How to implement Serial to Ethernet for W7100A

version 1.1

For more information, visit our website at http://www.wiznet.co.kr

Table of Contents

1 Introduction

RS-232 has been used as the connector for DTE ((Data Terminal Equipment) and DCE (Data Circuit-terminating Equipment) in the telecommunication field for a long time. Since PC came out RS-232 has been the standard for serial communication. Serial communication is being widely used even until now.

Since the wide spread of internet now, TCP/IP has been the most used internet protocol. Due to the recent growing internet environment, the need of communication between internet and other communication devices with an interface has increased. It is the same case with internet and serial communication devices. If Ethernet and RS-232 is connected, serial devices can be controlled through the internet, and use the internet service. Also, data collected from various serial sensors can be received via the internet.

In this document, we will use iMCU7100EVB to implement a 'serial to Ethernet converter' and describe some simple applied programs.

2 Basic Structure of Serial to Ethernet

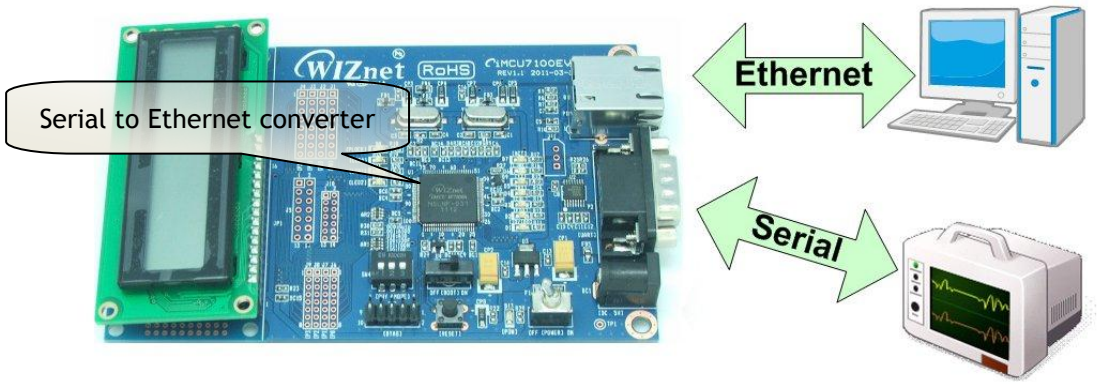


Fig 오류! 지정한 스타일은 사용되지 않습니다..1 Serial to Ethernet converter using W7100

The iMCU7100EVB can implement a 'Serial to Ethernet converter' without an extra device because it is contains a RJ45 connector, RS-232 connector, and a W7100 chip. In this document we will use the 'serial to Ethernet function' to send/receive messages between the serial device and Ethernet device. First examine in the Demonstration section and look over related codes in the Code section.

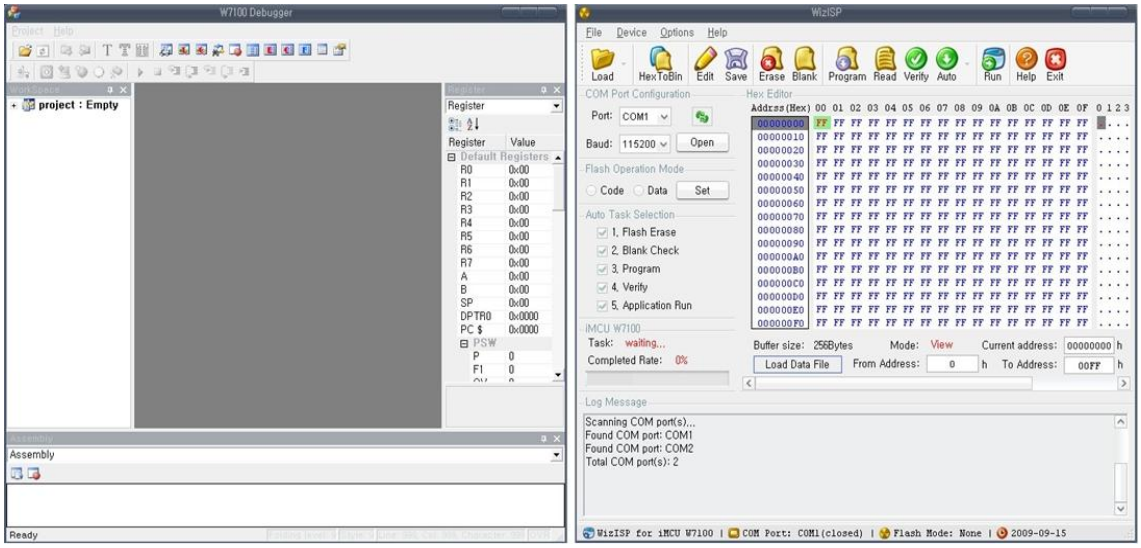
Please refer to document 'How to use UART in W7100' for related information on Serial. Also refer to document 'How to implement TCP for W7100' or 'How to implement UDP for W7100' for related information on Ethernet.

