TCP Application note for W5200

Version 1.0

Table of Contents

1 Introduction

The TCP (Transmission Control Protocol) controls data communication between networks. As one of the main protocols that forms the internet, more details are written in RFC 793 of IETF (Internet Engineering Task Force). TCP is a protocol that runs above IP; TCP guarantees the data transmission and allows receiving data in the order it was sent.

Since W7100 supports TCP protocol in the Transport Layer, user can use TCP/IP protocol without any composition.

2 TCP/IP SOCKET

User can use all eight SOCKETs provided by W5200 as TCP protocol. If the user wishes to use W5200 as TCP Protocol, user must first create the SOCKET that is going to be used. When creating SOCKET, SOCKET number, protocol to be used, port number to be used, and the flag to be set is required. This document is going to explain about TCP protocol; the protocol that is going to be used should set the Sn_MR (SOCKET n Mode Register in W5200.h) to Sn_MR_TCP (0x01). The SOCKET number means the existing eight SOCKETs and user can randomly number each one from 0 to 7. The port number that is going to be used can be assigned by user for the TCP protocol. There will be no problem if the above requirements for creating a SOCKET were assigned by using the SOCKET() function from WIZnet.

Since the TCP protocol of W5200 supports both sever mode and client mode, user can select one and use for its application. The difference between server mode and client mode are shown below.

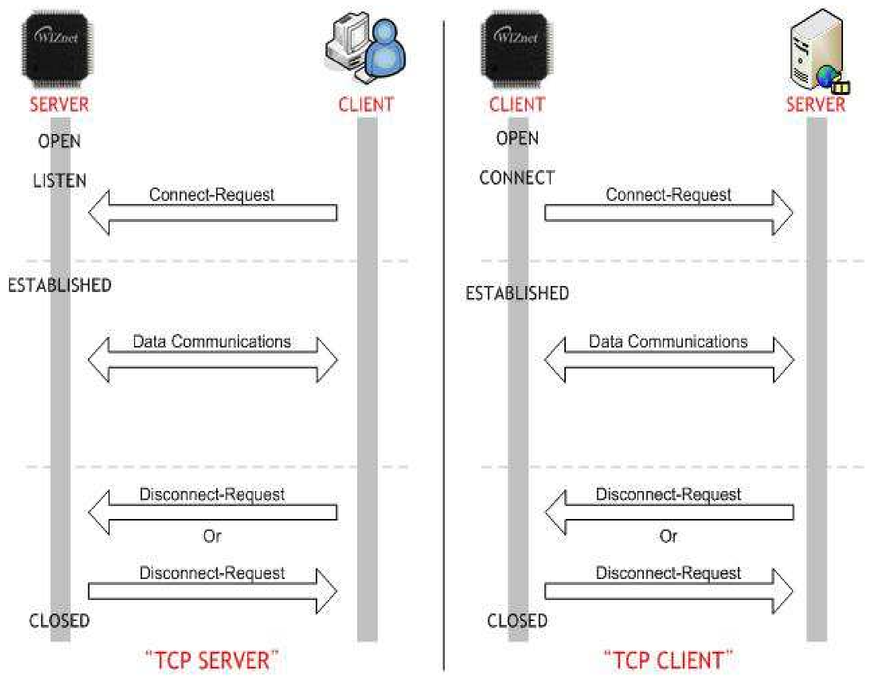


Fig. 2.1 TCP Server & TCP Client

As shown in Figure 2.1, all actions are the same except for one; whether the status is the LISTEN() or CONNECT() after opening the SOCKET. When TCP protocol runs in server mode, the SERVER will wait for CLIENT's connection request in LISTEN status. However, when TCP protocol runs in client mode, the CLIENT will try to connect to server in CONNECT status. Once the connection is successful, the status of SOCKET will change to ESTABLISHED(SOCK_ESTABLISHED, 0x17) status. SOCKETs that connect after this point can stay connected and exchange data until the SOCKET is closed.

The SOCKET lifecycle in server mode is consisted of OPEN, LISTEN, SEND, RECEIVE, DISCONNECT, and CLOSE. The SOCKET lifecycle in client mode is consisted of OPEN, CONNECT, SEND, RECEIVE, DISCONNECT, and CLOSE. Starting from section 2.1, the operations from each SOCKET Lifecycle will be explained in detail with example source code (Pseudo code).

2.1 OPEN

The first step of creating a SOCKET for both SERVER and CLIENT mode is OPEN. To create SOCKETn (the n th SOCKET), use the SOCKET() function to set the SOCKET number, protocol, port number, and flag. Since the protocol is TCP, set the protocol to Sn_MR_TCP(0x01). Setting of port number differs depending on whether it is server mode or client mode. When server mode is being used, server can set the source port number that the client is using. But when client mode is being used, it is best to choose a random port number and increase one number at a time until the SOCKET is connected since there can be a destination port number already in use. The flag of the TCP protocol is for 'No Delayed Ack flag' and etc. In general, it is set to 0. More details on protocol types, flag types, and etc. are explained in 'Sn_MR value' of 'W5200.h,' that is provided by WIZnet.

After all settings are completed, check Sn_SR(n) register to see whether the status of SOCKETn is changed to SOCK_INIT(0x13). User can use getSn_SR(SOCKETn) function when checking Sn_SR(n) register. If the status of SOCKETn is SOCK_INIT(0x13), SOCKET is created properly. If it is not created, user should recreate the SOCKET.

```
/* Method 1 : Server mode */
```

```
/* sets Protocol Number */
s = 0; // set SOCKET 0
/* OPEN SOCKET 0 */
socket(s, Sn_MR_TCP, port, mode);
while(getSn_SR(s) != SOCK_INIT);
```

```
/* Method 2 : Client mode */
```

```
/* sets Protocol Number */
s = 0; // set SOCKET 0
/* sets port number */
any_port = 1000;
/* OPEN SOCKET 0 */
socket(s, Sn_MR_TCP, any_port++, mode);
while(getSn_SR(s) != SOCK_INIT);
```

Example 2.1 Open Socket

2.2 LISTEN

The LISTEN step is only used during SERVER mode. After creating SOCKETn, change the status of SOCKET to LISTEN so that the CLIENT can connect. In order to change the status of SOCKET from SOCK_INIT(0x13) to LISTEN status, user can directly set the Sn_CR_LISTEN(0x02) to the Sn_CR(n) register or can use the LISTEN(n) function from 'SOCKET.c.' After the status changes to LISTEN the status of SOCKET will change to SOCK_LISTEN(0x14). Then, SOCKET will wait until there is a request to connect from CLIENT. Once CLIENT is connected, the status of SOCKET will change again to SOCK_ESTABLISHED(0x17). Then finally data transmission is possible with the CLIENT.

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

Example 2.2 set LISTEN state

2.3 CONNECT

The CONNECT stage is used during CLIENT mode to connect to the SERVER. The SOCKET number that is going to be used, destination IP, and destination port number are required to CONNECT. Set these requirements by using CONNECT() function; and once the connection is successful, the status of SOCKET will change to SOCK_ESTABLISHED(0x17).

```
s = 0; // set SOCKET 0
Serverip[4] = {192, 168, 11, 3}; // set Server(destination) IP
Serverport = 3000;    // set Server(destination) port
connect(s, Serverip, Serverport);
```

Example 2.3 set CONNECT state

2.4 SEND

In the case of TCP protocol, the connection between the peer is already established before sending data; therefore, not much information is required when sending data. Set the SOCKET number, address of the data that is going to be sent, data size, the use of resend (ON, OFF) when Windowfull occurs. The address of the data that is going to be sent is usually set by selecting the area, putting the data in, and setting the area with pointer.

In order to confirm the data transmission after the SEND command, the data length to be sent and the data length that is actually sent must be identical. The data length that is actually sent can be calculated by checking the Sn_TX_RD register value before and after the SEND command. The SEND process will complete if the data length to be sent and the data length actually sent matches identically. However, if not, Windowfull will occur in the peer's buffer and this means that parts of the data are not successfully sent. If Windowfull occurs in the send() process of the example code, the data is sent in unit of 1 byte and as much as the number of WINDOWFULL_MAX_RETRY_NYM untill the peer's buffer is ready to send the remaining data. After the data is sent, wait for a period of time (WINDOWFULL_WAIT_TIME), and check whether the send process was successful by comparing the data length to be sent and the data length actually sent. The send process will be completed if the data lengths match. Be aware that a loss of data can occur if data copy is operated during th process. Close the socket since there can be problems of the network if the number of retry exceeds the WINDOWFULL_MAX_RETRY_NUM.

