NAP7000P

User Manual

(Version 3.4)

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. assume 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, nor for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademark

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

License

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

Table of Contents

1 INTRODUCTION

There are six software packages, **7000 Utility**, **NAP7000S, NAP7000P, NAP7000V, NAP7000D and NAP7000X** are available for I-7000 series.

- The **7000 Utility** is a utility/diagnostic package for WINDOWS 95/98/NT user. With it, the user can setup the configuration of 7000 series module easily. The functions of **7000 Utility** are given as following:
    1. RS-232 Com port selection
    2. Search the 7000 series module connected in a PC
    3. To setup the configuration for a 7000 series module
    4. Module calibration
    5. analog Input/Output
    6. Digital Input/Output
    7. Hi/Lo alarm setting

- The **NAP7000S** is designed for DOS user. The NAP7000S provide utility, demo program, performance evaluation and data acquisition subroutines for the user to handle 7000 series modules. The NAP7000S can be executed under DOS, Windows 3.1, 95/98 and NT. All the NAP7000S program sources are given in the floppy disk.

- The **NAP7000D** is a hot-link DDE(Dynamic Data Exchange) Server. The DDE is a communication protocol that enables to exchange data between WINDOWS applications. The NAP7000D pass the data acquired from 7000 to the user application by means of the hot-link. The features of NAP7000D are given as following:
  When the hot-link established between NAP7000D(Server) to a user application(Client), the user application can receive the incoming data from NAP7000D without any further user involvement.
    1. Any Windows application that as a DDE client, such as Excel, LabView, TestPoint, FIX32, InTouch, ONSPECT, etc, can establish the hot-link to the NAP7000D. This means that a user can use these software package to communication with 7000 series module without any programming.
    2. The user may develop his DDE client applications with VC++, VB and Delphi.
    3. Many demo programs for NAP7000D are provided for easy starting.

- The **NAP7000V** is a collection of virtual instruments(VIs) designed for LabVIEW. The LabVIEW is an industrial's graphical programming system developed by National Instruments. These VIs will call UART.DLL & I7000.DLL automatically. The driver source of these VIs are also provided in the NAP7000V.

- The **NAP7000X** is an ActiveX Control (OCX) designed for Windows applications. With the NAP7000X, the user can develop the application rapidly. It support the standard OLE Container, such as VB5, Delphi3, BCB3…

- The **NAP7000P** is a Windows DLLs designed for Windows 95/98/NT user. It can be utilized by VC++, BC++, VB, Delphi, BC++ Builder. The features of NAP7000P are given as following:

1. provide general-purpose RS-232 application functions
2. provide general-purpose 7000 series module send/receive functions
3. provide high performance 7000 series module application functions
4. multi-speed demos
5. multi-data-format demos
6. lots of demo programs for VC++, VB, Delphi, BCB

1.1 INSTALLATION

- Installation Steps

step 1). Insert 'NAP7000P Setup' disk into floppy drive(either A: or B:).
step 2). clicking Start/Run in the task Bar
step 3). enter the path as:
A:\SETUP\SETUP.EXE (if floppy drive is A:)
B:\SETUP\SETUP.EXE (if floppy drive is B:)
step 4). Following those instructions in installation process to complete it.

[For Windows 95 User]
After installed, the UART.DLL & I7000.DLL will be copied into the folder of
C:\WINDOWS\SYSTEM

[For Windows NT User]
After installed, the UART.DLL & I7000.DLL will be copied into the folder of
C:\WINNT\SYSTEM32

1.2 README.TXT FILES

We will continuously upgrade this software, NAP7000P. Therefore some information may not be given in this manual. All the extra information will be given in the **readme.txt**. There are some **readme.txt** located in the companion floppy disk as following:

\NAP7000P\readme.txt → release notes of this version NAP7000P

\NAP7000P\readme.txt → for Windows 95/98/NT user
\NAP7000P\Demo\VC\readme.txt → VC++ demo program documentation
\NAP7000P\ Demo\VB\readme.txt → VB demo program documentation
\NAP7000P\ Demo\Delphi\readme.txt → Delphi demo program documentation
\NAP7000P\ Demo\BCB\readme.txt → BC Builder demo program documentation

The contents of readme.txt can be as following:

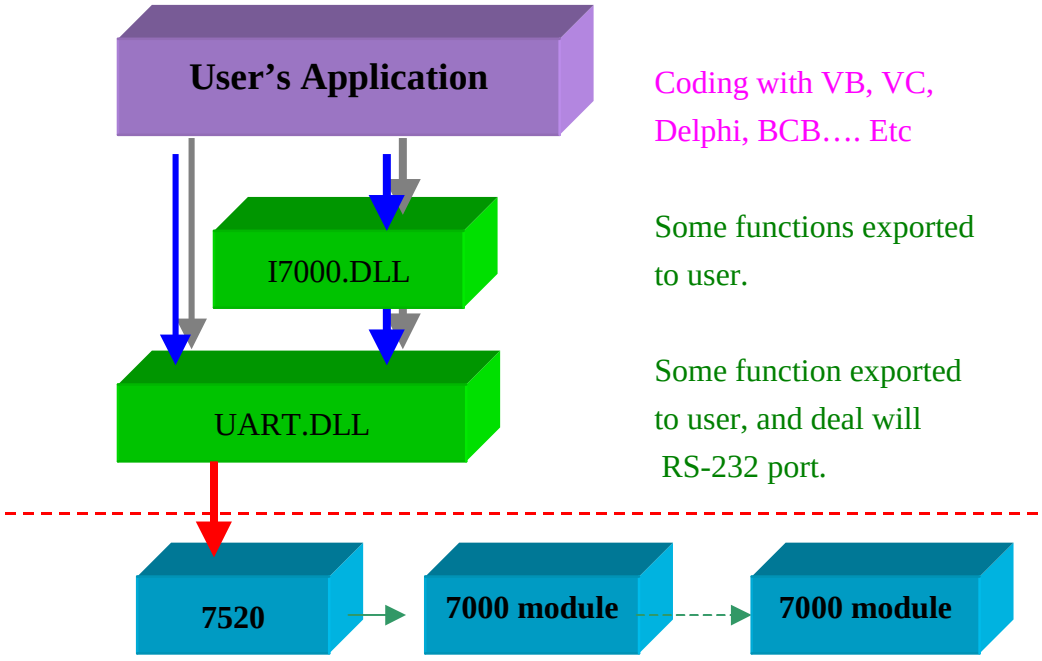
- release notes
- user manual
- demo program documentation
- compiler & link documentation
- application notes

It is recommended to read all the related readme.txt first before start to use this software.

1.3 ARCHITECTURES UNDER WINDOWS 95/NT APPLICATIONS

The **UART.DLL & I7000.DLL** are the dynamic linking library(DLL) designed for Windows 95/98 and Windows NT 3.51/4.0 applications. The user can use many programming languages such as VC++, BC++, BC++ Builder, VB, and Delphi to develop his application with UART.DLL and I7000.DLL. For your convenience, there are many demo program designed for VC++, VB, Delphi and BC++ Builder.

The relation between UART.DLL, I7000.DLL and user's application depicted as follows:



1.3.1 UART.DLL

The UART.DLL deal with PC's RS-232 port. The header file of UART.DLL is the I7000.H. The import library of UART.DLL is the UART.LIB. The related files are given in the NAP7000P as following:

(Refer to Sec. 1.3.3 for details.)

1.3.2 I7000.DLL

The I7000.DLL is designed for 7000 series applications. The I7000.DLL will call UART.DLL to send command and receive result to the 7000 series modules. The related files are given in the NAP7000P as following:

(Refer to Sec. 1.3.3 for details.)

1.3.3 INCLUDE FILES

The VC++ user has to include five files as following:

1. \NAP7000P\Driver\uart.dll → some functions to deal with RS-232
2. \NAP7000P\Driver\DLL\I7000.dll → some functions for A/D, D/A, D/I, D/O
3. \NAP7000P\Demo\VC\I7000.h → declaration file for uart.dll & I7000.dll
4. \NAP7000P\Demo\VC\uart.lib → import library of uart.dll
5. \NAP7000P\Demo\VC\I7000.lib → import library of I7000.dll

The BC++ & BC++ Builder user has to include five files as following:

1.\NAP7000P\Driver\uart.dll → RS232 related DLLs
2.\NAP7000P\Driver\I7000.dll → 7000 series related DLLs
3.\NAP7000P\Demo\ BCB\uartbc.lib → import library of uart.dll
4.\NAP7000P\ Demo\BCB\I7000bc.lib → import library of I7000.dll
5.\NAP7000P\ Demo\BCB\I7000.h → declaration file for uart.dll & I7000.dll
6.\NAP7000P\ Demo\BCB\I7000U.cpp → Some function for BCB

The VB user has to use four file as following:

1. \NAP7000P\Driver\uart.dll → RS232 related DLLs
2. \NAP7000P\Driver\I7000.dll → I7000 series related DLLs
3. \NAP7000P\Demo\VB\I7000.bas → declaration file for uart.dll & I7000.dll
4. \NAP7000P\Demo\VB\I7000u.bas → Some functions for VB

The Delphi user has to use three file as following:

1. \NAP7000P\Driver\uart.dll → RS232 related DLLs
2. \NAP7000P\Driver\I7000.dll → I7000 series related DLLs
3. \NAP7000P\Demo\Delphi\I7000.pas → declaration file for uart.dll & I7000.dll
4. \NAP7000P\Demo\Delphi\I7000u.pas → Some functions for Delphi

1.4 DEMO PROGRAM

There are many demo programs designed for VC++, VB, Delphi and BC++ Builder. These demo program are given in the NAP7000P as following:

1. \NAP7000P\DEMO\VC*.* → VC++ demo program
2. \NAP7000P\DEMO\VB*.* → VB demo program
3. \NAP7000P\DEMO\Delphi*.* → Delphi demo program
4. \NAP7000P\DEMO\BCB*.* → BC++ Builder demo program

These demp program are given as following:

Demo1 → send/receive command to 7000 with checksum disable
Demo2 → send/receive command to 7000 with checksum enable
Demo3 → send/receive command to Counter, ANC PC-202
Demo4 → send/receive command to OMRON PLC, CQM1 or C200
Demo5 → multi-speed demo
Demo6 → muti-data-format demo
Demo7 → multi-speed and multi-data-format demo

Demo20 → AnalogIn demo1
Demo21 → AnalogIn demo2
Demo23 → AnalogOut demo
Demo24 → DigitalIn demo
Demo25 → DigitalOut demo
Demo26 → multi-speed demo
Demo27 → AnalogIn8 demo
Demo28 → AnalogOutReadBack demo
Demo29 → DigitalOutReadBack demo

…….
…….
Refer to folder that you installed.

2 DLLs CALLING & DEMOs

Please refer to the following user manuals:

- **CallDll.pdf:**

  Describes how to call the DLL functions with VC++5, VB5, Delphi3 and Borland C++ Builder 3.

2.1 USING C / VISUAL C++

All the demo program given in \NAP7000P\Demo\Vc*.* are designed with VC++ language. It is testing OK under Windows 95/98/NT and Visual C++ 4.0 compiler. The key points are given as following:

1. Enter the DOS command prompt under Windows.
2. Make sure the environment variable, PATH, which include the Visual C++ compiler
3. Execute the \MSDEV\BIN\VCVARS32.BAT one time to setup the environment VARIABLE. The VCVARS32.BAT is provided by Visual C++.
4. The source program must include “I7000.H”
5. Copy the UART.LIB, I7000.LIB, UART.DLL and I7000.DLL to the same directory with source program
6. Edit the source program (refer to \nap7000p\Demo\vc\demo?\demo?.C)
7. Edit the NMAKE file
   (Please refer to \nap7000p\Demo\vc\demo?\demo?.MAK)
8. Edit the BATCH file (refer to \nap7000p\Demo\vc\demo?\c.bat)
9. Execute the batch file
10. Execute the execution file

2.2 USING MFC

The usage of NAP7000P DLLs for MFC user is very similar to that for C user. It is testing OK under Windows 95/98/NT and Visual C++ 4.0. The key points are given as following:

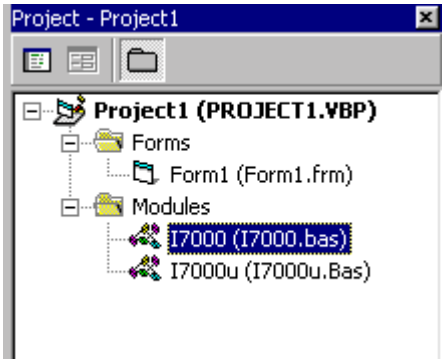
1. Use MFC wizard to create source code
2. The source program must include “I7000.H”
3. Copy the UART.LIB, I7000.LIB, UART.DLL and I7000.DLL to the same directory with source program
4. Select **Build/Settings/Link** and key-in “UART.LIB I7000.LIB” in the **object/library modules** field

2.3 USING VISUAL BASIC

\Driver\Uart.DLL → **DLLs**
\Driver\I7000.DLL → **DLLs**
\Demo\VB\I7000.BAS → declare file for I7000.DLL & Uart.DLL
\Demo\VB\I7000u.BAS → Some functions for VB

Note : Testing under Windows 95/98/NT and VB 5.0 (32 bits)

In the project file, we must include those declaration files.

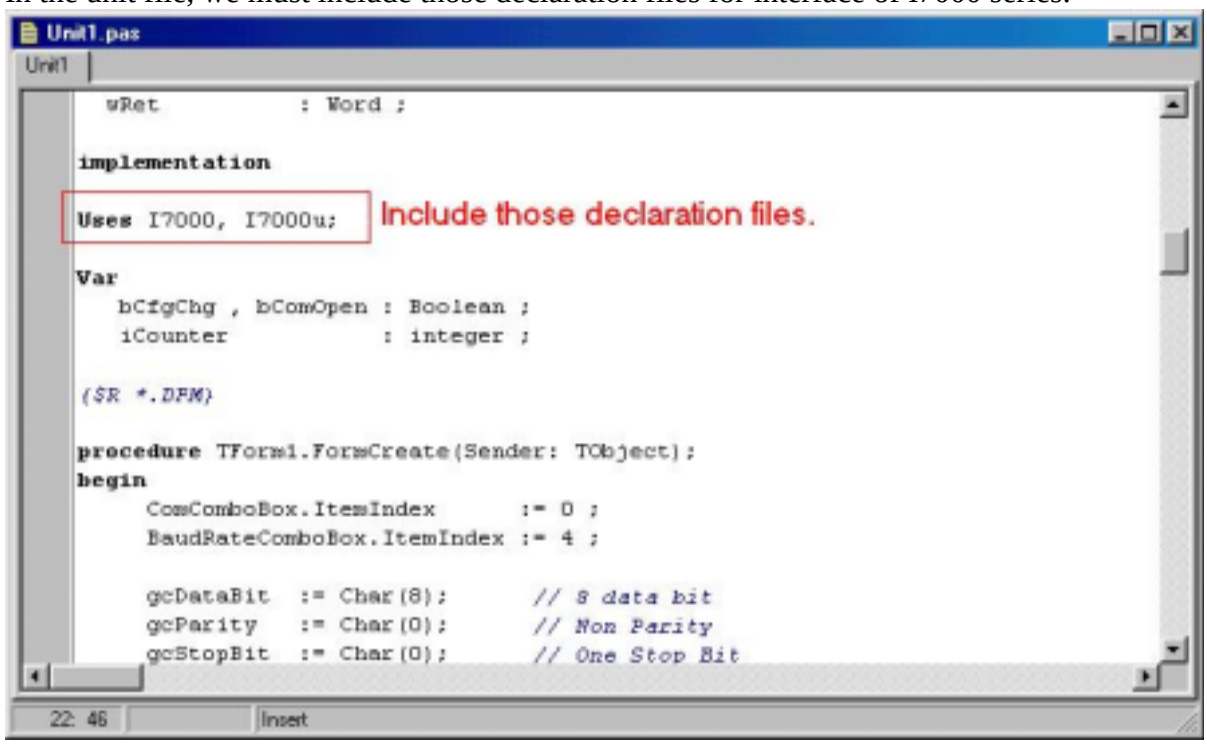


2.4 USING DELPHI

\Driver\Uart.DLL	→ **DLLs**
\Driver\I7000.DLL	→ **DLLs**
\Demo\Delphi\I7000.pas	→ declare file for I7000.dll & uart.dll
\Demo\Delphi\I7000u.pas	→ Some functions for Delphi.

Note : Testing under Windows 95/98/NT and Delphi 3.0 (32 bits)

In the unit file, we must include those declaration files for interface of I7000 series.

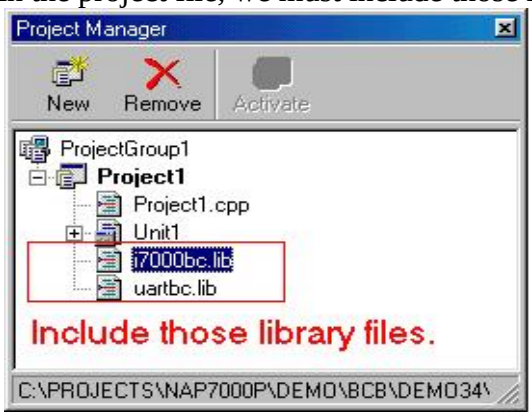


2.5 USING BORLAND C++ BUILDER

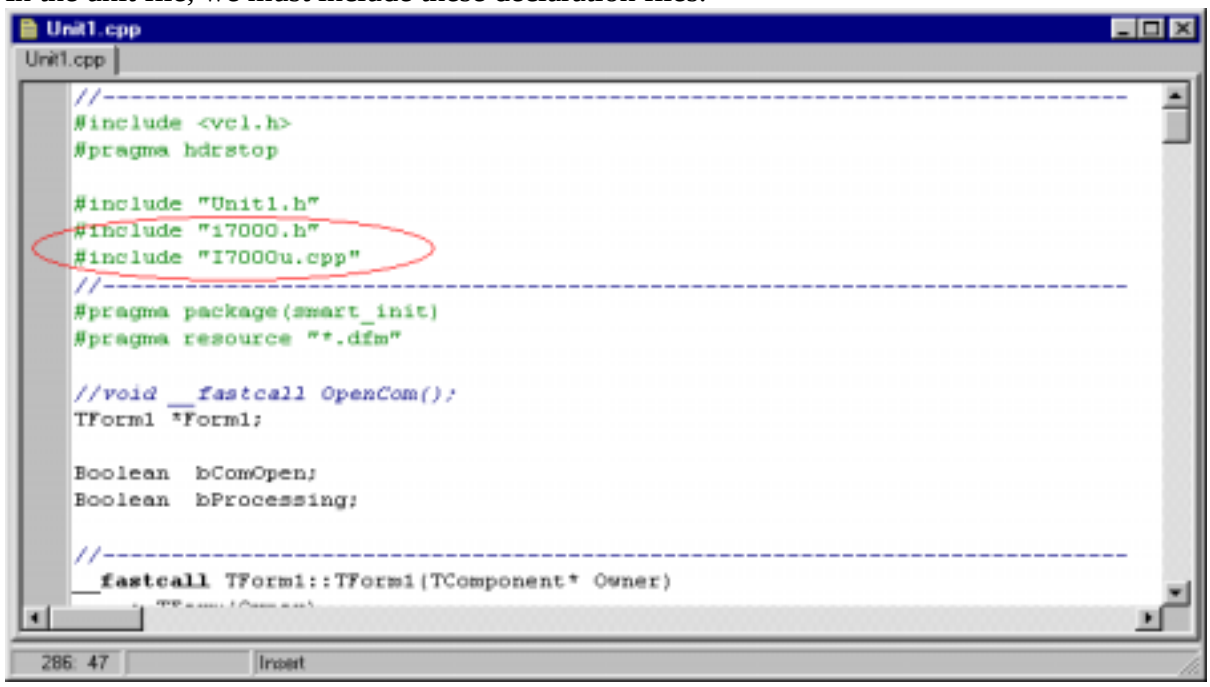
\Driver\Uart.DLL	→ **DLLs**
\Driver\I7000.DLL	→ **DLLs**
\Demo\BCB\I7000.h	→ declare file for I7000.dll & Uart.dll
\Demo\BCB\I7000U.cpp	→ Some functions for BCB
\Demo\BCB\Uartbc.Lib	→ LIBs
\Demo\BCB\I7000bc.Lib	→ LIBs

Note : Testing under Windows 95/98/NT and BC++Builder 3.0 (32 bits)

In the project file, we must include those library files.



In the unit file, we must include these declaration files.



2.6 VISUAL C++ DEMO PROGRAM

We use a common demo program for all C demo program. This demo program will accept the **cPort** and **wAddr** and **szSend** and call the different DLLs for usage demonstration.

Note 1: Some demo program does not use any of these three values

Note 2: Some demo program use these values in different name, format or functions.

Note 3: The demo program given in this section is used as reference. The user should refer to ????.C given in the floppy disk for details.