3 Serial to Ethernet Demonstration

In this section, we will try a demonstration of 'Serial to Ethernet' by using the TCP server. The iMCU7100EVB works as a TCP server with a serial interface. The device terminal program will be used for the 'Serial to Ethernet' demonstration; this program has both the Serial interface and the Ethernet interface and both of them can be tested simultaneously. The device terminal can be downloaded from WIZnet's homepage, Library => Download Center. First, connect iMCU7100EVB with the PC using a LAN

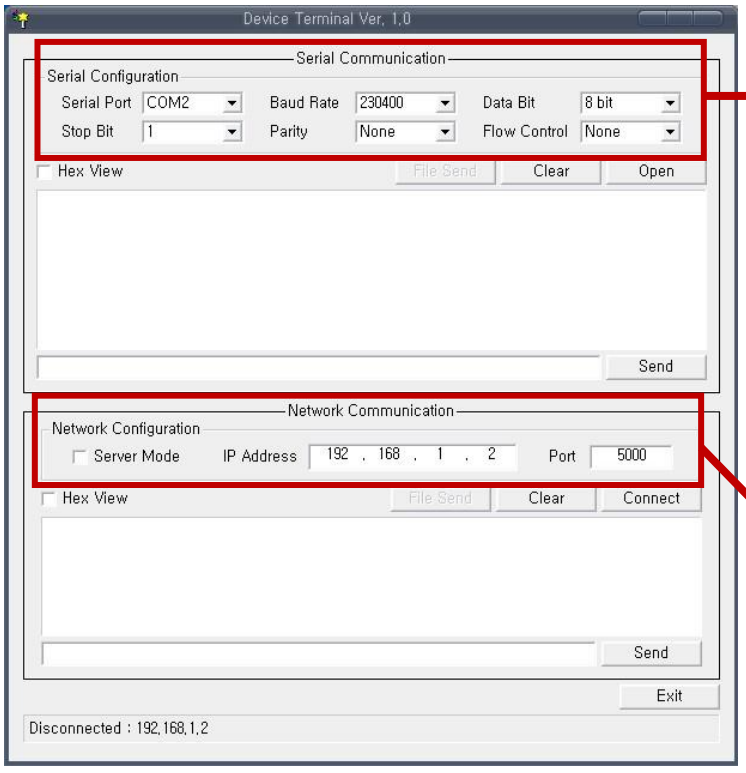
cable and serial cable. The PC will operate as both Ethernet device and serial device. For more details, refer to 'iMCU7100EVB User's Guide.'

After connecting iMCU7100EVB with the PC, turn on the board. Load the attached 'serial to Ethernet HEX file' in the board by using either 'W7100 debugger program' or 'WizSP program.' Please refer to 'W7100 Debugger Guide' for directions for iMCU7100 debugger program, and 'WizISP Program User Guide for W7100' for directions for WizSP program. Fig 3.1 is the captured screen of iMCU7100 debugger program and WizISP program.