Modify WINDOWFULL_WAIT_TIME, WINDOWFULL_MAX_RETRY_NUM, and Socket close according to user's network when dealing with Windowfull operation. If needed, WINDOWFULL_WAIT_TIME can be set longer or user can close Socket right away when Windowfull occurs.

Below are the steps for Send Process.

1. **Start SEND Command by using Data Length.**
2. **Calculate the Data Length that is actually sent.**
   If the total Data Length is 10 and the Data Length that is sent is 7, the remaining Data Length (=Sn_TX_RD_after_SEND-Sn_TX_RD_befor_SEND) is 3.
3. **Repeat the Send Command until the sum of the Data Length that is actually sent equals the Data Length to be sent.**
   Note: Do not operate Data Copy until the value of Return equals the value of Data Length to be sent.

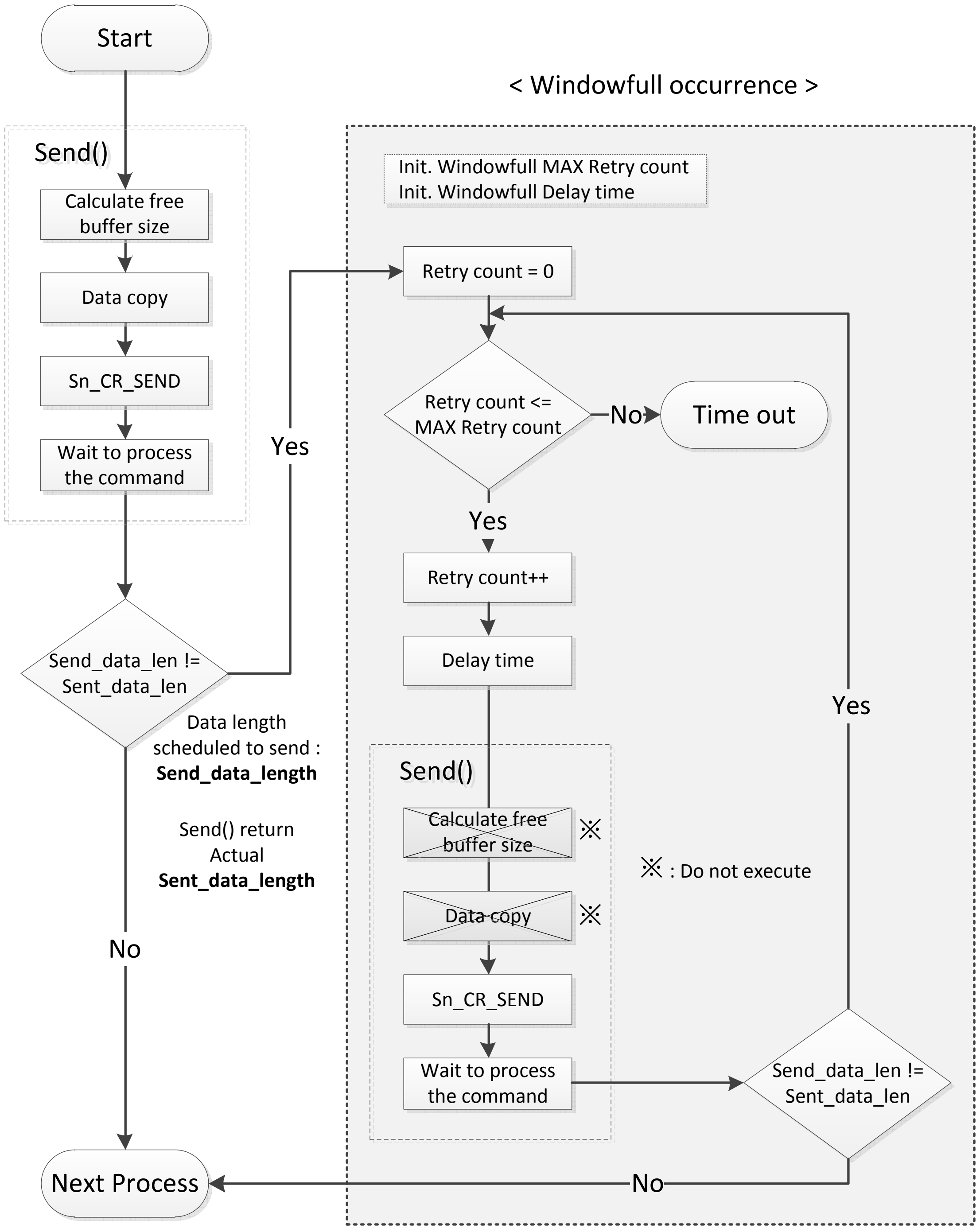

Start
Send()
Calculate free buffer size
Data copy
Sn_CR_SEND
Wait to process the command
Send_data_len != Sent_data_len
Yes
No
Data length scheduled to send : **Send_data_length**
Send() return Actual **Sent_data_length**
< Windowfull occurrence >
Init. Windowfull MAX Retry count
Init. Windowfull Delay time
Retry count = 0
Retry count <= MAX Retry count
No
Time out
Yes
Retry count++
Delay time
Send()
Calculate free buffer size ※
Data copy ※
Sn_CR_SEND
Wait to process the command
※ : Do not execute
Send_data_len != Sent_data_len
Yes
No
Next Process

Fig. 2.2 SEND() : Windowfull handling process

```
/* Send data to connected peer. */
// TX_RX_MAX_BUF_SIZE must be smaller than the maximum size of the TX buffer
s = 0; // set SOCKET 0
* data_buf; // set position of data buffer
RSR_len = getSn_RX_RSR(s); // set length for send
sent_data_len = send(s, data_buf, RSR_len, WINDOWFULL_FLAG_OFF); // send data and save the length
/* only assert when windowfull */
if(sent_data_len != received_len)
{
    Init_windowfull_retry_cnt(s); // windowfull_retry_cnt[s] set 0
    While(sent_data_len != received_len)
    {
        tmp_retry_cnt = incr_windowfull_retry_cnt(s);
        // default WINDOWFULL_MAX_RETRY_NUM = 3
        if(tmp_retry_cnt <= WINDOWFULL_MAX_RETRY_NUM)
        {
            // try to send the remaining data
            sent_data_len += sent(s, data_buf, received_len, WINDOWFULL_FLAG_ON);
            Delay_ms(WINDOWFULL_WAIT_TIME); // default delay = 1 sec
        }
        else
        {
            // if the 'Windowfull' occers and then send retry 3 times over, socket close
            close(s);
            while(1);
        }
    }
}
```

Example 2.4 SEND DATA

2.5 RECEIVE

RECEIVE is similar in usage method to SEND, but it checks the Sn_RX_RSR(n). The RECEIVE step is to move the data, which came into the RX buffer, to the user's data area. Therefore, user must check whether the value of Sn_RX_RSR(n) is larger than 0 before the RECEIVE step. If the value of Sn_RX_RSR(n) is larger than 0, it means that the data is in the RX buffer. User must use getSn_RX_RSR(n) function to check whether the data is received or not before the RECEIVE step.

```
/* Check received data */
// len indicates the received data size in the RX buffer.
// len must be smaller than the maximum size of the RX buffer
s = 0; // set SOCKET 0
* data_buf; // set position of data buffer
if ( (RSR_len = getSn_RX_RSR(s) ) > 0)
/* Received data */
// RSR_len is a length included the DATA packet.
received_len = recv(s, data_buf, RSR_len);
```

Example 2.5 RECEIVE DATA

2.6 DISCONNECT

There are two ways of closing a created SOCKET, and one is DISCONNECT(n). The DISCONNECT(n) is not used to just directly close the SOCKET; rather, it is used to send a disconnect-request (FIN packet) to a peer and wait for a disconnect-reply (FIN/ACK packet) to change the status of SOCKET to SOCK_CLOSED(0x00), and ultimately close the SOCKET. When a disconnect request comes in, W5200 generates a FIN/ACK packet to allow the peer to close SOCKET. If there is no answer from the peer after disconnect-request(FIN packet) is generated, TCP timeout occurs and after that, the status of SOCKET changes to SOCKET_CLOSED(0x00). When the user wants to DISCONNECT, use DISCONNECT(n) function and choose the SOCKET number that will generate the disconnect request.