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include "I7000.h"
```

include

```
void    READ_CMD(char *);
WORD    ASCII_TO_HEX(char);
void    TEST_CMD(HWND, int, int, int, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

char        cPort=2; /* COM-2 */
WORD    wAddr=1; /* module address=1 */
char        szSend[80], szReceive[80];
int         iLine;

/* ---------------------------------------------------------------------- */

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                PSTR szCmdLine, int iCmdShow)
    {
    static char szAppName[] = "Demo1";
    HWND        hwnd ;
    MSG         msg ;
```

Default setting

```
    WNDCLASSEX  wndclass ;
    wndclass.cbSize        = sizeof(wndclass);
    wndclass.style         = CS_HREDRAW|CS_VREDRAW;
    wndclass.lpfnWndProc   = WndProc;
    wndclass.cbClsExtra    = 0;
    wndclass.cbWndExtra    = 0;
    wndclass.hInstance     = hInstance;
    wndclass.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
    wndclass.hCursor       = LoadCursor(NULL, IDC_ARROW);
    wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName  = NULL;
    wndclass.lpszClassName = szAppName;
    wndclass.hIconSm       = LoadIcon(NULL, IDI_APPLICATION);

    RegisterClassEx(&wndclass) ;
    hwnd=CreateWindow(szAppName,"cPort=[0]  wAddr=[2][3]  szSend=[5..]",
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;
    ShowWindow(hwnd,SW_SHOWMAXIMIZED);
    UpdateWindow(hwnd);

    while (GetMessage(&msg, NULL, 0, 0))
        {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
        }
    return msg.wParam;
    }

/* ---------------------------------------------------------------- */

LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam,
LPARAM lParam)
{
static int  cxChar, cyChar, cxClient, cyClient, cxBuffer, cyBuffer,
          xCaret, yCaret;
```

```
static char cBuf[80];
HDC         hdc;
TEXTMETRIC  tm;
PAINTSTRUCT ps;
int         i;

switch (iMsg)
 {
 case WM_CREATE :                    // window initial
     hdc=GetDC(hwnd);
     SelectObject(hdc,GetStockObject(SYSTEM_FIXED_FONT));
     GetTextMetrics(hdc, &tm);
     cxChar=tm.tmAveCharWidth;
     cyChar=tm.tmHeight;
     ReleaseDC(hwnd, hdc);
     szSend[0]=0;     /* initial command */
     return 0;
 case WM_SIZE :
     cxClient=LOWORD(lParam);        // window size in pixels
     cyClient=HIWORD(lParam);
     cxBuffer=max(1,cxClient/cxChar); // window size in characters
     cyBuffer=max(1,cyClient/cyChar);
     return 0;
 case WM_SETFOCUS :
     CreateCaret(hwnd, NULL, cxChar, cyChar);
     SetCaretPos(xCaret * cxChar, yCaret * cyChar);
     ShowCaret(hwnd);
     return 0;
 case WM_KILLFOCUS :
     HideCaret(hwnd);
     DestroyCaret();
     return 0;
 case WM_CHAR :                      // user press KEYBOARD
     for (i = 0 ; i < (int) LOWORD(lParam) ; i++)
     {
     switch (wParam)
       {
```

Default setting

```
 case '\b' :                // backspace pressed
  if (xCaret > 0 )
  {
  xCaret-- ;
  cBuf[xCaret]=' ';
  HideCaret(hwnd);
  hdc=GetDC(hwnd);
  SelectObject(hdc,GetStockObject(SYSTEM_FIXED_FONT));
  TextOut(hdc, xCaret * cxChar, yCaret * cyChar,cBuf+xCaret,1);
  ShowCaret(hwnd);
  ReleaseDC(hwnd, hdc);
  }
  break ;
 case '\r' :  // carriage return  pressed
  cBuf[xCaret]=0;
  if (xCaret!=0) {xCaret=0; yCaret++;}
  READ_CMD(cBuf);
  TEST_CMD(hwnd,xCaret, cxChar, yCaret,cyChar);
  xCaret=0; yCaret+=iLine;
  if (yCaret >= cyBuffer) InvalidateRect(hwnd, NULL, TRUE);
  break ;
 case '\x1B' :              // escape pressed
  InvalidateRect (hwnd, NULL, TRUE)  ;
  xCaret=yCaret=0;
  break ;
 default :                  // other KEY pressed
  cBuf[xCaret]=(char) wParam;
  HideCaret(hwnd) ;
  hdc=GetDC (hwnd);
  SelectObject(hdc,GetStockObject(SYSTEM_FIXED_FONT));
  TextOut(hdc,xCaret*cxChar,yCaret*cyChar,cBuf+xCaret,1);
  ShowCaret(hwnd);
  ReleaseDC(hwnd, hdc);
  xCaret++;
  break ;
 }
}
SetCaretPos(xCaret*cxChar, yCaret*cyChar);
```

[Back Space] key is pressed
cancer last input key

[Enter] key is pressed
process the [keyin string]
call the test subroutine

[ESC] key is pressed
clear last input string

keyboard is pressed
save the key

```
      return 0;
  case WM_PAINT :                    // clr and show HELP
      InvalidateRect(hwnd, NULL, TRUE) ;
      hdc=BeginPaint(hwnd, &ps);

      SelectObject(hdc,GetStockObject(SYSTEM_FIXED_FONT));
      sprintf(cBuf,"NOW --> cPort=%d, wAddr=%d,
szSend=[%s]",cPort,wAddr,szSend);
      TextOut(hdc,0,0,cBuf,strlen(cBuf));
      xCaret = 0 ; yCaret=1;
      SetCaretPos(0,yCaret*cyChar);

      EndPaint(hwnd, &ps);
      return 0;
  case WM_DESTROY :
      PostQuitMessage(0);
      return 0 ;
  }
return DefWindowProc(hwnd, iMsg, wParam, lParam);
}

/* ------------------------------------------------------------------ */
/* [0]=cPort, [2][3]=wAddr, [5...]=szSend[]                          */

void READ_CMD(char szCmd[])
{
WORD wT1,wT2,wT3;

if (szCmd[0]==0) return;        // only press [Enter]

cPort=ASCII_TO_HEX(szCmd[0]);     // HEX format

wT1=ASCII_TO_HEX(szCmd[2]) ;
wT2=ASCII_TO_HEX(szCmd[3]) ;
wAddr=wT1*16+wT2;
```

Client Region is changed

accept current setting of cPort,wAddr and szSend

Fixed format : cPort=[0]

[1]=SPACE

Fixed format : wAddr=[2][3]

[4]=SPACE

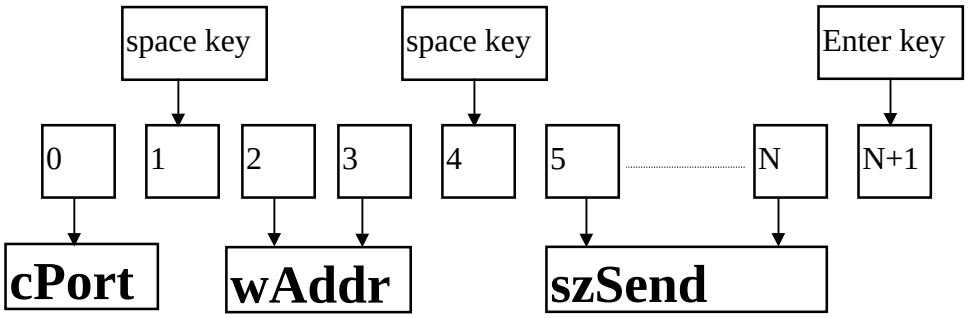
```
wT1=5; wT2=0;
for (;;)
   {
   if (szCmd[wT1]==0) break;
   szSend[wT2++]=szCmd[wT1++];
   }
szSend[wT2]=0;
}

WORD ASCII_TO_HEX(char cChar)
{
if (cChar<='9') return(cChar-'0');
else if (cChar<='F') return(cChar-'A'+10);
else return(cChar-'a'+10);
}

/* ---------------------------------------------------------------- */

void TEST_CMD(HWND hwnd, int x, int dx, int y, int dy)
{
```

Fixed format :
szSend=[5….]

Separate testing subroutine is putting here

```
}
```

The READ_COM only accept **fix format** command. The command format is given as below:



- if [0] = [Enter key] → accept current setting of **cPort,wAddr and szSend**

2.7 INCLUDE FILES

2.7.1 I7000.H

```
#define EXPORTS extern "C" __declspec (dllimport)
//#define EXPORTS

#define   Data5Bit                5
#define   Data6Bit                6
#define   Data7Bit                7
#define   Data8Bit                8

#define   NonParity               0
#define   OddParity               1
#define   EvenParity              2

#define   OneStopBit              0
#define   One5StopBit             1
#define   TwoStopBit              2

// return code
#define   NoError                 0
#define   FunctionError           1
#define   PortError               2
#define   BaudRateError           3
#define   DataError               4
#define   StopError                   5
#define   ParityError             6
#define   CheckSumError           7
#define   ComPortNotOpen          8
#define   SendThreadCreateError   9
#define   SendCmdError            10
#define   ReadComStatusError      11
#define   ResultStrCheckError     12
#define   CmdError                13
#define   TimeOut                 15
#define   ModuleIdError           17
```

```
#define    AdChannelError          18
#define    UnderInputRange         19
#define    ExceedInputRange        20
#define    InvalidateCounterNo     21
#define    InvalidateCounterValue  22
#define    InvalidateGateMode      23


EXPORTS WORD  CALLBACK Open_Com(char cPort, DWORD dwBaudRate, char
cData, char   cParity, char cStop);
EXPORTS BOOL  CALLBACK Close_Com(char cPort);
EXPORTS WORD  CALLBACK Send_Cmd(char cPort, char szCmd[], WORD
wTimeout, WORD    wChkSum);
EXPORTS WORD  CALLBACK Read_Com_Status(char cPort, char szResult[],
WORD
          *wT);
EXPORTS WORD  CALLBACK Send_Str(char cPort, char szCmd[], WORD
wTimeOut, WORD   wLenT, WORD wLenR);
EXPORTS WORD  CALLBACK Send_Receive_Cmd(char cPort, char szCmd[], char
          szResult[], WORD wTimeOut, WORD wChksum, WORD *wT);

//---------------------------------------------------------------------------
EXPORTS short CALLBACK Short_Sub_2(short nA, short nB);
EXPORTS float CALLBACK Float_Sub_2(float fA, float fB);
EXPORTS WORD  CALLBACK Get_Dll_Version(void);
EXPORTS WORD CALLBACK Test(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
// for driver testing
// strcpy(szSendTo7000,"Command String Send to I7000");
// strcpy(szReceiveFrom7000,"Command String Receive From I7000");
// for (i=0; i<4; i++)
//    {
//   w7000[i]++;
//   f7000[i]++;
//   }

EXPORTS WORD CALLBACK HostIsOK(WORD w7000[], float f7000[],
    char szSendTo7000[], char szReceiveFrom7000[]);
```

```
/*--- Read the module reset status, $AA5 --------*/
EXPORTS WORD CALLBACK ReadModuleResetStatus(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
//  w7000[0];   // RS-232 port number: 1/2/3/4
//  w7000[1];   // module address: 0x00 to 0xFF
//  w7000[2];   // don't care
//  w7000[3];   // checksum: 0=disable, 1=enable
//  w7000[4];   // TimeOut constant: normal=100
//  w7000[6];   // flag: 0=no save, 1=save send/receive string
//---------------------- output ---------------------------------
//  w7000[5];   // w7000[5]=0: module has not been reset
                // w7000[5]=1: module has been reset

/*--- To setup host watchdog, ~AA3ETT --------*/
EXPORTS WORD CALLBACK ToSetupHostWatchdog(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
//  w7000[0];   // RS-232 port number: 1/2/3/4
//  w7000[1];   // module address: 0x00 to 0xFF
//  w7000[2];   // don't care
//  w7000[3];   // checksum: 0=disable, 1=enable
//  w7000[4];   // TimeOut constant: normal=100
//  w7000[5];   // 0: Disable host watchdog
                // 1: Enable host watchdog
//  w7000[6];   // flag: 0=no save, 1=save send/receive string
//  w7000[7];   // a time interval for watchdog, unit is 0.1 second,
              // e.x  when w7000[7]=45, the time interval is 4.5 second

/*--- To read host watchdog setup value, ~AA2 --------*/
EXPORTS WORD CALLBACK ToReadHostWatchdog(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
//  w7000[0];   // RS-232 port number: 1/2/3/4
//  w7000[1];   // module address: 0x00 to 0xFF
```

```
//  w7000[2];   // don't care
//  w7000[3];   // checksum: 0=disable, 1=enable
//  w7000[4];   // TimeOut constant: normal=100
//  w7000[6];   // flag: 0=no save, 1=save send/receive string
//---------------------- output ----------------------------------
//  w7000[5];   // 0: Disable host watchdog
                // 1: Enable host watchdog
//  w7000[7];   // a time interval for watchdog, unit is 0.1 second,
              // e.x  when w7000[7]=45, the time interval is 4.5 second

/*--- Read the module status about the host watchdog, ~AA0 --------*/
EXPORTS WORD CALLBACK ReadModuleHostWatchdogStatus(WORD w7000[],
float f7000[],  char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
//  w7000[0];   // RS-232 port number: 1/2/3/4
//  w7000[1];   // module address: 0x00 to 0xFF
//  w7000[2];   // don't care
//  w7000[3];   // checksum: 0=disable, 1=enable
//  w7000[4];   // TimeOut constant: normal=100
//  w7000[6];   // flag: 0=no save, 1=save send/receive string
//---------------------- output ----------------------------------
//  w7000[5];   // w7000[5]=0:  module is OK
                //  w7000[5]=4:  host watchdog failure

/*--- Reset the module status about the host watchdog, ~AA1 --------*/
EXPORTS WORD CALLBACK ResetModuleHostWatchdogStatus(WORD w7000[],
float f7000[],  char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
//  w7000[0];   // RS-232 port number: 1/2/3/4
//  w7000[1];   // module address: 0x00 to 0xFF
//  w7000[2];   // don't care
//  w7000[3];   // checksum: 0=disable, 1=enable
//  w7000[4];   // TimeOut constant: normal=100
//  w7000[5];   // don't care
//  w7000[6];   // flag: 0=no save, 1=save send/receive string

/*--Set SafeValue for all DO module(42/43/44/50/60/65/67) ---*/
EXPORTS WORD CALLBACK SetSafeValueForDo(WORD w7000[], float f7000[],
```

```
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input --------------------------
//  w7000[0];   // RS-232 port number: 1/2/3/4
//  w7000[1];   // module address: 0x00 to 0xFF
//  w7000[2];   // module ID: 0x7050/60/67/42/43/44
//  w7000[3];   // checksum: 0=disable, 1=enable
//  w7000[4];   // TimeOut constant: normal=100
//  w7000[5];   // safe value
//  w7000[6];   // flag: 0=no save, 1=save send/receive string


/*--Set PowerOn value for all DO module(42/43/44/50/60/65/67) ---*/
EXPORTS WORD CALLBACK SetPowerOnValueForDo(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//  w7000[0];   // RS-232 port number: 1/2/3/4
//  w7000[1];   // module address: 0x00 to 0xFF
//  w7000[2];   // module ID: 0x7050/60/67/42/43/44
//  w7000[3];   // checksum: 0=disable, 1=enable
//  w7000[4];   // TimeOut constant: normal=100
//  w7000[5];   // PowerOn value
//  w7000[6];   // flag: 0=no save, 1=save send/receive string

/*--Set SafeValue for all Ao module(21/24), ~AA5 ---*/
EXPORTS WORD CALLBACK SetSafeValueForAo(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input --------------------------
//  w7000[0];   // RS-232 port number: 1/2/3/4
//  w7000[1];   // module address: 0x00 to 0xFF
//  w7000[2];   // module ID: 0x7021/0x7024
//  w7000[3];   // checksum: 0=disable, 1=enable
//  w7000[4];   // TimeOut constant: normal=100
//  w7000[5];   // Channel No
//  w7000[6];   // flag: 0=no save, 1=save send/receive string
//  f7000[0];   // safe value

/*--Set PowerOn value for all Ao module(21/24), $AA4 ---*/
EXPORTS WORD CALLBACK SetPowerOnValueForAo(WORD w7000[], float
```

```
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//  w7000[0];   // RS-232 port number: 1/2/3/4
//  w7000[1];   // module address: 0x00 to 0xFF
//  w7000[2];   // module ID: 0x7021/0x7024
//  w7000[3];   // checksum: 0=disable, 1=enable
//  w7000[4];   // TimeOut constant: normal=100
//  w7000[5];   // PowerOn value
//  w7000[6];   // flag: 0=no save, 1=save send/receive string
//  f7000[0];   // PowerOn value

/*-------- 7011/7012/7014 Set PowerOn/Safe value --------*/
EXPORTS WORD CALLBACK SetPowerOnSafeValue(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//  w7000[0];   // RS-232 port number: 1/2/3/4
//  w7000[1];   // module address: 0x00 to 0xFF
//  w7000[2];   // module ID: 0x7011/0x7012/0x7014
//  w7000[3];   // checksum: 0=disable, 1=enable
//  w7000[4];   // TimeOut constant: normal=100
//  w7000[5];   // PowerOn value
//  w7000[6];   // flag: 0=no save, 1=save send/receive string
//  w7000[7];   // safe value

/*-------- Read Config Status ---------*/
EXPORTS WORD CALLBACK  ReadConfigStatus(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
// w7000[0];   // RS-232 port number: 1/2/3/4
// w7000[1];   // module address: 0x00 to 0xFF
// w7000[2];   // module ID
// w7000[3];   // checksum: 0=disable, 1=enable
// w7000[4];   // TimeOut constant: normal=100
// w7000[5];   // Don't Care
// w7000[6];   // flag: 0=no save, 1=save send/receive string
//---------------------- Output ----------------------------------
// w7000[7]:   module address
// w7000[8]:   module Range Code
```

```
// w7000[9]:   module baudrate
// w7000[10]:  module data format


EXPORTS WORD CALLBACK AnalogIn(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module ID: 0x7011/12/13/14/17/18
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[5];          // channel number for multi-channel
// w7000[6];          // flag: 0=no save, 1=save send/receive string
//---------------------- output ----------------------------------
                  // f7000[0]=analog input value

EXPORTS WORD CALLBACK AnalogInFsr(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module ID: 0x7011/12/13/14/17/18
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[5];          // channel number for multi-channel
// w7000[6];          // flag: 0=no save, 1=save send/receive string
//---------------------- output ----------------------------------
                  // f7000[0]=analog input value in %FSR format

EXPORTS WORD CALLBACK AnalogInHex(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module ID: 0x7011/12/13/14/17/18
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
```

```
// w7000[5];          // channel number for multi-channel
// w7000[6];          // flag: 0=no save, 1=save send/receive string
//---------------------- output ------------------------------------
                      // w7000[7] : analog input value

EXPORTS WORD CALLBACK AnalogIn8(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -------------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module ID: 0x7017/18
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[6];          // flag: 0=no save, 1=save send/receive string
//---------------------- output ------------------------------------
                      // f7000[0]=analog input value of channel_0
                      // f7000[1]=analog input value of channel_1
                      // .....
                      // f7000[7]=analog input value of channel_7

EXPORTS WORD CALLBACK In8_7017(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -------------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module Type: 08/09/0A/0B/0C/0D
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[6];          // flag: 0=no save, 1=save send/receive string
//---------------------- output ------------------------------------
                      // f7000[0]=analog input value of channel_0
                      // f7000[1]=analog input value of channel_1
                      // .....
                      // f7000[7]=analog input value of channel_7


EXPORTS WORD CALLBACK AnalogOut(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
```

```
//---------------------- input ----------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module ID: 0x7021 or 0x7024
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[5];          // Not Used if module ID is 7021
//                    // Channel No(0 to 3) if module ID is 7024
// w7000[6];          // flag: 0=no save, 1=save send/receive string
// f7000[0];          // DA output value: from 0.0 to 10.0
//---------------------- output ---------------------------------
                      // void

EXPORTS WORD CALLBACK AnalogOutReadBack(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
// w7000[0];  // RS-232 port number: 1/2/3/4
// w7000[1];  // module address: 0x00 to 0xFF
// w7000[2];  // module ID: 0x7021 or 0x7024
// w7000[3];  // checksum: 0=disable, 1=enable
// w7000[4];  // TimeOut constant: normal=100
// w7000[5];  // 0: command read back ($AA6)
              // 1: when module ID is 0x7021
                //     analog output of current path read back ($AA8)
                //   when module ID is 0x7024
                //     the updating value in a specific Slew rate($AA8)
             //     (see 7024 manual for more detail information)
// w7000[6];  // flag: 0=no save, 1=save send/receive string
// w7000[7];  // Channel No.(0 to 3), if module ID is 7024
                //   Not used,     else
//---------------------- output ---------------------------------
                      // f7000[0]=analog output value read back

EXPORTS WORD CALLBACK DigitalOut(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
```

```
// w7000[2];          // module ID: 0x7050/60/67/42/43/44
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[5];          // 16-bit digital data to output
// w7000[6];          // flag: 0=no save, 1=save send/receive string
//---------------------- output -----------------------------------
                  // void

EXPORTS WORD CALLBACK DigitalOutReadBack(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ------------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module ID: 0x7050/60/67/42/43/44
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[6];          // flag: 0=no save, 1=save send/receive string
//---------------------- output -----------------------------------
                  // w7000[5]=digital output data read back

EXPORTS WORD CALLBACK DigitalIn(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ------------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module ID: 0x7050/52/53/60/41/44
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[6];          // flag: 0=no save, 1=save send/receive string
//---------------------- output -----------------------------------
                  // w7000[5]=digital input data

/*-------- 7011 Thermocouple Open Detection ---------*/
EXPORTS WORD CALLBACK ThermocoupleOpen_7011(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ------------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
```

```
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID: 0x7011
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100
//w7000[6];   // flag: 0=no save, 1=save send/receive string
//---------------------- output ----------------------------------
//w7000[5]   0: Close       1: Open

/*----- 7011/7012/7014 Set Power On and Safe value ---------*/
EXPORTS WORD CALLBACK SetPowerOnAndSafeValue(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID: 0x7011
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100  , 0.1 second
//w7000[5];   // 0xPPSS:  PP=Power On value   SS=safe value
          //          00: DO0 -> Off   DO1 -> Off
          //          01: DO0 -> On    DO1 -> Off
          //          02: DO0 -> Off   DO1 -> On
          //          03: DO0 -> On    DO1 -> On
//w7000[6];   // flag: 0=no save, 1=save send/receive string

/*----- 7011/7012/7014 Enable Alarm ---------*/
EXPORTS WORD CALLBACK EnableAlarm(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID: 0x7011
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100  , 0.1 second
//w7000[5];   // 0: Momentary,       1: Latch
//w7000[6];   // flag: 0=no save, 1=save send/receive string
```

```
/*----- 7011/7012/7014 Disable Alarm ---------*/
EXPORTS WORD CALLBACK DisableAlarm(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID: 0x7011/0x7012/0x7014
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100  , 0.1 second
//w7000[5];   don't care
//w7000[6];   // flag: 0=no save, 1=save send/receive string

/*-------- 7011/7012/7014 Clear Latch Alarm ---------*/
EXPORTS WORD CALLBACK ClearLatchAlarm(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID: 0x7011
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100  , 0.1 second
//w7000[5];   don't care
//w7000[6];   // flag: 0=no save, 1=save send/receive string

/*-------- 7011/7012/7014 Set Hi/Lo Alarm Limit Value ---------*/
EXPORTS WORD CALLBACK SetAlarmLimitValue(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID: 0x7011
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100  , 0.1 second
//w7000[5];   // 0: Lo Alarm    1: Hi Alarm
//w7000[6];   // flag: 0=no save, 1=save send/receive string
//f7000[0]:   // Alarm Limit Value

