MiniOS7 Framework API Functions Reference Manual

(For C Language)

Version 1.0.1, March 2012

Table of Contents

Chapter 1. Introducing the MiniOS7 Framework Library

The ICPDAS MiniOS7 Framework Library provides the user with a program development structure that can be used with I-7188EX/μPAC-7186EX/I-8000E (I-8x3x), etc. controllers.

The MiniOS7 Framework development structure helps developers to quickly and efficiently write custom programs for I-7188EX/μPAC-7186EX/I-8000E.

The MiniOS7 Framework Library includes the following API functions:

1. TCP server-side functions.
2. TCP client-side functions.
3. UDP connection functions.
4. ICMP functions.
5. Email transmission functions.
6. Simple Web server functions.
7. Simple X600 File functions.
8. .ini file access functions.

The MiniOS7 Framework Library also includes example programs that using the API functions shown above. To determine which MiniOS7 Framework Library function can be used to achieve a particular purpose, check the examples.

Note: An introduction to recently added functions may not be contained in this document, so it is recommended that users check the included examples.

Chapter 2. MiniOS7 Framework Library Revision History

[V0.9.3.00]

Version 0.9.3.00 or later is supported by 7188/7188x series modules, but some functions related to tcpipl.lib will not be called in default mode.

If the user needs tcpip functions for Ethernet modules (I-7188EX/ μPAC-7186EX /8000E etc), the code shown below must be added to the end of the UserInit():

```
/* For Ver. 0.9.3.00, the following two lines must be added. */
    XS_AddSystemLoopFun(XS_SocketLoopFun);
    XS_StartSocket();
```

If these lines are not added, tcpip functions cannot be used in Version 0.9.3.00 or later.

[V0.9.3.13]
For Version 0.9.3.13 or later.
XS_UserInit() replaces UserInit().
XS_UserEnd() replaces UserEnd().

Chapter 3. Starting the MiniOS7 Framework API functions

The ICPDAS tcpipL.LIB/tcpip_dm.LIB is needed in order to link to the user's project when using the MiniOS7 Framework Library. Either 7188EL.lib, 7186EL.lib or 8000EL.lib must also be linked to the project depending on whether I-7188EX, μPAC-7186EX or I-8000E hardware is used. If the X600 file system is used, the program must link to the X600L.LIB.

The file name of the MiniOS7 Framework Library is Xsnnnnn.LIB. The latest version of the MiniOS7 Framework is XS09313.LIB, which denotes Version 0.9.313.

The following items must be known before using the MiniOS7 Framework Library (XS_nnnnn.LIB):

3.1. The user program must include the following three subroutines

Main(), UserInit(), and UserEnd() must be included within the program.

(A) Main()

Main() is the beginning function in regular C programming. XS_main() is called in the main() function of the user program.

Format:

```
void main(int argc, char *argv[ ])
{
    XS_main(argc, argv);    /* call the main function of the MiniOS7 Framework Library */
}
```

[Note] Main() is a static format used in custom programs.

(B) void UserInit(int argc, char *argv[ ])

XS_main() will call UserInit() after rebooting. Any custom initialization actions should be added here, such as calling InstallCom_x(…) before attempting to access the COM Port.
It is best to add InitLib() to the first line of UserInit() so that the program can be used for either the 40M or 80M CPU with the MiniOS7 system.

Some functions should be initiated in UserInit(), including:

1. TCP server-side functions.
2. DHCP client functions.
3. Programs that will be called once every system loop (Using the XS_AddSystemLoopFun() function).
4. Programs that will be called once in a specific timespan (Using the DT_AddTimer/DT2_AddTimer functions).
5. Command line parameters that need to be handled (Command line parameters that will passed to UserInit() without being handled by the MiniOS7 Framework Library).
6. TCP/UDP client or server structure demo should call XS_StartSocket() function to initiate the socket.

(C) void UserEnd(void)

This function will be called by the library at the end of the program. Specific resources must be related here, such as:

1. Call RestoreCom_x() here if the program uses InstallCom_x().
2. The dynamic memory allocated after calling the malloc() function must be released by calling the free() function here.

***Note**
UserInit() is replaced by XS_UserInit() and UserEnd() is replaced by XS_UserEnd() in V0.9.3.13 or later.

3.2. Ending the program

Set the ‘QuitMain’ variable to 1 (QuitMain = 1) to terminate the main loop of XS_Main() and then quit the program.

The system will call UserEnd() before ending the program.

3.3. Adding TCP Server-side functions

3.3.1. TCP_SERVER definition

TCP_SERVER

typedef struct{
unsigned port;
int socket;
int state;
int connect;
void(*CallBackFun)(int skt, int mode);
} TCP_SERVER,*pTCP_SERVER;

The following is the definition of the TCP_SERVER structure in XS.H

port	Used to listen to the port of the TCP server
socket	The socket number used for connecting to the client
state	Reserved for the MiniOS7 Framework Library
connect	The total number of TCP client connections
CallBackFun	After a connection is established, this call back function will be called once when data is sent to the server from the client-side or the client breaks the connection. skt: The socket number used for connecting to a client. Mode: mode = 0, denotes that a connection has been established. mode = 1, denotes that the client has sent data to the server. readsocket() must be called to read the data in the CallBackFun. The return value of readsocket() is 0, which means that the connection has been terminated.

If a user wants to add the TCP Server function to their program, the first thing to do is to declare a variable for the TCP_SERVER structure and set the port number and CallBackFun() in the structure variable. The last thing is to call int XS_AddServer(TCP_SERVER *server) in the program.

When a client is connecting to the server for the first time, the MiniOS7 Framework Library will call the CallBackFun() and pass the socket number used for connecting. It will then set

mode = 1 to the parameters of the function.

The MiniOS7 Framework Library will then call the CallBackFun() again once the data has arrived from the client side. The difference between this and the first connection is that mode =1.

