```

```
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
DWORD AI
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]=m_NetID;
m_dwBuf[2]=0x8017h;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1; //Analog In ch1
m_dwBuf[6]=1;
m_dwBuf[7]=m_8017hSlot;
AnalogInHex_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
AI=m_dwBuf[8];
Close_Com(3);
```

Remark:

■ AnalogInFsr_8K

Description:

Obtain the analog input value in “FSR” format from I-8000 series modules. The “FSR” means “Percent” format.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x8017/0x8018
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The channel number of analog input module for 8013/17/18
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive.
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit.
fBuf:	Float Input/Output argument table
fBuf[0]:	[Output] The analog input value.
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_NetID=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_8017hSlot=1;
```

```
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float AI;
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]=m_NetID;
m_dwBuf[2]=0x8017h;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1; //Analog In ch1
m_dwBuf[6]=1;
m_dwBuf[7]=m_8017hSlot;
AnalogInFsr_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
AI=m_fBuf[0];
Close_Com(3);
```

Remark:

■ AnalogInAll_8K

Description:

Obtain the analog input values of all channels from I-8013, I-8017 or I-8018.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x8013/8017/8018
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit.
fBuf:	Float Input/Output argument table
fBuf[0]	[Output] Analog input value of channel_0
fBuf[1]	[Output] Analog input value of channel_1
fBuf[2]	[Output] Analog input value of channel_2
fBuf[3]	[Output] Analog input value of channel_3
fBuf[4]	[Output] Analog input value of channel_4
fBuf[5]	[Output] Analog input value of channel_5
fBuf[6]	[Output] Analog input value of channel_6
fBuf[7]	[Output] Analog input value of channel_7
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
```

```
DWORD m_NetID=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_8017hSlot=1;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float AI[12];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_NetID;
m_dwBuf[2]=0x8017h;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8017hSlot;
AnalogInAll_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
AI[0]=m_fBuf[0];
AI[1]=m_fBuf[1];
AI[2]=m_fBuf[2];
AI[3]=m_fBuf[3];
AI[4]=m_fBuf[4];
AI[5]=m_fBuf[5];
AI[6]=m_fBuf[6];
AI[7]=m_fBuf[7];
Close_Com(3);
```

Remark:

5.1.3 I-87K series modules

■ AnalogIn_87K

Description:

Obtain the analog input value from I-87K series analog input modules, for example: I-87017 or I-87018.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x87013/0x87017/0x87018
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] Channel number of analog input module for 87013/17/18/33
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive.
fBuf:	Float Input/Output argument table
fBuf[0]:	[Output] The analog input value return
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87017Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
```

```
DWORD m_dwBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
float AI;  
Open_Com(3, 9600, 8, 0, 0);  
m_dwBuf[0]=m_ComPort;  
m_dwBuf[1]= m_87017Addr;  
m_dwBuf[2]=0x87017;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[6]=1;  
AnalogIn_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
AI=m_fBuf[0];  
Close_Com(3);
```

Remark:

■ AnalogInHex_87K

Description:

Obtain the analog input value in “Hexadecimal” format from I-87K series analog input modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x87013/87017/87018
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The channel number of analog input module for 87013/17/18
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Output] The analog input value in “Hex” format.
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87017Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];  
DWORD AI;  
Open_Com(3, 9600, 8, 0, 0);  
m_dwBuf[0]=m_ComPort;  
m_dwBuf[1]= m_87017Addr;  
m_dwBuf[2]=0x87017;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[6]=1;  
AnalogInHex_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
AI=m_dwBuf[7];  
Close_Com(3);
```

Remark:

■ AnalogInFsr_87K

Description:

Obtain the analog input value in “FSR” format from I-87K series analog input modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x87013/87017/87018
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The channel number of analog input module for 87013/17/18
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0]:	[Output] The analog input value.
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87017Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];  
float AI;  
Open_Com(3, 9600, 8, 0, 0);  
m_dwBuf[0]=m_ComPort;  
m_dwBuf[1]= m_87017Addr;  
m_dwBuf[2]=0x87017;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[6]=1;  
AnalogInFsr_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
AI=m_fBuf[0];  
Close_Com(3);
```

Remark:

■ AnalogInAll_87K

Description:

Obtain the analog input values of all channels from I-87013, I-87017 and I-87018.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x87013/0x87017/87018
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0]	[Output] Analog input value of channel_0
fBuf[1]	[Output] Analog input value of channel_1
fBuf[2]	[Output] Analog input value of channel_2
fBuf[3]	[Output] Analog input value of channel_3
fBuf[4]	[Output] Analog input value of channel_4
fBuf[5]	[Output] Analog input value of channel_5
fBuf[6]	[Output] Analog input value of channel_6
fBuf[7]	[Output] Analog input value of channel_7
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_87017Addr=1;  
DWORD m_TimeOut=100;
```

```
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float AI[12];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87017Addr;
m_dwBuf[2]=0x87017;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[6]=1;
AnalogInAll_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
AI[0]=m_fBuf[0];
AI[1]=m_fBuf[1];
AI[2]=m_fBuf[2];
AI[3]=m_fBuf[3];
AI[4]=m_fBuf[4];
AI[5]=m_fBuf[5];
AI[6]=m_fBuf[6];
AI[7]=m_fBuf[7];
Close_Com(3);
```

Remark:

5.2 Module Alarm Functions

5.2.1 I-7000 series modules

■ EnableAlarm

Description:

Enable the alarm function of I-7000 series modules and configure it in the status of momentary alarm and latch alarm mode. This function currently supports I-7011, I-7012, I-7014, and I-7016.

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/7012/7014/7016
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] 0 → momentary alarm mode 1 → latch alarm mode
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7016Addr=1;  
WORD m_TimeOut=100;
```

```
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]=m_7016Addr;
m_wBuf[2]=0x7016;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=1;
m_wBuf[6]=1;
EnableAlarm(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ DisableAlarm

Description:

Disable alarm function of I-7000 series modules. This function currently supports I-7011, I-7012, I-7014, and I-7016.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/7012/7014/7016
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]	Not used
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7016Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
```

```
m_wBuf[0]=m_ComPort;  
m_wBuf[1]=m_7016Addr;  
m_wBuf[2]=0x7016;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[6]=1;  
DisableAlarm(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ ClearLatchAlarm

Description:

This function can clear the alarm latched of I-7000 series modules. This function currently supports I-7011, I-7012, I-7014, and I-7016.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/7012/7014/7016
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]	Not used
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7016Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
```

```
m_wBuf[0]=m_ComPort;  
m_wBuf[1]=m_7016Addr;  
m_wBuf[2]=0x7016;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[6]=1;  
ClearLatchAlarm(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ SetAlarmLimitValue

Description:

Set the high or low alarm limit value of I-7000 series modules. This function currently supports I-7011, I-7012, I-7014, and I-7016.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/0x7012/0x7014/0x7016
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] 0 → low alarm value setting 1 → high alarm value setting
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0]:	[Input] Alarm value
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7016Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];
float Alarm;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]=m_7016Addr;
m_wBuf[2]=0x7016;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=1;
m_wBuf[6]=1;
m_fBuf[0]= Alarm;
SetAlarmLimitValue(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ReadAlarmLimitValue

Description:

Obtain the high or low alarm limit value of I-7000 series modules. This function currently supports I-7011, I-7012, I-7014, and I-7016.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/0x7012/0x7014/7016.
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] 0 → low alarm value setting 1 → high alarm value setting
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0]:	[Output] Alarm value
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7016Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];
float Alarm;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]=m_7016Addr;
m_wBuf[2]=0x7016;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=1;
m_wBuf[6]=1;
ReadAlarmLimitValue(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Alarm=m_fBuf[0];
Close_Com(3);
```

Remark:

■ ReadOutputAlarmState

Description:

Obtain the alarm mode and alarm digital output value of I-7000 series modules. This function currently supports I-7011, I-7012, I-7014, and I-7016.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/0x7012/0x7014/0x7016.
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	No used
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
wBuf[7]:	[Output] 0 → alarm disable 1 → momentary alarm 2 → latch alarm
wBuf[8]:	[Output] 0 → DO:0 off DO:1 off 1 → DO:0 on DO:1 off 2 → DO:0 off DO:1 on 3 → DO:0 on DO:1 on
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7016Addr=1;
```

```
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
WORD Mode;
WORD State;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]=m_7016Addr;
m_wBuf[2]=0x7016;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[6]=1;
ReadOutputAlarmState(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Mode=m_wBuf[7];
State=m_wBuf[8];
Close_Com(3);
```

Remark:

5.2.2 I-8000 series modules

■ SetAlarmMode_8K

Description:

Disable or enable the alarm function of I-8000 series modules into momentary alarm or latch alarm mode. This function currently supports I-8013, I-8017h, I-8018, and I-8033.

Syntax:

```
[C++]  
WORD SetAlarmMode_8K (DWORD dwBuf[], float fBuf[], char szSend[], char  
szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x8013/8017/8018/8033
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The specified channel number for I-8013/8017/ 8018/8033
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit.
dwBuf[8]:	[Input] 0 → Low Alarm 1 → High Alarm
dwBuf[9]:	[Input] 0 → disable 1 → Momentary alarm mode 2 → Latch alarm mode
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_NetID=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_8017hSlot=1;
DWORD m_dwBuf[12];
DWORD Alarm;
DWORD Mode;
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]=m_NetID;
m_dwBuf[2]=0x8017h;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8017hSlot;
SetAlarmMode_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ SetAlarmConnect_8K

Description:

This function makes a connection effect between DO module and alarm function of analog input modules in I-8000 main unit. That is, if the alarm function of analog input has happened, then the specified DO channel of DO modules produce the defined output.

Syntax:

[C++]

WORD SetAlarmConnect_8K (DWORD dwBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x8013/8017/8018
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]	[Input] The specified channel number for I-8013/8017/8018/8033
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit.
dwBuf[8]:	[Input] 0 → Low Alarm 1 → High Alarm
dwBuf[9]:	[Input] The slot number of DO module .
dwBuf[10]:	[Input] The defined DO channel according to the alarm function
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
```

```
DWORD m_NetID=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_8017hSlot=1;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]=m_NetID;
m_dwBuf[2]=0x8017h;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8017hSlot;
m_dwBuf[8]=1;
m_dwBuf[9]=1;
m_dwBuf[10]=1;
SetAlarmConnect_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ClearLatchAlarm_8K

Description:

Clear the high or low latch alarm of analog input modules for I-8000 series modules.

Syntax:

[C++]

WORD ClearLatchAlarm_8K (DWORD dwBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x8013/8017/8018
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The defined analog input channel No.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit.
dwBuf[8]:	[Input] 0 → Low Alarm 1 → High Alarm
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules..

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_NetID=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_8017hSlot=1;
```

```
DWORD m_dwBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
Open_Com(3, 9600, 8, 0, 0);  
m_dwBuf[0]=m_ComPort;  
m_dwBuf[1]=m_NetID;  
m_dwBuf[2]=0x8017h;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[5]=1;  
m_dwBuf[6]=1;  
m_dwBuf[7]=m_8017hSlot;  
m_dwBuf[8]=1;  
ClearLatchAlarm_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ SetAlarmLimitValue_8K

Description:

Configure the high or low alarm limit value of analog input modules for I-8000 series modules.

Syntax:

```
[C++]  
WORD SetAlarmLimitValue_8K (DWORD dwBuf[], float fBuf[], char szSend[],  
char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x8013/8017/8018
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The defined analog input channel No.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit.
dwBuf[8]:	[Input] 0→ Set low alarm value 1→ Set high alarm value
fBuf:	Float Input/Output argument table
fBuf[0]:	[Input] Alarm value
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules..

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_NetID=1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;
```

```
DWORD m_8017hSlot=1;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]=m_NetID;
m_dwBuf[2]=0x8017h;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8017hSlot;
m_dwBuf[8]=1;
m_fBuf[0]=0.75;
SetAlarmLimitValue_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ReadAlarmLimitValue_8K

Description:

Obtain the high or low alarm limit value of analog input modules for I-8000 series modules.

Syntax:

```
[C++]  
WORD ReadAlarmLimitValue_8K(DWORD dwBuf[], float fBuf[], char szSend[],  
                             char szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x8013/8017/8018/8033
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The defined analog input channel No.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit.
dwBuf[8]:	[Input] 0 → Set low alarm value 1 → Set high alarm value
fBuf:	Float Input/Output argument table
fBuf[0]:	[Output] Alarm value
szSend:	[Input] Command string to be sent to 8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules..