/*-------- 7011/7012/7014 Read Hi/Lo Alarm Limit Value ---------*/
```

```
EXPORTS WORD CALLBACK ReadAlarmLimitValue(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ---------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID: 0x7011
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100  , 0.1 second
//w7000[5];   // 0: Lo Alarm    1: Hi Alarm
//w7000[6];   // flag: 0=no save, 1=save send/receive string
//---------------------- output ---------------------------------
//f7000[0]:   // Alarm Limit Value

/*---- Read Alarm State and Digital Output State ---------*/
EXPORTS WORD CALLBACK ReadOutputAlarmState(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ---------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100
//w7000[5];   // Don't Care
//w7000[6];   // flag: 0=no save, 1=save send/receive string
//---------------------- output ---------------------------------
//            Alarm State:
// w7000(7):0     Disable
// w7000(7):1     Momentary
// w7000(7):2     Latch
//            Digital Output
// w7000(8):0     Bit:1 Disable  Bit 0: Disable
// w7000(8):1     Bit:1 Disable  Bit 0: Enable
// w7000(8):2     Bit:1 Enable   Bit 0: Disable
// w7000(8):3     Bit:1 Enable   Bit 0: Enable

/*-------- 7011/7012/7014 Read Event Counter --------*/
```

```
EXPORTS WORD CALLBACK ReadEventCounter(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100
//w7000[5];   // Don't Care
//w7000[6];   // flag: 0=no save, 1=save send/receive string
//---------------------- output ----------------------------------
//w7000[7];   // The Event Counter Value

/*-------- 7011/7012/7014 Clear Event Counter --------*/
EXPORTS WORD CALLBACK ClearEventCounter(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100
//w7000[5];   // Don't Care
//w7000[6];   // flag: 0=no save, 1=save send/receive string

/**************************************************************************
****/
EXPORTS WORD CALLBACK CounterIn_7080(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module ID: 0x7080
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[5];          // 0: Counter 0,  1: Counter 1
// w7000[6];          // flag: 0=no save, 1=save send/receive string
//---------------------- output ----------------------------------
```

```
                // w7000[7]  Hi-Word of counter value
                // w7000[8]  Lo-Word of counter value

EXPORTS WORD CALLBACK ReadCounterMaxValue_7080(WORD w7000[], float
f7000[],
    char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
// w7000[0];        // RS-232 port number: 1/2/3/4
// w7000[1];        // module address: 0x00 to 0xFF
// w7000[2];        // module ID: 0x7080
// w7000[3];        // checksum: 0=disable, 1=enable
// w7000[4];        // TimeOut constant: normal=100
// w7000[5];        // 0: Counter 0,  1: Counter 1
// w7000[6];        // flag: 0=no save, 1=save send/receive string
//---------------------- output ----------------------------------
                // w7000[7]  Hi-Word of counter value
                // w7000[8]  Lo-Word of counter value

EXPORTS WORD CALLBACK SetCounterMaxValue_7080(WORD w7000[], float
f7000[],
    char szSendTo7000[], char szReceiveFrom7000[], double MaxValue);
//---------------------- input -----------------------------------
// w7000[0];        // RS-232 port number: 1/2/3/4
// w7000[1];        // module address: 0x00 to 0xFF
// w7000[2];        // module ID: 0x7080
// w7000[3];        // checksum: 0=disable, 1=enable
// w7000[4];        // TimeOut constant: normal=100
// w7000[5];        // Don't Care
// w7000[6];        // flag: 0=no save, 1=save send/receive string
// MaxValue    Setting the Counter's Max Value

EXPORTS WORD CALLBACK ReadAlarmLimitValue_7080(WORD w7000[], float
f7000[],
    char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
// w7000[0];        // RS-232 port number: 1/2/3/4
// w7000[1];        // module address: 0x00 to 0xFF
// w7000[2];        // module ID: 0x7080
```

```
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[5];          // 0: Counter 0,  1: Counter 1
// w7000[6];          // flag: 0=no save, 1=save send/receive string
//---------------------- output ------------------------------------
// w7000[7]  Hi-Word of counter value
// w7000[8]  Lo-Word of counter value

EXPORTS WORD CALLBACK SetAlarmLimitValue_7080(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[], double AlarmValue);
//---------------------- input -------------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module ID: 0x7080
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[5];          // 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
// w7000[6];          // flag: 0=no save, 1=save send/receive string

EXPORTS WORD CALLBACK EnableCounterAlarm_7080(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -------------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module ID: 0x7080
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[5];          // 0: counter 0 enable, 1: counter 1 enable
// w7000[6];          // flag: 0=no save, 1=save send/receive string

EXPORTS WORD CALLBACK DisableCounterAlarm_7080(WORD w7000[], float
```

```
f7000[],
    char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
// w7000[0];        // RS-232 port number: 1/2/3/4
// w7000[1];        // module address: 0x00 to 0xFF
// w7000[2];        // module ID: 0x7080
// w7000[3];        // checksum: 0=disable, 1=enable
// w7000[4];        // TimeOut constant: normal=100
// w7000[5];        // Don't care
// w7000[6];        // flag: 0=no save, 1=save send/receive string

EXPORTS WORD CALLBACK EnableCounterAlarm_7080D(WORD w7000[], float
f7000[],
    char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
// w7000[0];        // RS-232 port number: 1/2/3/4
// w7000[1];        // module address: 0x00 to 0xFF
// w7000[2];        // module ID: 0x7080
// w7000[3];        // checksum: 0=disable, 1=enable
// w7000[4];        // TimeOut constant: normal=100
// w7000[5];        // 0: momentary, 1: latch
// w7000[6];        // flag: 0=no save, 1=save send/receive string

EXPORTS WORD CALLBACK DisableCounterAlarm_7080D(WORD w7000[], float
f7000[],
    char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
// w7000[0];        // RS-232 port number: 1/2/3/4
// w7000[1];        // module address: 0x00 to 0xFF
// w7000[2];        // module ID: 0x7080
// w7000[3];        // checksum: 0=disable, 1=enable
// w7000[4];        // TimeOut constant: normal=100
// w7000[5];        // Don't care
// w7000[6];        // flag: 0=no save, 1=save send/receive string

EXPORTS WORD CALLBACK ReadCounterStatus_7080(WORD w7000[], float
f7000[],
    char szSendTo7000[], char szReceiveFrom7000[]);
```

```
//---------------------- input ----------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module ID: 0x7080
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[5];          // 0: Counter 0,  1: Counter 1
// w7000[6];          // flag: 0=no save, 1=save send/receive string
//---------------------- output ---------------------------------
// w7000[7]  0: Counting   1: Not Counting

EXPORTS WORD CALLBACK ClearCounter_7080(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module ID: 0x7080
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[5];          // 0: Counter 0,  1: Counter 1
// w7000[6];          // flag: 0=no save, 1=save send/receive string

EXPORTS WORD CALLBACK ReadOutputAlarmState_7080(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module ID: 0x7080
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[5];          // Don't Care
// w7000[6];          // flag: 0=no save, 1=save send/receive string
//---------------------- output ---------------------------------
  // w7000(7):0  for 7080 ->  Counter0:Disable   Counter1:Disable
  // w7000(7):1            Counter0:Disable   Counter1:Enable
  // w7000(7):2            Counter0:Enable      Counter1:Disable
  // w7000(7):3            Counter0:Enable      Counter1:Enable
```

```
  // w7000(7):0  for 7080D ->       Counter0:Disable
  // w7000(7):1              Counter0:Momentary
  // w7000(7):2              Counter0:Latch
  // w7000(8):0    Bit:1 Disable  Bit 0: Disable
  // w7000(8):1    Bit:1 Disable  Bit 0: Enable
  // w7000(8):2    Bit:1 Enable   Bit 0: Disable
  // w7000(8):3    Bit:1 Enable   Bit 0: Enable

EXPORTS WORD CALLBACK SetInputSignalMode_7080(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module ID: 0x7080
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[5];          // Counter 0's and 1's signal mode
// w7000[6];          // flag: 0=no save, 1=save send/receive string
//---------------------- Input -----------------------------------
               // w7000(5):0    Counter:0 TTL  Counter:0  TTL
               // w7000(5):1    Counter:1 Photo       Counter:1  Photo
               // w7000(5):2    Counter:0 TTL  Counter:1  Photo
               // w7000(5):3    Counter:1 Photo       Counter:0  TTL


EXPORTS WORD CALLBACK ReadInputSignalMode_7080(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
// w7000[0];          // RS-232 port number: 1/2/3/4
// w7000[1];          // module address: 0x00 to 0xFF
// w7000[2];          // module ID: 0x7080
// w7000[3];          // checksum: 0=disable, 1=enable
// w7000[4];          // TimeOut constant: normal=100
// w7000[5];          // Don't Care
// w7000[6];          // flag: 0=no save, 1=save send/receive string
//---------------------- output ----------------------------------
```

```
                // w7000(7):0    Counter:0 TTL  Counter:0  TTL
                // w7000(7):1    Counter:1 Photo       Counter:1  Photo
                // w7000(7):2    Counter:0 TTL  Counter:1  Photo
                // w7000(7):3    Counter:1 Photo       Counter:0  TTL


/* ------------ 7080 Preset Counter 0/1 Counter Value ----------------- */
EXPORTS WORD CALLBACK PresetCounterValue_7080(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[], double PresetValue);

/* ------------ 7080 Read Preset Counter 0/1 Counter Value ------------- */
EXPORTS WORD CALLBACK ReadPresetCounterValue_7080(WORD w7000[], float
f7000[],    char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- output ----------------------------------
                // w7000[7]= Hi-Word of Preset Counter Value
                // w7000[8]= Lo-Word of Preset Counter Value

/*-------- 7080 Start Counting ---------*/
EXPORTS WORD CALLBACK StartCounting_7080(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID: 0x7080
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100
//w7000[5];   // 0: Counter 0   1: Counter 1
//w7000[6];   // flag: 0=no save, 1=save send/receive string
//w7000[7];   // 0: Stop Counting  1: Start Counting

/*-------- Set 7080 module mode  ---------*/
EXPORTS WORD CALLBACK SetModuleMode_7080(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
```

```
//w7000[2];   // module ID: 0x7080
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100
//w7000[5];   // 0: 7080 mode1: 7080D mode
//w7000[6];   // flag: 0=no save, 1=save send/receive string

/*-------- Read 7080 module mode  ---------*/
EXPORTS WORD CALLBACK ReadModuleMode_7080(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID: 0x7080
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100
//w7000[5];   // don't care
//w7000[6];   // flag: 0=no save, 1=save send/receive string
//------------------ output ---------------------
//w7000[5]= 0: 7080 mode,  1: 7080D mode

/*-------- 7080 Set Level Volt ---------*/
EXPORTS WORD CALLBACK SetLevelVolt_7080(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID: 0x7080
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100
//w7000[5];   // 0: Low Level,        1: High Level
//w7000[6];   // flag: 0=no save, 1=save send/receive string
//f7000[0];   // Level Voltage

/*-------- 7080 Read Level Volt ---------*/
EXPORTS WORD CALLBACK ReadLevelVolt_7080(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ----------------------------------
```

```
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID: 0x7080
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100
//w7000[5];   // 0: Low Level,        1: High Level
//w7000[6];   // flag: 0=no save, 1=save send/receive string
//---------------------- output ---------------------------------
//f7000[0];         // Level Voltage

/* -------------- 7080 Set Min Signal Width ------------------------ */
EXPORTS WORD CALLBACK SetMinSignalWidth_7080(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[], long MinWidth);
//---------------------- input ----------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID: 0x7080
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100
//w7000[5];   // 0: Low Level,        1: High Level
//w7000[6];   // flag: 0=no save, 1=save send/receive string
//MinWidth    // the Input Signal Min Width

/* -------------- 7080 Read Min Signal Width ------------------------ */
EXPORTS WORD CALLBACK ReadMinSignalWidth_7080(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input ---------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID: 0x7080
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100
//w7000[5];   // 0: Low Level,        1: High Level
//w7000[6];   // flag: 0=no save, 1=save send/receive string
//---------------------- Output ---------------------------------
//w7000[7]    // the Reading Input Signal Min Width
```

```
/* ---------- 7080 Set Gate Mode ------------------------ */
EXPORTS WORD CALLBACK SetGateMode_7080(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID: 0x7080
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100
//w7000[5];   // 0: Low,  1: High,  2: None
//w7000[6];   // flag: 0=no save, 1=save send/receive string

/* ---------- 7080 Read Gate Mode ------------------------ */
EXPORTS WORD CALLBACK ReadGateMode_7080(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
//w7000[0];   // RS-232 port number: 1/2/3/4
//w7000[1];   // module address: 0x00 to 0xFF
//w7000[2];   // module ID: 0x7080
//w7000[3];   // checksum: 0=disable, 1=enable
//w7000[4];   // TimeOut constant: normal=100
//w7000[6];   // flag: 0=no save, 1=save send/receive string
//---------------------- output ----------------------------------
//w7000[5];   // 0: Low,  1: High,  2: None

/*-------- 7080 Send data to LED  ---------*/
EXPORTS WORD CALLBACK DataToLED_7080(WORD w7000[], float f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//---------------------- input -----------------------------------
//wPort=(char)w7000[0];   // RS-232 port number: 1/2/3/4
//wAddr=w7000[1];         // module address: 0x00 to 0xFF
//wID=w7000[2];     // module ID: 0x7080
//wCheckSum=w7000[3];       // checksum: 0=disable, 1=enable
//wTimeOut=w7000[4];   // TimeOut constant: normal=100
//wCounter=w7000[5];     // Don't Care
//wFlag=(char)w7000[6];   // flag: 0=no save, 1=save send/receive string
//fOutValue=f7000[0];     // Data to LED
```

```
/*-------- Set configuration for 7080 ---------*/
EXPORTS WORD CALLBACK SetConfiguration_7080(WORD w7000[], float
f7000[],
      char szSendTo7000[], char szReceiveFrom7000[]);
//*** Notice: if you change the Baudrate or Checksum, please short the INIT*(pin6) to
GND(pin10)
//---------------------- input -----------------------------------
//wPort=(char)w7000[0];   // RS-232 port number: 1/2/3/4
//wAddrOld=w7000[1];   // module original address: 0x00 to 0xFF
//wID=w7000[2];     // module ID: 0x7080
//wCheckSumOld=w7000[3];         // module original checksum: 0=disable, 1=enable
//wTimeOut=w7000[4];   // TimeOut constant: normal=100
//wFreqGateTime=w7000[5]; // desired frequency gate time:
                  //     0: 0.1 second
                  //     1: 1.0 second
                  // Don't care w7000[5],if set the module into Counter mode
//wFlag=(char)w7000[6];   // flag: 0=no save, 1=save send/receive string
//wAddrNew=w7000[7];         // desired new address
//wType=w7000[8];         // desired Type->  0:Counter mode
                  //              1:Frequency mode
//wBaudrate=w7000[9];   // 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
//wCheckSumNew=w7000[10]; // desired Checksum Address
```

2.7.2 I7000u.cpp

```
#include <vcl.h>
#include "I7000.h"

char gcPort;        //Global Char
long gwBaudRate;    //Global Word
long gdwBaudRate;
Word gwCheckSum;
char gcDataBit, gcParity, gcStopBit;

char gszSend[80];   //Global StringZ
char gszReceive[80];

float gfInData[1000]; //Global Float
Word  gw7000[11];
float gf7000[11];


//***************************************
void IFillWordArray04(Word *wArray, Word wVal0, Word wVal1, Word wVal2, Word
wVal3, Word wVal4)
{
      wArray[0]=wVal0;
      wArray[1]=wVal1;
      wArray[2]=wVal2;
      wArray[3]=wVal3;
      wArray[4]=wVal4;
}

//***************************************
void IFillWordArray59(Word *wArray, Word wVal5, Word wVal6, Word wVal7, Word
wVal8, Word wVal9)
{
      wArray[5]=wVal5;
```

```
    wArray[6]=wVal6;
    wArray[7]=wVal7;
    wArray[8]=wVal8;
    wArray[9]=wVal9;
}

//*************************************
void IFillFloatArray04(float *fArray, float fVal0, float fVal1, float fVal2, float fVal3,
float fVal4)
{
    fArray[0]=fVal0;
    fArray[1]=fVal1;
    fArray[2]=fVal2;
    fArray[3]=fVal3;
    fArray[4]=fVal4;
}

//*************************************
void IFillFloatArray59(float *fArray, float fVal5, float fVal6, float fVal7, float fVal8,
float fVal9)
{
    fArray[5]=fVal5;
    fArray[6]=fVal6;
    fArray[7]=fVal7;
    fArray[8]=fVal8;
    fArray[9]=fVal9;
}

//*************************************
Word IOpenCom(char cComPort, DWORD dwBaudRate)
{
    return Open_Com( cComPort, dwBaudRate, 8, 0 , 0 );
}


//*************************************
void ISetCOMString(TComboBox *cb, int iStart, int iEnd)
{
```

```
  int i;

  cb->Items->Clear();
  for (i=iStart;i<=iEnd;i++)
      cb->Items->Add( "COM" + IntToStr(i) );
  cb->ItemIndex = 0;

}

//************************************
void ISetBaudRateString(TComboBox *cb)
{
    cb->Items->Clear();
    cb->Items->Add("115200");
    cb->Items->Add("57600");
    cb->Items->Add("38400");
    cb->Items->Add("19200");
    cb->Items->Add("9600");
    cb->Items->Add("4800");
    cb->Items->Add("2400");
    cb->Items->Add("1200");
    cb->ItemIndex = 0;
}

//************************************
AnsiString GetErrorString(Word wErrCode)
{
   AnsiString ErrString;

   switch (wErrCode) {
      case 0: ErrString = "No Error" ;              break;
      case 1: ErrString = "Function Error";         break;
      case 2: ErrString = "Port Error";             break;
      case 3: ErrString = "Baud Rate Error";        break;
      case 4: ErrString = "Data Error";             break;
      case 5: ErrString = "Stop Error";             break;
      case 6: ErrString = "Parity Error";           break;
      case 7: ErrString = "CheckSum Error";         break;
```

```
    case 8: ErrString = "Not Open";                          break;
    case 9: ErrString = "Send Thread Create Error";              break;
    case 10: ErrString = "Send Command Error";                   break;
    case 11: ErrString = "Read Com Port Status Error";       break;
    case 12: ErrString = "Result String Check Error";        break;
    case 13: ErrString = "Command Error";                    break;
    //case 14: ErrString = "";                               break;
    case 15: ErrString = "Time Out";                             break;
    //case 16: ErrString = "";                               break;
    case 17: ErrString = "Module Id Error";                  break;
    case 18: ErrString = "AD Channel Error";                 break;
    case 19: ErrString = "Under Input Range";                break;
    case 20: ErrString = "Exceed Input Range";               break;
    case 21: ErrString = "Invalidate Counter No";            break;
    case 22: ErrString = "Invalidate Counter Value";         break;
    case 23: ErrString = "Invalidate Gate Mode";             break;
    default :
        ErrString = "Unknown Error";                             break;
  }
  return ErrString ;
}

AnsiString IGetErrorString(Word wErrCode)
{
    return GetErrorString( wErrCode );
}

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

2.7.3 I7000.PAS

```
unit I7000;
interface
type PSingle=^Single;
type PWord=^Word;

//Declare interface of UART.DLL
function Open_Com(cPort:Char; dwBaudRate:LongInt; cData:Char; cParity:Char; cStop:Char): Word; StdCall;
function Close_Com(cPort:Char): Word; StdCall;
function Read_Com_Status(cPort:Char; szRec:PChar; var wT:Word): Word; StdCall;
function Send_Cmd(cPort:Char; szCmd:PChar; wTimeOut:Word; wCheckSum:Word): Word;  StdCall;
function Send_Str(cPort:Char; szSend:PChar; wTimeOut:Word; wLenT,wLenR:Word): Word; StdCall;
function Send_Receive_Cmd
      (cPort:Char; szCmd,szResult:PChar; wTimeOut,wChkSum:Word; var wT:Word): Word; StdCall;

//Declare interface of I7000.DLL
function Short_Sub_2(nA: smallint; nB: smallint): smallint;  StdCall;
function Float_Sub_2(fA: single;   fB: single):   single;    StdCall;
function Get_Dll_Version: Word; StdCall;
function Test(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function ReadConfigStatus(w7000:PWord; f7000:PSingle; szSend,szReceive:PChar): Word; StdCall;
function AnalogIn(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function AnalogInFsr(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function AnalogInHex(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function AnalogIn8(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function In8_7017(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function AnalogOut(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function AnalogOutReadBack(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function DigitalOut(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function DigitalOutReadBack(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function DigitalIn(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function ThermocoupleOpen_7011(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function EnableAlarm(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function DisableAlarm(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function ClearLatchAlarm(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function SetAlarmLimitValue(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
```

function ReadAlarmLimitValue(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function ReadOutputAlarmState(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function ReadEventCounter(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function ClearEventCounter(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function CounterIn_7080(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function ReadCounterMaxValue_7080

(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function SetCounterMaxValue_7080

(w7000:PWord; f7000:PSingle; szSend:PChar;

szReceive:PChar; MaxValue:Double): Word; StdCall;

function ReadAlarmLimitValue_7080

(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function SetAlarmLimitValue_7080

(w7000:PWord; f7000:PSingle; szSend:PChar;

szReceive:PChar; MaxValue:Double): Word; StdCall;

function EnableCounterAlarm_7080

(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function DisableCounterAlarm_7080

(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function EnableCounterAlarm_7080D

(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function DisableCounterAlarm_7080D

(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function ReadCounterStatus_7080(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function ClearCounter_7080(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function ReadOutputAlarmState_7080

(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function SetInputSignalMode_7080(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function ReadInputSignalMode_7080

(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function PresetCounterValue_7080

(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar;