(a) W7100 Debugger (b) WizISP

Fig **오류! 지정한 스타일은 사용되지 않습니다.**.2 W7100 Debugger program and WizISP program



Serial Configuration	
Serial Port	COM2
Baud Rate	230400
Data Bit	8 bit
Stop Bit	1
Parity	None
Flow Control	None
Network Configuration	
Server Mode	Unchecked
IP Address	192.168.1.2
Port	5000

Fig 오류! 지정한 스타일은 사용되지 않습니다..3 Device terminal setting

After loading the HEX file on the iMCU7100EVB board, reset the board and open the device terminal program. Fig 3.3 is the captured screen of the device terminal and values for configuration.

From the Serial configuration, set the value of Serial Port to COM2. However, this value differs depending on user's PC setting. The Baud Rate should be set to 230400 because the 'iMCU7100EVB UART Baud Rate' from the attached HEX file is set to 230400. If the user modified the code and Baud Rate, use the modified Baud Rate. Please refer to 'How to use UART in W7100' for more details on Baud Rate.

The Network configuration is as followed. Leave the Server Mode as Unchecked status. Check this option when iMCU7100EVB is used as TCP client and the PC is used as TCP server. The IP address is set to 192.168.1.2 since the IP of iMCU7100EVB is set to that address. The IP can be modified by changing the code, and use the modified IP for the device terminal program.

After all configurations for the 'device terminal program' is done, connect with the board by clicking OPEN from the serial communication window and CONNECT from the network communication window. If the connection is successful, type a message in both the serial and network message window, and click SEND or press ENTER to check whether the 'serial to Ethernet' chatting works. Fig 2.4 shows this process. Type a random message in the Serial message window, and click SEND or press ENTER. Then the message will appear in the Ethernet conversation window. The same thing will happen if the same process is done in the serial conversation window.

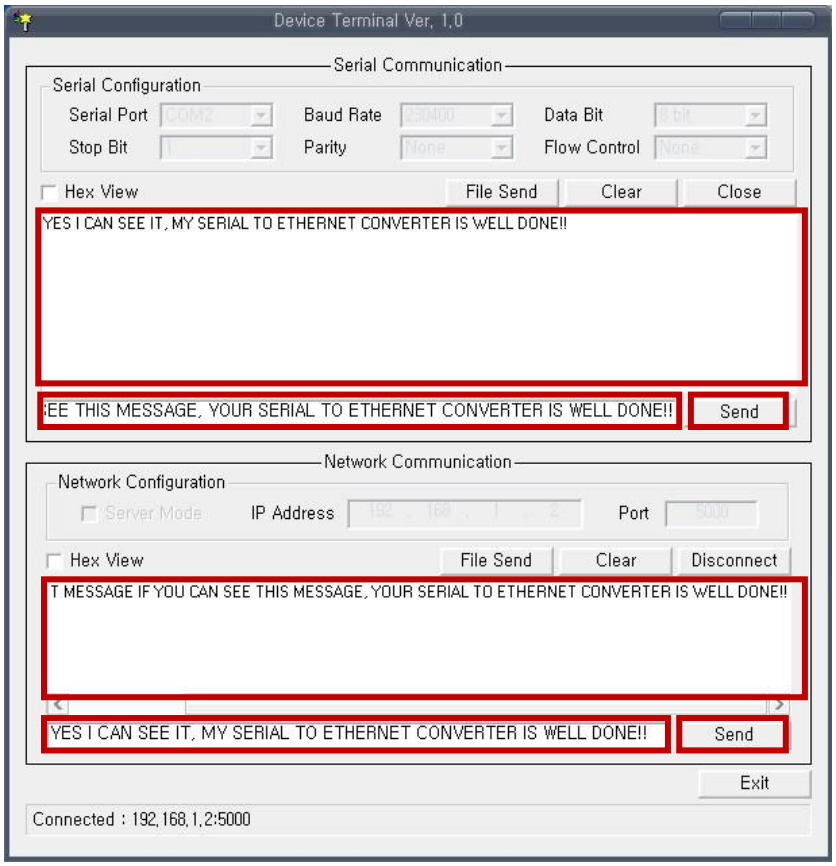


Fig 오류! 지정한 스타일은 사용되지 않습니다..4 Serial to Ethernet chat test using device terminal

Random files can be sent using the device terminal; click 'FILE SEND' and the 'file open window' will