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

Example 2.6 SET DISCONNECT

2.7 CLOSE

Unlike DISCONNECT, CLOSE directly changes the SOCKET to SOCK_CLOSED(0x00). User can use the CLOSE(n) function and choose the SOCKET number; it closes the SOCKET regardless of the peer. If a RST packet comes from a peer, SOCKET will unconditionally change to SOCK_CLOSED(0x00). Once the SOCKET changes to SOCK_CLOSED(0x00), that SOCKET is not usable unless it is opened again.

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

Example 2.7 SET CLOSE

3 TCP Loopback Program

3.1 Server Mode

TCP Loopback can check the performance of TCP protocol by using TCP protocol to send back the data that came in from a peer. This section will explain the example of Loopback in SERVER mode. The example codes are as followed.

```
void loopback_tcps(SOCKET s, uint16 port)
{
        uint16 RSR_len;
        uint16 received_len;
        uint8 * data_buf = TX_BUF;
        uint16 sent_data_len = 0;
        uint8 tmp_retry_cnt = 0;

         switch (getSn_SR(s))
         {
         case SOCK_ESTABLISHED:     /* if connection is established */
                 if(ch_status[s]==1)
                 {
                         ch_status[s] = 2;
                 }
                  if ((RSR_len = getSn_RX_RSR(s)) > 0)    /* check Rx data */
                 {
                         /* if Rx data size is lager than TX_RX_MAX_BUF_SIZE */
                         if (RSR_len > TX_RX_MAX_BUF_SIZE) RSR_len = TX_RX_MAX_BUF_SIZE;
                         received_len = recv(s, data_buf, RSR_len);   /* read the received data */
                         /* sent the received data */
                         sent_data_len = send(s, data_buf, received_len,
                                         (bool)WINDOWFULL_FLAG_OFF);
                     if(sent_data_len != received_len) /* only assert when windowfull */
                     {
                               init_windowfull_retry_cnt(s);
                               while(sent_data_len !=   received_len)
                               {
                                         tmp_retry_cnt = incr_windowfull_retry_cnt(s);
```

```
                                    if(tmp_retry_cnt <= WINDOWFULL_MAX_RETRY_NUM)
                                    {
                                            sent_data_len += send(s, data_buf,
                                          received_len, (bool)WINDOWFULL_FLAG_ON);
                                            Delay_ms(WINDOWFULL_WAIT_TIME);
                                     }
                                    else
                                    {
                                            close(s);
                                            while(1);
                                    }
                            }
                    }

            }
            break;
    case SOCK_CLOSE_WAIT:                                   /* If the client request to close */
            printf("\r\n%d : CLOSE_WAIT", s);
          if ((RSR_len = getSn_RX_RSR(s)) > 0)              /* check Rx data */
            {
                   /* if Rx data size is lager than TX_RX_MAX_BUF_SIZE */
                   if (RSR_len > TX_RX_MAX_BUF_SIZE) RSR_len = TX_RX_MAX_BUF_SIZE;
                   received_len = recv(s, data_buf, RSR_len); /* read the received data */
            }
            disconnect(s);
            ch_status[s] = 0;
            break;
    case SOCK_CLOSED:                                       /* if a socket is closed */
            if(!ch_status[s])
            {
                   printf("\r\n%d : Loop-Back TCP Server Started. port : %d", s, port);
                   ch_status[s] = 1;
            }
            if(socket(s,Sn_MR_TCP,port,0x00) == 0)      /* reinitialize the socket */
            {
                   printf("\r\n%d : Fail to create socket.",s);
                   ch_status[s] = 0;
```

```
                    }
                    break;
          case SOCK_INIT:     /* if a socket is initiated */
                  listen(s);
                  break;
          default:
                  break;
           }
}
```

Example 3.1 SET LOOPBACK SERVER

As shown in the example above, all functions from the SOCKET Lifecycle are used except the connect() function. At first use the getSn_SR(s) function to check the status of SOCKET. After W5200 is reset, all SOCKETs are in SOCK_CLOSED(0x00) status. Therefore, use the close(s) function to completely close the SOCKET, and use the socket(s, Sn_MR_TCP, port, 0x00) function to newly create SOCKET. If created properly, status will be SOCK_INIT, and then use the listen(s) function to change the SOCKET into LISTEN status. Once connected with a peer, SOCKET will change to SOCK_ESTABLISHED and wait for data. All received data are in the RX buffer. Use the recv(s, data_buf, len) function to save the length of received data in data_buf. Then use the send(s, data_buf, len) function to send back the data to the client. The client can compare the data before/after exchange and check whether the communication is working properly. If the received data length (received_len) and sent data length (sent_data_len) does not match each other, there is a possibility of the peer's Windowfull and Send Retry must be done as explained in section 2.4 to resend all remaining data.

The default WINDOWFULL_WAIT_TIME of the implemented Loopback example code is set to 1000ms, and WINDOWFULL_MAX_RETRY_NUM is set to 3; the Socket will close if the WINDOWFULL_MAX_ RETRY_NUM exceeds 3 before the entire send process is not completed due to windowfull.

3.2 Client Mode

This section will explain the example of Loopback in CLIENT mode. The example codes are as followed.

```
#define tick_second 1
void loopback_tcpc(SOCKET s, uint16 port)
{
            uint16 RSR_len;
           uint16 received_len;
           uint8 * data_buf = TX_BUF;
           uint16 sent_data_len = 0;
           uint8 tmp_retry_cnt = 0;

            switch (getSn_SR(s))
            {
            case SOCK_ESTABLISHED:                /* if connection is established */
                        if(ch_status[s]==1)
                        {
                                    printf("\r\n%d : Connected",s);
                                    ch_status[s] = 2;
                        }
                        if ((RSR_len = getSn_RX_RSR(s)) > 0)     /* check Rx data */
                        {
                                    /* if Rx data size is lager than TX_RX_MAX_BUF_SIZE */
                                    if (RSR_len > TX_RX_MAX_BUF_SIZE) RSR_len = TX_RX_MAX_BUF_SIZE;
                                    received_len = recv(s, data_buf, RSR_len);   /* read the received data */
                                    /* sent the received data */
                                    sent_data_len = send(s, data_buf, received_len,
                                                   (bool)WINDOWFULL_FLAG_OFF);
                               if(sent_data_len != received_len)   /* ohly assert when windowfull */
                               {
                                          init_windowfull_retry_cnt(s);
                                          while(sent_data_len !=   received_len)
                                          {
                                                      tmp_retry_cnt = incr_windowfull_retry_cnt(s);
                                                      if(tmp_retry_cnt <= WINDOWFULL_MAX_RETRY_NUM)
                                                      {
```

```
                                                sent_data_len += send(s, data_buf,
                                              received_len, (bool)WINDOWFULL_FLAG_ON);
                                                Delay_ms(WINDOWFULL_WAIT_TIME);
                                   }
                                  else
                                  {
                                                close(s);
                                                while(1);
                                  }
                          }
                  }
            }
            break;
     case SOCK_CLOSE_WAIT:                /* If the client request to close */
            printf("\r\n%d : CLOSE_WAIT", s);
            if ((RSR_len = getSn_RX_RSR(s)) > 0)      /* check Rx data */
            {
                    /* if Rx data size is lager than TX_RX_MAX_BUF_SIZE */
                   if (RSR_len > TX_RX_MAX_BUF_SIZE) RSR_len = TX_RX_MAX_BUF_SIZE;
                   received_len = recv(s, data_buf, RSR_len);    /* read the received data */
            }
            disconnect(s);
            ch_status[s] = 0;
            break;
     case SOCK_CLOSED:                    /* if a socket is closed */
            if(!ch_status[s])
            {
                   printf("\r\n%d : Loop-Back TCP Client Started. port: %d", s, port);
                   ch_status[s] = 1;
            }
            if(socket(s, Sn_MR_TCP, port, 0x00) == 0)       /* reinitialize the socket */
            {
                   printf("\a%d : Fail to create socket.",s);
                   ch_status[s] = 0;
            }
            break;
    case SOCK_INIT:         /* if a socket is initiated */
            /* For TCP client's connection request delay : 3 sec */
```

```
                    if(time_return() - presentTime >= (tick_second * 3)) {
                              /* Try to connect to TCP server(Socket, DestIP, DestPort) */
                              connect(s, Chconfig_Type_Def.destip, Chconfig_Type_Def.port);
                              presentTime = time_return();
                    }
                    break;
          default:
                    break;
          }
}
```

Example 3.2 SET LOOPBACK CLIENT

The example codes for TCP client are very similar to the example codes of TCP server. The only difference is that in case of SOCK_INIT, use the connect() function to connect to server instead of listen() function.