Example:

TCP Server example 1

```
void EchoServerCallBack(int skt, int mode)
{
        if(!mode) { // mode = 0, connection is successful
        }
        else { // mode = 1, data has arrived from the client
                //…
        }
}

TCP_SERVER server10000={
        10000, // listening TCP port is 10000
        -1,
        0,
        0,
        EchoServerCallBack
};

void UserInit(int argc, char *argv[ ])
{
//…
         XS_AddServer(&server10000);
//…
         XS_AddsystemLoopFun(XS_SocketLoopFun);
         XS_StartSocket();
}
```

The maximum number of TCP servers that can be added is 16. (MAX_SERVER_NUMBER is declared in XS.h)

[Note] When a connection is established, the user can save the client socket number from

CallBackFun() to a global variable. The user can then send data to the client using this socket number.

TCP Server example 2

```
int ClientSocket = -1; // socket number >=0 means the socket is in use
void ServerCallBack(int skt, int mode)
{ int cc, err;
        if(!mode) { // mode=0, the connection has been established.
              if(ClientSocket <0) ClientSocket = skt;
        }
        else { // mode = 1, the client-side is sending data to the server
               err = cc = readsocket(skt, buf, sizeof(buf)-1);
                if(err<= 0) { /* error or connection terminated by the remote station */
                      XS_CloseSocket(skt);
               }
               else {
                      err = writesocket(skt, buf, cc);
                      if(err<0) { /* write error */
                            Print("echo write error %d\r\n", err);
                            XS_CloseSocket(skt);
                      }
              }
        }
}
void UserLoop(void)
{
        //…
        if (ClientSocket >= 0 && HasDataToBeSend) {
                  XS_WriteSocket(ClientSocket, buf, datalen);
                  HasDataToBeSend = 0;
        }
        //…
}
```

Use XS_CloseSocket() to close a connection, and the MiniOS7 Framework Library will release the relevant resources. XS_WriteSocket() can be used to send data to the client. The return value for WriteSocket() will be <0 if there is a problem with the connection

between the client and the server. XS_CloseSocket() can be called to close this connection.

XS_CloseSocket() includes a call back function:

extern void(*SocketCloseCallBackFun)(int skt);

The MiniOS7 Framework Library will do nothing with this function in default mode. If the user needs to use a callback function, user can set up to call the function. This function can be found the XSD_C03.C. This demo shows how to remove records from the callback function while closing the socket. The records are used to save the connection count.

In addition to closing the connection to the TCP Server, the function is also used to close the TCP client or UDP function, but the user must determine the type of socket.

3.4. Adding TCP client functions

3.4.1. The definition of the TCP_CLIENT structure

TCP_CLIENT

typedef struct {
char *ip; // IP address of the server
unsigned port; // server TCP port to which the client is connect
int socket; // Socket number
int tmpState; // XS kernel reserved
int bConnected; // Connection status
int iConnectTryCount; // No. of times to attempt a connection
long IConnectTimeout; // Waiting time until the connection is established
long ItmpT; // XS reserved
void(*CallBackFun)(int skt, int mode); // Call this function when receiving a response from the server
void(*LoopFun)(void);
}TCP_CLIENT,*pTCP_CLIENT;

NOTES:

ip	The IP address of the server that the client is connected to. It must be a string where the format is “xxx.xxx.xxx.xxx”.
port	The Server TCP port that the client is connected to.
socket	The socket number that is used to connect to the client 7188. Initialization value is -1.
tmpState	MiniOS7 Framework Library Reserved.
bConnected	Connection status. When bConnected is set to -2, the MiniOS7 Framework Library will automatically handle the connection to server, and will set bConnected to 0 to wait for a successful connection to the server. If a connection is successfully established, bConnected will be set to 1. If the time taken to connect exceeds the IConnectTimeout, the system will continue to try to connect to the server based on the iConnectTryCount value. If the number of a connection attempts reach the iConnectTryCount value and be connection is still unsuccessful, bConnected will set to -1. After calling XS_CloseSocket(), bConnected will also be set to -1.
iConnectTryCount	No. of attempts to establish a connection.
IConnectTimeout	Waiting time until a connection is established.
ItmpT	XS reserved.
CallBackFun	Call this function when a connection is established or when receiving a response from the server. Mode = 0 denotes that the connection was been established. Mode = 1 denotes that data is being received from the server.
LoopFun	The function will be called once for every system loop when the connection is being established (Use only when needed). The advantage of using LoopFun is that the procedure will be handled while connecting, and the procedure will not be handled when the connection is broken. If the procedure is added by XS_AddSystemLoopFun(), the procedure will be handled regardless of whether a connection has been established or has been broken.

If a user wants to connect to a TCP Server function, the first thing to do is to declare a variable for the TCP_CLIENT structure and set the IP address and port number for connecting to the server and CallBackFun in the structure variable. The last thing that should be called in their program is int XS_AddClient(TCP_CLIENT *server).

If XS_AddClient() is called in UserInit(), and bConnected is set to -2, the XS will carry out the connection to the server immediately.

If a connection is not established immediately, call XS_AddClient() in UserInit(), but set bConnected to -1. Set bConnected to -2 until a request to connect to the server is received.

XSAddClient() can be called in other functions in addition to UserInit().

When a connection to the server is established, the MiniOS7 Framework Library will call the CallBackFun() using mode = 0, and a client process mechanism that usually sends data or commands to the server will be enabled.

If there is a response from the server, the MiniOS7 Framework Library will call CallBackFun() using mode = 1.
Use readsocket() in CallBackFun() to read the response.

3.5. Adding a User Loop function

In general, a loop is required to handle a routine. Using:

int XS_AddSystemLoopFun(void(*LoopFun)(void));
to add the procedure to the system loop. The maximum number is 16.

Use int XS_RemoveSystemLoopFun(void(*LoopFun)(void));
to remove the procedure from the system loop.

Each demo program uses the UserLoop() function as it is a necessary function within the XS structure. The function name can be changed if necessary.