appear. Sent files are saved in the folder where the device terminal is saved. But, when a file is sent through the Ethernet, the content of the file will appear on the serial conversation window. Since the transmission speed of TCP is faster than the transmission speed of serial, file transmission speed depends on the serial Baud Rate. If a file is sent using UDP, the data can't be trusted and therefore can be damaged.

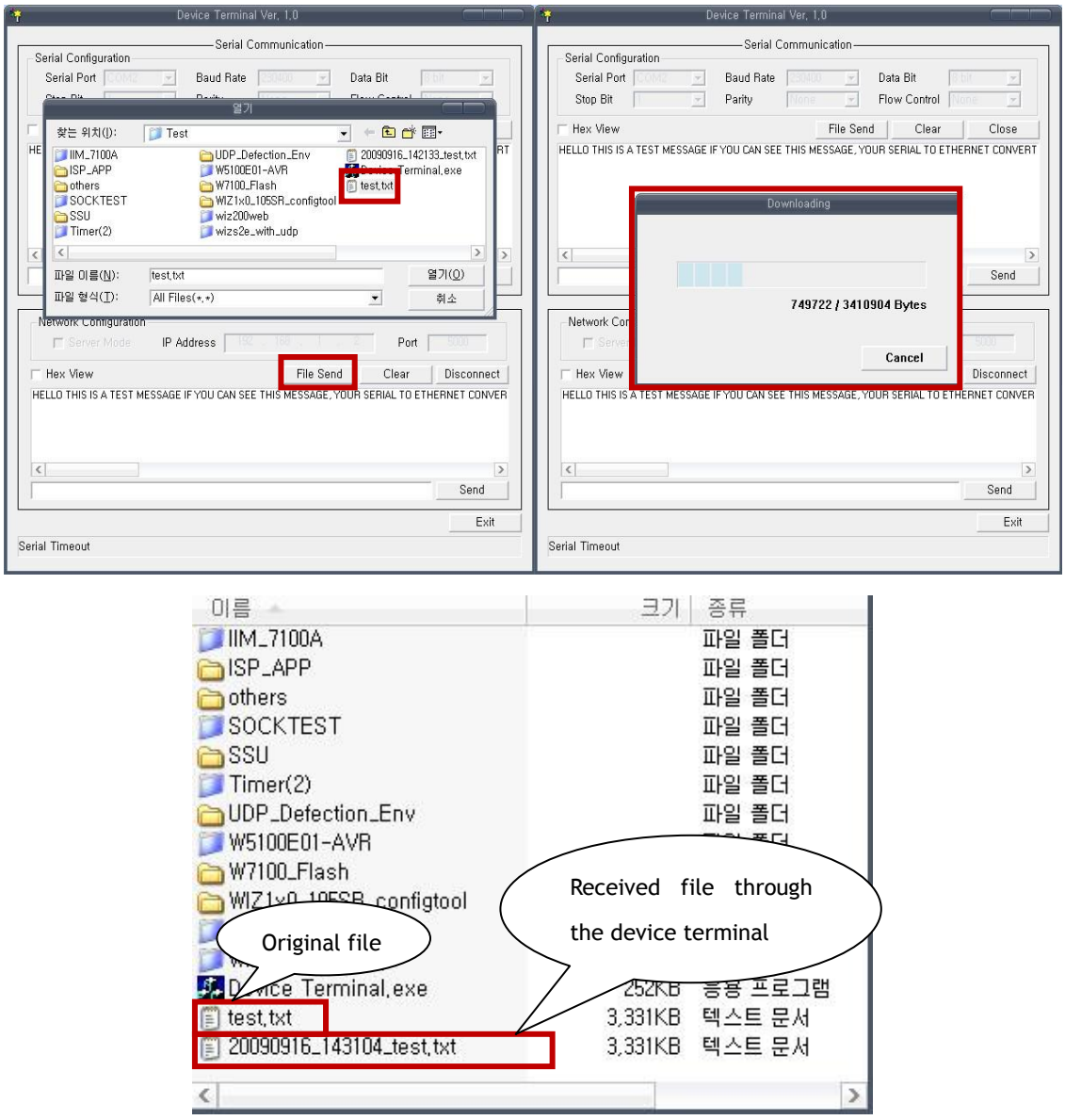


Fig 오류! 지정한 스타일은 사용되지 않습니다..5 Serial to Ethernet file sending using device terminal

4 Serial to Ethernet Code

In this section, we will go over the loaded code for 'Serial to Ethernet'. The codes for 'Serial to Ethernet' are implemented in 3 ways: TCP server, TCP client, and UDP. Each codes will be described over the next sections. For more details on TCP and UDP, please refer to 'How to implement TCP for W7100',

'How to implement UDP for W7100', and 'W7100 datasheet.'

4.1 Using TCP server

The LISTEN status in TCP server is where the server waits for the client's connection request. The client will request connection to the TCP in CONNECT status. If the server and client are connected, the SOCKET goes into ESTABLISHED status (SOCK_ESTABLISHED; defined by WIZnet). From this state, data can be sent/received until the SOCKET is closed. The life cycle of SOCKET in TCP server mode is composed of OPEN, LISTEN, SEND, RECEIVE, DISCONNECT, CLOSE.

The codes for each status are explained in the next few sections. The functions, socket(), listen(), recv(), send(), disconnect(), close(), are supported from WIZnet. For more details, please refer to 'W7100 TCPIPCore Driver Guide.'

4.1.1 OPEN

```
s = 0;          // set SOCKET 0 (From 0 to 7)
socket(s, Sn_MR_TCP, port, mode);     // OPEN SOCKET 0