PresetValue:Double): Word; StdCall;

function ReadPresetCounterValue_7080

(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function StartCounting_7080(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

function SetModuleMode_7080

(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;

```
function ReadModuleMode_7080(w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function SetLevelVolt_7080
        (w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function ReadLevelVolt_7080
        (w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function SetMinSignalWidth_7080
        (w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar;
        MinWidth:LongInt): Word; StdCall;
function ReadMinSignalWidth_7080
        (w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function SetGateMode_7080
        (w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function ReadGateMode_7080
        (w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function DataToLED_7080
        (w7000:PWord; f7000:PSingle; szSend:PChar; szReceive:PChar): Word; StdCall;
function SetConfiguration_7080
        (w7000:PWord; f7000:PSingle; szSendTo7000:PChar;
        szReceiveFrom7000:PChar):Word; StdCall;

// WatchDog
Function HostIsOK
        (w7000:PWord; f7000:PSingle; szSendTo7000:PChar;
        szReceiveFrom7000:PChar):Word; StdCall;
Function ReadModuleResetStatus
        (w7000:PWord; f7000:PSingle; szSendTo7000:PChar;
        szReceiveFrom7000:PChar):Word; StdCall;
Function ToSetupHostWatchdog
        (w7000:PWord; f7000:PSingle; szSendTo7000:PChar;
        szReceiveFrom7000:PChar):Word; StdCall;
Function ToReadHostWatchdog
        (w7000:PWord; f7000:PSingle; szSendTo7000:PChar;
        szReceiveFrom7000:PChar):Word; StdCall;
Function ReadModuleHostWatchdogStatus
        (w7000:PWord; f7000:PSingle; szSendTo7000:PChar;
        szReceiveFrom7000:PChar):Word; StdCall;
Function ResetModuleHostWatchdogStatus
        (w7000:PWord; f7000:PSingle; szSendTo7000:PChar;
```

```
        szReceiveFrom7000:PChar):Word; StdCall;
Function SetSafeValueForDo
        (w7000:PWord; f7000:PSingle; szSendTo7000:PChar;
        szReceiveFrom7000:PChar):Word; StdCall;
Function SetPowerOnValueForDo
        (w7000:PWord; f7000:PSingle; szSendTo7000:PChar;
        szReceiveFrom7000:PChar):Word; StdCall;
Function SetSafeValueForAo
        (w7000:PWord; f7000:PSingle; szSendTo7000:PChar;
        szReceiveFrom7000:PChar):Word; StdCall;
Function SetPowerOnValueForAo
        (w7000:PWord; f7000:PSingle; szSendTo7000:PChar;
        szReceiveFrom7000:PChar):Word; StdCall;
Function SetPowerOnSafeValue
        (w7000:PWord; f7000:PSingle; szSendTo7000:PChar;
        szReceiveFrom7000:PChar):Word; StdCall;

implementation
```

//Declare implementation of UART.DLL

```
function Open_Com;              external 'UART.DLL'  name 'Open_Com';
function Close_Com;             external 'UART.DLL'  name 'Close_Com';
function Read_Com_Status;       external 'UART.DLL'  name 'Read_Com_Status';
function Send_Cmd;              external 'UART.DLL'  name 'Send_Cmd';
function Send_Str;              external 'UART.DLL'  name 'Send_Str';
function Send_Receive_Cmd;      external 'UART.DLL'  name 'Send_Receive_Cmd';
```

//Declare implementation of I7000.DLL

```
function Short_Sub_2;           external 'I7000.DLL' name 'Short_Sub_2';
function Float_Sub_2;           external 'I7000.DLL' name 'Float_Sub_2';
function Get_Dll_Version;       external 'I7000.DLL' name 'Get_Dll_Version';
function Test;                  external 'I7000.DLL' name 'Test';
function ReadConfigStatus;      external 'I7000.DLL' name 'ReadConfigStatus';
function AnalogIn;              external 'I7000.DLL' name 'AnalogIn';
function AnalogInFsr;           external 'I7000.DLL' name 'AnalogInFsr';
function AnalogInHex;           external 'I7000.DLL' name 'AnalogInHex';
function AnalogIn8;             external 'I7000.DLL' name 'AnalogIn8';
function In8_7017;              external 'I7000.DLL' name 'In8_7017';
function AnalogOut;             external 'I7000.DLL' name 'AnalogOut';
```

```
function AnalogOutReadBack;              external 'I7000.DLL' name 'AnalogOutReadBack';
function DigitalOut;                     external 'I7000.DLL' name 'DigitalOut';
function DigitalOutReadBack;             external 'I7000.DLL' name 'DigitalOutReadBack';
function DigitalIn;                      external 'I7000.DLL' name 'DigitalIn';
function ThermocoupleOpen_7011;          external 'I7000.DLL' name 'ThermocoupleOpen_7011';
function EnableAlarm;                    external 'I7000.DLL' name 'EnableAlarm';
function DisableAlarm;                   external 'I7000.DLL' name 'EnableAlarm';
function ClearLatchAlarm;                external 'I7000.DLL' name 'ClearLatchAlarm';
function SetAlarmLimitValue;             external 'I7000.DLL' name 'SetAlarmLimitValue';
function ReadAlarmLimitValue;            external 'I7000.DLL' name 'ReadAlarmLimitValue';
function ReadOutputAlarmState;           external 'I7000.DLL' name 'ReadOutputAlarmState';
function ReadEventCounter;               external 'I7000.DLL' name 'ReadEventCounter';
function ClearEventCounter;                    external 'I7000.DLL' name 'ClearEventCounter';
function CounterIn_7080;                 external 'I7000.DLL' name 'CounterIn_7080';
function ReadCounterMaxValue_7080;       external 'I7000.DLL' name 'ReadCounterMaxValue_7080';
function SetCounterMaxValue_7080;        external 'I7000.DLL' name 'SetCounterMaxValue_7080';
function ReadAlarmLimitValue_7080;       external 'I7000.DLL' name 'ReadAlarmLimitValue_7080';
function SetAlarmLimitValue_7080;        external 'I7000.DLL' name 'SetAlarmLimitValue_7080';
function EnableCounterAlarm_7080;        external 'I7000.DLL' name 'EnableCounterAlarm_7080';
function DisableCounterAlarm_7080;       external 'I7000.DLL' name 'DisableCounterAlarm_7080';
function EnableCounterAlarm_7080D;       external 'I7000.DLL' name 'EnableCounterAlarm_7080D';
function DisableCounterAlarm_7080D;      external 'I7000.DLL' name 'DisableCounterAlarm_7080D';
function ReadCounterStatus_7080;         external 'I7000.DLL' name 'ReadCounterStatus_7080';
function ClearCounter_7080;              external 'I7000.DLL' name 'ClearCounter_7080';
function ReadOutputAlarmState_7080;      external 'I7000.DLL' name 'ReadOutputAlarmState_7080';
function SetInputSignalMode_7080;        external 'I7000.DLL' name 'SetInputSignalMode_7080';
function ReadInputSignalMode_7080;       external 'I7000.DLL' name 'ReadInputSignalMode_7080';
function PresetCounterValue_7080;        external 'I7000.DLL' name 'PresetCounterValue_7080';
function ReadPresetCounterValue_7080;    external 'I7000.DLL' name 'ReadPresetCounterValue_7080';
function StartCounting_7080;             external 'I7000.DLL' name 'StartCounting_7080';
function SetModuleMode_7080;             external 'I7000.DLL' name 'SetModuleMode_7080';
function ReadModuleMode_7080;            external 'I7000.DLL' name 'ReadModuleMode_7080';
function SetLevelVolt_7080;              external 'I7000.DLL' name 'SetLevelVolt_7080';
function ReadLevelVolt_7080;             external 'I7000.DLL' name 'ReadLevelVolt_7080';
function SetMinSignalWidth_7080;         external 'I7000.DLL' name 'SetMinSignalWidth_7080';
function ReadMinSignalWidth_7080;        external 'I7000.DLL' name 'ReadMinSignalWidth_7080';
function SetGateMode_7080;               external 'I7000.DLL' name 'SetGateMode_7080';
function ReadGateMode_7080;              external 'I7000.DLL' name 'ReadGateMode_7080';
```

```
function DataToLED_7080;              external 'I7000.DLL' name 'DataToLED_7080';
function SetConfiguration_7080;       external 'I7000.DLL' name 'SetConfiguration_7080';

// WatchDog
function HostIsOK;                    external 'I7000.DLL' name 'HostIsOK';
function ReadModuleResetStatus;       external 'I7000.DLL' name 'ReadModuleResetStatus';
function ToSetupHostWatchdog;         external 'I7000.DLL' name 'ToSetupHostWatchdog';
function ToReadHostWatchdog;          external 'I7000.DLL' name 'ToReadHostWatchdog';
function ReadModuleHostWatchdogStatus;
                                       external 'I7000.DLL' name 'ReadModuleHostWatchdogStatus';
function ResetModuleHostWatchdogStatus;
                                       external 'I7000.DLL' name 'ResetModuleHostWatchdogStatus';
function SetSafeValueForDo;           external 'I7000.DLL' name 'SetSafeValueForDo';
function SetPowerOnValueForDo;        external 'I7000.DLL' name 'SetPowerOnValueForDo';
function SetSafeValueForAo;           external 'I7000.DLL' name 'SetSafeValueForAo';
function SetPowerOnValueForAo;        external 'I7000.DLL' name 'SetPowerOnValueForAo';
function SetPowerOnSafeValue;         external 'I7000.DLL' name 'SetPowerOnSafeValue';

end.
```

2.7.4 I7000u.PAS

```
unit I7000u;

interface

uses
  SysUtils, StdCtrls;

type PSingle=^Single;
type PWord=^Word;

Function  IOpenCom(cComPort :char; dwBaudRate : LongInt):Word;
Function  GetErrorString(wErrCode : Word ): String;
Function  IGetErrorString(wErrCode : Word ): String;
Procedure ISetCOMString( cb : TComboBox; iStart, iEnd : integer);
Procedure ISetBaudRateString( cb : TComboBox);


Var
   gcPort             : Char;    //Global Char
   gwBaudRate         : LongInt; //Global Word
   gdwBaudRate        : LongInt;
   gwCheckSum         : Word;
   gcDataBit          : Char;
   gcParity           : Char;
   gcStopBit          : Char;

   gszSend            : PChar;   //Global StringZ
   gszReceive         : PChar;

   gfInData           : Array[0..999] of Single; //Global Float
   gw7000             : Array[0..10] of Word;
   gf7000             : Array[0..10] of Single;

implementation
```

```
uses I7000;


//************************************
Function IOpenCom(cComPort :char ; dwBaudRate : LongInt):Word;
begin
      IOpenCom := Open_Com( cComPort, dwBaudRate, char(8), char(0) , char(0) );
end;

//************************************
Procedure ISetCOMString( cb : TComboBox; iStart, iEnd : integer);
var
  i : integer;
begin
    cb.Items.Clear;
    for i := iStart to iEnd do
       cb.Items.Add( 'COM' + IntToStr(i) );
    cb.ItemIndex := 0;
end;

//************************************
Procedure ISetBaudRateString( cb : TComboBox);
begin
    cb.Items.Clear;
    cb.Items.Add('115200');          cb.Items.Add('57600');
    cb.Items.Add('38400');           cb.Items.Add('19200');
    cb.Items.Add('9600');                  cb.Items.Add('4800');
    cb.Items.Add('2400');                  cb.Items.Add('1200');
    cb.ItemIndex := 0;
end;


//************************************
function GetErrorString( wErrCode : Word ):String ;
Var
   ErrString : String ;
Begin
   Case wErrcode of
```

```
        0: ErrString    := 'No Error' ;
        1: ErrString    := 'Function Error';
        2: ErrString    := 'Port Error';
        3: ErrString    := 'Baud Rate Error';
        4: ErrString    := 'Data Error';
        5: ErrString    := 'Stop Error';
        6: ErrString    := 'Parity Error';
        7: ErrString    := 'CheckSum Error';
        8: ErrString    := 'ComPort Not Open';
        9: ErrString    := 'Send Thread Create Error';
       10: ErrString    := 'Send Command Error';
       11: ErrString    := 'Read Com Port Status Error';
       12: ErrString    := 'Result String Check Error';
       13: ErrString    := 'Command Error';
       //14: ErrString := '';
       15: ErrString    := 'Time Out';
       //16: ErrString := '';
       17: ErrString    := 'Module Id Error';
       18: ErrString    := 'AD Channel Error';
       19: ErrString    := 'Under Input Range';
       20: ErrString    := 'Exceed Input Range';
       21: ErrString    := 'Invalidate Counter No';
       22: ErrString    := 'Invalidate Counter Value';
       23: ErrString    := 'Invalidate Gate Mode';
    Else
          ErrString     := 'Unknown Error';
    end;
    GetErrorString    := ErrString ;
end;

function IGetErrorString( wErrCode : Word ):String ;
Begin
      IGetErrorString := GetErrorString( wErrCode );
end;
//************************************
end.
```

2.7.5 I7000.BAS

```
Attribute VB_Name = "I7000"
Option Explicit
'----------------------- Error Message ------------------
Global Const NoError          = 0
Global Const FunctionError    = 1
Global Const PortError        = 2
Global Const BaudRateError    = 3
Global Const DataError        = 4
Global Const StopError        = 5
Global Const ParityError      = 6
Global Const CheckSumError            = 7
Global Const ComPortNotOpen           = 8
Global Const SendThreadCreateError    = 9
Global Const SendCmdError             = 10
Global Const ReadComStatusError       = 11
Global Const ResultStrCheckError      = 12
Global Const CmdError                 = 13
Global Const TimeOut                  = 15
Global Const ModuleIdError            = 17
Global Const AdChannelError           = 18
Global Const UnderInputRange          = 19
Global Const ExceedInputRange         = 20
Global Const InvalidateCounterNo      = 21
Global Const InvalidateCounterValue   = 22

'----------------------  UART.DLL -----------------------------------------------
Declare Function Open_Com Lib "uart.dll" (ByVal Port As Byte, ByVal BaudRate As
Long, ByVal cData As Byte, ByVal cParity As Byte, ByVal cStop As Byte) As Integer
Declare Function Close_Com Lib "uart.dll" (ByVal Port As Byte) As Boolean
Declare Function Send_Cmd Lib "uart.dll"
            (ByVal Port As Byte, ByVal Cmd As String, _
          ByVal TimeOut As Integer, ByVal wChkSum As Integer) As Integer
Declare Function Read_Com_Status Lib "uart.dll"
            (ByVal Port As Byte, ByVal Buf As String,  status As Integer) As Integer
Declare Function Send_Str Lib "uart.dll"
```

```
        (ByVal Port As Byte, ByVal Buf As String, ByVal TimeOut As Integer,
         ByVal LenT As Integer, ByVal LenR As Integer) As Integer
Declare Function Send_Receive_Cmd Lib "uart.dll"
        (ByVal Port As Byte, ByVal szCmd As String, ByVal szResult As String,
         ByVal TimeOut As Integer, ByVal CheckSum As Integer, wT As Integer)
        As Integer

'---------------------  I7000.DLL ----------------------------------------
Declare Function ReadConfigStatus Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function Test Lib "i7000.dll" (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function AnalogIn Lib "i7000.dll" (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function AnalogInFsr Lib "i7000.dll" (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function AnalogInHex Lib "i7000.dll" (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function AnalogIn8 Lib "i7000.dll" (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function In8_7017 Lib "i7000.dll" (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function AnalogOut Lib "i7000.dll" (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function AnalogOutReadBack Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single,
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function DigitalOut Lib "i7000.dll" (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function DigitalOutReadBack Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single,
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function DigitalIn Lib "i7000.dll" (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function ThermocoupleOpen_7011 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String) As Integer
```

```
Declare Function EnableAlarm Lib "i7000.dll" (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function DisableAlarm Lib "i7000.dll" (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function ClearLatchAlarm Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function SetAlarmLimitValue Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single,
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function ReadAlarmLimitValue Lib "i7000.dll" (w7000 As Integer, f7000 As
Single,    ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function ReadOutputAlarmState Lib "i7000.dll" (w7000 As Integer, f7000 As
Single,    ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function ReadEventCounter Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function ClearEventCounter Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function CounterIn_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function ReadCounterMaxValue_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String) As Integer
Declare Function SetCounterMaxValue_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String, ByVal MaxValue As Double) As Integer
Declare Function ReadAlarmLimitValue_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String) As Integer
Declare Function SetAlarmLimitValue_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String, ByVal AlarmValue As Double)
        As Integer
Declare Function ReadCounterStatus_7080 Lib "i7000.dll" (w7000 As Integer, f7000 As
Single, ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
```

```
Declare Function ClearCounter_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function ReadOutputAlarmState_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String) As Integer
Declare Function EnableCounterAlarm_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String) As Integer
Declare Function DisableCounterAlarm_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String) As Integer
Declare Function EnableCounterAlarm_7080D Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String) As Integer
Declare Function DisableCounterAlarm_7080D Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String) As Integer
Declare Function SetInputSignalMode_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String) As Integer
Declare Function ReadInputSignalMode_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single,  ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String) As Integer
Declare Function ReadPresetCounterValue_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single,    ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String) As Integer
Declare Function PresetCounterValue_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single,  ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String, ByVal PresetValue As Double)
        As Integer
Declare Function StartCounting_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single,
       ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function ReadModuleMode_7080 Lib "i7000.dll" (w7000 As Integer, f7000 As
Single,   ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function SetModuleMode_7080 Lib "i7000.dll" (w7000 As Integer, f7000 As
Single,  ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
```

```
Declare Function ReadLevelVolt_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single,
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function ReadMinSignalWidth_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single,  ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String) As Integer
Declare Function SetMinSignalWidth_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single,  ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String, ByVal MinWidth As Long) As Integer
Declare Function SetGateMode_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single,
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function ReadGateMode_7080 Lib "i7000.dll" (w7000 As Integer, f7000 As
Single, ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function DataToLED_7080 Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, _
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function SetConfiguration_7080 Lib "i7000.dll" (w7000 As Integer, f7000 As
Single, ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function HostIsOK Lib "i7000.dll" (w7000 As Integer, f7000 As Single, _
       ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function ReadModuleResetStatus Lib "i7000.dll" (w7000 As Integer, f7000 As
Single, ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function ToSetupHostWatchdog Lib "i7000.dll" (w7000 As Integer, f7000 As
Single, ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function ToReadHostWatchdog Lib "i7000.dll" (w7000 As Integer, f7000 As
Single, ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function ReadModuleHostWatchdogStatus Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String) As Integer
Declare Function ResetModuleHostWatchdogStatus Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String) As Integer
Declare Function SetSafeValueForDo Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single, ByVal SendTo7000 As String,
        ByVal ReceiveFrom7000 As String) As Integer
Declare Function SetPowerOnValueForDo Lib "i7000.dll" (w7000 As Integer, f7000 As
Single, ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
```

```
Declare Function SetSafeValueForAo Lib "i7000.dll"
        (w7000 As Integer, f7000 As Single,
        ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function SetPowerOnValueForAo Lib "i7000.dll" (w7000 As Integer, f7000 As
Single, ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
Declare Function SetPowerOnSafeValue Lib "i7000.dll" (w7000 As Integer, f7000 As
Single,  ByVal SendTo7000 As String, ByVal ReceiveFrom7000 As String) As Integer
```

2.7.6 I7000u.BAS

```
Attribute VB_Name = "I7000u"

Option Explicit
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Global Port             As Byte
Global BaudRate         As Long
Global ToCheckSum       As Byte
Global DataBit          As Byte
Global Parity           As Byte
Global StopBit          As Byte
Global InData(0 To 999) As Single
Global Index            As Integer
Global COMOpen          As Integer

Global ConfigChange     As Integer
Global SendTo7000       As String
Global ReceiveFrom7000  As String

Global w7000(0 To 10)   As Integer
Global f7000(0 To 10)   As Single
Global Ret              As Integer
Global status           As Integer
Global Count            As Integer

'===========
Global gcPort           As Byte
Global gcDataBit        As Byte
Global gcParity         As Byte
Global gcStopBit        As Byte
Global gdwBaudRate      As Long
Global gszSend, gszReceive  As String
Global gw7000(0 To 10)      As Integer
Global gf7000(0 To 10)      As Single
```

```
'************************************
Public Sub IFillWordArray04(wArray As Integer, ByVal wVal0 As Integer,
ByVal wVal1 As Integer, ByVal wVal2 As Integer, ByVal wVal3 As Integer,
ByVal wVal4 As Integer)
    wArray(0) = wVal0
    wArray(1) = wVal1
    wArray(2) = wVal2
    wArray(3) = wVal3
    wArray(4) = wVal4
End Sub

'************************************
Public Sub IFillWordArray59(wArray As Integer, ByVal wVal5 As Integer,
ByVal wVal6 As Integer, ByVal wVal7 As Integer, ByVal wVal8 As Integer,
ByVal wVal9 As Integer)
    wArray(5) = wVal5
    wArray(6) = wVal6
    wArray(7) = wVal7
    wArray(8) = wVal8
    wArray(9) = wVal9
End Sub

'************************************
Public Sub IFillFloatArray04(fArray As Single, ByVal fVal0 As Single,
ByVal fVal1 As Single, ByVal fVal2 As Single, ByVal fVal3 As Single,
ByVal fVal4 As Single)
    fArray(0) = fVal0
    fArray(1) = fVal1
    fArray(2) = fVal2
    fArray(3) = fVal3
    fArray(4) = fVal4
End Sub

'************************************
Public Sub IFillFloatArray59(fArray As Single, ByVal fVal5 As Single,
ByVal fVal6 As Single, ByVal fVal7 As Single, ByVal fVal8 As Single,
ByVal fVal9 As Single)
```

```
        fArray(5) = fVal5
        fArray(6) = fVal6
        fArray(7) = fVal7
        fArray(8) = fVal8
        fArray(9) = fVal9
End Sub

'*************************************
Public Function IOpenCom(ByVal cCOMPort As Byte, ByVal dwBaudRate As Long)
As Integer
        IOpenCom = Open_Com(cCOMPort, dwBaudRate, 8, 0, 0)
End Function


'*************************************
Public Sub ISetCOMString(cb As ComboBox, iStart As Integer, iEnd As Integer)
    Dim i As Integer

    cb.Clear
    For i = iStart To iEnd
        cb.AddItem "COM" + Str(i)
    Next
    cb.ListIndex = 0
End Sub

'*************************************
Public Sub ISetBaudRateString(cb As ComboBox)
    cb.Clear
    cb.AddItem "115200"
    cb.AddItem "57600"
    cb.AddItem "38400"
    cb.AddItem "19200"
    cb.AddItem "9600"
    cb.AddItem "4800"
    cb.AddItem "2400"
    cb.AddItem "1200"
    cb.ListIndex = 0
End Sub
```

```
'************************************
Public Function GetErrorString(ByVal wErrCode As Integer) As String
   Dim ErrString As String

   Select Case wErrCode
   Case 0
      ErrString = "No Error"
   Case 1
      ErrString = "Function Error"
   Case 2
      ErrString = "Port Error"
   Case 3
      ErrString = "Baud Rate Error"
   Case 4
      ErrString = "Data Error"
   Case 5
      ErrString = "Stop Error"
   Case 6
      ErrString = "Parity Error"
   Case 7
      ErrString = "CheckSum Error"
   Case 8
      ErrString = "ComPort Not Open"
   Case 9
      ErrString = "Send Thread Create Error"
   Case 10
      ErrString = "Send Command Error"
   Case 11
      ErrString = "Read Com Port Status Error"
   Case 12
      ErrString = "Result String Check Error"
   Case 13
      ErrString = "Command Error"
   'Case 14
   '    ErrString = ""
   Case 15
      ErrString = "Time Out"
```

```
    'Case 16
    '    ErrString = ""
    Case 17
        ErrString = "Module Id Error"
    Case 18
        ErrString = "AD Channel Error"
    Case 19
        ErrString = "Under Input Range"
    Case 20
        ErrString = "Exceed Input Range"
    Case 21
        ErrString = "Invalidate Counter No"
    Case 22
        ErrString = "Invalidate Counter Value"
    Case 23
        ErrString = "Invalidate Gate Mode"
    Case Else
        ErrString = "Unknown Error"
    End Select

    GetErrorString = ErrString
End Function

Public Function IGetErrorString(ByVal wErrCode As Integer) As String
        IGetErrorString = GetErrorString(wErrCode)
End Function

'************************************
```

3 UART.DLL FOR WINDOWS95/NT

3.1 FUNCTION DESCRIPTION

There are six exported functions to consist the UART.DLLs. Refer to Sec 2.7 for "Include files". Refer to Sec. 2.1 to Sec. 2.6 for usage and demo program.