Refer to each demo program for more details.

3.6. Using the Timer functions

The MiniOS7 Framework Library includes two types of timer, as shown below:

```
int DT_AddTimer(long repeat, long dt, int id, void(*fun)(void));
int DT_DeleteTimer(int id, int number);
```

and

```
int DT2_AddTimer(long dt, int id, void(*fun)(void));
int DT2_DeleteTimer(int id);
```

DT2_AddTimer():
The DT2_AddTimer() function is used to call a function at regular intervals.
For instance, if you wish to switch on LED on every 500 ms(0.5 sec), then call the DT2_AddTimer() in the following manner:

```
id = DT2_AddTimer(500,1,LedToggle);
```

The function will return an ID value between 0 and 31 when DT2_AddTimer() is successfully added.
Call DT2_DeleteTimer(id) when attempting to terminate the subfunction DT2_AddTimer() called by the timer.

DT_AddTimer():
The DT_AddTimer() function is used to call function at regular intervals and then repeat the call for a set number of times.
Note that the DT_AddTimer() function with continually repeat if the repeat time is set to zero (repeat = 0).

A maximum of 32 terms are allowed for each (DT_AddTimer() / DT2_AddTimer()) function. Please check the demo program for more details.

3.7. Using the e-mail delivery (SMTP client) function

Email can be sent via the SMTP client by calling XS_SendMail() after setting the relevant variables shown below:

```
extern TCP_CLIENT SmtpClient;
```

→SmtpClient.ip should be set to the IP address of the SMTP server.

```
extern char *SMTP_MyName;
```

→The name of the module that is using the MiniOS7 Framework Library.

```
extern char *SMPT_MyEmailAddr;
```

→The e-mail address of the sender.

```
extern char *SMTP_DestEmailAddr[5];
```

→The e-mail address of the receiver, with a maximum of five addresses allowed.

```
extern int SMTP_DestEmailAddNo;
```

→The number of receiver e-mail addresses to be included and is related to SMTP_DestEmailAddr.

extern char *SMTP_Subject;

→The subject of the e-mail.

extern char *SMTP_MailData;

→The body content of the e-mail.

Please check the demo program xsd_011.C for more details.

3.8. Using the DHCP client

The DHCP client function is combined in the TCPIP.LIB and can be used by simply calling:

```
bUseDhcp = 1;
```

in the UserInit() section.

3.9. Using the HTTP (Web) Server

The HTTP server allows basic download functions that enable users to download any file. (Note: Executable files will not be downloaded unless the user knows the exact name of the file.)

Note: *Users can add CGI functionality by themselves.

3.9.1. Adding a Web Server to your program

All web server functions in the MiniOS7 Framework Library can be used after calling:

```
XS_AddServer(&XS_HttpServer);
```

in the UserInit() section.

To enable the web server function, call

```
XS_Http_StartHttpServer(void);
```

In the UserInit() section.

3.9.2. Setting the default response file

A default file can be set on the web server to respond to a connection without requiring an execution file, such as using command “http://192.168.1.20/” to build a connection. The

default response file is set in the following manner:

XS_Http_SetDefaultFile(“file name of default file”);

For example, the default file is index.htm, which then calls

XS_Http_SetDefaultFile(“index.htm”);

Note that the extension does not have to be *.htm, and can be another file extension or a CGI file.

3.9.3. Adding CGI subfunctions to custom programs

To add a CGI subfunction, call:

XS_Http_AddCgi(“CGI name”, CGI subfunction);

XS_Http_AddCgi(“DIR”,X600_Dir);

A CGI function named DIR, which is provided by the X600_Dir() function, is added.

When the command “http://192.168.1.20/DIR” is sent by the client, it calls X600_Dir() to respond.

The CGI subfunction must be in void CgiFun(int skt) format, in which skt is the socket number. The reply messages are sent from the defined skt.

3.9.4. Selecting the file system

The MiniOS7 file system is the default XS file system. All return files have to be saved in flash memory, which means that only the files displayed in the OS commands “dir” can be delivered. (Delivering files to DISK B is not yet implemented.)

If X600 series flash exists in the system, using files in X600 series flash is supported. Call XS_HttpUseX600Fs(); to switch the file system to X600.

The X600 system is now similar to MiniOS7 now, and files are saved in flash memory individually.

Uploading files to the X600 manually:

First, execute XSD_W600.EXE. After that, upload the file to the X600 using the connection

that is established with 7188E.EXE.

Uploading files to the X600 using a custom program:

(1) Add port 10000 (built-in):
    XS_AddServer(&Port10K);

(2) Add port 10000 CMD 33 to the upload file:
    pXS_Port10kCmd[33]= XS_LoadFile_33;

(3) Switch to the X600 file system:
    XS_ChangeToX600Fs();
    There is demo program in XSD_W600.X600_dir.C that includes the CGI function X600_Dir().

Chapter 4. Reading variables from the *.INI file

The MiniOS7 Framework Library provides a function for reading set data from a file, which helps the user to quickly accomplish.

Call ReadIniFile:

Syntax	int ReadIniFile(char *iniFileName, void (*CmdFun)(char *), int maxlength };
Description	The ReadIniFile function finds the file, reads the commands in the file line by line and passes the commands to CmdFun. The function has a length limitation. The command is separated by maxlength when the length exceeds one line.
Input variable	IniFileName: the name of the *.ini file CmdFun: the subfunction that executes the command maxlength: the length limitation for one line
Return value	0: NoError -1: There is not enough space in the flash memory, so memory cannot be allocated. (size = maxlength+1) (Failed to allocate maxlength+1 bytes of memory.) -2: Cannot find the .ini file. -3: CmdFun is NULL.

The MiniOS7 Framework Library provides another function for executing command in “/CMD=value” format where CMD is the command and value is the set value.

Each command must start with ‘/’ and be followed by the command name, then the equals sign followed by the set value. Lines beginning with ‘//’ are descriptions not commands.