while(getSn_SR(s) != SOCK_INIT);
```

Code **오류! 지정한 스타일은 사용되지 않습니다.**.1 Open SOCKET

OPEN status is where the SOCKET is created, and socket() function is used to create the SOCKET. SOCKET number, protocol, port number, and flag are input factors that need to entered in for socket() function; SOCKET number are between 0-7, for protocol, enter Sn_MR_TCP(0x01), for port number enter the port that is going to be used for communication with client, and the flag set to 0 such as the 'No Delayed Ack flag,'

After all the settings, call the socket () function. Check whether the Sn-SR() register, which tells the status of socket, changed to SOCK_INIT(0x13). The value of Sn_SR can be read easily using the getSn_SR() function. If SOCKETn did not change to SOCK_INIT(0x13), it means that the SOCKET was not created properly. In this case, call socket() function and create the SOCKET again.

4.1.2 LISTEN

```
s = 0;          // set SOCKET 0
listen(s);    // change the state of SOCKET 0 to LISTEN
```

Code **오류! 지정한 스타일은 사용되지 않습니다.**.2 LISTEN state

After the SOCKET is created, in order to connect with the client, the status of SOCKET needs to be in LISTEN status. There are two ways to change the status of SOCKET from SOCK_INIT(0x13) to SOCK_LISTEN(0x14). One way is for the user to command Sn_CR_LISTEN(0x02) in the Sn_CR() register. The other way is to call the listen() function. Both ways change the status of SOCKET from SOCK_INIT(0x13) to SOCK_LISTEN(0x14) immediately. Then in LISTEN status, the TCP server waits for the client's connection request. When the server is connected with the client, the status of SOCKET will change to SOCK_ESTABLISHED(0x17). Now, the TCP server can exchange data with the client.