3.1.1 Open_Com

- **Description:**

  This DLL will initialize the COM port. This DLL must be **called once before** the other DLLs are called to send/receive command.

- **Syntax:**

  WORD Open_Com(char cPort, DWORD dwBaudRate, char cData, char cParity, char cStop)

- **Input Parameter:**

cPort:	1=COM1, 2=COM2, 3=COM3, 4=COM4, others = invalidate
dwBaudRate:	1200/2400/4800/9600/19200/38400/57600/115200
cData:	5/6/7/8 data bit
cParity:	0=NonParity, 1=OddParity, 2=EvenParity
cStop:	0=1-stop, 1=1.5-stp, 2=2-stop

  NOTE: cData=8, cParity=0, cStop=0 for 7000 modules

- **Return Value:**

NoError =	OK
others =	Error code, refer to I7000.H in Sec. 2.7

- **Demo Program:** Sec. 3.2.1 to Sec. 3.2.1

3.1.2 Close_Com

- **Description:**

This DLL will free all the resources used by Open_Com. This DLL must be **called before** the program exit. The Open_Com will return error message if the program exit without calling Close_Com function.

- **Syntax:**

WORD Close_Com(char cPort)

- **Input Parameter:**

cPort : 1=COM1, 2=COM2, 3=COM3, 4=COM4, others = invalidate

- **Return Value:**

NoError = OK

others = Error code, refer to I7000.H in Sec. 2.7

- **Demo Program:** Sec. 3.2.1 to Sec. 3.2.7

```
Open_Com(COM_PORT,115200L,cData,cParity,cStop);    // open com first

    The other DLLs are called here

Close_Com(COM_PORT); // close com if stop the program
```

3.1.3 Send_Cmd

- **Description:**

This DLL will create a thread to send a command to 7000 and receive the response-result from 7000. If the wCheckSum=1, this DLL will automatically **add the two checksum bytes** to the input string. This DLL will **add the [0x0D]** to the end of the input string, szCmd. The Send_Cmd is a multi-task, multi-thread DLL.

- **Syntax:**

WORD Send_Cmd(char cPort, char szCmd[], WORD wTimeOut, WORD wCheckSum)

- **Input Parameter:**

cPort: 1=COM1, 2=COM2, 3=COM3, 4=COM4, others = invalidate

szCmd: the starting address of the original command string (terminated with 0)

wTimeOut: constant for time-out control, unit = 1ms

wCheckSum: 0=DISABLE, 1=ENABLE

- **Return Value:**

NoError : OK

others = Error code, refer to I7000.H in Sec. 2.7

- **Demo Program :** Sec. 3.2.1

```
Open_Com(COM_PORT,115200L,cData,cParity,cStop);// open com first
.
strcpy(cBuf,"$01M");                    // check module name
ret = Send_Cmd(1,cBuf,1000,0);          // create a thread
                                        // time-out=1000 ms=1 sec
                                        // checksum is disable
for (;;)                                // check the thread status
   {
   wRet=Read_Com_Status(1,cResult,&wT);
   if (wRet >= 0x100) break;
   }
sprintf(cShow,"Receive=%s wRet=%x, wT=%d",cResult,wRet,wT);
TextOut(hdc,1,yy,cShow,strlen(cShow)); yy+=16;
.
Close_Com(COM_PORT); // close com if stop the program
```

3.1.4 Read_Com_Status

- **Description:**

The **Send_Cmd(…)** will create a thread to send a command to 7000 and receive the response-result from 7000. The Read_Com_Status will return the status of this send/receive thread. If the thread is working, the status value will be smaller than 0x100. If the thread is finished, the status value will be larger than 0x100. **The return value will be equal to 0x105 if the send/receive operation is OK.**

If the wCheckSum in Send_Cmd is 1, the Recom_Com_Status will check the two checksum bytes of result string. **If the checksum is incorrect, the D13 of return value will be set to 1.**

- **Syntax:**

WORD Read_Com_Status(char cPort, char szResult[], WORD *wT)

- **Input Parameter:**

cPort :	1=COM1, 2=COM2, 3=COM3, 4=COM4, others = invalidate
szResult:	the starting address of the result string (terminated with 0)
wT:	time of send/receive interval, unit = 1 ms

- **Return Value:**

D0-D7=	thread status code, thread start=1 and stop=5
D8=	**0→ send/receive not finish, 1→ send/receive finish**
D9=	1→ send/receive timeout
D10=	reserved
D11=	1→ Com handle error
D12=	1→ send/receive overflow
D13=	1 → checksum error

- **Demo Program :** Sec. 3.2.1

```
strcpy(cBuf,"$01M");                    // check module name
ret = Send_Cmd(1,cBuf,1000,0);          // create a thread
for (;;) {                              // check the thread status
    wRet = Read_Com_Status(1,cResult,&wT);
    if (wRet >= 0x100) break;
    }
if  (wRet != 0x105)  {
                    // this is a error condition
                    }
```

3.1.5 Send_Str

- **Description:**

This DLL will create a thread to send a command and receive the response-result from a general purpose RS-232 device. The Send_Str is a multi-task, multi-thread DLL. This DLL is very similar to Send_Cmd except that this DLL will **not add any char** to the input string.

- **Syntax:**

WORD Send_Str(char cPort, char szCmd[], WORD wTimeOut, WORD wSendLen, WORD wReceiveLen)

- **Input Parameter:**

cPort: 1=COM1, 2=COM2, 3=COM3, 4=COM4, others = invalidate

szCmd: the starting address of the original command string (terminated with 0)

wTimeOut: constant for time-out control, unit = 1ms

wSendLen: string length of send-string

wReceiveLen: string length of receive-string

- **Return Value:**

NoError : OK

others = Error code, refer to I7000.H in Sec. 2.7

- **Demo Program :** Sec. 3.2.3

```
Open_Com(COM_PORT,9600L,cData,cParity,cStop);  // open com first
.
.
strcpy(cBuf,"<01C00007B22>");    // ANC PC202 counter command
wSendLen=13;     // send-string=<01C00007B22>
wReceiveLen=10;  // receive-string=(00007BC6)
wRet = Send_Str(cPort,cBuf,1000,wSendLen,wReceiveLen); // create a thread
for (;;)                                    // check the thread status
   {
   wRet=Read_Com_Status(1,cResult,&wT);
   if (wRet >= 0x100) break;
   }
.
.
Close_Com(COM_PORT); // close com if stop the program
```

3.1.6 Send_Receive_Cmd

- **Description:**

This DLL will send a command to 7000 and receive the response-result from 7000. If the wCheckSum=1, this DLL will automatically **add the two checksum bytes** to the input string and check the checksum status of the receive string. This DLL will **add the [0x0D]** to the end of the input string, szCmd. The Send_Receive_Cmd is not a multi-task DLL.

- **Syntax:**

WORD Send_Receive_Cmd(char cPort, char szCmd[], char szResult[], WORD wTimeOut, WORD wCheckSum, WORD *wT)

- **Input Parameter:**

cPort:	1=COM1, 2=COM2, 3=COM3, 4=COM4, others = invalidate
szCmd:	the starting address of the input string (terminated with 0)
szResult:	the starting address of the result string
wTimeOut:	constant for time-out control, unit = 1ms
wCheckSum:	0=DISABLE, 1=ENABLE
wT:	time of send/receive interval, unit = 1 ms

- **Return Value:**

NoError :	OK
others =	Error code, refer to I7000.H in Sec. 2.7

- **Demo Program :** Sec. 3.2.1

```
Open_Com(COM_PORT,115200L,cData,cParity,cStop);     // open com first
.
.
.
strcpy(cBuf,"$01M");                          // check module name
wRet = Send_Receive_Cmd(cPort,cBuf,cResult,wTimeOut,wCheckSum,&wT);
if      (wRet != NoError)
 {
 // something error
 }
.
.
Close_Com(COM_PORT); // close com if stop the program
```

3.2 VISUAL C++ DEMO PROGRAM

There are many demo program given in the NAP7000P. These demo program are used to demo driver usage. The disk contents of demo program are given as following:

\NAP7000P\Demo\VC\DEMO?\readme.doc	→ read this file first
\NAP7000P\Demo\VC\DEMO?\I7000.h	→ declaration file
\NAP7000P\Demo\VC\DEMO?\uart.dll	→ rs-232 related DLLs file
\NAP7000P\Demo\VC\DEMO?\uart.lib	→ import library of UART.DLL
\NAP7000P\Demo\VC\DEMO?\I7000.dll	→ 7000 related DLLs file
\NAP7000P\Demo\VC\DEMO?\I7000.lib	→ import library of 7000.DLL
\NAP7000P\Demo\VC\DEMO?\c.bat	→ batch file
\NAP7000P\Demo\VC\DEMO?\demo?.c	→ C program source
\NAP7000P\Demo\VC\DEMO?\demo?.mak	→ NMAKE file
\NAP7000P\Demo\VC\DEMO?\demo?.obj	→ object file of demo program
\NAP7000P\Demo\VC\DEMO?\demo?.exe	→ execution file of demo program

These demp program are given as following:
Demo01 → send/receive command to 7000 with checksum disable
Demo02 → send/receive command to 7000 with checksum enable
Demo03 → send/receive command to Counter, ANC PC-202
Demo04 → send/receive command to OMRON PLC, CQM1 or C200
Demo05 → multi-speed demo
Demo06 → muti-data-format demo
Demo07 → multi-speed and multi-data-format demo

…….
…….
Refer to folder that you installed.

3.2.1 Demo1: Send/Receive command to 7000 with CheckSum Disable

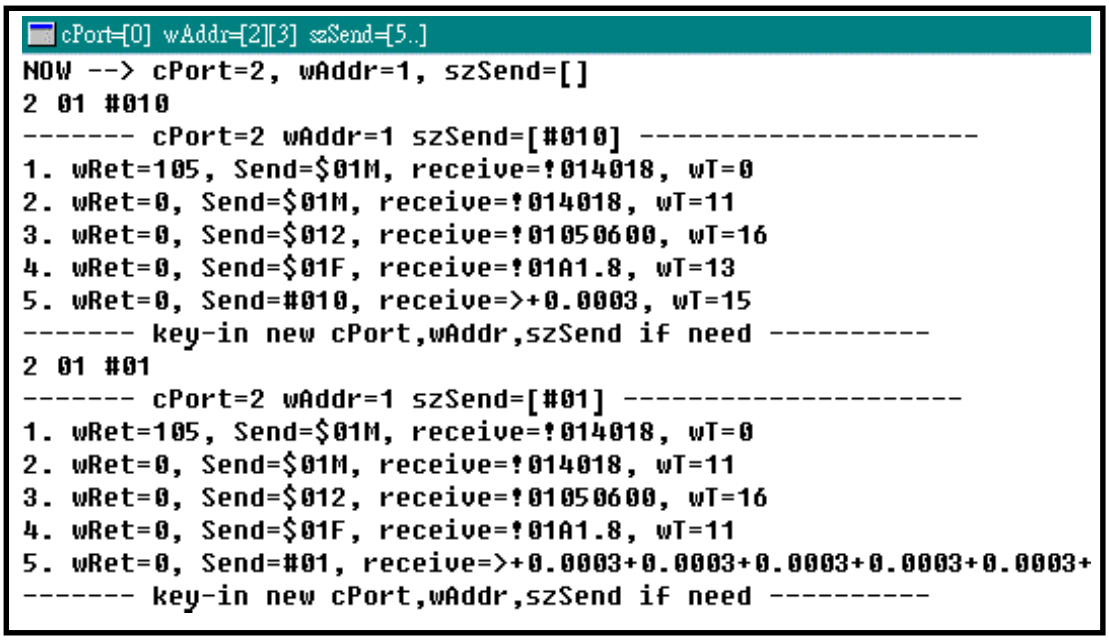
```
void TEST_CMD(HWND hwnd, int x, int dx, int y, int dy)
{
char cBuf[80],cShow[80],cResult[80];
HDC hdc;
WORD wRet,i,j,wT;
char cData=8, cStop=OneStopBit, cParity=NonParity;
DWORD dwBaudRate=9600L;
WORD wCheckSum=0;       /* check-sum disable */
WORD wTimeOut=100;      /* default time-out constant=100 */

iLine=0;
hdc=GetDC(hwnd);
HideCaret(hwnd);
SelectObject(hdc,GetStockObject(SYSTEM_FIXED_FONT));

sprintf(cShow,"------- cPort=%d wAddr=%d szSend=[%s] ----------
",cPort,wAddr,szSend);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

wRet=Open_Com(cPort,dwBaudRate,cData,cParity,cStop);
if (wRet!=NoError)
  {sprintf(cShow,"0. Open Error, return code=%d",wRet);
   TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
   goto ret_label;}
// ------------------- multi-task driver demo ----------------------
i=wAddr/16;
j=wAddr-i*16;
cBuf[0]='$';
cBuf[1]=HEX_TO_ASCII(i);
cBuf[2]=HEX_TO_ASCII(j);
cBuf[3]='M';
cBuf[4]=0;
```

```
wRet = Send_Cmd(cPort,cBuf,1000,wCheckSum); /* time-out = 1000 ms = 1 sec */
if (wRet!=NoError)
  {
  sprintf(cShow,"1. Send_Cmd error, return code=%d",wRet);
  TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
  goto ret_label;
  }

cResult[0]=0;
for (;;)
   { wRet=Read_Com_Status(cPort,cResult,&wT);
     if (wRet>=0x100) break;
     Sleep(1);}

sprintf(cShow,"1. wRet=%x, Send=%s, receive=%s, wT=%d",wRet,cBuf,cResult,wT);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
```

// ---------- Non-multi-task driver demo ----------

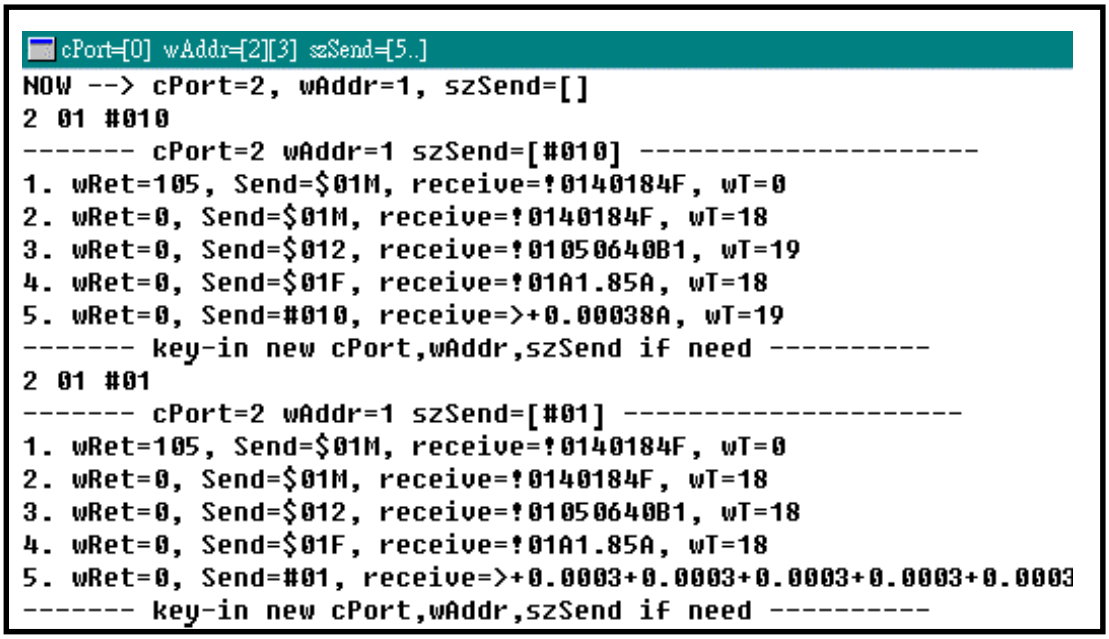
```
cBuf[4]=0;
wRet = Send_Receive_Cmd(cPort,cBuf,cResult,wTimeOut,wCheckSum,&wT);
sprintf(cShow,"2. wRet=%d, Send=%s, receive=%s, wT=%d",wRet,cBuf,cResult,wT);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

cBuf[3]='2';
cBuf[4]=0;
wRet = Send_Receive_Cmd(cPort,cBuf,cResult,wTimeOut,wCheckSum,&wT);
sprintf(cShow,"3. wRet=%d, Send=%s, receive=%s, wT=%d",wRet,cBuf,cResult,wT);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

cBuf[3]='F';
cBuf[4]=0;
wRet = Send_Receive_Cmd(cPort,cBuf,cResult,wTimeOut,wCheckSum,&wT);
sprintf(cShow,"4. wRet=%d, Send=%s, receive=%s, wT=%d",wRet,cBuf,cResult,wT);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
wRet = Send_Receive_Cmd(cPort,szSend,cResult,wTimeOut,wCheckSum,&wT);
sprintf(cShow,"5. wRet=%d, Send=%s, receive=%s,
wT=%d",wRet,szSend,cResult,wT);
```

```
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

ret_label:
wRet=Close_Com(cPort);
if (wRet!=NoError)
 {sprintf(cShow,"5. Close Error, return code=%d",wRet);
 TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;}

sprintf(cShow,"------- key-in new cPort,wAddr,szSend if need ----------");
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

ShowCaret(hwnd);
ReleaseDC (hwnd,hdc);
}
```



3.2.2 Demo2: Send/Receive command to 7000 with CheckSum Enable

Refer to /NAP7000P/Demo/Vc/DEMO2/DEMO2.C for program source



3.2.3 Demo3: Send/Receive command to Counter, ANC PC-202

```
void TEST_CMD(HWND hwnd, int x, int dx, int y, int dy)
{
char cBuf[80],cShow[80],cResult[80];
HDC hdc;
WORD wRet,wT;
char cData=8, cStop=OneStopBit, cParity=NonParity;
DWORD dwBaudRate=9600L;
WORD wSendLen,wReceiveLen;

iLine=0;
hdc=GetDC(hwnd);
HideCaret(hwnd);
SelectObject(hdc,GetStockObject(SYSTEM_FIXED_FONT));

sprintf(cShow,"------- cPort=%d wAddr=%d szSend=[%s] ----------------------
",cPort,wAddr,szSend);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

wRet=Open_Com(cPort,dwBaudRate,cData,cParity,cStop);
if (wRet!=NoError)
  {
  sprintf(cShow,"0. Open Error, return code=%d",wRet);
  TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
  goto ret_label;
  }

// ------------------- multi-task driver demo ------------------------

strcpy(cBuf,"<01C00007B22>");

wSendLen=13;     //  <01C00007B22>
wReceiveLen=10;  //  (00007Bc6)
```

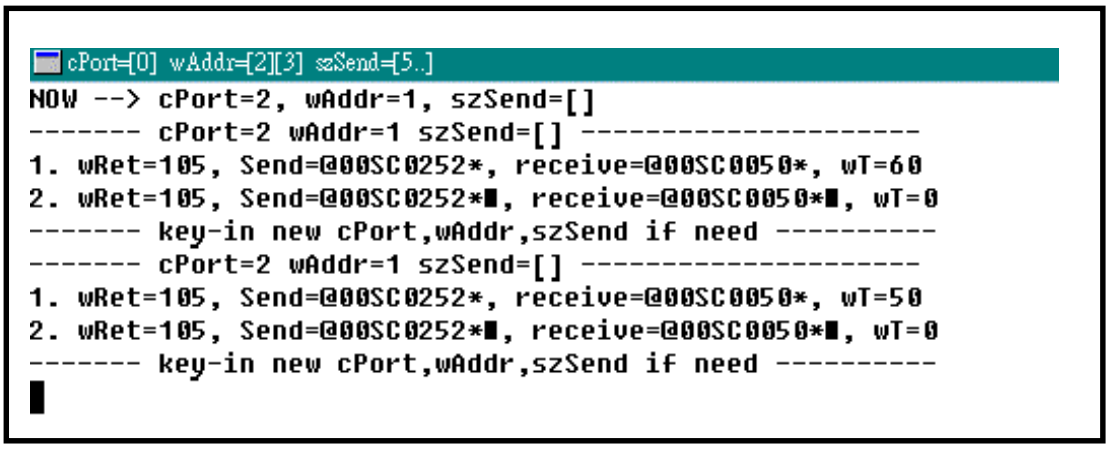
```
wRet = Send_Str(cPort,cBuf,1000,wSendLen,wReceiveLen); /* time-out = 1000 ms = 1
sec */
if (wRet!=NoError)
  {
  sprintf(cShow,"1. Send_Str error, return code=%d",wRet);
  TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
  goto ret_label;
  }

cResult[0]=0;
for (;;)
   {
   wRet=Read_Com_Status(cPort,cResult,&wT);
   if (wRet>=0x100) break;
   Sleep(1);
   }

sprintf(cShow,"1. wRet=%x, Send=%s, receive=%s, wT=%d",wRet,cBuf,cResult,wT);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

ret_label:
wRet=Close_Com(cPort);
if (wRet!=NoError)
  {
  sprintf(cShow,"2. Close Error, return code=%d",wRet);
  TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
  }

sprintf(cShow,"------- key-in new cPort,wAddr,szSend if need ----------");
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

ShowCaret(hwnd);
ReleaseDC (hwnd,hdc);
}
```