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_NetID=1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;
```

```
DWORD m_8017hSlot=1;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float Alarm;
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]=m_NetID;
m_dwBuf[2]=0x8017h;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8017hSlot;
m_dwBuf[8]=1;
ReadAlarmLimitValue_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Alarm=m_fBuf[0];
Close_Com(3);
```

Remark:

■ ReadAlarmMode_8K

Description:

Obtain the alarm mode setting of analog input modules for I-8000 series modules.

Syntax:

[C++]

WORD ReadAlarmMode_8K (DWORD dwBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x8013/8017/8018
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The defined analog input channel No.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit.
dwBuf[8]:	[Input] 0→ Read low alarm mode setting 1→ Read high alarm mode setting
dwBuf[9]:	[Output] 0 → Alarm disable 1 → Momentary alarm 2 → Latch alarm
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules..

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_NetID=1;
DWORD m_TimeOut=100;
```

```
DWORD m_Checksum=0;
DWORD m_8017hSlot=1;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
DWORD Alarm;
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]=m_NetID;
m_dwBuf[2]=0x8017h;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8017hSlot;
m_dwBuf[8]=1;
ReadAlarmMode_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Alarm=m_dwBuf[9];
Close_Com(3);
```

Remark:

■ ReadAlarmStatus_8K

Description:

Obtain the alarm status of analog input modules for I-8000 series modules.

Syntax:

```
[C++]  
WORD ReadAlarmStatus_8K (DWORD dwBuf[], float fBuf[], char szSend[], char  
szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0];	[Input] COM port number: 1 to 255
dwBuf[1];	[Input] Module address: 0x00 to 0xFF
dwBuf[2];	[Input] Module ID: 0x8013/17/18/33
dwBuf[3];	[Input] Checksum: 0=disable, 1=enable
dwBuf[4];	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5];	[Input] The defined analog input channel No.
dwBuf[6];	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7];	[Input] Slot number; the I/O module installed in I-8000 main unit.
dwBuf[8]	[Output] 1: High Alarm Occur 0: Don't Occur
dwBuf[9]	[Output] 1: Low Alarm Occur 0: Don't Occur
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_NetID=1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;  
DWORD m_8017hSlot=1;  
DWORD m_dwBuf[12];
```

```
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
DWORD HiAlarm;
DWORD LoAlarm;
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]=m_NetID;
m_dwBuf[2]=0x8017h;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8017hSlot;
ReadAlarmStatus_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
HiAlarm=m_dwBuf[8];
LoAlarm=m_dwBuf[9];
Close_Com(3);
```

Remark:

5.3 Strain Gauge Functions

■ SetupLinearMapping

Description:

Configure the linear mapping translation of I-7014 or I-7016 module from raw data range to target range data value. However, before using this function user need to get the module's range code of the module by calling "ReadConfigStatus () and set it into wBuf [7]. That is, this function provides linear mapping from range area [a, b] to [c, d], where fBuf[0]=a, fBuf[1]=b, fBuf[2]=c, fBuf[3]=d.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address: 0x00 to 0xFF
wBuf[2]:	[Input] Module ID: 0x7014, 0x7016
wBuf[3]:	[Input] Checksum: 0=disable, 1=enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	Not used.
wBuf[6]:	[Input] Flag: 0=no save, 1=save send/receive string
wBuf[7]:	[Input] Range code of this module (Analog input type 00~06)
fBuf:	Float Input/Output argument table
fBuf[0]:	[Input] Source low value, a
fBuf [1]:	[Input] Source high value, b
fBuf [2]:	[Input] Target low value, c
fBuf [3]:	[Input] Target high value, d
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
```

```
WORD m_7016Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7016Addr;
m_wBuf[2]=0x7016;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[6]=1;
m_wBuf[7]=06;
m_fBuf[0]=0;           //linear mapping from range area [0, 20] to [0, 100]
m_fBuf[1]=20;
m_fBuf[2]=0;
m_fBuf[3]=100;
SetupLinearMapping(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ EnableLinearMapping

Description:

Enable linear mapping function for I-7014 or I-7016 module.

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]	[Input] COM port number, 1 to 255
wBuf[1]	[Input] Module address: 0x00 to 0xFF
wBuf[2]	[Input] Module ID: 0x7014, 0x7016
wBuf[3]	[Input] Checksum: 0=disable, 1=enable
wBuf[4]	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]	Not used.
wBuf[6]	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive.
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7016Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;
```

```
m_wBuf[1]= m_7016Addr;  
m_wBuf[2]=0x7016;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[6]=1;  
EnableLinearMapping(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ DisableLinearMapping

Description:

Disable linear mapping function for I-7014 or I-7016 module.

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]	[Input] COM port number, 1 to 255
wBuf[1]	[Input] Module address: 0x00 to 0xFF
wBuf[2]	[Input] Module ID: 0x7014, 0x7016
wBuf[3]	[Input] Checksum: 0=disable, 1=enable
wBuf[4]	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]	Not used.
wBuf[6]	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive.
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7016Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;
```

```
m_wBuf[1]= m_7016Addr;  
m_wBuf[2]=0x7016;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[6]=1;  
DisableLinearMapping(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ ReadLinearMappingStatus

Description:

Obtain the status of the linear mapping function for I-7014 or I-7016 module.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address: 0x00 to 0xFF
wBuf[2]:	[Input] Module ID: 0x7014, 0x7016
wBuf[3]:	[Input] Checksum: 0=disable, 1=enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Output] 0: linear mapping is disable 1: linear mapping is enable
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7016Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
WORD Status;
```

```
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7016Addr;  
m_wBuf[2]=0x7016;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[6]=1;  
ReadLinearMappingStatus(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Status=m_wBuf[5];  
Close_Com(3);
```

Remark:

■ ReadSourceValueOfLM

Description:

Obtain the setting value of Linear Mapping source range for I-7014/7016 module.

Syntax:

[C++]

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

Parameter:

wBuf:	ORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]	[Input] Module address: 0x00 to 0xFF
wBuf[2]:	[Input] Module ID: 0x7014, 0x7016
wBuf[3]:	[Input] Checksum: 0=disable, 1=enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	float Input/Output argument table
fBuf[0]:	[Output] Low Source Value
fBuf[1]:	[Output] High Source Value
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7016Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float LoSValue;
```

```
float HiSValue;  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7016Addr;  
m_wBuf[2]=0x7016;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[6]=1;  
ReadSourceValueOfLM(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
LoSValue=m_fBuf[0];  
HiSValue=m_fBuf[1];  
Close_Com(3);
```

Remark:

■ ReadTargetValueOfLM

Description:

Obtain the setting value of Linear Mapping Target range for I-7014/7016 module.

Syntax:

[C++]

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

Parameter:

wBuf:	ORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]	[Input] Module address: 0x00 to 0xFF
wBuf[2]:	[Input] Module ID: 0x7014, 0x7016
wBuf[3]:	[Input] Checksum: 0=disable, 1=enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	float Input/Output argument table
fBuf[0]:	[Output] Low Target Value
fBuf[1]:	[Output] High Target Value
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7016Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float LoTValue;
```

```
float HiTValue;  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7016Addr;  
m_wBuf[2]=0x7016;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[6]=1;  
ReadTargetValueOfLM(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
LoTValue=m_fBuf[0];  
HiTValue=m_fBuf[1];  
Close_Com(3);
```

Remark:

5.4 Analog Output Functions

5.4.1 I-7000 series modules

■ AnalogOut

Description:

Output the analog value from Analog output module of I-7000 series modules.

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, ID: 0x7016/21/22/24
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] The analog output channel No.(0 to 3) of module I-7024; No used for single analog output module
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive.
fBuf:	Float Input/Output argument table
fBuf[0]:	[Input] Analog output value
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7016Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;
```

```
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7016Addr;
m_wBuf[2]=0x7016;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
//m_wBuf[5] ,I-7016 no used
m_wBuf[6]=1;
m_fBuf[0]=3.5; //Excitation Voltage Output +3.5V
AnalogOut(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ AnalogOutReadBack

Description:

Read back the analog output value of analog output modules for I-7000 series modules. There are two types of read back functions, as described in the following:

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

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7016/7021/7022/7024
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] When 7016: Don't care When 7021/7022/7024 0: command read back (\$AA6) 1: when module ID is 0x7021 or 0x7022 analog output of current path read back (\$AA8) When module ID is 0x7024 the updating value in a specific Slew rate(\$AA8) (see 7024 manual for more detail information)
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive.
wBuf[7]:	[Input] The analog output channel No. (0 to 3) of module I-7024 No used for single analog output module
fBuf:	Float Input/Output argument table
fBuf[0]:	[Output] Analog output read back value
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError: OK
Others: Error code

Example:

```
WORD m_ComPort=3;
WORD m_7016Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float Volt;

Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7016Addr;
m_wBuf[2]=0x7016;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=1;            //$AA6 command
m_wBuf[6]=1;
m_wBuf[7]=1;            //channel 1 "$017+2.57"
AnalogOutReadBack(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Volt=m_fBuf[0];        //Receive: "!01+2.57" excitation voltage, Volt=2.57
Close_Com(3);
```

Remark:

■ AnalogOutHex

Description:

Output the analog value of analog output modules through Hex format.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address: 0x00 to 0xFF
wBuf[2]:	[Input] Module ID: 0x7021/21P/22
wBuf[3]:	[Input] Checksum: 0=disable, 1=enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] The analog output channel No.(0 or 1) of module I-7022; No used for single analog output module
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
wBuf[7]:	[Input] Analog output value in Hexadecimal Data format
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7022Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];
```

```
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7022Addr;
m_wBuf[2]=0x7022;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=1;           // channel 1
m_wBuf[6]=1;
m_wBuf[7]=0x250;
AnalogOutHex(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ AnalogOutFsr

Description:

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

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address: 0x00 to 0xFF
wBuf[2]:	[Input] Module ID: 0x7021/21P/22
wBuf[3]:	[Input] Checksum: 0=disable, 1=enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] The analog output channel No.(0 or 1) of module I-7022; No used for single analog output module
wBuf[6]:	[Input] 0→ no save to szSend & szReceive 1→ save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0]:	[Input] Analog output value in % of Span data format.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7022Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
```

```
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7022Addr;
m_wBuf[2]=0x7022;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=1;           // channel 1
m_wBuf[6]=1;
m_fBuf[0]=50;
AnalogOutFsr(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ AnalogOutReadBackHex

Description:

Read back the analog output value of analog output modules in hex format for I-7000 series modules. There are two types of read back functions, as described in the following:

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

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0];	[Input] COM port number: 1 to 255
wBuf[1];	[Input] Module address: 0x00 to 0xFF
wBuf[2];	[Input] Module ID: 0x7021/21P/22
wBuf[3];	[Input] Checksum: 0=disable, 1=enable
wBuf[4];	[Input] Time out setting, normal=100, unit=ms.
wBuf[5];	[Input] 0: command \$AA6 read back 1: command \$AA8 read back
wBuf[6];	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
wBuf[7];	[Input] The analog output channel No. (0 to 1) of module I-7022 No used for single analog output module
dwBuf[9]	[Output] Analog output value in Hexadecimal Data format
fBuf:	Not used.
SzSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7022Addr=1;
```

```
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
WORD Volt;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7022Addr;
m_wBuf[2]=0x7022;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=0;           //command $AA6
m_wBuf[6]=1;
m_wBuf[7]=0;
AnalogOutReadBackHex(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Volt=m_wBuf[9];
Close_Com(3);
```

Remark:

■ AnalogOutReadBackFsr

Description:

Read back the analog output value of analog output modules through % of span data format for I-7000 series modules.