4.1.3 RS232 parameter initialization

```
/* TIMER setting for Baud Rate generating */
        ET1 = 0;            /* TIMER1 INT DISABLE */
        TMOD = 0x20;        /* TIMER MODE 2 */
        PCON |= 0x80;               /* SMOD = 1 */
        TH1 = 0xFC;                 /* X2 115200(SMOD=1) at 88.4736 MHz */
        TR1 = 1;                    /* Start the TIMER1 */
        SCON = 0x52;                /* Serial mode 1, REN=1, TI=1, RI=0 */
        /* Interrupt setting */
        RI   = 0;           /* Clear the receive interrupt flag of UART*/
        TI   = 0;           /* Clear the transmit interrupt flag of UART*/
        ES    = 1;          /* Serial interrupt enable */
```

Code 오류! 지정한 스타일은 사용되지 않습니다..3 RS232 initialize

After the TCP server is initialized, user should initialize the timer related registers for serial communication of W7100, as shown above in Code 4.3. For more details, please refer to 'How to use UART in W7100.'

4.1.4 Serial Interrupt processing

```
if(RI)                              //Check for Receive Interrupt flag
{                                   //s_buf : serial buffer
        s_buf[s_write_ptr] = SBUF;  /Write data from Serial buffer to uart rx buffer
        s_write_ptr++;              // Increment write pointer
        //Check if write pointer is bigger than Maximum size of Serial buffer
        if(s_write_ptr >= MAX_SBUF_SIZE){
                s_write_ptr = 0;
                if(s_write_ptr == s_read_ptr) overflow = 1;   // Buffer overflow check
        }
        else{
                if(s_write_ptr == s_read_ptr) overflow = 1;   // Buffer overflow check
        }
        RI = 0;     // Disable Receive Interrupt flag
}
```

Code 오류! 지정한 스타일은 사용되지 않습니다..4 Serial interrupt processing

Since serial interrupt is enabled in the 'serial initialization code,' a serial interrupt occurs when a serial message is sent or received. Therefore, a code for processing the serial interrupt needs to be entered. Code 4.4 is the code the saves the received data in the serial buffer. If the buffer is overflowed, the serial buffer needs to be controlled.

4.1.5 TCP to RS232

```
if((e_len = getSn_RX_RSR(s)) > 0)        // check Rx data
{                                        // e_len = length of the received Ethernet data, e_buf = Etherent RX buffer
    if(e_len > MAX_EBUF_SIZE) e_len = MAX_EBUF_SIZE; // Check the max Ethernet buffer size
    s_len = recv(s, e_buf, e_len);       // Receive the Ethernet data
    for(i=0; i < s_len; i++){
        putchar(*(e_buf + i));           // transmit the data to serial byusing putchar() function
    }
}
```

Code 오류! 지정한 스타일은 사용되지 않습니다..5 RECEIVE and SEND data

When data is received, the value of Sn_RX_RSR() becomes the length of the received data (refer to W7100 datasheet). Therefore, check the Sn_RX_RSR() register to see whether the value is larger than 0. If so, use the getSn_RX_RSR() function to read the value of Sn_RX_RSR(). Recv() function is used to receive data; among the factors from the recv() function, s is the socket number, e_buf is the buffer for saving the received data, e_len is the length of the received data. When the data is received, use the putchar() function to send the received data to the RS232 serial port.