The function is:

Void XS_SetupFun(char *cmd);

Call XS_SetupFun as the input of ReadIniFile for driving the XS_SetupFun. For example,

ReadIniFile(“xs.ini”, XS_SetupFun, 80);

XS_SetupFun() is an interface subfunction. There are two other types of subfunctions, one can be used to search for a command and the other allows a setting value to be changed. Please consult demo xs_cmd.c for details.

To set up a search command function, call:

Void XS_SetSearchCmdFun(int(*UserSearchCmdFun)(char *cmd));

The user can provide a subfunction that can be implemented to search for a command. The function was the command string as the input variable and returns the index of the command. The function returns -1 when there is no command matching the input command string. After returning the index number of the command, XS_SetupFun calls the other function to allow the setting value to be changed.

To set up the change set value function, call:

void XS_SetDoCmdFun(void(*UserDoCmd)(int idx, char *value));

The user can provide another subfunction that can be used to set the value. The function uses the command index and the set value string as input variables. The changed value does not notify XS_SetupFun().

To maintain the ability to expand the program, XS provides a function for users who code subfunctions that are used to identify neo-commands defined by themselves.

To set up the user function, call:

void XS_SetUserSetupFun(void(*UserDoCmdFun)(char *cmd));

XS_SetupFun() passes the command to the user’s function that has been set up using the function below if a command does not start with ‘/’ or the index of the command has not been found.

4.1. Demo xs_cmd.c:

Defines a new type that includes the command string and the command index.

```
typedef struct{
	char *cmd;
	int cmdId;
} XS_CMD, *pXS_CMD;
```

Define a command index with a readable name using enum.

```
enum{
_IP_=0,
_MASK_,
_GATEWAY_,
_DHCP_,
_ALIAS_,
_NAME_,
_UDP_SEARCH_,
_PASSWORD_,
_USER_,
};
```

Declare the command and the command index.

```
XS_CMD MySystemCmd[ ]={

{“ip”,_IP_},
{“mask”,_MASK_},
{“gateway”,_GATEWAY_},
{“dhcp”,_DHCP_},
{“alias”,_ALIAS_},
{“name”,_NAME_};
{“udpsearch”,_UDP_SEARCH_},
{“password”,_PASSWORD_},
{“user”,_USER_},
};
```

Here, use bsearch() provided in the C library to implement a command search subfunction. Before using bsearch(), use qsort() to call a compare function, and the contents that are compared here are strings, so ether strcmp or stricmp (case-insensitive) are available.

The function that compares the contents used by bsearch() and qsort() is shown below.

```
int XS_CMD_cmpfun(const void *xs_cmd1, const void *xs_cmd2)
{
  return stricmp(((pXS_CMD)xs_cmd1)-> cmd, ((pXS_CMD)xs_cmd2)->cmd);
  // strcmp replaces stricmp when the string is case-sensitive
}
```

The function used to sort commands is shown below:

```
void SortCmd(pXS_CMD pXsCmd, int no)
{
      qsort(pXsCmd, no, sizeof(XS_CMD), XS_CMD_cmpfun);
}
```

The function used to search for a command is shown below:

```
int MySearchCmd(char *cmd)
{
      XS_CMD XsCmd;
      pXS_CMD pXsCmd;
      XsCmd.cmd = cmd;
      pXsCmd = (pXS_CMD)bsearch(&XsCmd, MySystemCmd,
                  sizeof(MySystemCmd)/sizeof(XS_CMD), sizeof(XS_CMD),
                  XS_CMD_cmpfun);
      if (pXs_CMD) return MySystemCmd[pXsCmd-MySystemCmd],cmdId;
      else return -1;
}
```

The function used to change a set value is shown below: (In this demo, the function prints the value instead of changing the value to check whether the procedure has accessed the right command.)

```
void MyDoCmd(int cmdidx, char *value)
{
    switch(cmdidx){
        case _IP_:
            Print("IP=%s\r\n", value);
            break;
        case _MASK_:
            Print("MASK=%s\r\n", value);
            break;
        case _GATEWAY_:
            Print("GATEWAY=%s\r\n", value);
            break;
        case _DHCP_:
            Print("DHCP=%s\r\n", value);
            break;
        case _ALIAS_:
            Print("ALIAS=%s\r\n", value);
            break;
        case _NAME_:
            Print("NAME=%s\r\n", value);
            break;
        case _UDP_SEARCH_:
            Print("UDP_SEARCH=%s\r\n", value);
            break;
        case _PASSWORD_:
            Print("PASSWORD=%s\r\n", value);
            break;
        case _USER_:
            Print("USER=%s\r\n", value);
            break;
    }
}
```

Call XS_CmdInit(), which is shown in the UserInit() section below, to set the four subfunctions that are used to read variables from the *.ini file. (Here, the variables are read from 'xs.ini'.)

```
void XS_CmdInit(void)
```

```
{
   int ReadInFile(char *iniFileName, void (*CmdFun)(char *), int maxlength);
   void XS_SetupFun(char *cmd);
   SortCmd(MySystemCmd, sizeof(MySystemCmd)/sizeof(XS_CMD));
   XS_SetSearchCmdFun(MySearchCmd);
   XS_SetDoCmdFun(MyDoCmd);
   ReadIniFile("xs.ini", XS_SetupFun, 0);
}
```

Chapter 5. Using TCP Port 10000 with VCOM3

Call XS_AddServer(&Port10K);

In the UserInit() section, TCP/IP Port 10000 provided by the MiniOS7 Framework Library is ready for use, which means that the system automatically starts to listen to TCP Port 10000 and accepts incoming connections.

Port10K is of the type TCP_SERVER and is declared in XS.H as:

extern TCP_SERVER Port10K;

The commands supported by Port 10000 are declared in XS.H as:

extern int(*pXS_Port10kCmd[MAX_CMD_NUMBER])(PTCPREADDATA p);

Here, the function corresponding to the command number of Port 10000 is defined, and the first two characters of the Port 10000 command become the command number. i.e., ‘00’ is command 0, ‘01’ is command 1, etc. The number of commands is limited to 40 (00~39), and 0, 17, 18, 33 are already occupied by the MiniOS7 Framework Library.