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

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number: 1 to 255
wBuf[1]:	[Input] Module address: 0x00 to 0xFF
wBuf[2]:	[Input] Module ID: 0x7021/21P/22
wBuf[3]:	[Input] Checksum: 0=disable, 1=enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] 0: command \$AA6 read back 1: command \$AA8 read back
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
wBuf[7]:	[Input] The analog output channel No. (0 to 1) of module I-7022 No used for single analog output module
fBuf:	Float Input/Output argument table
fBuf[0]:	[Output] Analog output value in % of Span data format.
SzSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7022Addr=1;  
WORD m_TimeOut=100;
```

```
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float Volt;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7022Addr;
m_wBuf[2]=0x7022;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=0; //command $AA6
m_wBuf[6]=1;
m_wBuf[7]=0;
AnalogOutReadBackFsr(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Volt=m_fBuf[0];
Close_Com(3);
```

Remark:

5.4.2 I-8000 series modules

■ AnalogOut_8K

Description:

Output the analog value of analog output module for I-8000 series modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x8024
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The defined analog output channel No.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit
fBuf:	Float Input/Output argument table
fBuf[0]:	[Input] Analog output value
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_NetID=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
```

```
DWORD m_8024Slot=1;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]=m_NetID;
m_dwBuf[2]=0x8024;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8024Slot;
m_fBuf[0]=2.55;
AnalogOut_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ AnalogOutReadBack_8K

Description:

Read back the analog output value of analog output module for I-8000 series modules. This function currently supports I-8022/8024/8026 modules.

Syntax:

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x8024
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The defined analog output channel No.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit.
fBuf:	Float Input/Output argument table
fBuf[0]:	[Output] Analog output read back value
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_NetID=1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;  
DWORD m_8024Slot=1;  
DWORD m_dwBuf[12];
```

```
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float Volt;
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]=m_NetID;
m_dwBuf[2]=0x8024;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8024Slot;
AnalogOutReadBack_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Volt=m_fBuf[0];
Close_Com(3);
```

Remark:

■ ReadConfigurationStatus_8K

Description:

Read Configuration Status of analog output module for I-8000 series modules. This function currently supports I-8022/8024/8026 modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x8024, 0x8022, 0x8026
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The defined analog output channel No.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit.
dwBuf[8]:	[Output] Output range : 0x30, 0x31, 0x32
dwBuf[9]:	[Output] Slew rate
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_NetID=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_8024Slot=1;
```

```
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
DWORD Status;
DWORD Rate;
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]=m_NetID;
m_dwBuf[2]=0x8024;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8024Slot;
ReadConfigurationStatus_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Status=m_dwBuf[8];
Rate=m_dwBuf[9];
Close_Com(3);
```

Remark:

■ SetStartUpValue_8K

Description:

Setting Start-Up Value for I-8000 series modules. This function currently supports I-8022/8024/8026 modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x8024, 0x8022, 0x8026
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The defined analog output channel No.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit.
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_NetID=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_8024Slot=1;
DWORD m_dwBuf[12];
float m_fBuf[12];
```

```
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]=m_NetID;
m_dwBuf[2]=0x8024;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8024Slot;
SetStartUpValue_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ReadStartUpValue_8K

Description:

Read Start-Up Value for I-8000 series modules. This function currently supports I-8022/8024/8026 modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x8024, 0x8022, 0x8026
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The defined analog output channel No.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-8000 main unit.
fBuf:	Float Input/Output argument table
fBuf[0]:	Start-Up Value
szSend:	[Input] Command string to be sent to I-8000 series modules
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_NetID=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_8024Slot=1;
DWORD m_dwBuf[12];
```

```
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float StartUp;
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]=m_NetID;
m_dwBuf[2]=0x8024;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8024Slot;
ReadStartUpValue_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
StartUp=m_fBuf[0];
Close_Com(3);
```

Remark:

5.4.3 I-87K series modules

■ AnalogOut_87K

Description:

Output the analog value of analog output module for I-87K series modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0] :	[Input] COM port number, 1 to 255
dwBuf[1] :	[Input] Module address, from 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87016/ 0x87021/0x87022 /0x87024/0x87026
dwBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0] :	[Input] Analog output value
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87024Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
```

```
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87024Addr;
m_dwBuf[2]=0x87024;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_fBuf[0]=2.55;      //+2.55V
AnalogOut_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ AnalogOutReadBack_87K

Description:

Read back the analog output value of analog output modules for I-87K series modules. This function currently supports I-87021/22/24/26. There are two types of read back functions, as described in the following:

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

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0X87016/87021/87024/87026
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The defined analog output channel No
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0] :	[Output] Analog output read back value
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_87024Addr=1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;  
DWORD m_dwBuf[12];
```

```
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float Volt;
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87024Addr;
m_dwBuf[2]=0x87024;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
AnalogOutReadBack_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Volt=m_fBuf[0];
Close_Com(3);
```

Remark:

■ AnalogOutHex_87K

Description:

Output the analog value of analog output I-87K series modules through Hex format.

Syntax:

[C++]

WORD AnalogOutHex_87K (DWORD dwBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address: 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID: 0x87022/87026 (0x87024 has no Hex type)
dwBuf[3]:	[Input] Checksum: 0=disable, 1=enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The analog output channel No. (0 or 1)
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Analog output value in Hexadecimal Data format
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87022Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
```

```
Open_Com(3, 9600, 8, 0, 0);  
m_dwBuf[0]=m_ComPort;  
m_dwBuf[1]= m_87022Addr;  
m_dwBuf[2]=0x87022;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[5]=1;  
m_dwBuf[6]=1;  
m_dwBuf[7]=0x250;  
AnalogOutHex_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ AnalogOutFsr_87K

Description:

Output the analog value of analog output through % of span data format for I-87K series modules. This function only can be used after analog output module is set as “FSR” output mode.

Syntax:

```
[C++]  
WORD AnalogOutFsr_87K (DWORD dwBuf[], float fBuf[], char szSend[], char  
szReceive[])
```

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0] :	[Input] COM port number, 1 to 255
dwBuf[1] :	[Input] Module address, from 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID: 0x87022/87026 (0x87024 has no Fsr type)
dwBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5] :	[Input] The defined analog output channel No.
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0] :	[Input] Analog output value in % of Span data format.
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_87022Addr=1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;  
DWORD m_dwBuf[12];  
float m_fBuf[12];  
char m_szSend[100];
```

```
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87022Addr;
m_dwBuf[2]=0x87022;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_fBuf[0]=50;           //50%
AnalogOutFsr_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ AnalogOutReadBackHex_87K

Description:

Read back the analog output value of analog output modules in hex format for I-87K series modules. There are two types of read back functions, as described in the following:

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

Syntax:

[C++]

WORD AnalogOutReadBackHex_87K (DWORD dwBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0];	[Input] COM port number: 1 to 255
dwBuf[1];	[Input] Module address: 0x00 to 0xFF
dwBuf[2];	[Input] Module ID: 0x87022/87026 (0x87024 has no Hex type)
dwBuf[3];	[Input] Checksum: 0=disable, 1=enable
dwBuf[4];	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5];	[Input] 0: command \$AA6 read back 1: command \$AA8 read back
dwBuf[6]	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7] :	[Input] The analog output channel No. (0 or 1)
dwBuf[9]	[Output] Analog output value read back in Hexadecimal Data format
fBuf:	Not used.
SzSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_87022Addr=1;
```

```
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
DWORD Volt;
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87022Addr;
m_dwBuf[2]=0x87022;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_dwBuf[7]=1; //channel 1
AnalogOutReadBackHex_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Volt=m_dwBuf[9];
Close_Com(3);
```

Remark:

■ AnalogOutReadBackFsr_87K

Description:

Read back the analog output value of analog output modules through % of span data format for I-87K series modules. There are two types of read back functions, as described in the following:

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

Syntax:

[C++]

WORD AnalogOutReadBackFsr_87K (DWORD dwBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0];	[Input] COM port number: 1 to 255
dwBuf[1];	[Input] Module address: 0x00 to 0xFF
dwBuf[2];	[Input] Module ID: 0x87022/87026 (0x87024 has no Fsr type)
dwBuf[3];	[Input] Checksum: 0=disable, 1=enable
dwBuf[4];	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5];	[Input] 0: command \$AA6 read back 1: command \$AA8 read back
dwBuf[6]	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7] :	[Input] The analog output channel No. (0 or 1)
fBuf:	Float Input/Output argument table
fBuf[0]:	[Output] Analog output value read back in % of Span data format.
SzSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_87022Addr=1;
```

```
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float Volt;
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87022Addr;
m_dwBuf[2]=0x87022;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_dwBuf[7]=1; //channel 1
AnalogOutReadBackFsr_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Volt=m_fBuf[0];
Close_Com(3);
```

Remark:

■ ReadConfigurationStatus_87K

Description:

Read Configuration Status of analog output module for I-87K series modules. This function currently supports I-87024 modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x87024
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The defined analog output channel No.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-87K main unit.
dwBuf[8]:	[Output] Output range : 0x30, 0x31, 0x32
dwBuf[9]:	[Output] Slew rate
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-87K series modules
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87024Addr=1
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_87024Slot=1;
```

```
DWORD m_dwBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
DWORD Status;  
DWORD Rate;  
Open_Com(3, 9600, 8, 0, 0);  
m_dwBuf[0]=m_ComPort;  
m_dwBuf[1]= m_87024Addr;  
m_dwBuf[2]=0x87024;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[5]=0;  
m_dwBuf[6]=1;  
m_dwBuf[7]=m_87024Slot;  
ReadConfigurationStatus_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
Status=m_dwBuf[8];  
Rate=m_dwBuf[9];  
Close_Com(3);
```

Remark:

■ SetStartUpValue_87K

Description:

Setting Start-Up Value for I-87K series modules. This function currently supports I-87024 modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x80724
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The defined analog output channel No.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-87K main unit.
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-87K series modules
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87024Addr =1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_87024Slot=1;
DWORD m_dwBuf[12];
float m_fBuf[12];
```

```
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87024Addr;
m_dwBuf[2]=0x87024;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=0;
m_dwBuf[6]=1;
m_dwBuf[7]=m_87024Slot;
SetStartUpValue_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ReadStartUpValue_87K

Description:

Read Start-Up Value for I-87K series modules. This function currently supports I-87024 modules.

Syntax:

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x87024
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The defined analog output channel No.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] Slot number; the I/O module installed in I-87K main unit.
fBuf:	Float Input/Output argument table
fBuf[0]:	Start-Up Value
szSend:	[Input] Command string to be sent to I-87K series modules
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_87024Addr =1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;  
DWORD m_87024Slot=1;  
DWORD m_dwBuf[12];
```

```
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
float StartUp;
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87024Addr;
m_dwBuf[2]=0x87024;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=0;
m_dwBuf[6]=1;
m_dwBuf[7]=m_87024Slot;
ReadStartUpValue_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
StartUp=m_fBuf[0];
Close_Com(3);
```

Remark:

5.5 Digital Input Functions

5.5.1 I-7000 series modules

■ DigitalIn

Description:

Obtain the digital input value from I-7000 series modules.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0] :	[Input] COM port number, 1 to 255
wBuf[1] :	[Input] Module address, from 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7050/7052/7053/7060/7041/7044
wBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
wBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Output] 16-bit digital input data
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7050Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
```

```
char m_szSend[100];
char m_szReceive[100];
WORD DI;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7050Addr;
m_wBuf[2]=0x7050;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[6]=1;
DigitalIn(m_wBuf, m_fBuf, m_szSend, m_szReceive);
DI=m_wfBuf[5];
Close_Com(3);
```

Remark:

■ DigitalInLatch

Description:

Obtain the latch value of the high or low latch mode of Digital Input module.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address: 0x00 to 0xFF
wBuf[2]:	[Input] Module ID: 0x7050/52/53/60/63/65/41/44
wBuf[3]:	[Input] Checksum: 0=disable, 1=enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] 0: low Latch mode , 1: high Latch mode
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
wBuf[7]:	[Output] Latch value
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7050Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
```

```
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7050Addr;
m_wBuf[2]=0x7050;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=1;
m_wBuf[6]=1;
m_wBuf[7]=0x03;
DigitalInLatch(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ClearDigitalInLatch

Description:

This function can clear the latch status of digital input module when latch function has been enabled.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address: 0x00 to 0xFF
wBuf[2]:	[Input] Module ID: 0x7050/52/53/60/63/65/41/44
wBuf[3]:	[Input] Checksum: 0=disable, 1=enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	Not used.
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7050Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
```

```
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7050Addr;  
m_wBuf[2]=0x7050;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[6]=1;  
ClearDigitalInLatch(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ DigitalInCounterRead

Description:

Obtain the counter event value of the channel number of Digital Input module.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address: 0x00 to 0xFF
wBuf[2]:	[Input] Module ID: 0x7050/52/53/60/63/65/41/44
wBuf[3]:	[Input] Checksum: 0=disable, 1=enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] The digital input Channel No.
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
wBuf[7]:	[Output] Counter value of the digital input channel No.
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7050Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
```

```
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7050Addr;
m_wBuf[2]=0x7050;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=0;
m_wBuf[6]=1;
m_wBuf[7]=100;
DigitalInCounterRead(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ClearDigitalInCounter

Description:

Clear the counter value of the channel number of Digital Input module.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address: 0x00 to 0xFF
wBuf[2]:	[Input] Module ID: 0x7050/52/53/60/63/65/41/44
wBuf[3]:	[Input] Checksum: 0=disable, 1=enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] The digital input Channel No.
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7050Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
```

```
m_wBuf[1]= m_7050Addr;  
m_wBuf[2]=0x7050;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[5]=0;  
m_wBuf[6]=1;  
ClearDigitalInCounter(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ ReadEventCounter

Description:

Obtain the value of event counter of I-7000 series modules. This function only supports I-7011, I-7012, and I-7014 modules.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument tablew
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/0x7012/0x7014/0x7016
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	Not used
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
wBuf[7]:	[Output] The value of event counter
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7012Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
```

```
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7012Addr;  
m_wBuf[2]=0x7012;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[6]=1;  
m_wBuf[7]=100;  
ReadEventCounter(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ ClearEventCounter

Description:

Clear the value of event counter of I-7000 series modules. This function only supports I-7011, I-7012, and I-7014 modules.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument tablew
Buf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/0x7012/0x7014/0x7016
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	Not used
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7012Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
```

```
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7012Addr;  
m_wBuf[2]=0x7012;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[6]=1;  
ClearEventCounter(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

5.5.2 I-8000 series modules

■ DigitalIn_8K

Description:

Obtain the digital input value from I-8000 series modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0] :	[Input] COM port number, 1 to 255
dwBuf[1] :	[Input] Module address, from 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8040/42/51/52/53/54/55/58/63/77
dwBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5] :	[Output] 16-bit digital input data
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-8000 main unit.
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_NetID=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_8040Slot=1;
```

```
DWORD m_dwBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
DWORD DI;  
Open_Com(3, 9600, 8, 0, 0);  
m_dwBuf[0]=m_ComPort;  
m_dwBuf[1]= m_NetID;  
m_dwBuf[2]=0x8040;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[6]=1;  
m_dwBuf[7]=m_8040Slot;  
DigitalIn_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
DI= m_dwBuf[5];  
Close_Com(3);
```

Remark:

■ DigitalInCounterRead_8K

Description:

Obtain the counter value of the digital input channel No. of I-8K series modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0] :	[Input] COM port number, 1 to 255
dwBuf[1] :	[Input] Module address, from 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8051/52/53/54/55/58/63/40
dwBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5] :	[Input] Channel No.
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-8000 main unit.
dwBuf[8] :	[Output] DigitalIn Counter Value
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_NetID=1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;  
DWORD m_8040Slot=1;  
DWORD m_dwBuf[12];  
float m_fBuf[12];
```

```
char m_szSend[100];
char m_szReceive[100];
DWORD DIcounter;
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_NetID;
m_dwBuf[2]=0x8040;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=0;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8040Slot;
DigitalInCounterRead_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
DIcounter= m_dwBuf[8];
Close_Com(3);
```

Remark:

■ ClearDigitalInCounter_8K

Description:

Clear the counter value of the digital input channel No. of I-8K series modules.