The Loopback test is implemented in polling method; TCP client continuously uses connect() function to connect with Server. Since the repetition of request for connection with network can cause overhead, timer interrupt is used to enable connection request in every 3 seconds.

4 TCP Loopback Program Demonstration

The demonstration of TCP Loopback will be covered in this section. Download the TCP loopback application binary file or Source code and follow the demonstration steps (For more details, please refer to the 'W5200E01-M3 User's Guide').

The status of the W5200E01-M3 board is shown in serial message using the serial terminal; users can check the connection status with the serial message.

Since the use of Serial terminal program and Telnet Client depends on the user's OS, check the Windows version according to Table 4.1. If the user uses Windows Vista or 7, additional program settings are required.

Tab. 4.1 Windows setting for serial terminal

Serial terminal program	
Windows XP or older version	The Hyper terminal is included as Windows basic application program. Use the Hyper terminal to check serial message.
Windows Vista / Windows 7	The Hyper terminal program is not included as basic application program. Other serial terminal program must be used. TeraTerm can be used as open serial terminal program, and the port settings are identical as Hyper terminal. TeraTerm is an open serial terminal program with BSD license. TeraTerm can be downloaded at http://ttssh2.sourceforge.jp/

This document will use Hyper terminal, which is the most used serial terminal program, to check the serial message of W5200E01-M3 when performing TCP Loopback test.

4.1 Setting for Running TCP Loopback

Connect USB mini cable and LAN cable to W5200E01-M3 for TCP Loopback demonstration. Use 'Flash Loader Demonstrator' from STMicroelectronics to load the TCP Loopback binary image on W5200E01-M3 board. Please refer to the 'W5200E01-M3 User's Guide' for more details on how to use W5200E01-M3 and Flash Loader Demonstrator.

The process of loading image to W5200E01-M3 is as followed.

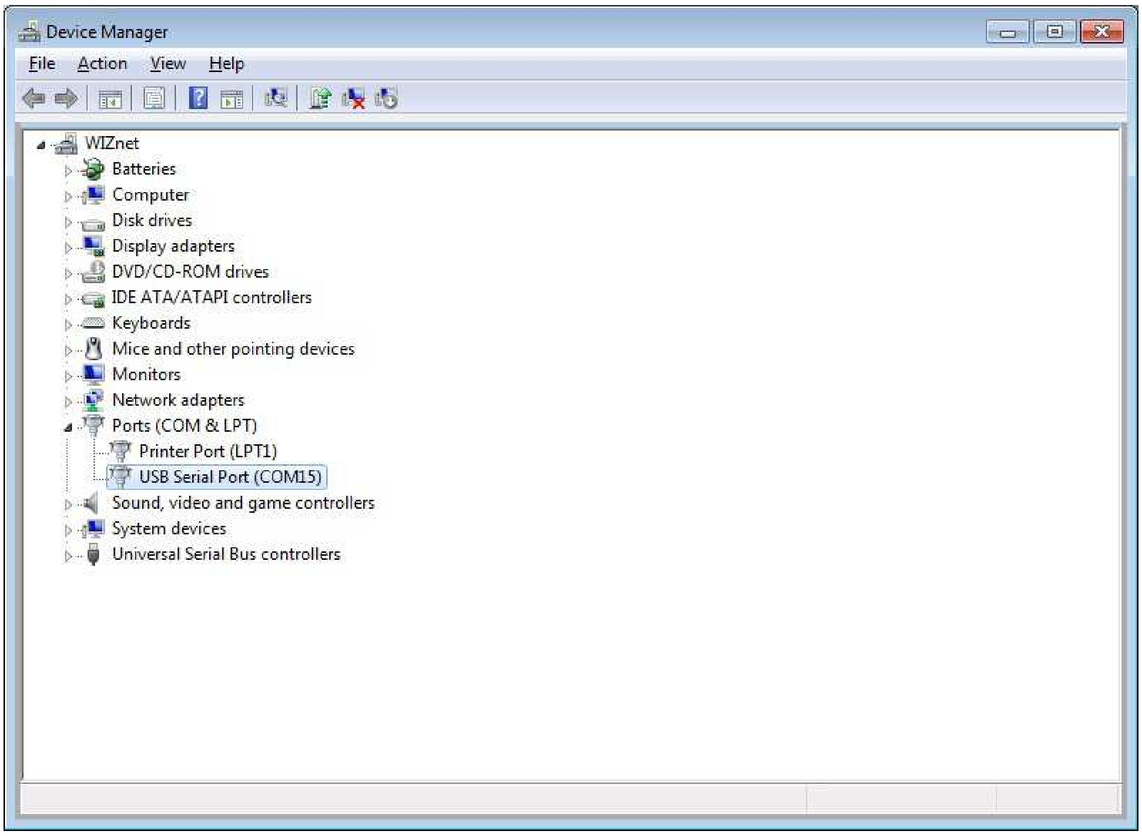
1. Connect the USB mini cable and LAN cable to W5200E01-M3.
2. Modify the provided source code from the IAR Compiler according to the user's network.
3. Compile the modified source code and create an application image.

4. Select PROG, and use the Flash Loader Demonstrator to download the created image to the board.
5. After the download is complete, change the PROG S/W to RUN. Set the Serial terminal port and check the application.

4.1.1 Physical Connection

Once the user's PC and W5200E01-M3 is connected through USB mini cable, the COM port is assigned to W5200E01-M3. Check the assigned COM port in order to use Flash Loader Demonstrator to download image on the board. The document uses COM 15 as COM port, but this can change according to the user's setting.

User can check the COM port at [Control Panel – Device Manager].



- WIZnet
  - Batteries
  - Computer
  - Disk drives
  - Display adapters
  - DVD/CD-ROM drives
  - IDE ATA/ATAPI controllers
  - Keyboards
  - Mice and other pointing devices
  - Monitors
  - Network adapters
  - Ports (COM & LPT)
    - Printer Port (LPT1)
    - USB Serial Port (COM15)
  - Sound, video and game controllers
  - System devices
  - Universal Serial Bus controllers

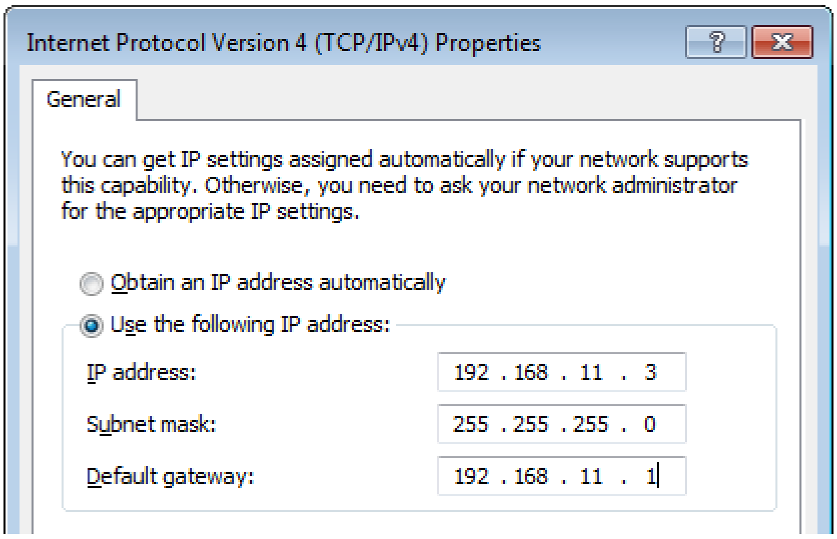
Fig. 4.1 Check the COM port in Windows device manager