3.2.4 Demo4: Send/Receive command to PLC

```
void TEST_CMD(HWND hwnd, int x, int dx, int y, int dy)
{
char cBuf[80],cShow[80],cResult[80];
HDC hdc;
WORD wRet,wT;
DWORD dwBaudRate=9600L;
WORD wSendLen,wReceiveLen;
char cData=7, cStop=TwoStopBit, cParity=EvenParity;

iLine=0;
hdc=GetDC(hwnd);
HideCaret(hwnd);
SelectObject(hdc,GetStockObject(SYSTEM_FIXED_FONT));

sprintf(cShow,"------- cPort=%d wAddr=%d szSend=[%s] ----------------------
",cPort,wAddr,szSend);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

wRet=Open_Com(cPort,dwBaudRate,cData,cParity,cStop);
if (wRet!=NoError)
  {
  sprintf(cShow,"0. Open Error, return code=%d",wRet);
  TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
  goto ret_label;
  }

// ------------------- Send-Cmd driver demo -----------------------

strcpy(cBuf,"@00SC0252*");
wRet = Send_Cmd(cPort,cBuf,1000,0); /* time-out = 1000 ms = 1 sec */
                          /* check-sum must disable */
                          /* auto add 0x0D in the input string */
if (wRet!=NoError)
  {
```

```
  sprintf(cShow,"1. Send_Cmd error, return code=%d",wRet);
  TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
  goto ret_label;
  }

cResult[0]=0;
for (;;)
   {
   wRet=Read_Com_Status(cPort,cResult,&wT);
   if (wRet>=0x100) break;
   Sleep(1);
   }

sprintf(cShow,"1. wRet=%x, Send=%s, receive=%s, wT=%d",wRet,cBuf,cResult,wT);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;


// ------------------- Send-Str driver demo ----------------------

strcpy(cBuf,"@00SC0252*");
cBuf[10]=0x0d;
cBuf[11]=0;
wSendLen=11;     //@00SC0252*+[0x0D]
wReceiveLen=11;  //@00SC0050*+[0x0D]
wRet = Send_Str(cPort,cBuf,1000,wSendLen,wReceiveLen);
                        /* will not add 0x0D in the input string */
if (wRet!=NoError)
  {
  sprintf(cShow,"2. Send_Str error, return code=%d",wRet);
  TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
  goto ret_label;
  }

cResult[0]=0;
for (;;)
   {
   wRet=Read_Com_Status(cPort,cResult,&wT);
   if (wRet>=0x100) break;
```

```
   Sleep(1);
   }

sprintf(cShow,"2.wRet=%x, Send=%s, receive=%s, wT=%d",wRet,cBuf,cResult,wT);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

ret_label:
wRet=Close_Com(cPort);
if (wRet!=NoError)
  {
  sprintf(cShow,"3. Close Error, return code=%d",wRet);
  TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
  }

sprintf(cShow,"------- key-in new cPort,wAddr,szSend if need ----------");
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

ShowCaret(hwnd);
ReleaseDC (hwnd,hdc);
}
```



3.2.5 Demo5: Multi-speed Demo

```
void TEST_CMD(HWND hwnd, int x, int dx, int y, int dy)
{
char cBuf[80],cShow[80],cResult[80];
HDC hdc;
WORD wRet,wT;
char cData=8, cStop=OneStopBit, cParity=NonParity;
DWORD dwBaudRate=9600L;
WORD wCheckSum=0;       /* check-sum disable */
WORD wTimeOut=100;      /* default time-out constant=100 */

iLine=0;
hdc=GetDC(hwnd);
HideCaret(hwnd);
SelectObject(hdc,GetStockObject(SYSTEM_FIXED_FONT));

sprintf(cShow,"------- cPort=%d wAddr=%d szSend=[%s] ----------------------
",cPort,wAddr,szSend);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

/* ---------- 9600 driver demo ------------------------- */

dwBaudRate=9600L;
wRet=Open_Com(cPort,dwBaudRate,cData,cParity,cStop);
if (wRet!=NoError)
  {
  sprintf(cShow,"1. Open Error, return code=%d",wRet);
  TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
  goto ret_label;
  }

strcpy(cBuf,"$01M");
wRet = Send_Receive_Cmd(cPort,cBuf,cResult,wTimeOut,wCheckSum,&wT);
sprintf(cShow,"1. wRet=%d, Send=%s, receive=%s, wT=%d",wRet,cBuf,cResult,wT);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
```

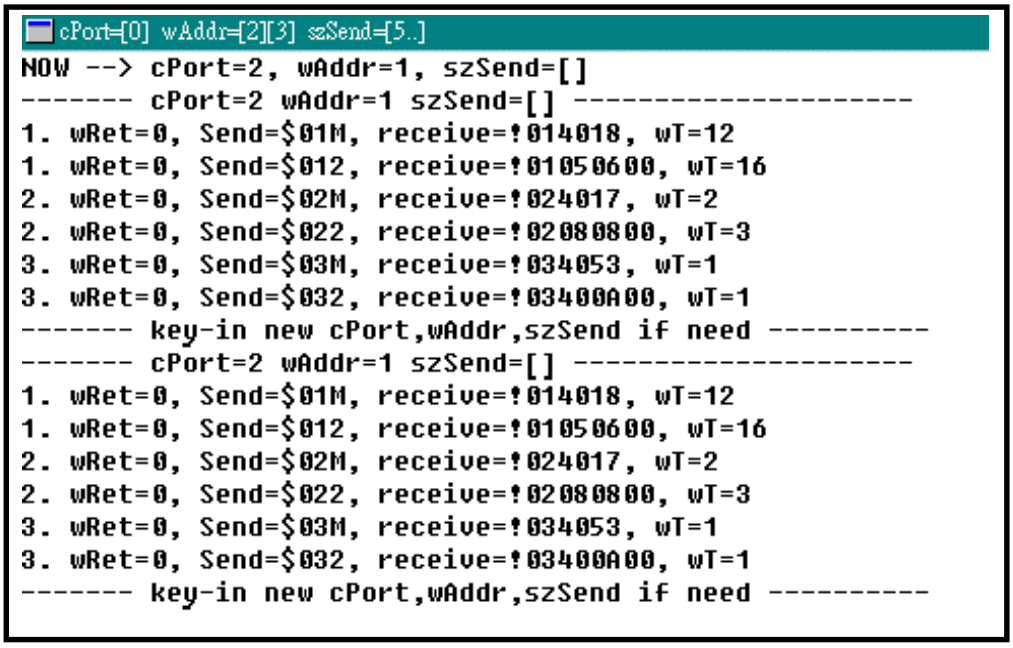
```
strcpy(cBuf,"$012");
wRet = Send_Receive_Cmd(cPort,cBuf,cResult,wTimeOut,wCheckSum,&wT);
sprintf(cShow,"1. wRet=%d, Send=%s, receive=%s, wT=%d",wRet,cBuf,cResult,wT);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

wRet=Close_Com(cPort);
if (wRet!=NoError)
  {
  sprintf(cShow,"1. Close Error, return code=%d",wRet);
  TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
  }
```

/* ---------- 38400 driver demo ---------- */

```
dwBaudRate=38400L;
wRet=Open_Com(cPort,dwBaudRate,cData,cParity,cStop);
if (wRet!=NoError)
  {
  sprintf(cShow,"2. Open Error, return code=%d",wRet);
  TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
  goto ret_label;
  }

strcpy(cBuf,"$02M");
wRet = Send_Receive_Cmd(cPort,cBuf,cResult,wTimeOut,wCheckSum,&wT);
sprintf(cShow,"2. wRet=%d, Send=%s, receive=%s, wT=%d",wRet,cBuf,cResult,wT);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

strcpy(cBuf,"$022");
wRet = Send_Receive_Cmd(cPort,cBuf,cResult,wTimeOut,wCheckSum,&wT);
sprintf(cShow,"2. wRet=%d, Send=%s, receive=%s, wT=%d",wRet,cBuf,cResult,wT);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

wRet=Close_Com(cPort);
if (wRet!=NoError)
  {
  sprintf(cShow,"2. Close Error, return code=%d",wRet);
```

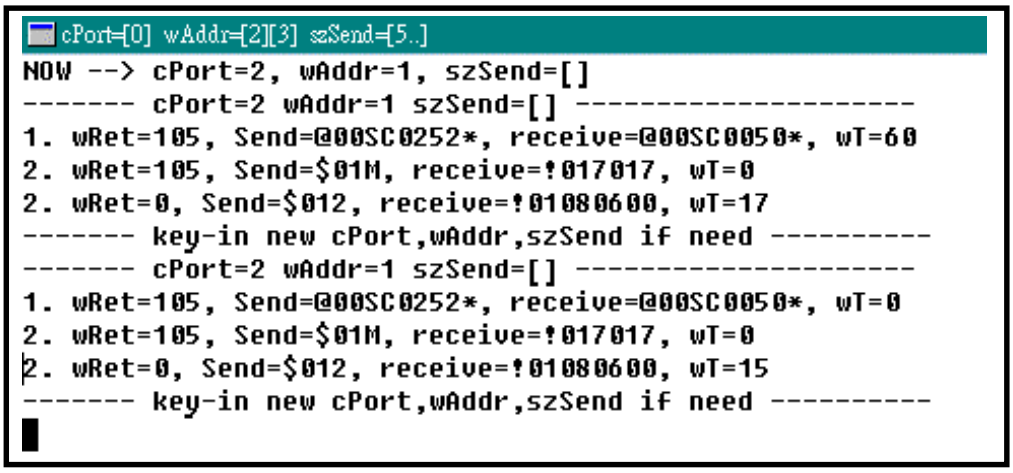
```
  TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
  }

/* ---------- 115200 driver demo ------------------------- */

dwBaudRate=115200L;
wRet=Open_Com(cPort,dwBaudRate,cData,cParity,cStop);
if (wRet!=NoError)
  {
  sprintf(cShow,"3. Open Error, return code=%d",wRet);
  TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
  goto ret_label;
  }

strcpy(cBuf,"$03M");
wRet = Send_Receive_Cmd(cPort,cBuf,cResult,wTimeOut,wCheckSum,&wT);
sprintf(cShow,"3. wRet=%d, Send=%s, receive=%s, wT=%d",wRet,cBuf,cResult,wT);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

strcpy(cBuf,"$032");
wRet = Send_Receive_Cmd(cPort,cBuf,cResult,wTimeOut,wCheckSum,&wT);
sprintf(cShow,"3. wRet=%d, Send=%s, receive=%s, wT=%d",wRet,cBuf,cResult,wT);
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

wRet=Close_Com(cPort);
if (wRet!=NoError)
  {
  sprintf(cShow,"3. Close Error, return code=%d",wRet);
  TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;
  }

ret_label:

sprintf(cShow,"------- key-in new cPort,wAddr,szSend if need ----------");
TextOut(hdc,x*dx,(y+iLine)*dy,cShow,strlen(cShow)); iLine++;

ShowCaret(hwnd);
ReleaseDC (hwnd,hdc);
```

}


cPort=[0] wAddr=[2][3] szSend=[5..]
NOW --> cPort=2, wAddr=1, szSend=[]
------- cPort=2 wAddr=1 szSend=[] ----------
1. wRet=0, Send=$01M, receive=!014018, wT=12
1. wRet=0, Send=$012, receive=!01050600, wT=16
2. wRet=0, Send=$02M, receive=!024017, wT=2
2. wRet=0, Send=$022, receive=!02080800, wT=3
3. wRet=0, Send=$03M, receive=!034053, wT=1
3. wRet=0, Send=$032, receive=!03400A00, wT=1
------- key-in new cPort,wAddr,szSend if need ----------
------- cPort=2 wAddr=1 szSend=[] ----------
1. wRet=0, Send=$01M, receive=!014018, wT=12
1. wRet=0, Send=$012, receive=!01050600, wT=16
2. wRet=0, Send=$02M, receive=!024017, wT=2
2. wRet=0, Send=$022, receive=!02080800, wT=3
3. wRet=0, Send=$03M, receive=!034053, wT=1
3. wRet=0, Send=$032, receive=!03400A00, wT=1
------- key-in new cPort,wAddr,szSend if need ----------

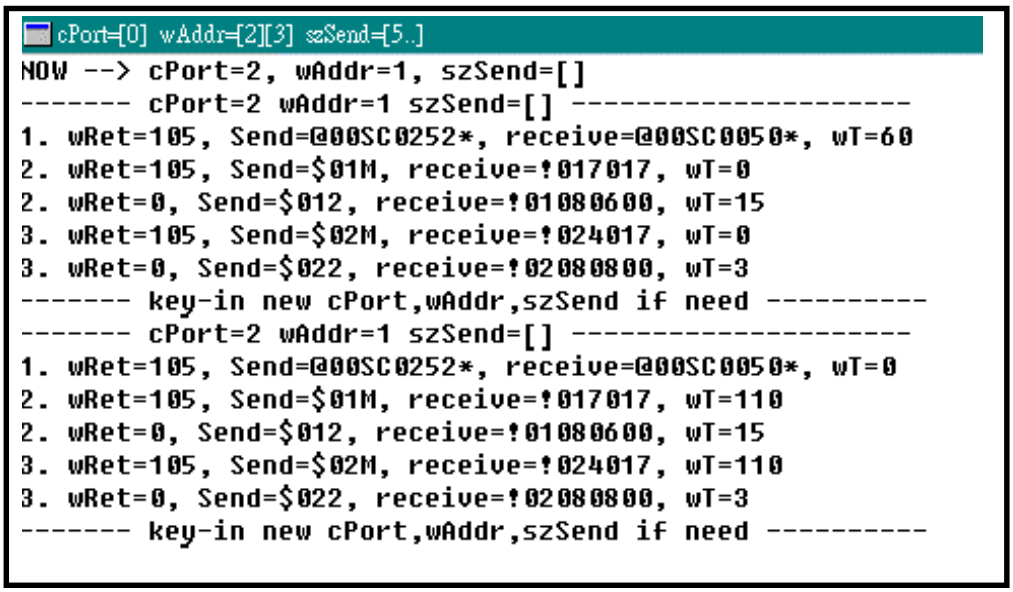
3.2.6 Demo6: Multi-data-format Demo

Refer to /NAP7000P/Demo/Vc/DEMO6/DEMO6.C for program source



3.2.7 Demo7: Multi-speed & Multi-data-fomat Demo

Refer to /NAP7000P/Demo/Vc/DEMO7/DEMO7.C for program source



4 I7000.DLL

4.1 Test Functions

Please refer to Sec. 2.7 for “Include Files”, and Sec. 4.2 for demo programs.

4.1.1 Short_Sub_2

- **Description:**

  Compute C=A-B in **short** format, **short=16 bits sign integer.** This function is provided for testing purpose. To test this DLLs can be called by your programming language, call this subroutine for testing. If this subroutine return the correct value, the other DLLs will work OK also.

- **Syntax:**

  short Short_Sub_2(short nA, short nB);

- **Input Parameter:**

  nA : short integer

  nB : short integer

- **Return Value:** return=nA-nB → short integer

- **Demo Program:** void

4.1.2 Float_Sub_2

- **Description:**

  Compute C=A-B in **float** format, **float=32 bits floating pointer number.** This function is provided for testing purpose. To test this DLLs can be called by your programming language, call this subroutine for testing. If this subroutine return the correct value, the other DLLs will work OK also.

- **Syntax:**

  float Float_Sub_2(float fA, float fB);

- **Input Parameter:**

  fA : floating point value

  fB : floating point value

- **Return Value:** return=fA-fB → floating point value

- **Demo Program:** void

4.1.3 Get_Dll_Version

- **Description:**
  Read the software version of the NAP7000P DLLs.

- **Syntax:**
  WORD Get_Dll_Version(void) ;

- **Input Parameter:** void

- **Return Value:**
  return=0x202 → Version 2.2

- **Demo Program:** void

4.1.4 Test

- **Description:** Test function.

- **Syntax:**

  Test(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:** NoError

- **Demo Program:** void

4.2 Analog Input/Output Functions

4.2.1 AnalogIn

- **Description:**

  Read the analog input value from 7000.

- **Syntax:**

  AnalogIn(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **Demo Program:** Sec. 4.2

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4
w7000[1] :	module address, from 0x00 to 0xFF
w7000[2] :	**module ID, 0x7011/7012/7013/7014/7017/7018/7033**
w7000[3] :	0=checksum disable, 1=checksum enable
w7000[4] :	TimeOut constant, normal=100
w7000[5] :	channel number for 7017/7018/7033
w7000[6] :	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  f7000[0] : analog input value return

- **Example:**

```
w7000[0]=cPort;          // port
w7000[1]=wAddr;          // Address
w7000[2]=0x7033;              // ID
w7000[3]=0;              // CheckSum disable
w7000[4]=wTimeOut;       // TimeOut constant, normal=100
w7000[5]=0;              // Channel 0
w7000[6]=1;              // string debug
wRet= AnalogIn(w7000, f7000, szSend, szReceive);
if (wRet != 0 ) {
        // Something Error .
}
else {
        fFloatVal = f7000[0];  // The analog input value
}
```

4.2.2 AnalogInHex

- **Description:**

  Read the analog input value(in Hex format) from 7000.

- **Syntax:**

  AnalogInHex(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **Demo Program:** Sec. 4.2

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4
w7000[1] :	module address, from 0x00 to 0xFF
w7000[2] :	**module ID, 0x7011/7012/7013/7014/7017/7018/7033**
w7000[3] :	0=checksum disable, 1=checksum enable
w7000[4] :	TimeOut constant, normal=100
w7000[5] :	channel number for 7017/7018/7033
w7000[6] :	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000
w7000[7]:	The analog input value in Hex format.

- **F7000: Float Input/Output Table**

  Not used.

- **Example:**

```
w7000[0]=cPort;          // port
w7000[1]=wAddr;          // Address
w7000[2]=0x7033;              // ID
w7000[3]=0;              // CheckSum disable
w7000[4]=wTimeOut;       // TimeOut constant, normal=100
w7000[5]=0;              // Channel 0
w7000[6]=1;              // string debug
wRet= AnalogInHex(w7000, f7000, szSend, szReceive);
if (wRet != 0 ) {
        // Something Error .
}
else {
        wHexVal = w7000[7];  // The analog input value
}
```

4.2.3 AnalogInFsr

- **Description:**

  Read the analog input value(in FSR format) from 7000.

- **Syntax:**

  AnalogInFsr(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **Demo Program:** Sec. 4.2

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4	
w7000[1] :	module address, from 0x00 to 0xFF	
w7000[2] :	**module ID, 0x7011/7012/7013/7014/7017/7018/7033**	
w7000[3] :	0=checksum disable, 1=checksum enable	
w7000[4] :	TimeOut constant, normal=100	
w7000[5] :	channel number for 7017/7018/7033	
w7000[6] :	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  f7000[0]: The analog input value.

- **Example:**

```
w7000[0]=cPort;          // port
w7000[1]=wAddr;          // Address
w7000[2]=0x7033;              // ID
w7000[3]=0;              // CheckSum disable
w7000[4]=wTimeOut;       // TimeOut constant, normal=100
w7000[5]=0;              // Channel 0
w7000[6]=1;              // string debug
wRet= AnalogInFsr(w7000, f7000, szSend, szReceive);
if (wRet != 0 ) {
        // Something Error .
}
else {
        fFsrVal = f7000[0];  // The analog input value
}
```

4.2.4 AnalogIn8

- **Description:**

  Read the 8 channels of analog input values from 7017 or 7018.

- **Syntax:**

  AnalogIn8(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **Demo Program:** Sec. 4.2

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4
w7000[1] :	module address, from 0x00 to 0xFF
w7000[2] :	**module ID, 0x7017/7018**
w7000[3] :	0=checksum disable, 1=checksum enable
w7000[4] :	TimeOut constant, normal=100
w7000[6] :	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  f7000[0] : analog input value of channel_0

  f7000[1] : analog input value of channel_1

  ……

  f7000[7] : analog input value of channel_7

4.2.5 AnalogOut

- **Description:**

  Send the analog output command to a 7000 module.

- **Syntax:**

  AnalogOut(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000 (for debug)
szReceiveFrom7000:	result string read from 7000 (for debug)

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **Demo Program:** Sec. 4.2

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4
w7000[1] :	module address, from 0x00 to 0xFF
w7000[2] :	module ID, 0x7021/ 0x7024
w7000[3] :	0=checksum disable, 1=checksum enable
w7000[4] :	TimeOut constant, normal=100
w7000[5] :	Channel No. (0 to 3) if module ID is 7024 ; // Not used if module ID is 7021
w7000[6] :	0 → no save to szSendTo7000&szReceiveFrom7000 1 → szSendTo7000=command string send to 7000 → szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  f7000[0] : analog output value

- **Example:**

```
w7000[0]=cPort;     // port                          w7000[1]=wAddr;    // Address
w7000[2]=0x7021;    // ID                            w7000[3]=0;        // CheckSum disable
w7000[4]=wTimeOut; // TimeOut constant               w7000[6]=1;        // string debug
f7000[0]=5.432;     // DA output value
wRet=AnalogOut(w7000, f7000, szSend, szReceive);
```

4.2.6 AnalogOutReadBack

- **Description:**

  Read back the current D/A output value of 7021/7024. There are two types of analog output read back described as following:

  1. **command read back by $AA6 command**
  2. **analog output of current path read back by $AA8 command**

- **Syntax:**

  AnalogOutReadBack(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000 (for debug)
szReceiveFrom7000:	result string read from 7000 (for debug)

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **Demo Program:** Sec. 4.2

- **W7000: WORD Input/Output Table**

w7000[0]: RS-232 port number, 1/2/3/4

w7000[1]: module address, from 0x00 to 0xFF

w7000[2]: module ID, 0x7021 / 0x7024

w7000[3]: 0=checksum disable, 1=checksum enable

w7000[4]: TimeOut constant, normal=100

w7000[5]: 0: command read back ($AA6)

1: analog output of current path read back ($AA8)

w7000[6]: 0 → no save to szSendTo7000&szReceiveFrom7000

1 → szSendTo7000=command string send to 7000

→ szReceiveFrom7000=result string receive from 7000

w7000[7] : Channel No.(0 to 3), if wID is 7024

Not used if module ID is 7021 .