Syntax:

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0] :	[Input] COM port number, 1 to 255
dwBuf[1] :	[Input] Module address, from 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8051/52/53/54/55/58/63/40
dwBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5] :	[Input] Channel No.
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-8000 main unit.
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_NetID=1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;  
DWORD m_8040Slot=1;  
DWORD m_dwBuf[12];  
float m_fBuf[12];  
char m_szSend[100];
```

```
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_NetID;
m_dwBuf[2]=0x8040;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=0;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8040Slot;
ClearDigitalInCounter_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ DigitalInLatch_8K

Description:

Obtain the digital Input latch value of the high or low latch mode of I-8K series modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0] :	[Input] COM port number, 1 to 255
dwBuf[1] :	[Input] Module address, from 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8051/52/53/54/55/58/63/40
dwBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5] :	[Input] 0 → select to latch low 1 → select to latch high
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-8000 main unit.
dwBuf[8] :	[Output] Latched data
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_NetID=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_8040Slot=1;
```

```
DWORD m_dwBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
DWORD Dllatch;  
Open_Com(3, 9600, 8, 0, 0);  
m_dwBuf[0]=m_ComPort;  
m_dwBuf[1]= m_NetID;  
m_dwBuf[2]=0x8040;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[5]=0;  
m_dwBuf[6]=1;  
m_dwBuf[7]=m_8040Slot;  
DigitalInLatch_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
Dllatch= m_dwBuf[8];  
Close_Com(3);
```

Remark:

■ ClearDigitalInLatch_8K

Description:

This function can clear the latch status of digital input module when latch function has been enabled.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0] :	[Input] COM port number, 1 to 255
dwBuf[1] :	[Input] Module address, from 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8051/52/53/54/55/58/63/40
dwBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5] :	[Input] 0 → select to latch low 1 → select to latch high
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-8000 main unit.
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_NetID=1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;  
DWORD m_8040Slot=1;  
DWORD m_dwBuf[12];
```

```
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
DWORD Dllatch;
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_NetID;
m_dwBuf[2]=0x8040;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=0;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8040Slot;
ClearDigitalInLatch_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

5.5.3 I-87K series modules

■ DigitalIn_87K

Description:

Obtain the digital input value from I-87K series modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0] :	[Input] COM port number, 1 to 255
dwBuf[1] :	[Input] Module address, from 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87054/55/56/57/60/63/64/65/66/68
dwBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5] :	[Output] 16-bit digital input data
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87054Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];  
DWORD DI;  
Open_Com(3, 9600, 8, 0, 0);  
m_dwBuf[0]=m_ComPort;  
m_dwBuf[1]= m_87054Addr;  
m_dwBuf[2]=0x87054;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[6]=1;  
DigitalIn_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
DI= m_dwBuf[5];  
Close_Com(3);
```

Remark:

■ DigitalInLatch_87K

Description:

Obtain the digital Input latch value of the high or low latch mode of I-87K series modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address: 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID: 0x87051/52/53/54/58/63
dwBuf[3]:	[Input] Checksum: 0=disable, 1=enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] 0: low Latch mode , 1: high Latch mode
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Output] Latch value
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87051Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
```

```
Open_Com(3, 9600, 8, 0, 0);  
m_dwBuf[0]=m_ComPort;  
m_dwBuf[1]= m_87051Addr;  
m_dwBuf[2]=0x87051;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[5]=1;  
m_dwBuf[6]=1;  
m_dwBuf[7]=0x01;  
DigitalInLatch_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ ClearDigitalInLatch_87K

Description:

This function can clear the latch status of digital input module when latch function has been enabled.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address: 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID: 0x87051/52/53/54/63
dwBuf[3]:	[Input] Checksum: 0=disable, 1=enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	Not used.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_87051Addr=1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;  
DWORD m_dwBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
Open_Com(3, 9600, 8, 0, 0);
```

```
m_dwBuf[0]=m_ComPort;  
m_dwBuf[1]= m_87051Addr;  
m_dwBuf[2]=0x87051;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[6]=1;  
ClearDigitalInLatch_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ DigitalInCounterRead_87K

Description:

Obtain the counter value of the digital input channel No. of I-87K series modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address: 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID: 0x87051/52/53/54/63
dwBuf[3]:	[Input] Checksum: 0=disable, 1=enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The digital input Channel No.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
dwBuf[7]:	[Output] Counter value of the digital input channel No.
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87051Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
```

```
m_dwBuf[0]=m_ComPort;  
m_dwBuf[1]= m_87051Addr;  
m_dwBuf[2]=0x87051;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[5]=0;  
m_dwBuf[6]=1;  
m_dwBuf[7]=16;  
DigitalInCounterRead_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ ClearDigitalInCounter_87K

Description:

Clear the counter value of the digital input channel No. of I-87K series modules.

Syntax:

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address: 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID: 0x87051/52/53/54/63
dwBuf[3]:	[Input] Checksum: 0=disable, 1=enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] The digital input Channel No.
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive
fBuf:	Not used.
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_87051Addr=1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;  
DWORD m_dwBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
Open_Com(3, 9600, 8, 0, 0);  
m_dwBuf[0]=m_ComPort;
```

```
m_dwBuf[1]= m_87051Addr;  
m_dwBuf[2]=0x87051;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[5]=0;  
m_dwBuf[6]=1;  
ClearDigitalInCounter_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

5.6 Digital Output Functions

5.6.1 I-7000 series modules

■ DigitalOut

Description:

Output the value of the digital output module for I-7000 series modules.

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0] :	[Input] COM port number, 1 to 255
wBuf[1] :	[Input] Module address, from 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7011/12/14/42/43/44/50/ 0x7060/63/65/66/67/80
wBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
wBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] 16-bit digital output data
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive.
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7011Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];
```

```
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7011Addr;
m_wBuf[2]=0x7011;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=0x03
m_wBuf[6]=1;
DigitalOut(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ DigitalBitOut

Description:

Set the digital output value of the channel No. of I-7000 series modules. The output Value is “0” or “1”.

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address: 0x00 to 0xFF
wBuf[2]:	[Input] Module ID: 0x7050/60/63/65/66/67/42/43/44
wBuf[3]:	[Input] Checksum: 0=disable, 1=enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	Not used
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → Save to szSend & szReceive.
wBuf[7]:	[Input] The digital output channel No.
wBuf[8]:	[Input] Logic value (0 or 1)
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7065Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];
```

```
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7065Addr;
m_wBuf[2]=0x7065;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[6]=1;
m_wBuf[7]=0x08; //RL4 relay On
m_wBuf[8]=1;
DigitalBitOut(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ DigitalOutReadBack

Description:

Read back the digital output value of I-7000 series modules.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0] :	[Input] COM port number, 1 to 255
wBuf[1] :	[Input] Module address, from 0x00 to 0xFF
wBuf[2] :	[Input] Module ID, 0x7050/60/66/67/42/43/44
wBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
wBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Output] 16-bit digital output data read back
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7060Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
WORD DO;
Open_Com(3, 9600, 8, 0, 0);
```

```
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7060Addr;  
m_wBuf[2]=0x7060;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[6]=1;  
DigitalOutReadBack(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
DO= m_wBuf[5];  
Close_Com(3);
```

Remark:

■ DigitalOut_7016

Description:

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

Syntax:

[C++]

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

Parameter:

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

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7016Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
```

```
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7016Addr;
m_wBuf[2]=0x7016;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=1;
m_wBuf[6]=1;
m_wBuf[7]=1;
DigitalOut_7016(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

5.6.2 I-8000 series modules

■ DigitalOut_8K

Description:

Set the digital output value of digital output module for I-8000 series modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0] :	[Input] COM port number, 1 to 255
dwBuf[1] :	[Input] Module address, from 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8041/42/54/55/56/57/60 0x8063/64/65/66/68/77
dwBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5] :	[Input] 16-bit digital output data
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-8000 main unit.
fBuf:	Not Used
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules..

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_NetID=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
```

```
DWORD m_8041Slot=1;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]=m_NetID;
m_dwBuf[2]=0x8041;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=10;
m_dwBuf[6]=1;
m_dwBuf[7]=m_8041Slot;
DigitalOut_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ DigitalBitOut_8K

Description:

Set the digital value of the digital output channel No. of I-8000 series modules.
The output value is "0" or "1"

Syntax:

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0] :	[Input] COM port number, 1 to 255
dwBuf[1] :	[Input] Module address, from 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x8041/42/54/55/56/57/60 0x8063/64/65/66/68/77
dwBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5] :	[Input] Output digital data; 0 or 1
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7] :	[Input] Slot number; the I/O module installed in I-8000 main unit.
dwBuf[8];	[Input] The output channel No.
fBuf:	Not Used
szSend:	[Input] Command string to be sent to I-8000 series modules.
szReceive:	[Output] Result string receiving from I-8000 series modules..

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_NetID=1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;  
DWORD m_8041Slot=1;
```

```
DWORD m_dwBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
Open_Com(3, 9600, 8, 0, 0);  
m_dwBuf[0]=m_ComPort;  
m_dwBuf[1]=m_NetID;  
m_dwBuf[2]=0x8041;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[5]=1;  
m_dwBuf[6]=1;  
m_dwBuf[7]=m_8041Slot;  
m_dwBuf[8]=3; // Channel 3 output high logic  
DigitalBitOut_8K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

5.6.3 I-87K series modules

■ DigitalOut_87K

Description:

Set the digital output value of the digital output module for I-87K series modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0] :	[Input] COM port number, 1 to 255
dwBuf[1] :	[Input] Module address, from 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87054/55/56/57/60/63/64/65/66/68
dwBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5] :	[Input] 16-bit digital output data
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87054Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87054Addr;
m_dwBuf[2]=0x87054;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=3;
m_dwBuf[6]=1;
DigitalOut_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ DigitalOutReadBack_87K

Description:

Read back the digital output value of the digital output module for I-87K series modules.

Syntax:

[C++]

WORD DigitalOutRaedBack_87K (DWORD dwBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0] :	[Input] COM port number, 1 to 255
dwBuf[1] :	[Input] Module address, from 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87054/55/56/57/60/63/64/65/66/68
dwBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5] :	[Output] 16-bit digital output data read back
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87054Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
DWORD DO;
```

```
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87054Addr;
m_dwBuf[2]=0x87054;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[6]=1;
DigitalOutReadBack_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
DO=m_dwBuf[5];
Close_Com(3);
```

Remark:

■ DigitalBitOut_87K

Description:

Set the digital output value of the specific digital output channel No. of the digital output module for I-87K series modules. The output value is only for “0” or “1”

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0] :	[Input] COM port number, 1 to 255
dwBuf[1] :	[Input] Module address, from 0x00 to 0xFF
dwBuf[2] :	[Input] Module ID, 0x87054/55/56/57/60/63/64/65/66/68
dwBuf[3] :	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4] :	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5] :	[Input] 1-bit digital output data
dwBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
dwBuf[7]:	[Input] The digital output channel No.
dwBuf[8]:	[Input] Data to output (0 or 1)
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-87K series modules.
szReceive:	[Output] Result string receiving from I-87K series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87054Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87054Addr;
m_dwBuf[2]=0x87054;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=1;
m_dwBuf[6]=1;
m_dwBuf[7]=1;
m_dwBuf[8]=1;
DigitalBitOut_87K(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

5.7 Counter Functions

■ CounterIn_7080

Description:

Obtain the value of the selected counter in module I-7080.