4.1.2 Network Configuration

Once the COM port is checked, prepare the binary image to program the board. The Tcp Loopback source code is implemented using the IAR Embedded Workbench IDE and is provided at WIZnet's homepage / [SUPPORT – Download]. The network information of the board is included in the main.c file of the source code, and the network information must be modified according to the user's settings.

```
/* main.c */
…
/* Configure Network Information of W5200 */
uint8 MAC[6] = {0x00, 0x08, 0xDC, 0x01, 0x02, 0x03};   // MAC Address
uint8 IP[4] = {192, 168, 11, 4};   // IP Address
uint8 GateWay[4] = {192, 168, 11, 1};   // Gateway Address
uint8 SubNet[4] = {255, 255, 255, 0};   // SubnetMask Address
…
/* FOR TCP Client */
/* Configure Network Information of TEST PC */
uint8 Dest_IP[4] = {192, 168, 11, 3}; // DST_IP Address
uint16 Dest_PORT = 3000; // DST_IP port


…
```

Test MAC address and IP address have been used in this document for demonstration purpose. The IP address of the Test PC was set to 192.168.11.3; the IP address of the board and gateway must be set according to the Test PC. Users have to check or change IP address of PC: [Local Area Connection – Properties – Internet Protocol – Properties]


Internet Protocol Version 4 (TCP/IPv4) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address: 192 . 168 . 11 . 3
Subnet mask: 255 . 255 . 255 . 0
Default gateway: 192 . 168 . 11 . 1

Fig. 4.2 Internet Protocol Properties

The test MAC address is 00:08:DC:01:02:03 and test IP address is 192.168.11.4 for W5200e01-M3. When setting the network, enter the identical Gateway as the Test PC's.

After setting the network, use ping test to check the communication between W5200E01-M3 and the network. Open the command window and enter [ping 192.168.11.4], and if W5200e01-M3 is connected, the screen will appear as shown in Figure 3.4.

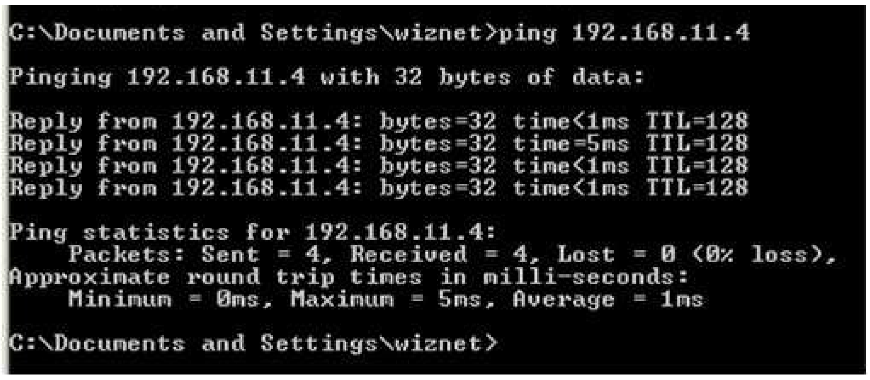

```
C:\Documents and Settings\wiznet>ping 192.168.11.4

Pinging 192.168.11.4 with 32 bytes of data:

Reply from 192.168.11.4: bytes=32 time<1ms TTL=128
Reply from 192.168.11.4: bytes=32 time=5ms TTL=128
Reply from 192.168.11.4: bytes=32 time<1ms TTL=128
Reply from 192.168.11.4: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.11.4:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 5ms, Average = 1ms

C:\Documents and Settings\wiznet>
```

Fig. 4.3 Ping Test at Command Prompt

If the ping test is successful, network settings are completed.

4.1.3 Compile

Once the network setting is completed, the source code must be compiled and linked in order to create the TCP Loopback firmware image for programming W5200E01-M3. Use [Project – Make] from the menu of IAR Embedded Workbench IDE or press the [F7] key to operate the 'Make' process, and then the

W5200EVB_App.bin file will be created in the project directory's [\Debug\Exe].

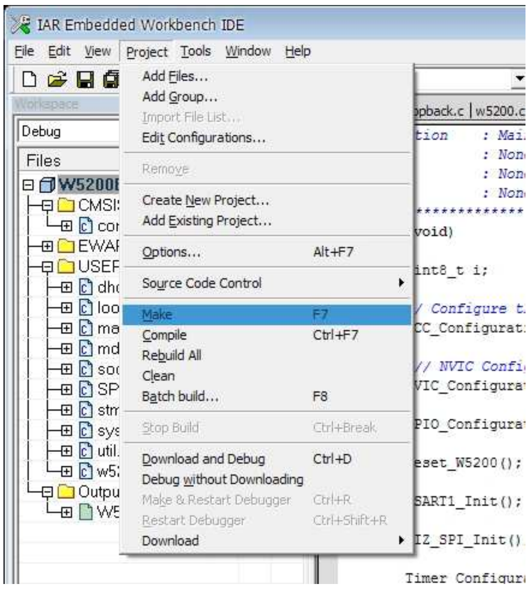
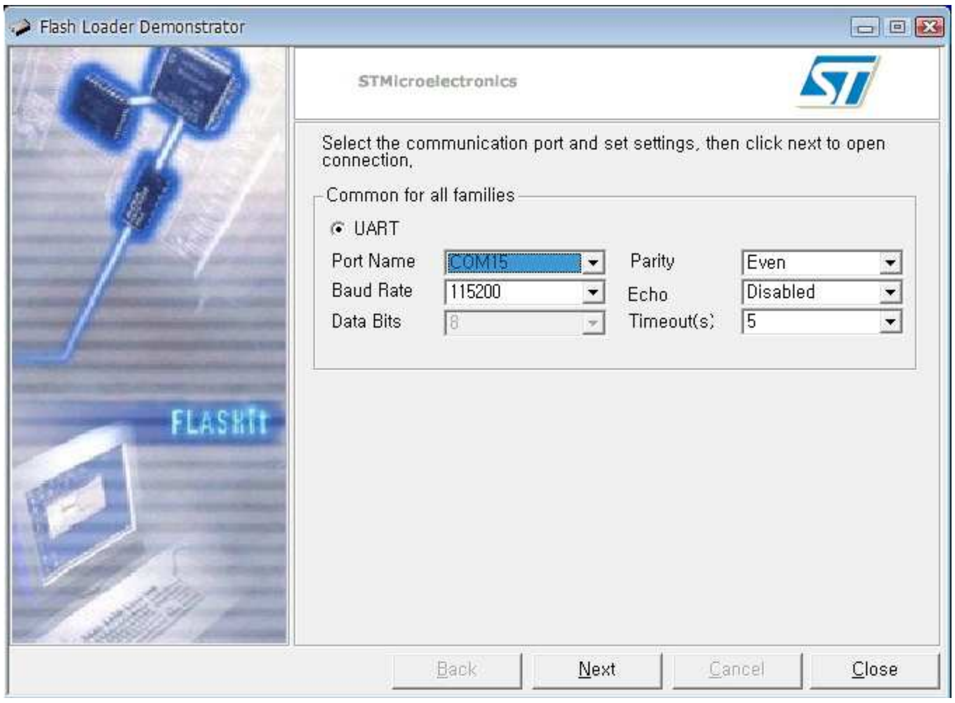


Fig. 4.4 Compile on IAR Embedded Workbench IDE

4.1.4 Download

The process of downloading the created W5200EVB_App.bin file to the W5200E01-M3 board is as followed.

1. Create W5200EVB_App.bin in IAR complier
2. Select PROG for PROG S/W of W5200E01-M3 and reset the board
3. Run the Flash Loader Demonstrator and set serial port
4. Select Target device (STM32_Med-density_64K)
5. Select 'Download to device' and set the image file path
   (image file path : [\Work\App\Debug\Exe] / located in project directory)
6. Once download is completed, change the PROG S/W to RUN and reset the board



Select the communication port and set settings, then click next to open connection.