5.1.1. Using the PASSWORD command

Command 0 is used to set up a password. To add command 0 to a program, call:

```
pXS_Port10kCmd[0] = XS_SetPassword_00; // built-in function
```

Other command numbers can also be set as command 0:

```
pXS_Port10kCmd[30] = XS_SetPassword_00;
```

This sets command 30 as the command for setting up the password. The user can define any command they need in their own way.

XS_SetPassword_00 is a built-in function which allows the user to set up a password. There are other settings available when using the password command, as shown below:

(1) bNeedPassword = 1;

This enables the password checking function so that a password must be entered when connecting to Port 10000. After checking that the password is correct, the user can continue to other processes. If the password is incorrect, access will be denied.

(2) pXS_CheckPassword = XS_CheckPassword;
This is the password checking function. XS_CheckPassword is a built-in function that allows the user to use their own password checking function, but it should correspond with set password function, which means that the user should use their own set password function instead of the built-in function (XS_SetPassword_00) when they use their own password checking function.

(3) PasswordMsg = “Need Password:”;
A message is sent when the connection is established on Port 10000 depending on the user.

Based on the above, use password with MiniOS7 Framework LIBRARY should do as below:

```
bNeedPassword = 1; // enables the password function.
pXS_CheckPassword = XS_CheckPassword; // built-in function
pXS_Port10kCmd[0] = XS_SetPassword_00; // built-in function
PasswordMsg = “Need Password:”;
```

5.1.2. Using the INPORT command

Command 17 is the import command in VCOM3 and can input hardware I/O data. There are two modes: 8-bit mode and 16-bit mode. To add command 17 to a program, call:

```
pXS_Port10kCmd[17] = XS_Inport_17; // built-in function
```

As with command 0, other command numbers can be set as command 17.

Format of command 17:
17pppp: 8-bit mode, pppp is the I/O address, there are four characters.
17ppppW: 16-bit mode, pppp is the I/O address, there are four characters and ‘W’ can also be lowercase.

5.1.3. Using the OUTPORT command

Command 18 is the output command in VCOM3 and can output hardware I/O data. There are two modes: 8-bit mode and 16-bit mode. To add command 18 to a program, call:

pXS_Port10kCmd[18] = XS_Outport_18; // built-in function

As with command 0, other command numbers can be set as command 18.

Format of command 18:

18ppppHH: 8-bit mode, LIB calls outp(pppp, HH);

18ppppHHHH: 16-bit mode, LIB calls outpw(pppp, HHHH);

pppp is the I/O address, there are four characters. HH (two characters) is 8-bit mode, and HHHH (four characters) is 16-bit mode.

5.1.4. Using the DOWNLOAD command

Downloading files is the main function for using port 10000 in VCOM3, and makes updating via TCP/IP more convenient.

Command 33 is used to download files in VCOM3 and is used to download files to modules that use the MiniOS7 Framework Library in coordination with 7188.exe, as shown below:

pXS_Port10kCmd[33] = XS_LoadFile_33; // built-in function

Other command numbers can be set as the download function, but command 33 must be used when downloading files using 7188e.exe. It is blocked in VCOM3.

5.1.5. Downloading a file using 7188E.EXE

After establishing a connection with the server, the user can begin to download the files.

Common command of CMD33:
“33DIR”: The system displays the contents of DISK A.
“33DIR B”: The system displays the contents of DISK B. (There is no DISK B in the x600 file system)
“33DEL A”: Deletes all the files on DISK A.
“33DEL B”: Deletes all the files on DISK B.
“33DEL file name”: Deletes a file from the x600 file system.
Downloading files using the auto-download function in 7188e.exe:

Downloading a single file –

Press F2 to set the file name and then press F9 to download the file to DISK A. Press ALT+F9 to download the file to DISK B.

Downloading multiple files –

Press ALT+F2 to set the file name. The maximum number of files is 10. Then press F10 to download all the files to DISK A individually. Press ALT+F10 to download all the files to DISK B.

Note: There is no DISK B in the X600 file system, so none of the functions using DISK B will work.

Chapter 6. Using the UDP/IP

6.1. UDP_SOCKET structure

The UDP_SOCKET structure is declared in XS.H

UDP_SOCKET
typedef struct{ int socket; unsigned MyPort; char *RemoteIp; unsigned RemotePort; int fNonBlock; int fEnabledBroadcast; int status; void (*CallBackFun)(int skt); } UDP_SOCKET, *pUDP_SOCKET;

The following describes the structure variables of UDP_SOCKET.

socket: This is the socket identifier. Set the initial value of the socket to -1 in order to indicate that this UDP_SOCKET has not yet obtained a socket identifier. This socket variable will be set to -1 when the XS_AddUdpSocket() function is called.

MyPort: This variable is used to specify the port number for a connection. The port number is the port that the server will be listening on.

RemoteIp: This variable is used to specify the IP address of the remote Server. The IP address string will be “xxx.xxx.xxx.xxx”. (Note that the IP address cannot be retrieved using DNS in this version.)

RemotePort: This variable is used to specify the port number to which the Service binds.

Note 1: If you want to implement an UDP client application that can establish a connection with a remote server, it is necessary to specify the RemoteIp and RemotePort variables.

Note 2: If you want to implement an UDP server application, the RemoteIp variable must be set to NULL and the MyPort variable must be set to 0.

fNonBlock: 1: Sets the connection for non-blocking operations.
0: Sets the connection for blocking operations.

fEnableBroadcast: When this variable is set to 1, this UDP socket will be allowed to transmit UDP broadcast packets.

status: This variable is used by the system only and is set to 1 during a normal period of operation. If an error occurs, it will be a negative value.