Syntax:

```
[C++]  
WORD CounterIn_7080 (WORD wBuf[], float fBuf[], char szSend[], char  
szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] 0 → Set counter 0 1 → Set counter 1
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceiveseries
wBuf[7]:	[Output] High word of counter value from the selected counter
wBuf[8]:	[Output] Low word of counter value from the selected counter
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7080Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;
```

```
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
WORD CountH;
WORD CountL;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7080Addr;
m_wBuf[2]=0x7080;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=0;
m_wBuf[6]=1;
CounterIn_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);
CountH=m_wBuf[7];
CountL=m_wBuf[8];
Close_Com(3);
```

Remark:

■ StartCounting_7080

Description:

Start/stop the counting process of the selected counter in module I-7080.

Syntax:

```
[C++]  
WORD StartCounting_7080 (WORD wBuf[], float fBuf[], char szSend[], char  
szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] 0 → Set counter 0 1 → Set counter 1
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceiveseries
wBuf[7]:	[Input] 0 → Stop Counting 1 → Start Counting
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7080Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];
```

```
char m_szReceive[100];  
WORD CountH;  
WORD CountL;  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7080Addr;  
m_wBuf[2]=0x7080;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[5]=0;  
m_wBuf[6]=1;  
m_wBuf[7]=1;  
StartCounting_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ ClearCounter_7080

Description:

Clear the value of the selected counter in module I-7080.

Syntax:

[C++]

WORD ClearCounter_7080 (WORD wBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] 0 → Clear the value of counter 0 1 → Clear the value of counter 1
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceiveseries
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7080Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
```

```
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7080Addr;
m_wBuf[2]=0x7080;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=0;
m_wBuf[6]=1;
ClearCounter_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ReadCounterMaxValue _7080

Description:

Obtain the maximum setting value of the selected counter in module I-7080.

Syntax:

[C++]

WORD ReadCounterMaxValue_7080 (WORD wBuf[], float fBuf[], char szSend[],
char szReceive[])

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] 0 → Set counter 0 1 → Set counter 1
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceiveseries
wBuf[7]:	[Output] High word of maximum setting value from the selected counter
wBuf[8]:	[Output] Low word of maximum setting value from the selected counter
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7080Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];
```

```
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
WORD CountMaxH;
WORD CountMaxL;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7080Addr;
m_wBuf[2]=0x7080;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=0;
m_wBuf[6]=1;
ReadCounterMaxValue_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);
CountMaxH=m_wBuf[7];
CountMaxL=m_wBuf[8];
Close_Com(3);
```

Remark:

■ SetCounterMaxValue _7080

Description:

Configure the maximum value of the selected counter for module I-7080.

Syntax:

```
[C++]  
WORD SetCounterMaxValue_7080 (WORD wBuf[], float fBuf[], char szSend[],  
char szReceive[], unsigned long MaxValue)
```

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5]:	[Input] 0 → Set counter 0 1 → Set counter 1
wBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceiveseries
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.
MaxValue:	[Input] Setting the Counter's Max Value

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7080Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];
```

```
unsigned long MaxValue
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7080Addr;
m_wBuf[2]=0x7080;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut;
m_wBuf[5]=0;
m_wBuf[6]=1;
MaxValue=1000;
ReadCounterMaxValue_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive, MaxValue);
Close_Com(3);
```

Remark:

■ EnableCounterAlarm_7080

Description:

Enable counter alarm (for alarm-mode 0) of I-7080 module.

Syntax:

[C++]

WORD EnableCounterAlarm_7080 (WORD wBuf[], float fBuf[], char szSend[],
char szReceive[])

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] 0 : momentary 1 : latch
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7080Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
```

```
m_wBuf[1]= m_7080Addr;  
m_wBuf[2]=0x7080;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut;  
m_wBuf[5]=0;  
m_wBuf[6]=1;  
EnableCounterAlarm_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ DisableCounterAlarm_7080

Description:

Disable counter alarm (for alarm-mode 0) of I-7080 module.

Syntax:

```
[C++]  
WORD DisableCounterAlarm_7080 (WORD wBuf[], float fBuf[], char szSend[],  
char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] Not used
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7080Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;
```

```
m_wBuf[1]= m_7080Addr;  
m_wBuf[2]=0x7080;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[6]=1;  
DisableCounterAlarm_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ ReadInputSignalMode _7080

Description:

Obtain the setting value of input signal mode in I-7080 module. For more detail information for “Input signal mode” please refer user’s manual.

Syntax:

[C++]

WORD ReadInputSignalMode_7080 (WORD wBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] Not used
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
wBuf[7] :	[Output] 0 → Counter:0 TTL Counter:1 TTL 1 → Counter:0 Photo Counter:1 Photo 2 → Counter:0 TTL Counter:1 Photo 3 → Counter:0 Photo Counter:1 TTL
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7080Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
```

```
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
WORD Status;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7080Addr;
m_wBuf[2]=0x7080;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[6]=1;
ReadInputSignalMode_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Status=m_wBuf(7);
Close_Com(3);
```

Remark:


```
char m_szReceive[100];  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7080Addr;  
m_wBuf[2]=0x7080;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[5]=0;  
m_wBuf[6]=1;  
SetInputSignalMode_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ PresetCounterValue_7080

Description:

Configure the preset value of the selected counter in I-7080 module.

Syntax:

[C++]

WORD PresetCounterValue_7080 (WORD wBuf[], float fBuf[], char szSend[],
char szReceive[], unsigned long PresetValue)

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] 0 → Configure counter 0 1 → Configure counter 1
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7080Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
unsigned long PresetValue;
```

```
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7080Addr;  
m_wBuf[2]=0x7080;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[5]=0;  
m_wBuf[6]=1;  
PresetValue=100;  
PresetCounterValue_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive, PresetValue);  
Close_Com(3);
```

Remark:

■ ReadPresetCounterValue_7080

Description:

Obtain the preset value of the selected counter in I-7080 module.

Syntax:

```
[C++]  
WORD ReadPresetCounterValue_7080 (WORD wBuf[], float fBuf[], char  
szSend[], char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] 0 → Read counter 0 1 → Read counter 1
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
wBuf[7]:	[Output] The high word of preset value
wBuf[8]:	[Output] The low word of preset value
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7080Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];
```

```
char m_szReceive[100];  
WORD PresetValueH;  
WORD PresetValueL;  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7080Addr;  
m_wBuf[2]=0x7080;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[5]=0;  
m_wBuf[6]=1;  
ReadPresetCounterValue_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
PresetValueH=m_wBuf[7];  
PresetValueL=m_wBuf[8];  
Close_Com(3);
```

Remark:

■ SetModuleMode_7080

Description:

Configure the alarm mode of the selected counter in I-7080 module. There are two counter alarm modes; alarm mode 0 and alarm mode 1. These two alarm modes can be used in both of I-7080 & I-7080D. For more detail information please refer user's manual.

Syntax:

[C++]

WORD SetModuleMode_7080 (WORD wBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] 0 → to set into 7080 alarm mode(mode 0) 1 → to set into 7080D alarm mode (mode 1)
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7080Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
```

```
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7080Addr;
m_wBuf[2]=0x7080;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[5]=0;
m_wBuf[6]=1;
SetModuleMode_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ReadModuleMode_7080

Description:

Obtain the setting status of counter alarm mode in I-7080 module.

Syntax:

[C++]

WORD ReadModuleMode_7080 (WORD wBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Output] 0→ to set into 7080 alarm mode(mode 0) 1→ to set into 7080D alarm mode (mode 1)
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7080Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
WORD ModuleMode;
```

```
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7080Addr;  
m_wBuf[2]=0x7080;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[6]=1;  
SetModuleMode_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
ModuleMode= m_wBuf[5];  
Close_Com(3);
```

Remark:

■ SetLevelVolt_7080

Description:

Configure the high or low trigger level value of non-isolated input in I-7080 module.

Syntax:

```
[C++]  
WORD SetLevelVolt_7080 (WORD wBuf[], float fBuf[], char szSend[], char  
szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] 0 → Set the low trigger level 1 → Set the high trigger level
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0]:	[Input] The trigger level value
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7080Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];
```

```
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7080Addr;
m_wBuf[2]=0x7080;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[5]=0;
m_wBuf[6]=1;
m_fBuf[0]=1.25;      //Low level voltage= 1.25 V
SetLevelVolt_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ReadLevelVolt_7080

Description:

Obtain the high or low trigger level setting value of non-isolated input in I-7080 module.

Syntax:

[C++]

WORD ReadLevelVolt_7080 (WORD wBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] 0 → Set the low trigger level 1 → Set the high trigger level
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0]:	[Output] The trigger level value
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7080Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];  
float LevelVolt;  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7080Addr;  
m_wBuf[2]=0x7080;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[5]=0;  
m_wBuf[6]=1;  
ReadLevelVolt_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
LevelVolt=m_fBuf[0];  
Close_Com(3);
```

Remark:

■ SetMinSignalWidth_7080

Description:

Configure the width value of the minimum high or low input signal level in I-7080 module.

Syntax:

[C++]

WORD SetMinSignalWidth_7080 (WORD wBuf[], float fBuf[], char szSend[], char szReceive[], long MinWidth)

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] 0 → set the min. width at low level 1 → set the min. width at high level
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.
MinWidth:	[Input] The minimum input signal width of the high or low trigger

level. The value unit is uS and the corresponding range is from 2 uS to 65535 uS. For Example: when MinWidth =2000, it means the minimum with is 2 mS.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7080Addr=1;
WORD m_TimeOut=100;
```

```
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
long MinWidth;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7080Addr;
m_wBuf[2]=0x7080;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[5]=0;
m_wBuf[6]=1;
MinWidth=100;
ReadPresetCounterValue_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive, MinWidth);
Close_Com(3);
```

Remark:

■ ReadMinSignalWidth_7080

Description:

Obtain the setting width value of the minimum high or low input signal level in I-7080 module.

Syntax:

```
[C++]  
WORD ReadMinSignalWidth_7080 (WORD wBuf[], float fBuf[], char szSend[],  
char szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] 0 → set the min. width at low level 1 → set the min. width at high level
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
wBuf[7]:	[Output] The input Signal Min Width
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7080Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];
```

```
char m_szReceive[100];  
WORD MinWidth;  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7080Addr;  
m_wBuf[2]=0x7080;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[5]=0;  
m_wBuf[6]=1;  
ReadPresetCounterValue_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive, MinWidth);  
MinWidth= m_wBuf[7];  
Close_Com(3);
```

Remark:

■ SetGateMode_7080

Description:

Configure the gate control mode of I-7080 module. There are 3 type modes:

- 0 → gate input signal must be low to enable counter
- 1 → gate input signal must be high to enable counter
- 2 → gate input signal is ignored. The counter will be always enable

Syntax:

```
[C++]  
WORD SetGateMode_7080 (WORD wBuf[], float fBuf[], char szSend[], char  
szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] 0 → the gate is low active 1 → the gate is high active 2 → the gate is disable
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7080Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];
```

```
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7080Addr;
m_wBuf[2]=0x7080;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[5]=0;
m_wBuf[6]=1;
SetGateMode_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ReadGateMode_7080

Description:

Obtain the setting status of the gate control mode in I-7080 module.

Syntax:

```
[C++]  
WORD ReadGateMode_7080 (WORD wBuf[], float fBuf[], char szSend[], char  
szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Output] 0 → the gate is low active 1 → the gate is high active 2 → the gate is disable
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7080Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];
```

```
WORD GateMode;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7080Addr;
m_wBuf[2]=0x7080;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[6]=1;
ReadGateMode_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);
GateMode=m_wBuf[5];
Close_Com(3);
```

Remark:

■ ReadOutputAlarmState_7080

Description:

Obtain the alarm digital output and the corresponding alarm setting status of I-7080 module.

Syntax:

[C++]

WORD ReadOutputAlarmState_7080 (WORD wBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	Not used
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
wBuf[7]:	[Output] For 7080 mode(alarm mode 0) 0 → Counter:0 disable Counter:1 disable 1 → Counter:0 enable Counter:1 disable 2 → Counter:0 disable Counter:1 enable 3 → Counter:0 enable Counter:1 enable For 7080D mode (alarm mode 1) 0 → Counter:0 disable 1 → Counter:0 momentary alarm mode 2 → Counter:0 latch alarm mode Counter: 1 No used in Alarm mode 1
wBuf[8]:	[Output] Alarm digital output 0 → DO:0 off DO:1 off 1 → DO:0 on DO:1 off 2 → DO:0 off DO:1 on 3 → DO:0 on DO:1 on
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.