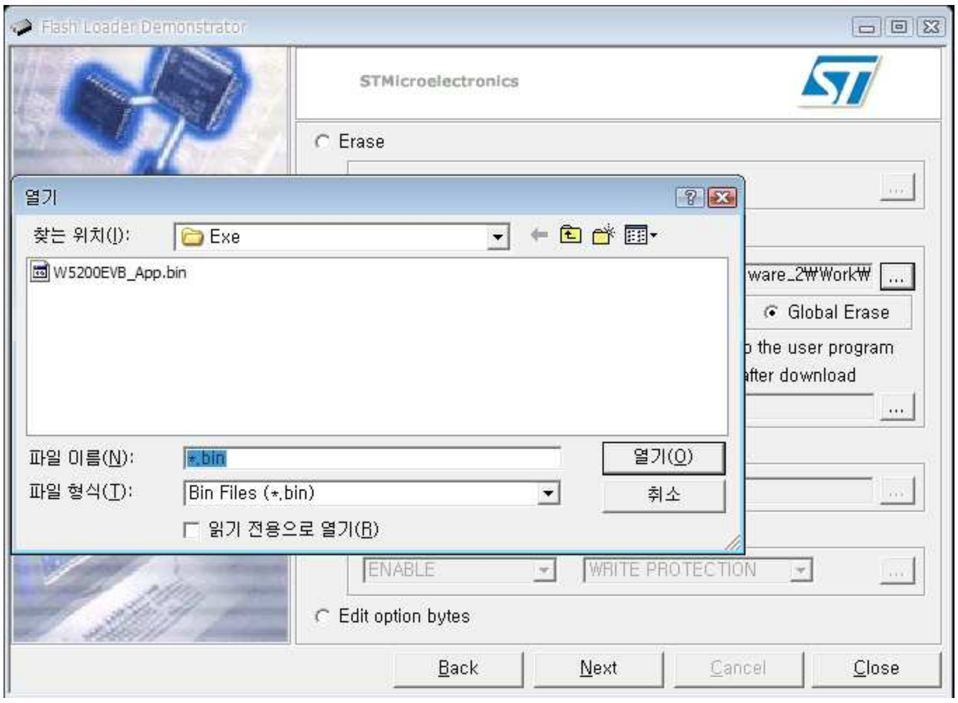
Common for all families

UART

Port Name	COM15	Parity	Even
Baud Rate	115200	Echo	Disabled
Data Bits	8	Timeout(s)	5

Fig. 4.5 Flash Loader Demonstrator setting

Select the COM port that is connected with W5200E01-M3 for Port Name. The settings must be done according to user's PC's serial communication settings.

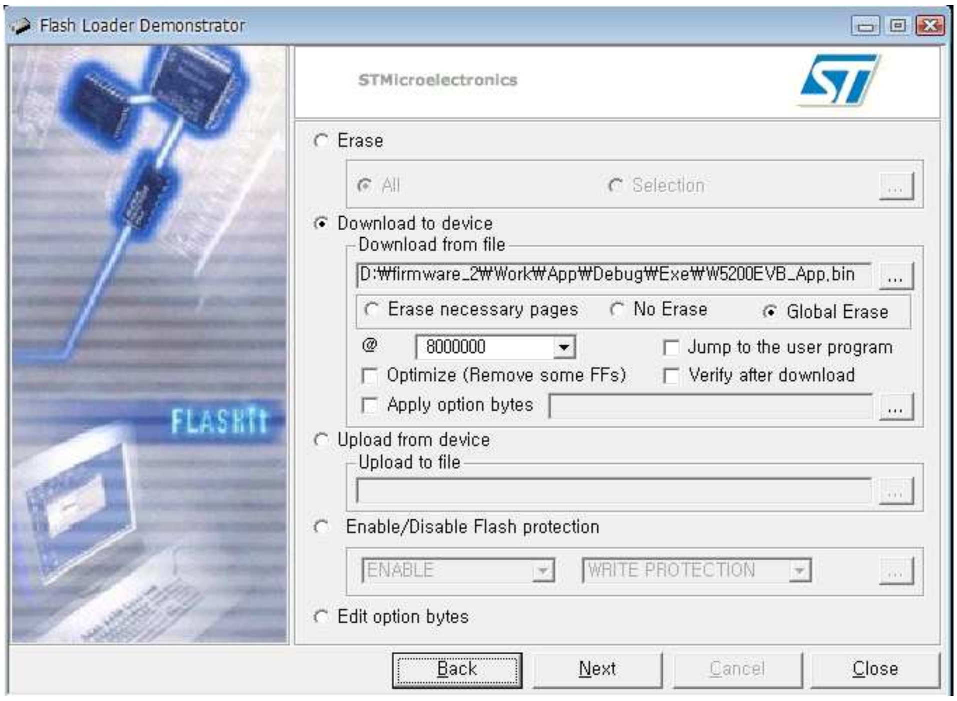


파일 이름(N): *.bin

파일 형식(T): Bin Files (*.bin)

읽기 전용으로 열기(R)

Fig. 4.6 Select Binary image for Flash Loader Demonstrator



Flash Loader Demonstrator

STMicroelectronics

Erase: All / Selection

[✓ Download to device]

Download from file: D:₩firmware_2₩Work₩App₩Debug₩Exe₩W5200EVB_App.bin

Erase necessary pages / No Erase / [✓ Global Erase]

@ 8000000

Jump to the user program

Optimize (Remove some FFs)

Verify after download

Apply option bytes

Upload from device

Upload to file

Enable/Disable Flash protection

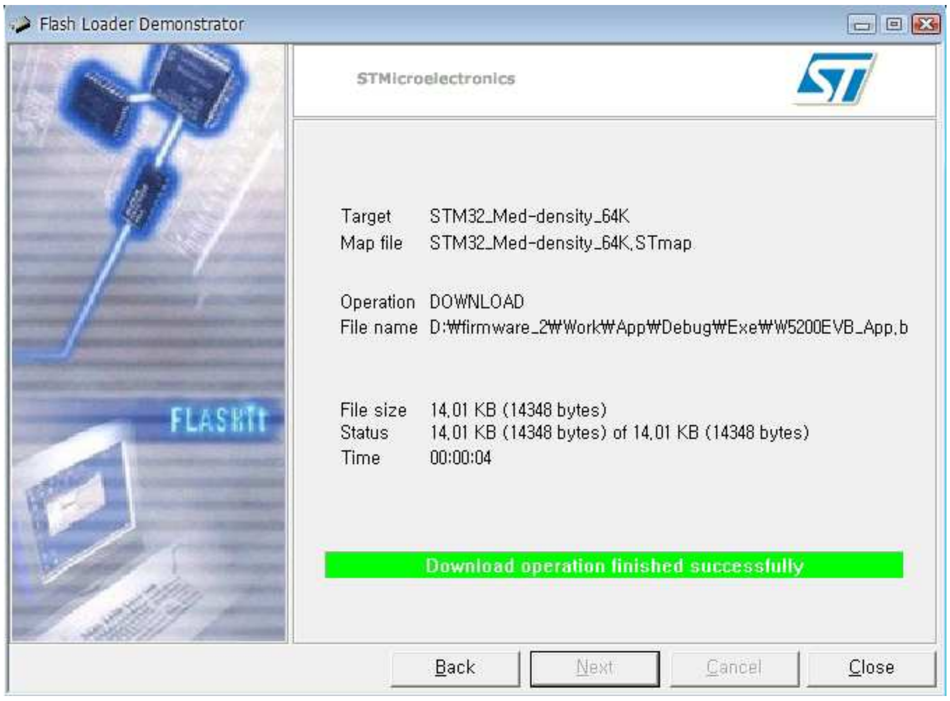
ENABLE / WRITE PROTECTION

Edit option bytes

Fig 3.7 Download to device

Start the download from @ 0x08000000 of the memory address.

User may use a different starting address for download if needed.