CallBackFun: This function pointer is used to specify a callback function for receiving data via UDP. The system will automatically call the specified function when data arrives. In the specified callback function, both the recvfrom() and recv()/readsocket() functions can be used to receive messages from the connection.

6.2. Writing an UDP client program

1. Declare the structure of an UDP_Socket type.

2. Call the XS_AddUdpSocket() function.

3. After successfully creating a socket, call the XS_WriteSocket() function to send a message to the destination server.

4. The system will automatically call the specified callback function when data arrives.

6.3. Writing an UDP server program

1. Declare the structure of an UDP_Socket type.

2. Call the XS_AddUdpSocket() function.

3. The system will automatically call the specified callback function if data arrives from an

UDP client.

6.4. Using an UDP socket to establish multiple UDP connection

TCP is a connection-oriented protocol, which means that upon receiving a communication request it requires a handshaking process to set up an end-to-end connection. According to the end-to-end principle, TCP only allows packets to be transmitted between a server and the client side. Unlike TCP, UDP can send packets to any destination client in the LAN (even if the destination client does not have a port number specified for listening on). Thus, one UDP socket can communicate with multiple clients. The recvfrom() function can be used to receive UDP packets. The sendto() function can be used to send packets to the specified destination. UDP is suitable for packet broadcast (sending to all destinations on a local device network) as it can transmit a packet that will be received by every device on a local network.

An UDP server is particularly suitable for request-response protocols such as the Modbus protocol. After using the recvfrom() function to receive a request from a device and processing the request, the sendto() function is used to return a message to the specified device in response.

When the UDP server sends messages to a destination client on its own initiative, it is necessary to know the destination address of the client that will receive the UDP messages.

Chapter 7. Using the COM Ports

7.1. COMPORT structure

The COM Port structure is declared in XS.H

struct ComPort

```
typedef struct ComPort COMPORT, *pCOMPORT;

struct ComPort{
   int Port; //0: COM0, 1: COM1, … etc.
   char *Buf;
   unsigned Bufsize;
   char EndChar;
   unsigned TriggerLevel;
   long Timeout;
   long StartTimeTicks;
   long LastTimeTicks;
   long MasterTimeout;
   long MasterStartTimeTicks;
   unsigned Size;
   int (*ReadComn)(unsigned char *buf, int n);
   void (*CallbackFun)(pCOMPORT, int mode);
   pCOMPORT next;
}:
```

Declaring a variable of ComPort type:

```
struct ComPort com;
```

Declaring a pointer of ComPort type:

```
Struct ComPort *com;
or
pCOMPORT com;
or
COMPORT *com;
```

The following describes the structure variables of ComPort.

Port: This variable is used to specify a COM Port number. Port numbers range between 1 and 8 for both I-7188E and μPAC-7186EX series modules.

Buf: This variable is used to point to a buffer that the function can read the bytes into. All data that is received from a COM Port will be temporarily stored in this data buffer. The specified callback function will be called while a set-up condition exists.

Bufsize: This variable is used to set the size of the buffer. The callback function will be called when the data buffer is full.

EndChar: This variable is used to specify a terminating character. The callback function will be called when the end character of a received string is same as the specified terminating character. A NULL character indicates that this function is disabled.

TriggerLevel: This variable is used to specify the trigger level of the data buffer. The callback function will be called when the amount of data in the buffer reaches this TriggerLevel. This variable is particularly suitable for receiving data of a fixed length.

Timeout: This variable is used to specify a period of time that is allowed to elapse since the COM Port last received data. The callback function will be called when this timeout expires.

StartTimeTicks: This variable is only used by the system. It is used to record the time ticks since the first byte that is currently in the buffer was received.

LastTimeTicks: This variable is only used by the system. It is used to record the time ticks since the last byte that is currently in the buffer was received.

MasterTimeout: This variable is used to specify a period of time that will be allowed to elapse before receiving a response from a slave device. This variable is only used by a Master device. This function is disabled when the MasterTimeout variable is set to 0.

MasterStartTimeTicks: After sending a command to a slave device, this variable must be set to the current time ticks value. This function is disabled when the MasterStartTimeTicks variable is set to 0.

Size: This variable is used to indicate the current number of bytes that are stored in the data buffer. After accomplishing the callback function, this variable must be set to 0.

ReadComn: A function pointer. The system will call this function to retrieve the data that was temporarily stored in the input buffer of the COM Port with each loop.

CallbackFun: This function pointer is used to specify a callback function for processing the data that is received from the COM Port.

When any set-up condition exists, the specified callback function will be called. The mode parameter indicates which condition existed.

mode = 0: Timeout expired

mode = 1: EndChar

mode = 2: TriggerLevel

mode = 3: The data buffer was full and the Size variable should be equal to the Bufsize variable

mode = 4: Master Timeout expired

Next: This variable is used by the system only, and it is used to link to the next COM Port that will be handled.

7.2. When will the callback function be called by the system?

The Bufsize, EndChar, TriggerLevel and Timeout variables will affect when the callback function is called. These four conditions exist only when data arrives. When any one of the conditions exists, the specified callback function will be called instantly. In other words, the specified callback function might not be called if no data was received.

The Bufsize variable is used to specify the size of the data buffer. This variable cannot be set to 0, or no data will be able to be stored in the data buffer. In other words, the system will be unable to receive any data using the COM Port. If the buffer is full, it will be necessary to instantly receive the data that is stored in the buffer. Otherwise the buffer will no longer store any data.

The EndChar variable can be set to 0 if the user wishes to disable this function. The function is not suitable for receiving a string where the terminating character is NULL, but is particularly suitable for applications where the response, such the CR character is used as the terminating character as using an I-7188E series module to connect with I-7000 series modules based on the DCON protocol. In this case, the EndChar variable can be set to “0x0d”.

TriggerLevel is suitable for applications that send a fixed length command or receive a fixed length response. This function is disabled when the TriggerLevel variable is set to 0.