4.1.6 RS232 to TCP

```
EA = 0;                                  //Disable all interrupt
tmp_read = s_read_ptr;                   // Get current value of read pointer
tmp_write = s_write_ptr;                 // Get current value of write pointer
EA = 1;                                  //Enable all interrupt
If(overflow){
      length = (MAX_SBUF_SIZE - tmp_read) + tmp_write;   //Calculate the length of data to be read
      send(s, &s_buf[tmp_read], MAX_SBUF_SIZE - tmp_read));
      send(s, &s_buf[0], tmp_write);
      overflow = 0;                      //Clear the overflow flag
}
if(tmp_write == tmp_read) return 0;      // Read and write pointer can not be the same
if(tmp_write > tmp_read){
      length = tmp_write - tmp_read;     //Calculte the length of data to be read
      send(s, &s_buf[tmp_read], length);
}
else{
      length = (MAX_SBUF_SIZE - tmp_read) + tmp_write;   //Calculate the length of data to be read
      send(s, &s_buf[tmp_read], MAX_SBUF_SIZE - tmp_read));
      send(s, &s_buf[0], tmp_write);
}
```

```
s_read_ptr += length;              //Updata the read pointer
//Check if read pointer is bigger than Maximum size of Serial buffer
if(s_read_ptr >= MAX_SBUF_SIZE) s_read_ptr -= MAX_SBUF_SIZE;
return length;                     //return length of data readed
```

Code 오류! 지정한 스타일은 사용되지 않습니다..6 Serial interrupt processing

The basic concept of 'RS232 to TCP conversion' is similar to the concept of 'TCP to RS232' mentioned above. When the serial data is received, save the data in temporal buffer according to 'interrupt service routine,' and use SEND() function to send the data to the TCP server. Be cautious about the data when the buffer of interrupt service routine is overflowed. In Code 2.2.3, it is programmed to immediately send the whole received data when the overflow flag is asserted.

Once the conversion is done and there are no data to send/receive, the TCP connection should end. Use disconnect() function or close() function to end the connection.

4.1.7 DISCONNECT

```
s = 0; // set SOCKET 0
disconnect(s);
```

Code 오류! 지정한 스타일은 사용되지 않습니다..7 DISCONNECT

disconnect() function sends the disconnect-request (FIN packet) and waits for the disconnect-reply (FIN/ACK packet). When the disconnect() function is used, the status of SOCKET changes to SOCK_CLOSED(0x00), and then close the SOCKET. When the disconnect-request arrives, W7100 creates the FIN/ACK packet, so that the other user can close the SOCKET. If there is no reply after a period of time passed since the disconnect-request was sent, TCP timeout will occur and the status of the SOCKET will change to SOCK_CLSOED(0x00).

4.1.8 CLOSE

```
s = 0; // set SOCKET 0
close(s);
```

Code 오류! 지정한 스타일은 사용되지 않습니다..8 CLOSE

Unlike the disconnect() function, close() function changes the status of socket to SOCK_CLOSED(0x00) immediately. User can disconnect with other user right away. If the RST packet was received by the other user the status of socket will change to SOCK_CLOSED(0x00). Please refer to W7100 datasheet for more details on RST packet. Once the status of SOCKET has changed to SOCK_CLOSED(0x00), it needs to be reopened in order to reuse it.

4.2 Using TCP Client

4.2.1 CONNECT

The code for implementing 'serial to Ethernet' in TCP client mode is similar to TCP server mode except the LISTEN step. Because the client has to connect to the server, socket number that will be used,

destination IP, and destination port number is required. If the settings are complete using the connect() function, the connection to the server will be successful and the status of SOCKET will change to SOCK_ESTABLISHED(0x17), and data transmission will be possible.

```
s = 0; // set SOCKET 0
serverip[4] = {192, 168, 1, 2}; // set server(destination) IP
serverport = 5000;                // set server(destination) port
connect(s, serverip, serverport);
```

Code 오류! 지정한 스타일은 사용되지 않습니다..9 CONNECT

All codes except the Connect process are the same with the codes from TCP server section, so please refer to section 4.1.

4.3 Using UDP

Since UDP does not need connection, the steps: listen, connect, disconnect, and close are not necessary. When 'serial to Ethernet' is implemented using UDP, the following steps are the only requires the steps: socket open, UDP to RS232 conversion, RS232 to UDP conversion.