Flash Loader Demonstrator

STMicroelectronics

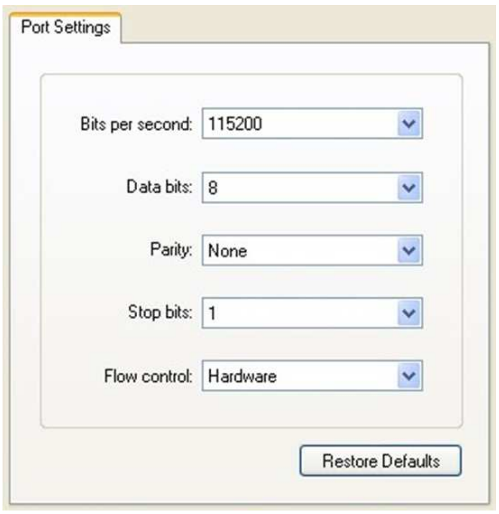
```
Target     STM32_Med-density_64K
Map file   STM32_Med-density_64K.STmap

Operation  DOWNLOAD
File name  D:₩firmware_2₩Work₩App₩Debug₩Exe₩W5200EVB_App.b

File size  14.01 KB (14348 bytes)
Status     14.01 KB (14348 bytes) of 14.01 KB (14348 bytes)
Time       00:00:04
```

Download operation finished successfully

Fig. 4.8 Download Completetion

4.1.5 Setting for Serial Terminal

Once the image is successfully loaded to W5200E01-M3, set the serial terminal to test the board. As mentioned earlier, Hyper Terminal, which is the most used serial terminal, will be used to check the board's serial message. Run the Hyper terminal to set the Port information for communication with the board. Click the [Properties] in File menu and select COM port. Press the [Configure] button. The screen will appear as shown in Figure 4.9.



Bits per second: 115200
Data bits: 8
Parity: None
Stop bits: 1
Flow control: Hardware

Fig. 4.9 Hyper terminal setting

Table. 4.2 Hyper terminal setting

Port Settings	Value
Bits/sec(Baud rate)	115200
Data bits	8
Parity	None
Stop bits	1
Flow control	Hardware

The Loopback test for w5200E01-M3 is ready after the settings in Table 4.2 Is done.

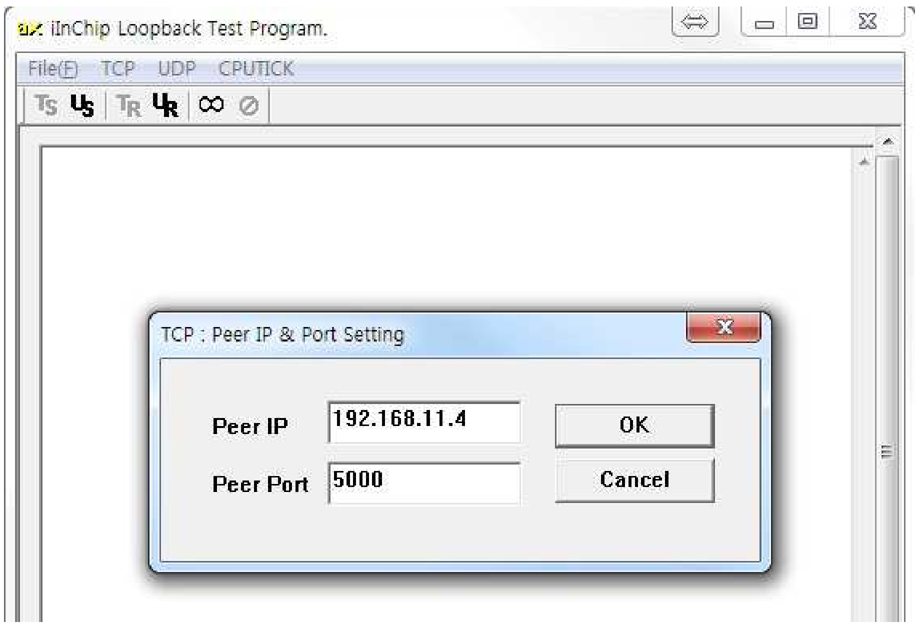
4.2 AX1

4.2.1 AX1 Setting for TCP Server

When W5200E01-M3 runs in server mode, use AX1 program in PC to connect to W5200E01-M3 as a TCP client. In the AX1, select the TCP => CONNCET menu for connecting to the W5200E01-M3, using peer IP address 192.168.11.4 and peer port 5000.

Once connection is successful, select TCP => SEND menu and send data as shown in Figure 4.10 Since W5200E01-M3 is the server, the AX1 program window will show the client's status.

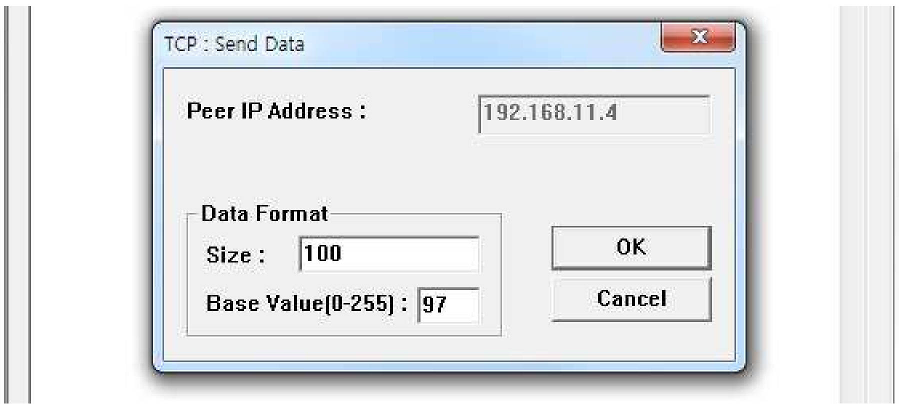
User can download AX1 program at WIZnet's website www.wiznet.co.kr.



Peer IP 192.168.11.4

Peer Port 5000

Fig 4.10 AX1 Peer IP & Port Setting



Peer IP Address : 192.168.11.4

Size : 100

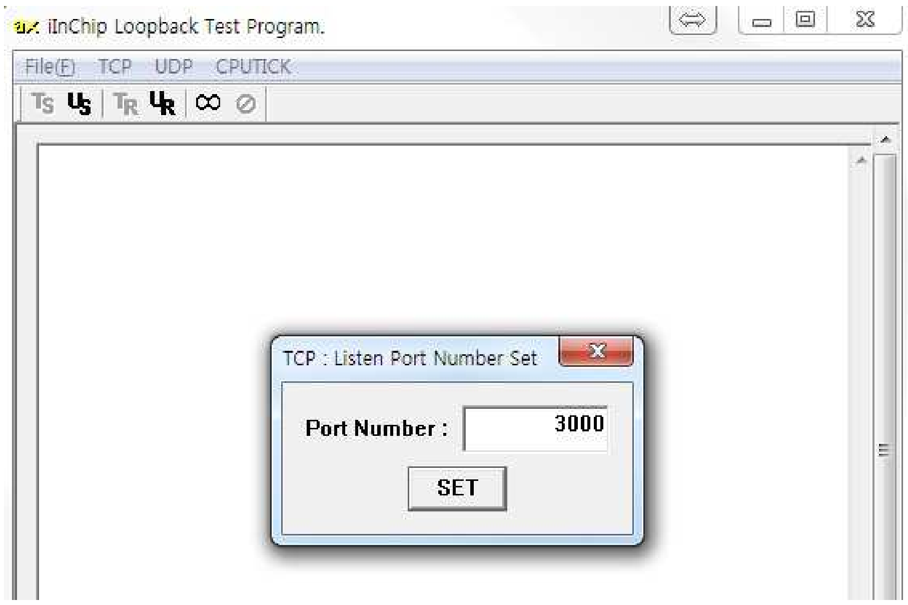
Base Value(0-255) : 97

Fig. 4.11 AX1 Send data

4.2.2 AX1 Setting for TCP Client

When W5200E01-M3 runs in client mode, users must use AX1 program in PC as a TCP server. Then, connect W5200E01-M3 to the server port that is set by the AX1 program. If the server port number is opened as 5000 in AX1 (listen state), W5200E01-M3 will connect to server IP address and server port (192.168.11.xxx, 5000). Note that the IP address must be the IP address of the PC that opened AX1.

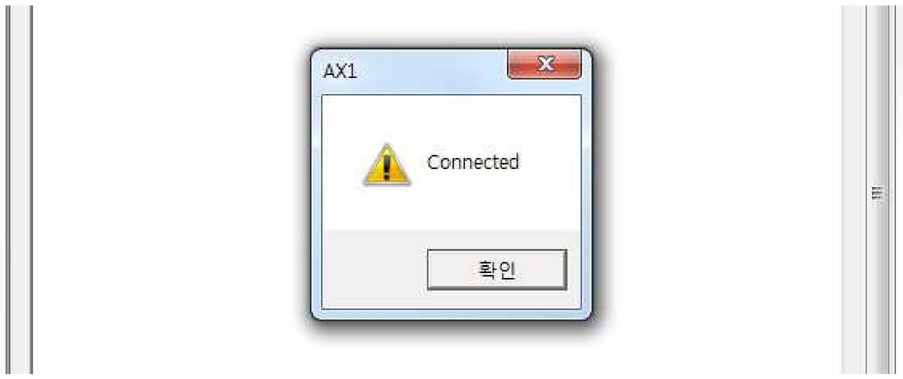
Once connection is successful, select TCP => SEND menu and send data as shown in Figure 4.13 since W5200E01-M3 is the client, the AX1 program window will show the server's status.