- **F7000: Float Input/Output Table**

f7000[0] : analog output read back value

4.3 Digital Input/Output Functions

4.3.1 DigitalIn

- **Description:**

  Read the digital input value from a 7000 module.

- **Syntax:**

  DigitalIn(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **Demo Program:** Sec. 4.2

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4
w7000[1] :	module address, from 0x00 to 0xFF
w7000[2] :	**module ID, 0x7050/7052/7053/7060/7041/7044**
w7000[3] :	0=checksum disable, 1=checksum enable
w7000[4] :	TimeOut constant, normal=100
w7000[5] :	16-bit digital input data
w7000[6] :	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **f7000: Float Input/Output Table**

  Not used

- **Example :**

```
w7000[0]=cPort;    // port
w7000[1]=wAddr;    // Address
w7000[2]=0x7053;   // ID
w7000[3]=0;        // CheckSum disable
w7000[4]=wTimeOut; // TimeOut constant
w7000[6]=1;        // string debug
wRet=DigitalIn(w7000, f7000, szSend, szReceive);
```

4.3.2 DigitalOut

- **Description:**

  To set the digital output value for a 7000 module.

- **Syntax:**

  DigitalOut(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **Demo Program:** Sec. 4.2

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4
w7000[1] :	module address, from 0x00 to 0xFF
w7000[2] :	**module ID, 0x7050/7060/7067/7042/7043/7044/7011/7012/7014**
w7000[3] :	0=checksum disable, 1=checksum enable
w7000[4] :	TimeOut constant, normal=100
w7000[5] :	16-bit digital output data
w7000[6] :	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Void

- **Example:**

```
w7000[0]=cPort;     // port
w7000[1]=wAddr;    // Address
w7000[2]=0x7053;   // ID
w7000[3]=0;        // CheckSum disable
w7000[4]=wTimeOut; // TimeOut constant
w7000[5]=wDoVal;  // digital output value
w7000[6]=1;        // string debug
wRet=DigitalOut(w7000, f7000, szSend, szReceive);
```

4.3.3 DigitalOutReadBack

- **Description:**

  Read back the digital output value of a 7000 module.

- **Syntax:**

  DigitalOutReadBack(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to 7000.H in Sec. 2.7.

- **Demo Program:** Sec. 4.2

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4
w7000[1] :	module address, from 0x00 to 0xFF
w7000[2] :	**module ID, 0x7050/7060/7067/7042/7043/7044**
w7000[3] :	0=checksum disable, 1=checksum enable
w7000[4] :	TimeOut constant, normal=100
w7000[5] :	16-bit digital output data read back
w7000[6] :	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **f7000: Float Input/Output Table**

  void

- **Example:**

```
w7000[0]=cPort;     // port
w7000[1]=wAddr;    // Address
w7000[2]=0x7053;   // ID
w7000[3]=0;        // CheckSum disable
w7000[4]=wTimeOut; // TimeOut constant
w7000[6]=1;        // string debug
wRet=DigitalOutReadBack(w7000, f7000, szSend, szReceive);
// w7000[5] = digital output read back
```

4.4 Misc Functions

NetworkAnalogIn

- **Description:**

  Read the multi-module analog input value from 7000 RS-485 network. The user can call AnalogIn to read analog input value one by one or call this function once for easy programming.

- **Syntax:**

  NetworkAnalogIn(WORD wPort, WORD wTotal, WORD wT, WORD wID[], WORD wConfig[], WORD wChksum[], float f7000[])

- **Input Parameter:**

wPort:	RS-232 port number, 1/2/3/4
wTotal:	number of modules to read
wT:	time out constant, normal=100
wID:	wID[?]=module address of module_?, from 0x00 to 0xFF
wConfig:	if wID[?]=0x7017 then wConfig[?]=08/09/0A/0B/0C/0D
	if wID[?] !=0x7017 then wConfig[?] is ignored
wChksum:	if wChksum[?]=1 → the checksum of module_? is enable

f7000[0]: analog value of module_0, channel_0 f7000[1]: analog value of module_0, channel_1 …….. f7000[7]: analog value of module_0, channel_7

f7000[8]: analog value of module_1, channel_0 …….. f7000[15]: analog value of module_1, channel_7

f7000[n*8]: analog value of module_n, channel_0 f7000[n*8+1]: analog value of module_n, channel_1 …….. f7000[n*8+7]: analog value of module_n, channel_7

- **Return Value:**

  NoError: OK

  others: Error code, refer to I7000.H in Sec. 2.7.

- **Demo Program:** Sec. 4.2
- **Example:**

```
Open_Com(COM_PORT,9600L,cData,cParity,cStop);       // open com firstly

wTotal=4;                   // total 4 modules
wAddr[0]=3;                 // module address of module_0 = 0x03
wID[0]=0x7050;              // module_0 is 7050
wCheckSum[0]=0;             // the checksum of module_0 is disable

wAddr[1]=0x11;              // module address of module_1 = 0x11
wID[1]=0x7052;              // module_1 is 7052
wCheckSum[1]=1;             // the checksum of module_1 is enable

wAddr[2]=0x20;              // module address of module_2 = 0x20
wID[2]=0x7053;              // module_2 is 7053
wCheckSum[2]=1;             // the checksum of module_2 is enable

wAddr[3]=0x21;              // module address of module_3 = 0x21
wID[3]=0x7060;              // module_3 is 7060
wCheckSum[3]=0;             // the checksum of module_3 is disable

wRet=NetworkDigitalIn(cPort, wTotal, wTimeOut, wAddr, wID, wConfig,
wCheckSum, w7000);
if    (wRet == NoError)
      {
            // w7000[0]=16-bit digital value of module_0 (7050 is 7-bit)
            // w7000[1]=16-bit digital value of module_1 (7052 is 8-bit)
            // w7000[2]=16-bit digital value of module_2 (7053 is 16-bit)
            // w7000[3]=16-bit digital value of module_3 (7060 is 4-bit)
      }
Close_Com(COM_PORT); // close com if stop the program
```

- **Description:**

  Read the multi-module digital input value from 7000 RS-485 network. The user can call DigitalIn to read analog input value one by one or call this function once for easy programming.

- **Syntax:**

  NetworkDigitalIn(WORD wPort, WORD wTotal, WORD wT, WORD wID[], WORD wConfig[], WORD wChksum[], WORD w7000[])

- **Input Parameter:**

wPort:	RS-232 port number, 1/2/3/4
wTotal:	number of modules to read
wT:	time out constant, normal=100
wID:	wID[?]=module address of module_?, from 0x00 to 0xFF
wConfig:	reserved.
wChksum:	if wChksum[?]=1 → the checksum of module_? is enable
w7000[0]:	16-bit digital value of module_0
w7000[1]:	16-bit digital value of module_1
……	
w7000[n]:	16-bit digital value of module_n

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **Demo Program:** Sec. 4.2

- Open_Com(COM_PORT,9600L,cData,cParity,cStop); // open com first

```
wTotal=4;               // total 4 modules
wAddr[0]=3;             // module address of module_0 = 0x03
wID[0]=0x7012;          // module_0 is 7012
wCheckSum[0]=0;         // the checksum of module_0 is disable

wAddr[1]=0x11;          // module address of module_1 = 0x11
wID[1]=0x7013;          // module_1 is 7013
wCheckSum[1]=1;         // the checksum of module_1 is enable

wAddr[2]=0x20;          // module address of module_2 = 0x20
wID[2]=0x7018;          // module_2 is 7018
wCheckSum[2]=1;         // the checksum of module_2 is enable

wAddr[3]=0x21;          // module address of module_3 = 0x21
wID[3]=0x7017;          // module_3 is 7017
wConfig[3]=0x08;        // +/-10V range
wCheckSum[3]=0;         // the checksum of module_3 is disable

wRet=NetworkAnalogIn(cPort, wTotal, wTimeOut, wAddr, wID, wConfig,
wCheckSum, f7000);
if    (wRet == NoError)
      {
            // f7000[0]=analog value of module_0 (7012 is single channel)
            // f7000[8]=analog value of module_1 (7013 is single channel)
            // f7000[16+0]=analog value of module_2, channel_0
            // ……..
            // f7000[16+7]=analog value of module_2, channel_7
            // f7000[24+0]=analog value of module_3, channel_0
            // ……..
            // f7000[24+7]=analog value of module_3, channel_7
      }

Close_Com(COM_PORT); // close com if stop the program
```

4.4.1 ThermocoupleOpen_7011 [7011 only]

- **Description:**

  To detect if the thermocouple is open for a 7011 module.

- **Syntax:**

  ThermocoupleOpen_7011(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4	
w7000[1] :	module address, from 0x00 to 0xFF	
w7000[2] :	**module ID, 0x7011**	
w7000[3] :	0=checksum disable, 1=checksum enable	
w7000[4] :	TimeOut constant, normal=100	
w7000[5] :	0	→ the thermocouple is close
	1	→ the thermocouple is open
w7000[6] :	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

- **Example:**

Code Fragment

```
w7000[0]=cPort;          // port
w7000[1]=wAddr;          // Address
w7000[2]=0x7011;         // ID
w7000[3]=0;              // CheckSum disable
w7000[4]=wTimeOut;       // TimeOut constant, normal=100
w7000[6]=1;              // string debug
wRet=Thermocouple_7011(w7000, f7000, szSend, szReceive);
if( w7000[5]==0 )
{
        // the thermocouple is close to this 7011 module
}
else
{
        // the thermocouple is open to this 7011 module
}
```

4.4.2 GetLedDisplay_7033

- **Description:**

The module 7033 has three channels of Analog Input. The LED can display the Analog Input value of specify channel. The function GetLedDisplay_7033() can get the active channel number for 7033 module.

- **Syntax:**

GetLedDisplay_7033(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0] : RS-232 port number, 1/2/3/4
w7000[1] : module address, from 0x00 to 0xFF
w7000[2] : module ID, 0x7033
w7000[3] : 0=checksum disable, 1=checksum enable
w7000[4] : TimeOut constant, normal=100
w7000[5] : Return the channel number that is actived (Displayed)
w7000[6] : 0 → no save to szSendTo7000&szReceiveFrom7000
1 → szSendTo7000=command string send to 7000
→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

- **Example:**

Code Fragment

```
w7000[0]=cPort;          // port
w7000[1]=wAddr;          // Address
w7000[2]=0x7033;         // ID
w7000[3]=0;              // CheckSum disable
w7000[4]=wTimeOut;       // TimeOut constant, normal=100
w7000[6]=1;              // string debug
wRet=GetLedDisplay_7033(w7000, f7000, szSend, szReceive);
if( wRet != 0) {
        // Something Error
}
else {
        wChannelNo = w7000[5];
}
```

4.4.3 SetLedDisplay_7033

- **Description:**

The module 7033 has three channels of Analog Input. The LED can display the Analog Input value of specify channel. The function SetLedDisplay_7033() is used to set the active channel number for 7033 module.

- **Syntax:**

SetLedDisplay_7033(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0] : RS-232 port number, 1/2/3/4
w7000[1] : module address, from 0x00 to 0xFF
w7000[2] : module ID, 0x7033
w7000[3] : 0=checksum disable, 1=checksum enable
w7000[4] : TimeOut constant, normal=100
w7000[5] : The channel number that is setted to actived (Displayed)
w7000[6] : 0 → no save to szSendTo7000&szReceiveFrom7000
1 → szSendTo7000=command string send to 7000
→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

- **Example:**

Code Fragment

```
w7000[0]=cPort;          // port
w7000[1]=wAddr;          // Address
w7000[2]=0x7033;         // ID
w7000[3]=0;              // CheckSum disable
w7000[4]=wTimeOut;       // TimeOut constant, normal=100
w7000[5]=0;              // Setting the channel 0 to be dispalyed.
w7000[6]=1;              // string debug
wRet=SetLedDisplay_7033(w7000, f7000, szSend, szReceive);
if( wRet != 0) {
        // Something Error
}
```

4.5 Alarm Functions

4.5.1 EnableAlarm [7011/7012/7014 only]

- **Description:**

  To let 7000 module enter *momentary alarm mode* or *latch alarm mode*.

- **Syntax:**

  EnabledAlarm(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4	
w7000[1] :	module address, from 0x00 to 0xFF	
w7000[2] :	**module ID, 0x7011/7012/7014**	
w7000[3] :	0=checksum disable, 1=checksum enable	
w7000[4] :	TimeOut constant, normal=100	
w7000[5] :	0	→ momentary alarm mode
	1	→ latch alarm mode
w7000[6] :	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

4.5.2 DisableAlarm [7011/7012/7014 only]

- **Description:**

  To disable alarm function for a 7000 module.

- **Syntax:**

  DisableAlarm(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4
w7000[1] :	module address, from 0x00 to 0xFF
w7000[2] :	**module ID, 0x7011/7012/7014**
w7000[3] :	0=checksum disable, 1=checksum enable
w7000[4] :	TimeOut constant, normal=100
w7000[5] :	not used
w7000[6] :	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

4.5.3 ClearLatchAlarm [7011/7012/7014 only]

- **Description:**

  To clear the latch alarm for a 7000 mode.

- **Syntax:**

  ClearLatchAlarm(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4
w7000[1] :	module address, from 0x00 to 0xFF
w7000[2] :	**module ID, 0x7011/7012/7014**
w7000[3] :	0=checksum disable, 1=checksum enable
w7000[4] :	TimeOut constant, normal=100
w7000[5] :	not used
w7000[6] :	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

4.5.4 SetAlarmLimitValue [7011/7012/7014 only]

- **Description:**

  To set a high or low alarm limit value for a 7000 module.

- **Syntax:**

  SetAlarmLimitValue(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7011/0x7012/0x7014**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5]:	0	→ low alarm value setting
	1	→ high alarm value setting
w7000[6]:	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

f7000[0] :	alarm value

- **Example:**

Code Fragment

```
w7000[0]=cPort;              // port
w7000[1]=wAddr;              // Address
w7000[2]=0x7011;             // ID
w7000[3]=0;                  // CheckSum disable
w7000[4]=wTimeOut;           // TimeOut constant, normal=100
w7000[5]=0;                  //  to set low alarm value
w7000[6]=1;                  // string debug
f7000[0]=300.0;              //  the low alarm value is 300
wRet=SetAlarmLimitValue(w7000, f7000, szSend, szReceive);
if( wRet>0 )
{
        // the setting alarm value process is error
}
else
{
        // the setting alarm value process is successful
}
```

4.5.5 ReadAlarmLimitValue [7011/7012/7014 only]

- **Description:**

  To get the high or low alarm limit value for a 7000 module.

- **Syntax:**

  GetAlarmLimitValue(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7011/0x7012/0x7014**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5]:	0	→ low alarm value setting
	1	→ high alarm value setting
w7000[6]:	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

f7000[0] :	alarm value

- **Example:**

Code Fragment

```
w7000[0]=cPort;              // port
w7000[1]=wAddr;              // Address
w7000[2]=0x7011;             // ID
w7000[3]=0;                  // CheckSum disable
w7000[4]=wTimeOut;           // TimeOut constant, normal=100
w7000[5]=0;                  //  to get the low alarm value
w7000[6]=1;                  // string debug
wRet=GetAlarmLimitValue(w7000, f7000, szSend, szReceive);
if( wRet>0 )
{
        // the setting alarm value process is error
}
else
{
        // the get alarm limit value process is successful
        // and the alarm limit value stored in the f7000[0]
}
```

4.5.6 ReadOutputAlarmState [7011/7012/7014 only]

- **Description:**

  Reading the alarm mode and digital output value for a 7000 module.

- **Syntax:**

  ReadOutputAlarmState(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7011/0x7012/0x7014**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5]:	no used	
w7000[6]:	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000
w7000[7]:	0	→ alarm disable
	1	→ momentary alarm
	2	→ latch alarm

w7000[8]: 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

- **F7000: Float Input/Output Table**

Not used

- **Example:**

```
w7000[0]=cPort;              // port
w7000[1]=wAddr;              // Address
w7000[2]=0x7012;             // ID
w7000[3]=0;                  // CheckSum disable
w7000[4]=wTimeOut;           // TimeOut constant, normal=100
w7000[6]=1;                  // string debug
wRet=ReadOutputAlarmState(w7000, f7000, szSend, szReceive);
if( wRet>0 )  {
        // some error occur !!!
}
else {
        switch( w7000[7] )   // to detect the alarm status
        {
            case 0:  // the alarm is disable
                     break;
            case 1:  // the momentary alarm
                     break;
            case 2:  // the latch alarm
                     break;
        }
        switch( w7000[8] )   // to detect the digital output status
        {
            case 0:  // the DO:0 is off,     DO:1 is off
                     break;
            case 1:  // the DO:0 is on,      DO:1 is off
                     break;
            case 2:  // the DO:0 is off,     DO:1 is on
                     break;
            case 3:  // the DO:0 is on,      DO:1 is on
                     break;
        }
}
```

4.6 Event Counter Functions

4.6.1 ReadEventCounter [7011/7012/7014 only]

- **Description:**

  To read the value of event counter for a 7000 module.

- **Syntax:**

  ReadEventCounter(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	module address, from 0x00 to 0xFF
w7000[2]:	**module ID, 0x7011/0x7012/0x7014**
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	not used
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000
w7000[7]:	the reading event counter value

- **F7000: Float Input/Output Table**

  Not used

4.6.2 ClearEventCounter [7011/7012/7014 only]

- **Description:**

  To clear the value of event counter for a 7000 module.

- **Syntax:**

  ClearEventCounter(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	module address, from 0x00 to 0xFF
w7000[2]:	**module ID, 0x7011/0x7012/0x7014**
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	not used
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

4.7 7080 Functions

4.7.1 CounterIn_7080

- **Description:**

  To read the counter value of 7080 module.

- **Syntax:**

  CounterIn_7080(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	module address, from 0x00 to 0xFF
w7000[2]:	**module ID, 0x7080**
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	0 → to read 7080's counter 0
	1 → to read 7080's counter 1
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000
w7000[7]:	high word of counter value
w7000[8]:	low word of counter value

- **F7000: Float Input/Output Table**

  Not used

- **Example:**

```
unsigned long    CounterValue;
w7000[0]=cPort;          // port
w7000[1]=wAddr;          // Address
w7000[2]=0x7080;         // module ID
w7000[3]=0;              // CheckSum disable
w7000[4]=wTimeOut;   // TimeOut constant, normal=100
w7000[5]=0;              // to read counter 0's value
w7000[6]=1;                      // string debug
wRet=CounterIn_7080(w7000, f7000, szSend, szReceive);
if( wRet>0 )
{
            // some error occur !!!
}
else   // estimating counter value
{
      CounterValue=w7000[7]*65536+w7000[8];
}
```

4.7.2 ReadCounterMaxValue_7080

- **Description:**

  To read the maximum value of the counter for a 7080 module.

- **Syntax:**

  ReadCounterMaxValue_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	module address, from 0x00 to 0xFF
w7000[2]:	**module ID, 0x7080**
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	0 → to read counter 0's max value
	1 → to read counter 1's max value
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000
w7000[7]:	high word of counter value
w7000[8]:	low word of counter value

- **F7000: Float Input/Output Table**

  Not used

4.7.3 SetCounterMaxValue_7080

- **Description:**

  To set the maximum value of the counter for a 7080 module.

- **Syntax:**

  SetCounterMaxValue_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[],double MaxValue)

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000
MaxValue:	the max value set to counter

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7080**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5]:	0	→ to set counter 0's max value
	1	→ to set counter 1's max value
w7000[6]:	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

- **Example:**

```
double      CounterMaxValue;
w7000[0]=cPort;          // port
w7000[1]=wAddr;          // Address
w7000[2]=0x7080;         // module ID
w7000[3]=0;              // CheckSum disable
w7000[4]=wTimeOut;       // TimeOut constant, normal=100
w7000[5]=0;              // to set counter 0's max value
w7000[6]=1;              // string debug
CounterMaxValue=2000000.0;
wRet=SetCounterMaxValue_7080(w7000, f7000, szSend,
                        szReceive,CounterMaxValue);
if( wRet>0 )
{
        // some error occur !!!
}
else
{
    // successful !!!
}
```

4.7.4 EnableCounterAlarm_7080 [for 7080 mode only]

- **Description:**

  To set 7080 module enter *momentary alarm mode* or *latch alarm mode*.