If the Timeout variable is set to 0, this function is disabled, although it is not recommended to disable this function. The function may be used in conjunction with the TriggerLevel and EndChar functions. Both TriggerLevel and EndChar are chiefly used to assist the Timeout variable in checking for data loss. TriggerLevel will break down if data is lost and an inappropriate Timeout could cause a program to be blocked. The same situation could occur with the EndChar setting and the program could be blocked if the last character which matches the EndChar is lost. Setting an appropriate Timeout value is the only way to check whether data has been completely received if the data has no constant length or end character. Data is sent in a series, so by setting the Timeout to just a little bit longer than the calculated sending period, means that, basically, the MiniOS7 Framework Library can check whether an individual command has been completely received, although this is not absolutely correct. The crux of the matter is how long should the timeout be? If it is set too long, more than one set of data may be received at one time. If it is set too short, a single

set of data may be split into pieces. Both of these situations make the process of reading a command more complicated. There is another variable that is also related to the Timeout – the FIFO trigger level in the UART. The FIFO trigger level is a variable that indicates the waiting time for any data stored in the FIFO. If the value of their variable is longer than the Timeout value, then the receiving timeout might expire while data is still waiting for the FIFO trigger level in the UART FIFO.

The MasterTimeout and MasterStartTimeTicks variables are used for Master-slave mode operations. Set the MasterTimeout variable to a non-zero value in order to enable Master-slave mode. After a command is sent from a master to a destination client, the MasterStartTimeTicks must be set to the system timeticks. Thus, the MiniOS7 Framework Library can check whether the MasterTimeout has expired. Notice that the MasterStartTimeTicks variable must be set to 0 when the sending of commands to Slave has stopped. This prevents the CallBack function from being triggered when the master timeout has elapsed after the sending of commands has stopoped. (The MiniOS7 Framework Library will set the MasterTimeout to zero when a master timeout occurs, but does not change the MasterTimeout for a normal response.)

Chapter 8. Using the ICMP/IP

8.1. ICMP/IP structure

The ICMP_SOCKET structure is declared in XS.H

ICMP_SOCKET
typedef struct{ int socket; char *RemoteIp; int status; void (*CallBackFun)(int skt); } ICMP_SOCKET, *pICMP_SOCKET;

The following describes the structure variables of ICMP_SOCKET.

socket: Socket identifier. Set the initial value of the socket to -1 in order to indicate that this UDP_SOCKET has not yet obtained a socket identifier.

RemoteIp: This variable is used to specify the IP address of the destination.

status: This variable is used by the system only.

CallBackFun: This function pointer is used to specify a callback function for receiving data. The system will automatically call the function when data arrives. In the callback function, the XS_ReadSocket() function can be used to receive messages from the connection.

8.2. ICMP functions

8.2.1. int XS_AddIcmpSocket(ICMP_SOCKET *IcmpSocket);

In order to add an ICMP function to an application, it is necessary to specify the RemoteIp and CallBackFun variables before calling the XS_AddIcmpSocket() function. The system will

provide a socket for an ICMP connection after successfully calling the XS_AddIcmpSocket() function. If the IcmpSocket->socket variable is not a negative value when evaluated in the LoopFun() function, this indicates that this socket can be used to transmit ICMP packets.

8.2.2. int XS_RemoveIcmpSocket (ICMP_SOCKET *IcmpSocket);

This function is used to remove the specified ICMP socket.

8.3. Implementing a ping function (ICMP echo)

Sample code: ping.c
To enable ping function, the demo code should include XS_Ping.c file.

int XS_AddPing_fun(char *IpStr,unsigned id,unsigned startSeq,int PingLength,

unsigned PingCnt,long PingPeriod,void (*fun)(int skt,int mode,int id,char *data, int length))

IpStr: The server IP.

id: The Id.

startSeq: The start sequence.

PingLength: The ping length.

PingCnt: The ping count.

PingPeriod: The ping period.

(*fun)(int skt,int mode,int id,char *data,int length): The callback function.

skt: The socket.

mode: -2: ping to server failed. -1: ping executing. 1: ping to server success.

id: The Id.

*data: Received data.

length: Received data length.

Chapter 9. XS Functions

9.1. Necessary functions in user program

9.1.1. void main(int argc, char *argv[ ]);

Please refer to the demo programs for the method of calling this function.

9.1.2. void UserInit(int argc, char * argv[ ]);

The XS_main() function is called at the start stage of the XS program. All initialization code can be added in the XS_main() function, such as retrieving the configuration settings that are stored in the EEPROM or perhaps calling the InstallCom() function to initialize the configuration settings of the COM Port.

9.1.3. void UserEnd(void);

The UserEnd() function will be called before performing the XS_main() function. This function is typically used to free resources allocated or to back up the settings if the program is going to be closed, for example storing the configuration settings in the EEPROM or perhaps calling the RestoreCom() function.

9.2. Basic Functions

9.2.1. void XS_GetVersion(char *ver);

This function is used to obtain the version number of the library.

ver: This argument is used to store the address of the string that indicates the version number, and the length of the string must be more than 8 characters.

9.2.2. void XS_GetLibData(char *date);

This function is used to obtain the version date of the library.

date: This argument is used to store the address of the string that indicates the version date, and the length of the string must be more than 8 character.

9.2.3. extern char SocketType[32];

The SocketType[ ] can be used to indicate the current status of each socket.