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

Return Value:

NoError: OK
Others: Error code

Example:

```
WORD m_ComPort=3;
WORD m_7080Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
WORD AlarmMode;
WORD DO;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7080Addr;
m_wBuf[2]=0x7080;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[6]=1;
ReadOutputAlarmState_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);
AlarmMode=m_wBuf[7];
DO=m_wBuf[8];
Close_Com(3);
```

Remark:

■ ReadAlarmLimitValue_7080

Description:

Obtain the maximum value of the can read alarm limit value of I-7080 (for alarm-mode 0).

Syntax:

[C++]

WORD ReadAlarmLimitValue_7080 (WORD wBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] 0 → Read counter 0 1 → Read counter 1
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
wBuf[7]:	[Output] The high word of counter value
wBuf[8]:	[Output] The low word of counter value
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7080Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
```

```
char m_szSend[100];
char m_szReceive[100];
WORD AlarmValueH;
WORD AlarmValueL;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7080Addr;
m_wBuf[2]=0x7080;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[5]=0;
m_wBuf[6]=1;
ReadAlarmLimitValue_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);
AlarmValueH=m_wBuf[7];
AlarmValueL=m_wBuf[8];
Close_Com(3);
```

Remark:

■ SetAlarmLimitValue_7080

Description:

Users can set alarm limit value for I-7080 (for alarm-mode 0).

Syntax:

[C++]

WORD SetAlarmLimitValue_7080 (WORD wBuf[], float fBuf[], char szSend[], char szReceive[], unsigned long AlarmValue)

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] When in 7080 alarm mode(mode 0) 0: To set Counter 0 alarm value 1: To set Counter 1 alarm value When in 7080D alarm mode (mode 0) 0: To set Counter 0 high alarm value 1: To set Counter 0 high-high alarm value
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7080Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
```

```
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
unsigned long AlarmValue;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7080Addr;
m_wBuf[2]=0x7080;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[5]=0;
m_wBuf[6]=1;
AlarmValue=100;
SetAlarmLimitValue_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive, AlarmValue);
Close_Com(3);
```

Remark:

■ ReadCounterStatus_7080

Description:

Obtain the counter working status (reading/stop) of I-7080 module.

Syntax:

[C++]

WORD ReadCounterStatus_7080 (WORD wBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] 0 → Read counter 0 status 1 → Read counter 1 status
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
wBuf[7]:	[Output] 0 → Counting. 1 → Stop.
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7080Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];  
WORD CounterStatus;  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7080Addr;  
m_wBuf[2]=0x7080;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[5]=0;  
m_wBuf[6]=1;  
ReadCounterStatus_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
CounterStatus =m_wBuf[7];  
Close_Com(3);
```

Remark:

■ SetConfiguration_7080

Description:

Set the configuration of I-7080 or I-7080D module.

Syntax:

[C++]

WORD SetConfiguration_7080 (WORD wBuf[], float fBuf[], char szSend[], char szReceive[])

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] Desired frequency gate time: 0 → 0.1 second 1 → 1.0 second Don't care wBuf[5],if set the module in Counter mode
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
wBuf[7];	[Input] Desired new address
wBuf[8];	[Input] Desired Type 0 → Counter mode 1 → Frequency mode
wBuf[9];	[Input] Desired Baudrate: 3: 1200 BPS 4: 2400 BPS 5: 4800 BPS 6: 9600 BPS 7: 19200 BPS 8: 38400 BPS 9: 57600 BPS 10: 115200 BPS
wBuf[10];	[Input] Desired Checksum Address
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError: OK
Others: Error code

Example:

```
WORD m_ComPort=3;
WORD m_7080Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7080Addr;
m_wBuf[2]=0x7080;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[5]=0;
m_wBuf[6]=1;
m_wBuf[7]=0;
m_wBuf[8]=0;
m_wBuf[9]=6;
m_wBuf[10]=0;
SetConfiguration_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ DataToLED_7080

Description:

Output the defined data to LED display of I-7080D module.

Syntax:

```
[C++]  
WORD DataToLed_7080 (WORD wBuf[], float fBuf[], char szSend[], char  
szReceive[])
```

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7080
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	Not used
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0]:	[Input] Output data to LED display
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7080Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
Open_Com(3, 9600, 8, 0, 0);
```

```
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7080Addr;
m_wBuf[2]=0x7080;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[6]=1;
m_fBuf[0]=2.5;
DataToLED_7080(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ReadCounter

Description:

This function is used for the Up Counter mode to read the Up Counter value.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x7080, 0x87082, 0x8080
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] channel number
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceiveseries
dwBuf[7]:	[Input] Slot number 1~7
dwBuf[8]:	[Input] 0 → Read counter 1 → Read counter and clear counter 2 → Read counter and overflow
dwBuf[9]:	[Output] Counter Value
dwBuf[10]:	[Output] Overflow Value
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_87082Addr=1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;
```

```
DWORD m_dwBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
DWORD Counter;  
DWORD Overflow;  
Open_Com(3, 9600, 8, 0, 0);  
m_dwBuf[0]=m_ComPort;  
m_dwBuf[1]= m_87082Addr;  
m_dwBuf[2]=0x87082;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[5]=0;  
m_dwBuf[6]=1;  
m_dwBuf[7]=1;  
m_dwBuf[8]=2;  
ReadCounter(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
Counter=m_dwBuf[9];  
Overflow=m_dwBuf[10];  
Close_Com(3);
```

Remark:

■ ReadCounter_All

Description:

This function is used for the Up Counter mode to read all channel.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x7080, 0x87082, 0x8080
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] channel number
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceiveseries
dwBuf[7]:	[Input] Slot number 1~7
dwBuf[8]:	[Input] 0 → Read counter 1 → Read counter and overflow
dwBuf[9]:	[Output] Counter Value for channel 0
dwBuf[10]:	[Output] Counter Value for channel 1
dwBuf[11]:	[Output] Counter Value for channel 2
dwBuf[12]:	[Output] Counter Value for channel 3
dwBuf[13]:	[Output] Counter Value for channel 4
dwBuf[14]:	[Output] Counter Value for channel 5
dwBuf[15]:	[Output] Counter Value for channel 6
dwBuf[16]:	[Output] Counter Value for channel 7
dwBuf[17]:	[Output] Overflow Value for channel 0
dwBuf[18]:	[Output] Overflow Value for channel 1
dwBuf[19]:	[Output] Overflow Value for channel 2
dwBuf[20]:	[Output] Overflow Value for channel 3
dwBuf[21]:	[Output] Overflow Value for channel 4
dwBuf[22]:	[Output] Overflow Value for channel 5
dwBuf[23]:	[Output] Overflow Value for channel 6

dwBuf[24]: [Output] Overflow Value for channel 7
fBuf: Not used
szSend: [Input] Command string to be sent to I-7000 series modules.
szReceive: [Output] Result string receiving from I-7000 series modules.

Return Value:

NoError: OK
Others: Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87082Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[36];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
DWORD Counter[8];
DWORD Overflow[8];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87082Addr;
m_dwBuf[2]=0x87082;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=0;
m_dwBuf[6]=1;
m_dwBuf[7]=1;
m_dwBuf[8]=2;
ReadCounter_All(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Counter[0]=m_dwBuf[9];
Counter[1]=m_dwBuf[10];
Counter[2]=m_dwBuf[11];
Counter[3]=m_dwBuf[12];
Counter[4]=m_dwBuf[13];
Counter[5]=m_dwBuf[14];
Counter[6]=m_dwBuf[15];
Counter[7]=m_dwBuf[16];
```

```
Overflow[0]=m_dwBuf[17];  
Overflow[1]=m_dwBuf[18];  
Overflow[2]=m_dwBuf[19];  
Overflow[3]=m_dwBuf[20];  
Overflow[4]=m_dwBuf[21];  
Overflow[5]=m_dwBuf[22];  
Overflow[6]=m_dwBuf[23];  
Overflow[7]=m_dwBuf[24];  
Close_Com(3);
```

Remark:

■ ReadFreq

Description:

This function is used to read the Measured Frequency value in the Frequency mode.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x7080, 0x87082, 0x8080
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] channel number
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceiveseries
dwBuf[7]:	[Input] Slot number 1~7
dwBuf[9]:	[Output] Frequency Value
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87082Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];  
DWORD Frequency;  
Open_Com(3, 9600, 8, 0, 0);  
m_dwBuf[0]=m_ComPort;  
m_dwBuf[1]= m_87082Addr;  
m_dwBuf[2]=0x87082;  
m_dwBuf[3]=m_Checksum;  
m_dwBuf[4]=m_TimeOut;  
m_dwBuf[5]=0;  
m_dwBuf[6]=1;  
m_dwBuf[7]=1;  
ReadFreq(m_dwBuf, m_fBuf, m_szSend, m_szReceive);  
Frequency =m_dwBuf[9];  
Close_Com(3);
```

Remark:

■ ReadFreq_All

Description:

This function is used to read all channel in the Frequency mode.

Syntax:

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x7080, 0x87082, 0x8080
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] channel number
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceiveseries
dwBuf[7]:	[Input] Slot number 1~7
dwBuf[9]:	[Output] Frequency Value for channel 0
dwBuf[10]:	[Output] Frequency Value for channel 1
dwBuf[11]:	[Output] Frequency Value for channel 2
dwBuf[12]:	[Output] Frequency Value for channel 3
dwBuf[13]:	[Output] Frequency Value for channel 4
dwBuf[14]:	[Output] Frequency Value for channel 5
dwBuf[15]:	[Output] Frequency Value for channel 6
dwBuf[16]:	[Output] Frequency Value for channel 7
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
```

```
DWORD m_87082Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[36];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
DWORD Frequency[8];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87082Addr;
m_dwBuf[2]=0x87082;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=0;
m_dwBuf[6]=1;
m_dwBuf[7]=1;
m_dwBuf[8]=2;
ReadFreq_All(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Frequency[0]=m_dwBuf[9];
Frequency[1]=m_dwBuf[10];
Frequency[2]=m_dwBuf[11];
Frequency[3]=m_dwBuf[12];
Frequency[4]=m_dwBuf[13];
Frequency[5]=m_dwBuf[14];
Frequency[6]=m_dwBuf[15];
Frequency[7]=m_dwBuf[16];
Close_Com(3);
```

Remark:

■ ClearCounter

Description:

This function is used to clear the specific channel counter value in a specific slot.

Syntax:

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x7080, 0x87082, 0x8080
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] channel number
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceiveseries
dwBuf[7]:	[Input] Slot number 1~7
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;  
DWORD m_87082Addr=1;  
DWORD m_TimeOut=100;  
DWORD m_Checksum=0;  
DWORD m_dwBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
Open_Com(3, 9600, 8, 0, 0);
```

```
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87082Addr;
m_dwBuf[2]=0x87082;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=0;
m_dwBuf[6]=1;
m_dwBuf[7]=1;
ClearCounter(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ SetCounterStatus

Description:

This function obtains the counter working status (reading/stop) of 0x7080, 0x87082, 0x8080 modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x7080, 0x87082, 0x8080
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] channel number
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceiveseries
dwBuf[7]:	[Input] Slot number 1~7
dwBuf[8]:	[Input] Counter or Frequency Status
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87082Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87082Addr;
m_dwBuf[2]=0x87082;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=0;
m_dwBuf[6]=1;
m_dwBuf[7]=1;
m_dwBuf[8]=1;
SetCounterStatus(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ReadCounterStatus

Description:

This function obtains the counter working status (reading/stop) of 0x7080, 0x87082, 0x8080 modules.