- **Syntax:**

  EnableCounterAlarm_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4	
w7000[1] :	module address, from 0x00 to 0xFF	
w7000[2] :	**module ID, 0x7080**	
w7000[3] :	0=checksum disable, 1=checksum enable	
w7000[4] :	TimeOut constant, normal=100	
w7000[5] :	0	→ to enable alarm for counter 0
	1	→ to enable alarm for counter 1
w7000[6] :	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

- **Example:**

```
/** please refer to Sec 1.8.2 of  7080 User Manual for detailed information  about **/
/** the alarm mode 0(7080 alarm mode) and alarm mode 1(7080D alarm mode)   **/

/***** Set the module into 7080 alarm mode(mode 1) firstly *****/
double          CounterAlarmValue;
w7000[0]=cPort;                 // port
w7000[1]=wAddr;                 // Address
w7000[2]=0x7080;                // module ID
w7000[3]=0;                     // CheckSum disable
w7000[4]=wTimeOut;              // TimeOut constant, normal=100
w7000[5]=0;                     // set the module into 7080 alarm mode
w7000[6]=1;                     // string debug
wRet=SetModuleMode_7080(w7000, f7000, szSend, szReceive,CounterMaxValue);
if( wRet>0 )
{
        // some error occur !!!
        return( wRet );
}

CounterAlarmValue=200000.0;
w7000[5]=0;    // set counter 0's alarm value
wRet=SetAlarmLimitValue_7080(w7000, f7000, szSend, szReceive,CounterAlarmValue);
if( wRet>0 )
{
        // some error occur !!!
        return( wRet );
}

w7000[5]=0;     // enable alarm for counter 0
wRet=EnableCounterAlarm_7080(w7000, f7000, szSend, szReceive,CounterMaxValue);
if( wRet>0 )
{
        // some error occur !!!
        return( wRet );
}

//
//      After some processed
//
w7000[5]=0;     // disable alarm for counter 0
wRet=DisableCounterAlarm_7080(w7000, f7000, szSend, szReceive,CounterMaxValue);
if( wRet>0 )
{
        // some error occur !!!
        return( wRet );
}
```

- **Example 2:**

```
/** please refer to Sec 1.8.2 of  7080 User Manual for detailed information  about **/
/** the alarm mode 0(7080 alarm mode) and alarm mode 1(7080D alarm mode)     **/

/***** Set the module into 7080D alarm mode(mode 1) firstly *****/
double          HiAlarmValue,HiHiAlarmValue;
w7000[0]=cPort;                 // port
w7000[1]=wAddr;                 // Address
w7000[2]=0x7080;                // module ID
w7000[3]=0;                     // CheckSum disable
w7000[4]=wTimeOut;              // TimeOut constant, normal=100
w7000[5]=1;                     // set the module into 7080D alarm mode
w7000[6]=1;                     // string debug
wRet=SetModuleMode_7080(w7000, f7000, szSend, szReceive,CounterMaxValue);
if( wRet>0 )
{
        // some error occur !!!
        return( wRet );
}

HiCounterAlarmValue=100000.0;
w7000[5]=0;    // set counter 0's high alarm value
wRet=SetAlarmLimitValue_7080(w7000, f7000, szSend, szReceive,CounterAlarmValue);
if( wRet>0 )
{
        // some error occur !!!
        return( wRet );
}

HiHiCounterAlarmValue=200000.0;
w7000[5]=1;    // set counter 0's high-high alarm value
wRet=SetAlarmLimitValue_7080(w7000, f7000, szSend, szReceive,CounterAlarmValue);
if( wRet>0 )
{
        // some error occur !!!
        return( wRet );
}

w7000[5]=1;     // enable latch alarm
wRet=EnableCounterAlarm_7080D(w7000, f7000, szSend, szReceive,CounterMaxValue);
if( wRet>0 )
{
        // some error occur !!!
        return( wRet );
}

//  After some processed
wRet=DisableCounterAlarm_7080D(w7000, f7000, szSend, szReceive,CounterMaxValue);
if( wRet>0 )
{
        // some error occur !!!
        return( wRet );
}
```

4.7.5 DisableCounterAlarm_7080 [for 7080 mode only]

- **Description:**

  To disable 7080's alarm.

- **Syntax:**

  DisableCounterAlarm_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4
w7000[1] :	module address, from 0x00 to 0xFF
w7000[2] :	**module ID, 0x7080**
w7000[3] :	0=checksum disable, 1=checksum enable
w7000[4] :	TimeOut constant, normal=100
w7000[5] :	not used
w7000[6] :	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

4.7.6 EnableCounterAlarm_7080D [for 7080D mode only]

- **Description:**

  To set 7080D module enter *momentary alarm mode* or *latch alarm mode*.

- **Syntax:**

  EnableCounterAlarm_7080D(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4	
w7000[1] :	module address, from 0x00 to 0xFF	
w7000[2] :	**module ID, 0x7080**	
w7000[3] :	0=checksum disable, 1=checksum enable	
w7000[4] :	TimeOut constant, normal=100	
w7000[5] :	0	→ momentary alarm mode
	1	→ latch alarm mode
w7000[6] :	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

4.7.7 DisableCounterAlarm_7080D [for 7080D mode only]

- **Description:**

  To disable 7080D's alarm.

- **Syntax:**

  DisableCounterAlarm_7080D(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4
w7000[1] :	module address, from 0x00 to 0xFF
w7000[2] :	**module ID, 0x7080**
w7000[3] :	0=checksum disable, 1=checksum enable
w7000[4] :	TimeOut constant, normal=100
w7000[5] :	not used
w7000[6] :	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

4.7.8 ReadCounterStatus_7080

- **Description:**

  To get counter's reading/stop status.

- **Syntax:**

  ReadCounterStatus_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0] :	RS-232 port number, 1/2/3/4	
w7000[1] :	module address, from 0x00 to 0xFF	
w7000[2] :	**module ID, 0x7080**	
w7000[3] :	0=checksum disable, 1=checksum enable	
w7000[4] :	TimeOut constant, normal=100	
w7000[5] :	0	→ to read Counter 0's status
	1	→ to read Counter 1's status
w7000[6] :	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000
w7000[7]:	0	→ now is in counting status
	1	→ now is in stoping status

- **F7000: Float Input/Output Table**

  Not used

4.7.9 ClearCounter_7080

- **Description:**

  To clear the assigned counter value for a 7000 module.

- **Syntax:**

  ClearCounter_7080(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
Others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7080**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5] :	0	→ to clear Counter 0's value
	1	→ to clear Counter 1's value
w7000[6]:	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

4.7.10 ReadOutputAlarmState_7080

- **Description:**

  Reading the status of Digital output and alarm.

- **Syntax:**

  ReadOutputAlarmState_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	module address, from 0x00 to 0xFF
w7000[2]:	**module ID, 0x7080**
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	no used
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

w7000[7]: 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 Not used in Alarm mode 1

w7000[8]: 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

- **F7000: Float Input/Output Table**

Not used

4.7.11 SetInputSignalMode_7080

- **Description:**

  To set a input signal mode.

- **Syntax:**

  SetInputSignalMode_7080(WORD w7000[], float f7000[], char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
Others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7080**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5] :	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
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000	
	1 → szSendTo7000=command string send to 7000	
	→ szReceiveFrom7000=result string receive from 7000	

- **F7000: Float Input/Output Table**

  Not used

4.7.12 ReadInputSignalMode_7080

- **Description:**

  To read input signal mode.

- **Syntax:**

  ReadInputSignalMode_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
Others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7080**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5]:	not used	
w7000[6]:	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000
w7000[7] :	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

- **F7000: Float Input/Output Table**

  Not used

4.7.13 PresetCounterValue_7080

- **Description:**

  To set the counter's preset value for a 7080 module.

- **Syntax:**

  PresetCounterValue_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[],double PresetValue)

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000
PresetValue:	The counter's preset value

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7080**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5]:	0	→ to set the preset value of counter 0
	1	→ to set the preset value of counter 1
w7000[6]:	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

4.7.14 ReadPresetCounterValue_7080

- **Description:**

  To read the preset value of the assigned counter.

- **Syntax:**

  ReadPresetCounterValue_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	module address, from 0x00 to 0xFF
w7000[2]:	**module ID, 0x7080**
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	0 → to read 7080's counter 0
	1 → to read 7080's counter 1
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000
w7000[7]:	The high word of preset value
w7000[8]:	The low word of preset value

- **F7000: Float Input/Output Table**

  Not used

4.7.15 StartCounting_7080

- **Description:**

  To start/stop counting for an assigned counter.

- **Syntax:**

  StartCounting_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7080**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5]:	0	→ to activate counter 0
	1	→ to activate counter 1
w7000[6]:	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000
w7000[7]:	0	→ Stop Counting
	1	→ Start Counting

- **F7000: Float Input/Output Table**

  Not used

4.7.16 SetModuleMode_7080

- **Description:**

  To set counter alarm mode.

- **Syntax:**

  SetModuleMode_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7080**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5]:	0	→ to set into 7080 alarm mode(mode 0)
	1	→ to set into 7080D alarm mode(mode 1)
w7000[6]:	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

4.7.17 ReadModuleMode_7080

- **Description:**

  To read counter alarm mode.

- **Syntax:**

  ReadModuleMode_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7080**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5]:	0	→ now is in 7080 alarm mode(mode 0)
	1	→ now is in 7080D alarm mode(mode 1)
w7000[6]:	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

4.7.18 SetLevelVolt_7080

- **Description:**

  To set the high/low trigger level of non-isolated input.

- **Syntax:**

  SetLevelVolt_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7080**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5]:	0	→ set the low trigger level
	1	→ set the high trigger level
w7000[6]:	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

f7000[0]:	the trigger level

4.7.19 ReadLevelVolt_7080

- **Description:**

  To get the high/low trigger level of non-isolated input.

- **Syntax:**

  ReadLevelVolt_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7080**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5]:	0	→ Read the low trigger level
	1	→ Read the high trigger level
w7000[6]:	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

f7000[0]:	the trigger level

4.7.20 SetMinSignalWidth_7080

- **Description:**

  To set the minimum input signal width of high/low level.

- **Syntax:**

  SetMinSignalWidth_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[], long MinWidth)

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000
MinWidth:	minimum width value. The unit is uS and range from 2 uS to 65535 uS. For Example: when MinWidth=2000, the minimum with is 2 mS.

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7080**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5]:	0	→ set the min. width at low level
	1	→ set the min. width at high level
w7000[6]:	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

4.7.21 ReadMinSignalWidth_7080

- **Description:**

  To read the minimum input signal width of high/low level.

- **Syntax:**

  ReadMinSignalWidth_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	module address, from 0x00 to 0xFF
w7000[2]:	**module ID, 0x7080**
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	0 → read the min. width at low level
	1 → read the min. width at high level
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000
w7000[7]:	the min. width

- **F7000: Float Input/Output Table**

4.7.22 SetGateMode_7080

- **Description:**

  To set the gate control mode.

- **Syntax:**

  SetGateMode_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7080**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5]:	0	→ the gate is low active
	1	→ the gate is high active
	3	→ the gate is disable
w7000[6]:	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

4.7.23 ReadGateMode_7080

- **Description:**

  To read the gate control mode.

- **Syntax:**

  WORd ReadGateMode_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4	
w7000[1]:	module address, from 0x00 to 0xFF	
w7000[2]:	**module ID, 0x7080**	
w7000[3]:	0=checksum disable, 1=checksum enable	
w7000[4]:	TimeOut constant, normal=100	
w7000[5]:	0	→ the gate is low active
	1	→ the gate is high active
	2	→ the gate is disable
w7000[6]:	0	→ no save to szSendTo7000&szReceiveFrom7000
	1	→ szSendTo7000=command string send to 7000
		→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

4.7.24 DataToLed_7080

- **Description:**

  To setting theLED value.

- **Syntax:**

  WORD DataToLed_7080(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	module address, from 0x00 to 0xFF
w7000[2]:	**module ID, 0x7080**
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	not used
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

f7000[0]:	output data to led

4.8 Dual Watchdog Functions

4.8.1 HostIsOK

- **Description:**

  To tell all modules “Host is OK” by send this command “~**”.

- **Syntax:**

  WORD HostIsOK(WORD w7000[], float f7000[],
  char szSendTo7000 [], char szReceiveFrom7000 [])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	Not used
w7000[2]:	Not used
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	Not used
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

  Not used

4.8.2 ReadModuleResetStatus

- **Description:**

  To read the module reset status.

- **Syntax:**

  WORD ReadModuleResetStatus(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

  w7000[0]: RS-232 port number, 1/2/3/4
  w7000[1]: module address, from 0x00 to 0xFF
  w7000[2]: Module ID, 0x7011, 0x7012 ……
  w7000[3]: 0=checksum disable, 1=checksum enable
  w7000[4]: TimeOut constant, normal=100
  w7000[6]: 0 → no save to szSendTo7000&szReceiveFrom7000
  1 → szSendTo7000=command string send to 7000
  → szReceiveFrom7000=result string receive from 7000

  ------- output -----
  w7000[5]: 0: module has not been reset
  1: module has been reset

- **F7000: Float Input/Output Table**

  void

4.8.3 ToSetupHostWatchdog

- **Description:**

  To setup the module's Host Watchdog.

- **Syntax:**

  WORD ToSetupHostWatchdog(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	module address, from 0x00 to 0xFF
w7000[2]:	Not used
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	0: Disable host watchdog 1: Enable host watchdog
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000 1 → szSendTo7000=command string send to 7000 → szReceiveFrom7000=result string receive from 7000
w7000[7]:	a time interval for watchdog, unit is 0.1 second, e.x when w7000[7]=45, the time interval is 4.5 second

- **F7000: Float Input/Output Table**

4.8.4 ToReadHostWatchdog

- **Description:**

  To read the module's Host Watchdog setup values.

- **Syntax:**

  WORD ToReadHostWatchdog(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

  w7000[0]: RS-232 port number, 1/2/3/4
  w7000[1]: module address, from 0x00 to 0xFF
  w7000[2]: Not used
  w7000[3]: 0=checksum disable, 1=checksum enable
  w7000[4]: TimeOut constant, normal=100
  w7000[6]: 0 → no save to szSendTo7000&szReceiveFrom7000
  1 → szSendTo7000=command string send to 7000
  → szReceiveFrom7000=result string receive from 7000
  ---- output ----
  w7000[5]: 0: Host watchdog is disabled.
  1: Host watchdog is enabled.
  w7000[7]: a time interval for watchdog, unit is 0.1 second,
  e.x when w7000[7]=45, the time interval is 4.5 second

- **F7000: Float Input/Output Table**

4.8.5 ReadModuleHostWatchdogStatus

- **Description:**

  To read the module's Host Watchdog status.

- **Syntax:**

  WORD ReadModuleHostWatchdogStatus(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

  w7000[0]: RS-232 port number, 1/2/3/4
  w7000[1]: module address, from 0x00 to 0xFF
  w7000[2]: Not used
  w7000[3]: 0=checksum disable, 1=checksum enable
  w7000[4]: TimeOut constant, normal=100
  w7000[6]: 0 → no save to szSendTo7000&szReceiveFrom7000
  1 → szSendTo7000=command string send to 7000
  → szReceiveFrom7000=result string receive from 7000
  ---- output ----
  w7000[5]: 0: Module's Host watchdog is in NORMAL mode.
  4: Module's Host watchdog is in HOST FAILURE mode.

- **F7000: Float Input/Output Table**

4.8.6 ResetModuleHostWatchdogStatus

- **Description:**

  To reset the module's Host Watchdog status.

- **Syntax:**

  WORD ReadModuleHostWatchdogStatus(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	module address, from 0x00 to 0xFF
w7000[2]:	Not used
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	Not used
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

4.8.7 SetSafeValueForDo

- **Description:**

  To setup the safe value for DO modules.

- **Syntax:**

  WORD SetSafeValueForDo(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	module address, from 0x00 to 0xFF
w7000[2]:	module ID: 0x7050/60/67/42/43/44
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	safe value
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

4.8.8 SetPowerOnValueForDo

- **Description:**

  To setup the power on value for DO modules.

- **Syntax:**

  WORD SetPowerOnValueForDo(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	module address, from 0x00 to 0xFF
w7000[2]:	module ID: 0x7050/60/67/42/43/44
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	PowerOn value
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

4.8.9 SetSafeValueForAo

- **Description:**

  To setup the safe value for AO modules.

- **Syntax:**

  WORD SetSafeValueForAo(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	module address, from 0x00 to 0xFF
w7000[2]:	module ID: 0x7021/0x7024
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	Channel No
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

f7000[0]:	safe value

4.8.10 SetPowerOnValueForAo

- **Description:**

  To setup the power on value for AO modules.

- **Syntax:**

  WORD SetPowerOnValueForAo (WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	module address, from 0x00 to 0xFF
w7000[2]:	module ID: 0x7021/0x7024
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	Channel No
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000

- **F7000: Float Input/Output Table**

f7000[0]:	Power On value

4.8.11 SetPowerOnSafeValue

- **Description:**

  To setup the power on and safe value for modules.

- **Syntax:**

  WORD SetPowerOnSafeValue(WORD w7000[], float f7000[],
  char szSendTo7000[], char szReceiveFrom7000[])

- **Input Parameter:**

w7000:	WORD Input/Output argument table
f7000:	float Input/Output argument table
szSendTo7000:	command string send to 7000
szReceiveFrom7000:	result string read from 7000

- **Return Value:**

NoError:	OK
others:	Error code, refer to I7000.H in Sec. 2.7.

- **W7000: WORD Input/Output Table**

w7000[0]:	RS-232 port number, 1/2/3/4
w7000[1]:	module address, from 0x00 to 0xFF
w7000[2]:	module ID: 0x7011/0x7012/0x7014
w7000[3]:	0=checksum disable, 1=checksum enable
w7000[4]:	TimeOut constant, normal=100
w7000[5]:	PowerOn value
w7000[6]:	0 → no save to szSendTo7000&szReceiveFrom7000
	1 → szSendTo7000=command string send to 7000
	→ szReceiveFrom7000=result string receive from 7000
w7000[7]:	safe value

- **F7000: Float Input/Output Table**

4.9 VISUAL C++ DEMO PROGRAM

These files are too large to print in this manual, refer to floppy disk for details.

\NAP7000P\Demo\VC

Demo20 → AnalogIn demo1
Demo21 → AnalogIn demo2
Demo23 → AnalogOut demo
Demo24 → DigitalIn demo
Demo25 → DigitalOut demo
Demo26 → multi-speed demo
Demo27 → AnalogIn8 demo
Demo28 → AnalogOutReadBack demo
Demo29 → DigitalOutReadBack demo
Demo30 → NetworkAnalogIn demo
Demo31 → NetworkDigitalIn demo

…….
…….
Refer to folder that you installed.

4.10 DELPHI DEMO PROGRAM

These files are too large to print in this manual, refer to floppy disk for details.
\NAP7000P\Demo\Delphi

Start1 (for 7011/7012/7013/7014)
Getting start demo program
demo02 (for 7011/7012/7013/7014)
AnalogIn demo & performance evalution
demo20 (for 7011/7012/7013/7014)
AnalogIn demo & performance evalution
demo21 (for 7017/7018)
AnalogIn demo & performance evalution
demo23 (for 7021)
AnalogOut demo & performance evalution
demo24 (for 7041/7044/7050/7052/7053/7060)
DigitalIn demo & performance evalution
demo25 (for 7042/7043/7044/7050/7060/7067)
DigitalOut demo & performance evalution
demo27 (for 7017/7018)
AnalogIn8 demo & performance evalution
demo33 (for 7050/7060)
DigitalIn/Out demo & performance evalution
demo34 (for 7024)
AnalogOut demo & performance evalution
demo36 (for 7011/7012/7014/7015/7016/7017/7018)
demo program for send synchronise command
and read analog input value from several modules
SafeAI Demo program for Host Watchdog, safe/power on value for AI modules
SafeAO Demo program for Host Watchdog, safe/power on value for AO modules
SafeDO Demo program for Host Watchdog, safe/power on value for DO modules
Watchdog Demo program for Module Watchdog.
…….
…….
Refer to folder that you installed.

4.11 VISUAL BASIC DEMO PROGRAM

These files are too large to print in this manual, refer to floppy disk for details.

\NAP7000P\Demo\VB

Start1 (for 7011/7012/7013/7014)
Getting start demo program

demo20 (for 7011/7012/7013/7014)
AnalogIn demo & performance evalution

demo21 (for 7017/7018)
AnalogIn demo & performance evalution

demo23 (for 7021)
AnalogOut demo & performance evalution

demo24 (for 7041/7044/7050/7052/7053/7060)
DigitalIn demo & performance evalution

demo25 (for 7042/7043/7044/7050/7060/7067)
DigitalOut demo & performance evalution

demo26 (7012 and one 7021) Multi-speed demo

demo27 (for 7017/7018) AnalogIn8 demo & performance evalution

demo 30 (for 7080) reading 7080/7080D Counter/Frequency value

demo 31 7080/7080D Alarm demo

demo 32 7011/7012/7014 Alarm Demo

demo36 (for 7011/7012/7014/7015/7016/7017/7018)
demo program for send synchronise command
and read analog input value from several modules

SafeAI Demo program for Host Watchdog, safe/power on value for AI modules

SafeAO Demo program for Host Watchdog, safe/power on value for AO modules

SafeDO Demo program for Host Watchdog, safe/power on value for DO modules

Watchdog Demo program for Module Watchdog.

…….

…….

Refer to folder that you installed.

4.12 BORLAND C++ BUILDER DEMO PROGRAM

These files are too large to print in this manual, refer to floppy disk for details.

\NAP7000P\Demo\BCB

Start1 (for 7011/7012/7013/7014)
Getting start demo program
demo20 (for 7011/7012/7013/7014)
AnalogIn demo & performance evalution
demo21 (for 7017/7018)
AnalogIn demo & performance evalution
demo23 (for 7021)
AnalogOut demo & performance evalution
demo24 (for 7041/7044/7050/7052/7053/7060)
DigitalIn demo & performance evalution
demo25 (for 7042/7043/7044/7050/7060/7067)
DigitalOut demo & performance evalution
demo27 (for 7017/7018)
AnalogIn8 demo & performance evalution
demo33 (for 7050/7060)
DigitalIn/Out demo & performance evalution
demo36 (for 7011/7012/7014/7015/7016/7017/7018)
demo program for send synchronise command
and read analog input value from several modules
SafeAI Demo program for Host Watchdog, safe/power on value for AI modules
SafeAO Demo program for Host Watchdog, safe/power on value for AO modules
SafeDO Demo program for Host Watchdog, safe/power on value for DO modules
Watchdog Demo program for Module Watchdog.
…….
…….
Refer to folder that you installed.

4.13 I7000.DLL DRIVER SOURCE

The driver source of I7000.DLL are given as following:

\NAP7000P\Driver\Source\I7000.H	→ declaration file
\NAP7000P\Driver\Source\I7000.C	→ program source file
\NAP7000P\Driver\Source\I7000.DEF	→ definition file
\NAP7000P\Driver\Source\I7000.MAK	→ VC++ 4.0 make file
\NAP7000P\Driver\Source\UART.LIB	→ import library of UART.DLL

- These files are too large to print in this manual, refer to floppy disk for details
- To Use VC++ 4.0 to make this DLLs
- The I7000.DLL will call UART.DLL, therefore the UART.LIB must be put in the same directory with 7000.MAK

5 PROBLEMS REPORT

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

icpdas@ms8.hinet.net

on the Internet.

When reporting problems, please include the following information:

1) Is the problem reproducible? If so, how?
2) What kind and version of Platform that you using? For example, Windows 3.1, Windows for Workgroups, Windows NT 4.0, etc.
3) What kinds of our products that 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 that you using?
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 testing the problems that you said. And then reply as soon as possible to you. Please check that we had received your comments? And please keeps contact with us.

E-mail: **icpdas@ms8.hinet.net**
Web site: http://www.icpdas.com.tw