0: This socket is not in the mode that is supported by the library.
1: A server socket for a connection.
2: A client socket for a connection.
3: The socket is listening for a connection.
4: An UDP socket.
5: An ICMP socket.

```
void XS_Install_DHCP_Client(void);
/*
  for use as a DHCP client function:
  1.  call the “Install_DHCP_Client();” function
*/

int XS_AddServer(TCP_SERVER *server);
int XS_AddClient(TCP_CLIENT *client);

/* for a SMTP(send E-mail) */
extern TCP_CLIENT SmtpClient;
extern char *SMTP_MyName;
extern char *SMTP_MyEmailAddr;
extern char *SMTP_DestEmailAddr[5];
extern int SMTP_DestEmailAddrNo;
extern char *SMTP_Subject;
extern char *SMTP_MailData;
int XS_SendMail(void);

/* for a timer function */
int DT_AddTimer(long repeat, long dt, int id, void (*fun)(void));
int DT_DeleteTimer(int id, int number);

int DT2_AddTimer(long dt, int id,void (*fun)(void));
int DT2_DeleteTimer(int id);

/* for a system loop function */
int XS_AddSystemLoopFun(void (*LoopFun)(void));
int XS_RemoveSystemLoopFun(void (*LoopFun)(void));

extern char FILE_NOT_FOUND[ ];
extern char FILE_IS_TEXT[ ];
extern char FILE_IS_TEXT_NoLength[ ];
extern char FILE_IS_IMAGE[ ];
extern char FILE_IS_APPLICATION[ ];
extern char FILE_UPLOAD[ ];
extern char FILE_UPLOAD_END[ ];
```

```
extern int SizeOfFileNotFound;

extern char *ImageFileType[ ];
extern char *ApplicationFileType[ ];

extern TCP_SERVER_XS_HttpServer;
extern METHOD {_GET= 1, _POST};

int XS_WriteSocket (int skt, char *buf, int len);
void XS_Http_SetDefaultFile(char *fname);
int XS_Http_AddCgi (char *name, void (*cgifun)(int skt));

extern void(*SocketCloseCallBackFun)(int);

void XS_main(int argc, int *argv[ ]);

int XS_CloseSocket(int socket);

extern int bNeedPassword;
extern int iPasswordBlock; /* default value is 6 */
extern int iPasswordAddr; /* default value is 250 */
extern int (*pXS_CheckPassword)(PTCPREADDATA p);
int XS_CheckPassword(PTCPREADDATA p);
void XS_SetPassword_00(PTCPREADDATA p);

int TcpFlush(void);
int TcpPuts(int skt, int mode, char *str);
int TcpPrint(int skt, int mode, char *fmt, …);

void ResetCrc16(void);
unsigned ReadCrc16(void);
void UpdataCrc16(unsigned char data);
void MakeCRC16Table(void);

/* command 17 */
int XS_Inport_17(PTCPREADDATA p);

/* command 18 */
```

```
int XS_Output_18(PTCPREADDATA p);

/* command 33 */
void XS_LoadFile_33(PTCPREADDATA p);

#ifndef_PORT10K_CMD33_
	#define_PORT10K_CMD33_
	typedef struct{
		char *cmd;
		int (*fun)(int);
	}CMD33;
#endif
extern CMD33 Cmd_33[ ]; /* these are only 5 items in Cmd_33[ ] */
extern int (*Cmd33_WriteFileHeader)(FILE_DATA *fdata);
extern int (*Cmd33_WriteFileData)(char *data, int no);
extern int (*Cmd33_FinishFileHeader)(void);

void XS_ChangeToX600Fs(void);
void XS_ChangeToOS7Fs(void);
char *X600Fs_GetTmpBuf(void);

void XS_HttpUseOs7Fs(void);
void XS_HttpUseX600Fs(void);

/*
  For an UDP socket
*/
typedef struct{
	int socket;
	unsigned MyPort;
	char *RemoteIP;
	unsigned RemotePort;
	int fNonBlock;
	int fEnableBroadcast;
	int status;
	void (*CallbackFun)(int skt);
} UDP_SOCKET, *pUDP_SOCKET;
extern pUDP_SOCKET pUdpSocket[MAX_UDP_NUMBER];
```

```
int XS_AddUdpSocket(UDP_SOCKET *UdpSocket);

int XS_RemoveUdpSocket(UDP_SOCKET *UdpSocket);

/*
For an UDP search function (The same as VCOM320 or later version.)
*/
char *GetModuleName(void);    /*User's program must support this function*/
char *GetAliasName(void);       /*User's program must support this function*/
extern UDP_SOCKET XS_UdpSearch;

typedef struct ComPort COMPORT, *pCOMPORT;

struct ComPort{
  int Port;    //0: COM0, 1: COM1, …etc.
  char *Buf;
  unsigned Bufsize;
  char EndChar;
  unsigned TriggerLevel;
  long Timeout;
  long StartTimeTicks;
  long LastTimeTicks;
  long MasterTimeout;
  long MasterStartTimeTicks;
  unsigned Size;
  int (*ReadComn)(unsigned char *buf, int n);
  void (*CallbackFun)(pCOMPORT, int mode);
  pCOMPORT next;
};
#endif

void XS_AddComPort(pCOMPORT pcomport);
void XS_RemoveComPort(pCOMPORT pcomport);

/* for an ICMP socket /*
typedef struct{
      int socket;
      char *RemoteIP;
```

```
    int status;
    void(*CallBackFun)(int skt);
}ICMP_SOCKET,*plCMP_SOCKET;

extern pICMP_SOCKET plcmpSocket[MAX_ICMP_NUMBER];

int XS_AddlcmpSocket(ICMP_SOCKET *IcmpSocket);
int XS_RemoveIcmpSocket(ICMP_SOCKET *IcmpSocket);
```

Chapter 10. DEMO statement

Connect a 7188E_UDP OS(UDP port 23) with an UDP
This is a demo program for an UDP client.

Use an UDP search
Demo: XSD_001.c 1. Add an UDP search function.

Use TCP Port 10000
Demo: XSD_001.c 1. Use Port 10000 supplied by the MiniOS7 Framework library. 2. Add a load file function.(command 33) 3. Use the password function.(command 00) 4. Use command 19/23. 5. Add an inp/inpw/outp/outpw function.(command 17/18)

Use a COM Port
Demo: xsd_com2.c

Use ping
Demo: xsd_ping.c

Use xs.ini setting
Demo: xsd_com2.c