Syntax:

[C++]

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

Parameter:

dwBuf:	DWORD Input/Output argument table
dwBuf[0]:	[Input] COM port number, 1 to 255
dwBuf[1]:	[Input] Module address, from 0x00 to 0xFF
dwBuf[2]:	[Input] Module ID, 0x7080, 0x87082, 0x8080
dwBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
dwBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
dwBuf[5]:	[Input] channel number
dwBuf[6]:	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceiveseries
dwBuf[7]:	[Input] Slot number 1~7
dwBuf[8]:	[Output] Counter or Frequency Status
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
DWORD m_ComPort=3;
DWORD m_87082Addr=1;
DWORD m_TimeOut=100;
DWORD m_Checksum=0;
DWORD m_dwBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];
DWORD Status;
Open_Com(3, 9600, 8, 0, 0);
m_dwBuf[0]=m_ComPort;
m_dwBuf[1]= m_87082Addr;
m_dwBuf[2]=0x87082;
m_dwBuf[3]=m_Checksum;
m_dwBuf[4]=m_TimeOut;
m_dwBuf[5]=0;
m_dwBuf[6]=1;
m_dwBuf[7]=1;
SetCounterStatus(m_dwBuf, m_fBuf, m_szSend, m_szReceive);
Status=m_dwbuf[8];
Close_Com(3);
```

Remark:

5.8 Dual Watchdog Functions

All ICPDAS DCON (I-7000/8000/87K) series modules equip a hardware module watchdog and software host watchdog. The DCON series modules are designed for industry applications, therefore they can work in the harsh environment. About the detail “Dual Watchdog “ description please refer to “Appendix A”.

■ HostIsOK

Description:

This function provides a method to tell all module “Host PC is OK” by sending command string “~***”. If the module can’t receive “~***” during a time interval, the host “WatchDog” function will be enabled. The related functions are “ToSetupHostWatchdog” and “ToReadHostWatchdog”.

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	Not used
wBuf[2]:	Not used
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	Not used
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[6]=1;
HostIsOK(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ToSetupHostWatchdog

Description:

Configure the timer working interval of the Host Watchdog for I-7000 series modules. Also, it can enable or disable “WatchDog” function.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	Not used
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] 0 → Disable host watchdog 1 → Enable host watchdog If TimeOut value = 0 then disable the Host Watchdog
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
wBuf[7]:	[Input] Timer setting for watchdog, unit is 0.1 second, For example: when wBuf[7]=45, the timer interval is 4.5 second
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
```

```
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= 1;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[5]=1;
m_wBuf[6]=1;
m_wBuf[7]=30;          //when wBuf[7]=30, the timer interval is 3 second
ToSetupHostWatchdog(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ToReadHostWatchdog

Description:

Obtain the setting value of timer interval and WatchDog status of the module for I-7000 series modules.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	Not used
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Output] 0 → Disable host watchdog 1 → Enable host watchdog
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
wBuf[7]:	[Output] Timer setting for watchdog, unit is 0.1 second, For example: when wBuf[7]=45, the timer interval is 4.5 second
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];  
WORD WatchdogStatus;  
WORD WatchdogTime;  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= 1;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[6]=1;  
ToReadHostWatchdog(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
WatchdogStatus=m_wBuf[5];  
WatchdogTime=m_wBuf[7];  
Close_Com(3);
```

Remark:

■ ReadModuleResetStatus

Description:

Obtain the module reset status. If the return value is “0”, it means “module has not been reset”. If the return value is “1”, it means “module has been reset”.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011, 0x7012
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Output] 0 → module has not been reset 1 → module has been reset
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7011Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
```

```
WORD ResetStatus;  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7011Addr;  
m_wBuf[2]=0x7011;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[6]=1;  
ReadModuleResetStatus(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
ResetStatus= m_wBuf[5];  
Close_Com(3);
```

Remark:

■ ReadModuleHostWatchdogStatus

Description:

Obtain the module's Host Watchdog status for I-7000 series modules. If the return value is "0", it means "Module's Host watchdog is in NORMAL mode". If the return value is "4", it means "Module's Host watchdog is in HOST FAILURE mode".

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	Not used
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Output] 0 → NORMAL mode. 4 → HOST FAILURE mode.
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
```

```
WORD WatchdogStatus;  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= 1;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[6]=1;  
ReadModuleHostWatchdogStatus(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
WatchdogStatus= m_wBuf[5];  
Close_Com(3);
```

Remark:

■ ResetModuleHostWatchdogStatus

Description:

Reset the module's Host Watchdog status for I-7000 series modules. The related function is "ReadModuleHostWatchdogStatus".

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	Not used
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	Not used
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
```

```
m_wBuf[1]= 1;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[6]=1;  
ResetModuleHostWatchdogStatus(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ SetSafeValueForDo

Description:

Configure the safe value of DO modules for I-7000 series modules when “WatchDog” function of the module has been enabled.

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7050/60/63/66/67/42/43/44
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] Safe digital output value in Hex format
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7050Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
Open_Com(3, 9600, 8, 0, 0);
```

```
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7050Addr;
m_wBuf[2]=0x7050;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[5]=0x03;
m_wBuf[6]=1;
SetSafeValueForDo(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ SetPowerOnValueForDo

Description:

Configure the initial digital output value of digital output module for I-7000 series modules when its power is on.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7050/60/63/66/67/42/43/44
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] Power On value in Hex format
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7050Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
WORD ResetStatus;
```

```
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7050Addr;  
m_wBuf[2]=0x7050;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[5]=0x03;  
m_wBuf[6]=1;  
SetPowerOnValueForDo(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ SetSafeValueForAo

Description:

Configure the channel No.safe value of analog output module for I-7000 series modules when the “WatchDog” function of the module has been enabled.

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7021/0x7024
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] The analog output channel No.
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0]:	[Input] Analog output safe value
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7024Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];
```

```
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7024Addr;
m_wBuf[2]=0x7024;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[5]=0;
m_wBuf[6]=1;
m_fBuf[0]=2.5;
SetSafeValueForAo(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ SetPowerOnValueForAo

Description:

Configure the initial analog output of analog output module for I-7024 module when its power is on.

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7021/0x7024
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] The analog output channel No.
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0]:	[Input] Power On analog output value
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7024Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];
```

```
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7024Addr;  
m_wBuf[2]=0x7024;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[5]=0;  
m_wBuf[6]=1;  
m_fBuf[0]=2.5;  
SetPowerOnValueForAo(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
Close_Com(3);
```

Remark:

■ SetPowerOnSafeValue

Description:

Configure the power on and safe value of the digital output channels for I-7011, I-7012, and I-7014 modules.

Power On value:

00 → DO0 off , DO1 off ; 01 → DO0 on , DO1 off ;
02 → DO0 off , DO1 on ; 03 → DO0 on , DO1 on ;

Safe value:

00 → DO0 off , DO1 off ; 01 → DO0 on , DO1 off ;
02 → DO0 off , DO1 on ; 03 → DO0 on , DO1 on ;

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7011/0x7012/0x7014
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] Power On value in hex format
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
wBuf[7] :	[Input] Safe value in hex format.
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7014Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7014Addr;
m_wBuf[2]=0x7014;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[5]=0;
m_wBuf[6]=1;
m_wBuf[7]=0x03;
SetPowerOnSafeValue(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Close_Com(3);
```

Remark:

■ ReadSafeValueForAo

Description:

Obtain the safe setting value of analog output module for I-7000 series modules.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7021/0x7022/0x7024
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] Not used if module ID is 7021 Channel No (0 to 1) if module ID is 7022 Channel No (0 to 3) if module ID is 7024
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0]:	[Output] Safe value
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7024Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
```

```
char m_szReceive[100];
float SafeValue;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7024Addr;
m_wBuf[2]=0x7024;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[5]=0;
m_wBuf[6]=1;
ReadSafeValueForAo(m_wBuf, m_fBuf, m_szSend, m_szReceive);
SafeValue=m_fBuf[0];
Close_Com(3);
```

Remark:

■ ReadPowerOnValueForAo

Description:

Obtain the initial output setting value of analog output module for I-7000 series module when the power is on.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7024
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Input] Channel No.(0 to 3) if module ID is 7024
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Float Input/Output argument table
fBuf[0]:	[Output] The initial output value when the power is on.
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7024Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
```

```
float PowerOnValue;  
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7024Addr;  
m_wBuf[2]=0x7024;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[5]=0;  
m_wBuf[6]=1;  
ReadPowerOnValueForAo(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
PowerOnValue=m_fBuf[0];  
Close_Com(3);
```

Remark:

■ ReadPowerOnValueForDo

Description:

Obtain the initial output setting value of digital output module for I-7000 series module when the power is on.

Syntax:

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7050/60/63/65/66/67/42/43/44
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Output] Power on value in hex format
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;  
WORD m_7050Addr=1;  
WORD m_TimeOut=100;  
WORD m_Checksum=0;  
WORD m_wBuf[12];  
float m_fBuf[12];  
char m_szSend[100];  
char m_szReceive[100];  
WORD PowerOnValue;
```

```
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7050Addr;  
m_wBuf[2]=0x7050;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[6]=1;  
ReadPowerOnValueForDo(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
PowerOnValue=m_wBuf[5];  
Close_Com(3);
```

Remark:

■ ReadSafeValueForDo

Description:

Obtain the safe setting value of digital output module for I-7000 series modules when “WatchDog” function of the module has been enabled.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID, 0x7050/60/63/65/66/67/42/43/44
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	[Output] Safe value in hex format
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7050Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
WORD SafeValue;
```

```
Open_Com(3, 9600, 8, 0, 0);  
m_wBuf[0]=m_ComPort;  
m_wBuf[1]= m_7050Addr;  
m_wBuf[2]=0x7050;  
m_wBuf[3]=m_Checksum;  
m_wBuf[4]=m_TimeOut  
m_wBuf[6]=1;  
ReadSafeValueForDo(m_wBuf, m_fBuf, m_szSend, m_szReceive);  
SafeValue=m_wBuf[5];  
Close_Com(3);
```

Remark:

■ ReadConfigStatus

Description:

Obtain the configuration status of the module for I-7000 series modules. For more detail information of the parameter please refer to the user's manual.

Syntax:

[C++]

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

Parameter:

wBuf:	WORD Input/Output argument table
wBuf[0]:	[Input] COM port number, 1 to 255
wBuf[1]:	[Input] Module address, from 0x00 to 0xFF
wBuf[2]:	[Input] Module ID (for all modules)
wBuf[3]:	[Input] 0=checksum disable, 1=checksum enable
wBuf[4]:	[Input] Time out setting, normal=100, unit=ms.
wBuf[5] :	Not used
wBuf[6] :	[Input] 0 → no save to szSend & szReceive 1 → save to szSend & szReceive
wBuf[7]:	[Output] Module address
wBuf[8]:	[Output] Module Range Code
wBuf[9]:	[Output] Module baudrate
wBuf[10]:	[Output] Module data format
fBuf:	Not used
szSend:	[Input] Command string to be sent to I-7000 series modules.
szReceive:	[Output] Result string receiving from I-7000 series modules.

Return Value:

NoError:	OK
Others:	Error code

Example:

```
WORD m_ComPort=3;
WORD m_7050Addr=1;
WORD m_TimeOut=100;
WORD m_Checksum=0;
WORD m_wBuf[12];
```

```
float m_fBuf[12];
char m_szSend[100];
char m_szReceive[100];
WORD Address;
WORD RangeCode;
WORD Baudrate;
WORD DataFormat;
Open_Com(3, 9600, 8, 0, 0);
m_wBuf[0]=m_ComPort;
m_wBuf[1]= m_7050Addr;
m_wBuf[2]=0x7050;
m_wBuf[3]=m_Checksum;
m_wBuf[4]=m_TimeOut
m_wBuf[6]=1;
ReadConfigStatus(m_wBuf, m_fBuf, m_szSend, m_szReceive);
Address=m_wBuf[7];
RangeCode=m_wBuf[8];
Baudrate=m_wBuf[9];
DataFormat=m_wBuf[10];
Close_Com(3);
```

Remark:

APPENDIX A WatchDog

Operation Principle

All ICPDAS DCON (I-7000/8000/87K) series modules equip a hardware module watchdog and software host watchdog. The DCON series modules are designed for industry applications, therefore they can work in the harsh environment. The modules may be down if its application environment is very bad and produces the noise effect to not be overcome by modules. Therefore, **the built-in hardware module watchdog can reset the module if it is down under the effect of too large noise signal** (refer to Figure 5). Sometimes even the host-PC may be down for hardware or software reasons. **The software host watchdog can monitor the status of host PC.** This **host watchdog** can be applied to two situations:(1) If the host PC is down, all the output of DCON series modules will go to their predefined safe states for safety protection reason (refer to Figure 1 to 3).; (2) If the RS-485 network is broken, all the host command can not send to remote modules. This is very dangerous in real world application. The DCON series output modules will force their output going to their predefined safe state for safety consideration if the host watchdog is active. Therefore, these dual watchdog features, module and host watchdog, will increase the reliability of system.

1. **Host watchdog:** (software): If the host is down, all modules output go to their predefined safe value. (refer to Figure 1 to 4).
2. **Module watchdog:** (hardware):If the module is down, module will reset itself and output go to safe value. (refer to Figure 5 to 6).

■ Host WatchDog

The host may be down under the effect of the following condition:

1. Noise too large → make host hardware going error
2. Software problem → make host going to the dead lock state
3. Hardware problem → host hardware is damaged
4. The RS485 network is broken → can't send out command to modules

The software host watchdog is designed to monitor the host computer. If the host computer is down, the output value of the modules will automatically go to their predefined safe states to avoid unpredictable damaged. Followings are the three methods for Host watchdog after module host watchdog is enabled.

(1) Host PC sends command “~**” to every module to notify that the Host PC is OK. If module host watchdog is enabled, this command “~**” must be sent to the module within the timeout period of watchdog timer and reset timer to mean that Host PC is OK.



Figure 1. Host PC send command “~**” to every module to notify that the Host PC is OK.