4.3.1 OPEN

```
s = 0;          // set SOCKET 0 (From 0 to 7)
socket(s, Sn_MR_UDP, port, mode);     // OPEN SOCKET 0
while(getSn_SR(s) != SOCK_INIT);
```

Code 오류! 지정한 스타일은 사용되지 않습니다..10 Open SOCKET

The basic codes for the OPEN step are the same with Code 4.1. The only difference is to enter Sn_MR_UDP instead of Sn_MR_TCP in protocol factor. For more details, please refer to section 4.1.1.

4.3.2 UDP to RS232

```
if((e_len = getSn_RX_RSR(s)) > 0)          // check Rx data
{                                  // e_len = length of the received Ethernet data, e_buf = Etherent RX buffer
    if(e_len > MAX_EBUF_SIZE) e_len = MAX_EBUF_SIZE; // Check the max Ethernet buffer size
    s_len = recvfrom(s, e_buf, e_len, (uint8 *)destIP, &destport);       // Receive the Ethernet data
    for(i=0; i < s_len; i++)
    {
        putchar(*(e_buf + i));                // transmit the data to serial byusing putchar() function
    }
}
```

Code 오류! 지정한 스타일은 사용되지 않습니다..11 RECEIVE and SEND data

The basic steps for 'UDP to RS232' are the same as the section 4.1.5. The only difference is using recvfrom() function instead of recv() function. The recvfrom() function needs two more input factors than the recv() function; destIP is the factor for the peer's IP data and destport is the factor for the

peer's port data. By using these two factors, data can be sent to a peer by using sendto() function without connection. When data is received, send the data to RS232 serial port. Please refer to section 4.1.5 for other more details.

4.3.3 RS232 to UDP

```
EA = 0;                                  //Disable all interrupt
tmp_read = s_read_ptr;                   // Get current value of read pointer
tmp_write = s_write_ptr;                 // Get current value of write pointer
EA = 1;                                  //Enable all interrupt
If(overflow){
      length = (MAX_SBUF_SIZE - tmp_read) + tmp_write;   //Calculate the length of data to be read
      sendto(s, (uint8*)&s_buf[tmp_read], MAX_SBUF_SIZE - tmp_read, (uint8*)destIP, destport);
      sendto(s, (uint8*)&s_buf[0], tmp_write);
      overflow = 0;                      //Clear the overflow flag
}
if(tmp_write == tmp_read) return 0;      // Read and write pointer can not be the same
if(tmp_write > tmp_read){
      length = tmp_write - tmp_read;     //Calculte the length of data to be read
      sendto(s, (uint8*)&s_buf[tmp_read], length, (uint8*)destIP, destport);
}
Else{
      length = (MAX_SBUF_SIZE - tmp_read) + tmp_write;   //Calculate the length of data to be read
      sendto(s, (uint8*)&s_buf[tmp_read], MAX_SBUF_SIZE - tmp_read, (uint8*)destIP, destport);
      sendto(s, (uint8*)&s_buf[0], tmp_write, (uint8*)destIP, destport);
}
s_read_ptr += length;              //Updata the read pointer
//Check if read pointer is bigger than Maximum size of Serial buffer
if(s_read_ptr >= MAX_SBUF_SIZE) s_read_ptr -= MAX_SBUF_SIZE;
return length;                     //return length of data readed
```

Code **오류! 지정한 스타일은 사용되지 않습니다.**.12 Serial interrupt processing

The basic steps are the same with section 4.1.6. The only difference is using sendto() function instead of send() function to send the received data of serial buffer to Ethernet. The sendto() function needs two more input factors than the send() function; destIP is the factor for the peer's IP data and destport is the factor for the peer's port data. Please refer to section 4.1.6. for other more details.

Document History Information

Version	Date	Descriptions
Ver. 0.9 Beta	Sep, 2009	Release with W7100 launching
Ver. 1.0	Mar, 2011	Modify for W7100A 64pin QFN package

Copyright Notice

Technical Support: support@wiznet.co.kr
Sales & Distribution: sales@wiznet.co.kr

For more information, visit our website at http://www.wiznet.co.kr