(2) When the host PC command “~**” can not be sent to every module, which may be caused by host PC is down or RS-485 network is broken, that is, module watchdog timer can not be reset within default time period. Therefore, the module figures out the host PC is down and then it set the output value to the predefined safety value to avoid unpredictable damaged.

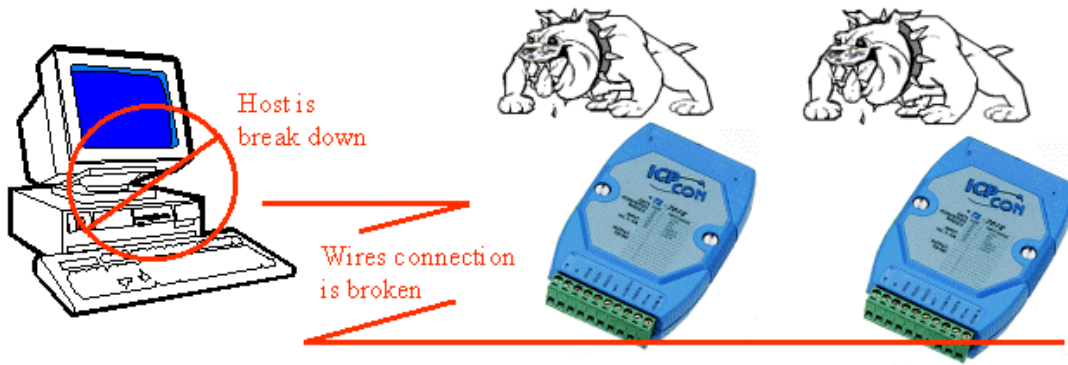


Figure 2. If Host PC can not send command to every module, then every Host Watchdog of the module set its output the predefined value for safety reason.

(3) Once the timer of host watchdog is not reset by host PC and module safety output is produced, then any command will be ignored by the module. In addition, if PC is work and try to control the module again, host PC can send command “~AA0” to read in the module status and then use command “ ~AA1” to clear the module status to let the module work again.



Figure 3. Host PC must use command ~AA0 to read the module status, and ~AA1 to clear the module status.

The flow chart of the host computer is given as Figure 4. Note that host OK command must be sent out to notify the module after every function sent.

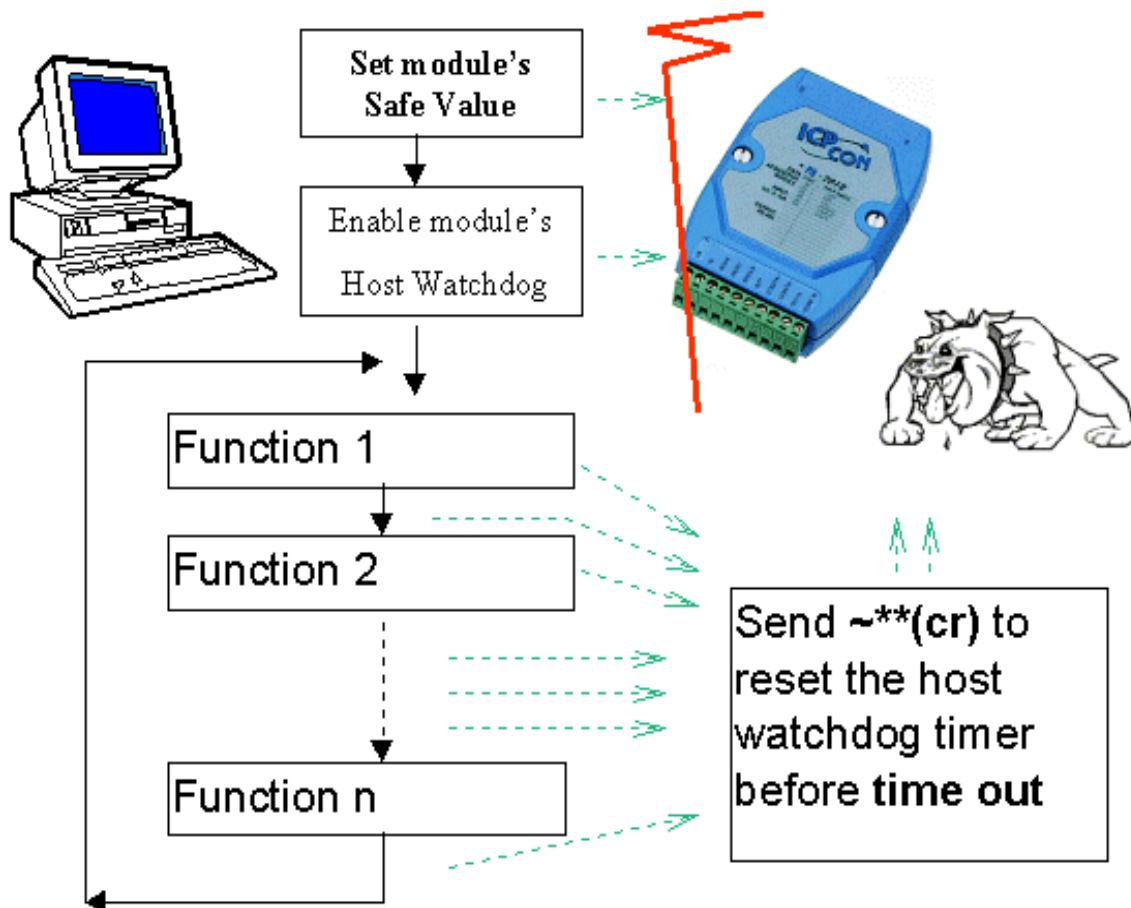


Figure 4. The flow chart of Host Watchdog.

■ Module WatchDog

The module watchdog is a hardware reset circuit to monitor the modules's operation status. While working in a harsh or noisy environment, the module may be down by the external signal. The circuit may let the module to work continuous and never halt.

The reset status is set while the module power on or reset by module watchdog. And it can be cleared while the command (\$AA5) of read reset status is applied. This command is useful for user to check the module working status. When the reset status is set means that the module is reset and the output may be change to the PowerOn value. When the reset status is clear means that the module is not reseted and the output is not change to PowerOn value. Note that the power on value can be set as different output value before module is reset. **Therefore, the user needs to send output command (\$AA5) to module for checking and keeping the same output state before and after module watchdog reset.**



Figure 5. Module Watchdog will reset the module when the module is hanged.

The flow chart of the failure detection for module hardware watchdog is given as Figure 6.

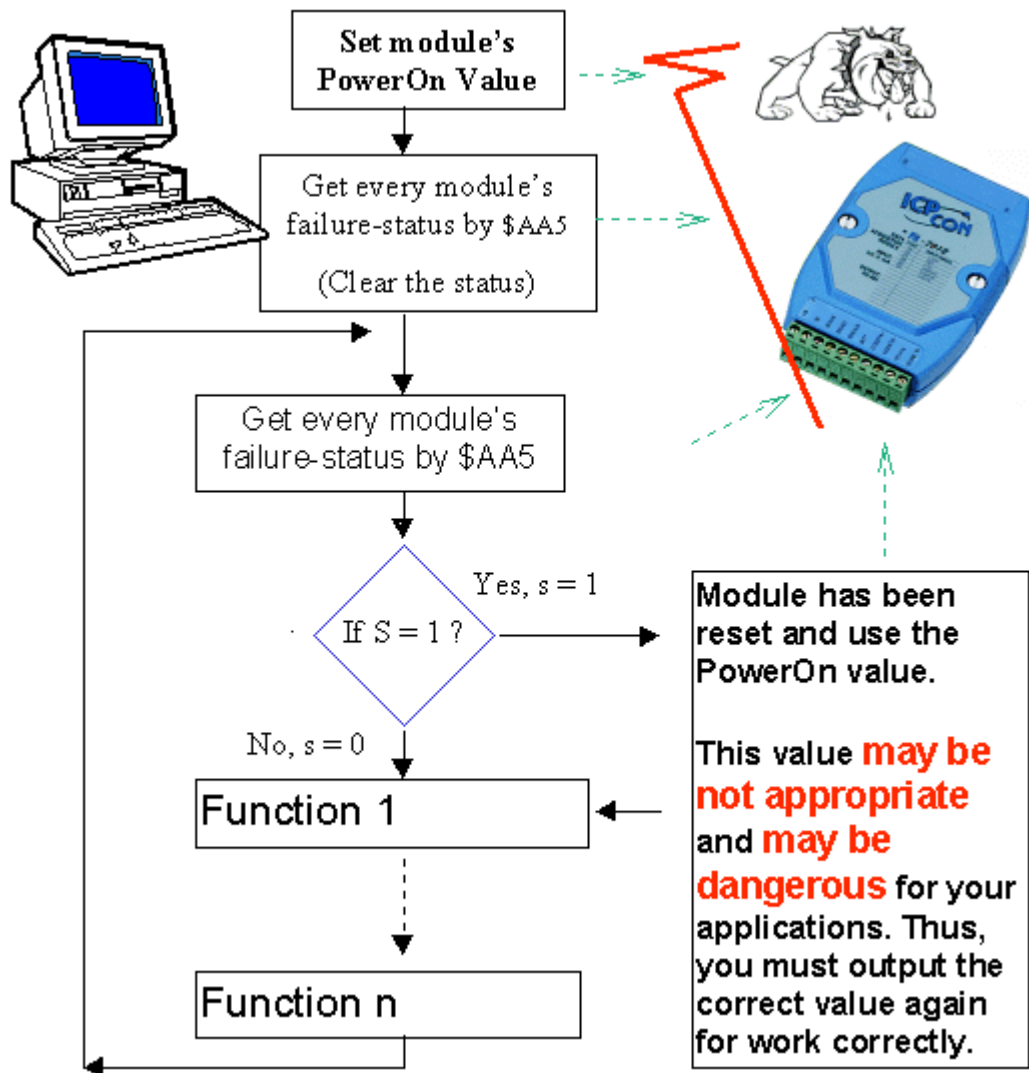


Figure 6. The flow chart of Module Watchdog.

■ Comparison of Host and Module Watchdog

	Host Watchdog	Module Watchdog
Software or Hardware	<ul style="list-style-type: none"> ● Software Watchdog ● Built-in firmware 	<ul style="list-style-type: none"> ● Hardware Watchdog ● Circuit in module
Purpose	<ul style="list-style-type: none"> ● Monitor the Host PC ● Used in all output modules 	<ul style="list-style-type: none"> ● Monitor the Module ● Used in all modules
When to occur	<ul style="list-style-type: none"> ● Host is down ● Communication line is broken 	<ul style="list-style-type: none"> ● Module is hanged ● Noise is too large in the working environment
What to do	<ul style="list-style-type: none"> ● Module go to safe state ● Module status S = 0x04 ● Module's output go to safe value ● All output command will be ignored. 	<ul style="list-style-type: none"> ● Reset the Module ● Module Reset status S = 0x01 ● Module's output go to PowerOn value
CLEAR module-status	<ul style="list-style-type: none"> ● ~AA1 ● S set to 0 	
READ module-status	<ul style="list-style-type: none"> ● ~AA0 ● S = 4 → Host is down ● S = 0 → Host is OK 	
READ module-reset status		<ul style="list-style-type: none"> ● \$AA5 ● S = 1 → Module Reset ● S = 0 → Not reset
Setup steps	<ul style="list-style-type: none"> ● Setup the safe value ● Setup the timer interval value of Host Watchdog and enable the Host Watchdog 	<ul style="list-style-type: none"> ● Setup the PowerOn value
Send "Host is OK"	<ul style="list-style-type: none"> ● ~** Send this command to modules before timeout of Host Watchdog's timer. 	

APPENDIX B Error Code

Error Code

NoError	0	Functions work normally.
FunctionError	1	Call wrong function error.
PortError	2	Use wrong COM Port error.
BaudRateError	3	Baud rate error.
DataError	4	Data Bit error.
StopError	5	Stop Bit error.
ParityError	6	Parity Bit error.
ChecksumError	7	Checksum mechanism error.
ComPortNotOpen	8	COM is not open error.
SendThreadCreateError	9	Send thread create error.
SendCmdError	10	Send command error.
ReadComStatusError	11	Read COM Port status error.
ResultStrCheckError	12	Result string check error.
CmdError	13	Command error.
TimeOut	15	TimeOut error.
ModuleIdError	17	Module ID error.
AdChannelError	18	Channel number error.
UnderInputRange	19	Under input range error.
ExceedInputRange	20	Exceed input range error.
InvalidateCounterNo	21	Invalidate counter number error.
InvalidateCounterValue	22	Invalidate counter value error
InvalidateGateMode	23	Invalidate gate mode error.
InvalidateChannelNo	24	Invalidate channel No error.
ComPortInUse	25	COM Port is in use 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 3.1, Windows for Workgroups, Windows NT 4.0, 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>