TCP : Listen Port Number Set

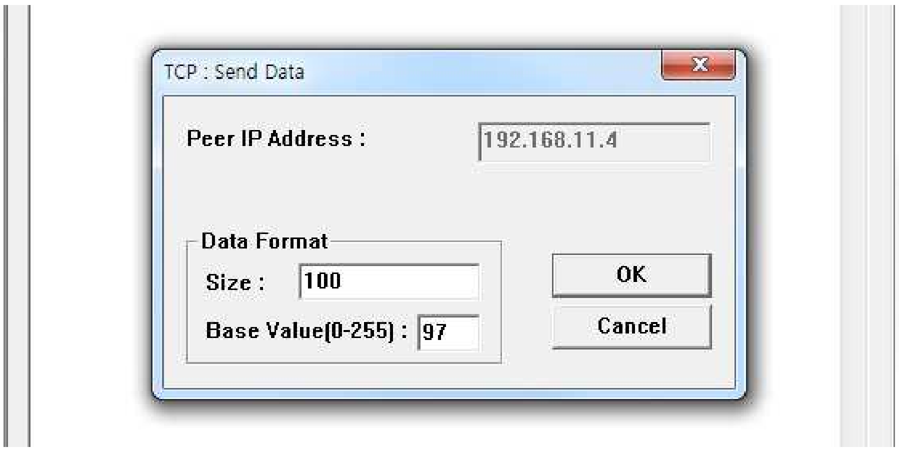
Port Number : 3000

Fig. 4.12 AX1 Port Number Setting for Listen



Connected

Fig. 4.13 AX1 TCP Client Connected


TCP : Send Data

Peer IP Address : 192.168.11.4

Data Format

Size : 100

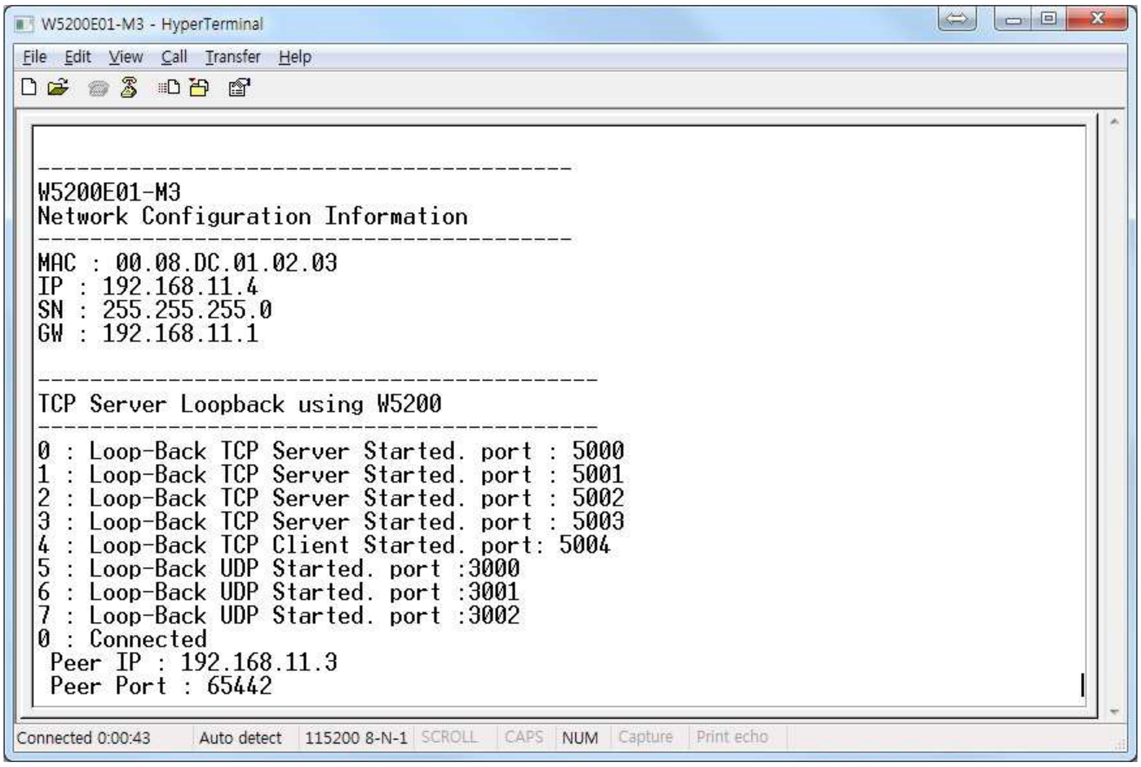
Base Value(0-255) : 97

Fig. 4.14 AX1 Send Data

4.3 Demonstration

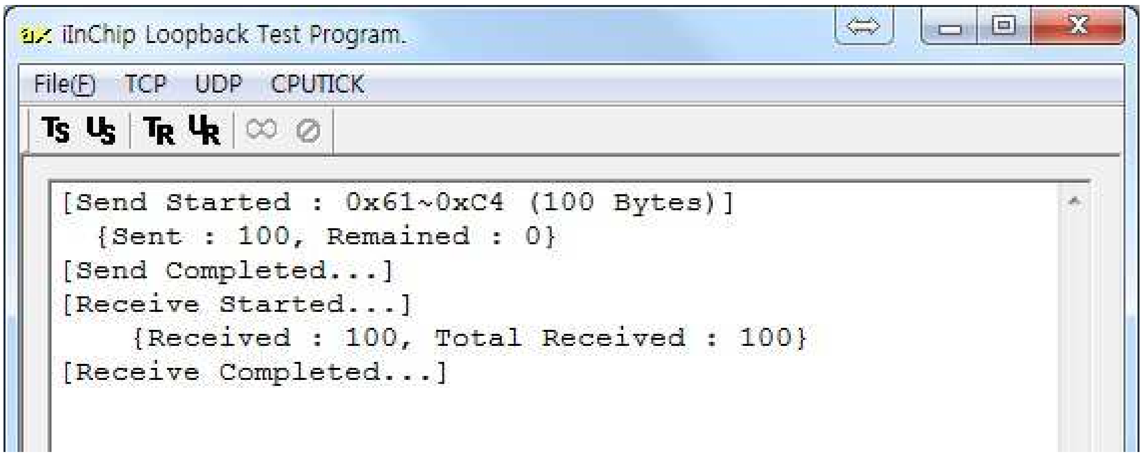
4.3.1 TCP Server Demonstration

After all settings for TCP server, click the TCP send. Then the AX1 shows process of the client PC as in the Fig. 4.16 it shows the size of SEND/RECEIVE data. Also the Hyper Terminal shows the process of iMCU7100EVB.


W5200E01-M3 - HyperTerminal
File Edit View Call Transfer Help
Connected 0:00:43 | Auto detect | 115200 8-N-1 | SCROLL | CAPS | NUM | Capture | Print echo

```
-----------------------------------------
W5200E01-M3
Network Configuration Information
-----------------------------------------
MAC : 00.08.DC.01.02.03
IP : 192.168.11.4
SN : 255.255.255.0
GW : 192.168.11.1

-------------------------------------------
TCP Server Loopback using W5200
-------------------------------------------
0 : Loop-Back TCP Server Started. port : 5000
1 : Loop-Back TCP Server Started. port : 5001
2 : Loop-Back TCP Server Started. port : 5002
3 : Loop-Back TCP Server Started. port : 5003
4 : Loop-Back TCP Client Started. port: 5004
5 : Loop-Back UDP Started. port :3000
6 : Loop-Back UDP Started. port :3001
7 : Loop-Back UDP Started. port :3002
0 : Connected
 Peer IP : 192.168.11.3
 Peer Port : 65442
```

Fig 4.15 TCP Server Demonstration (Hyper terminal)

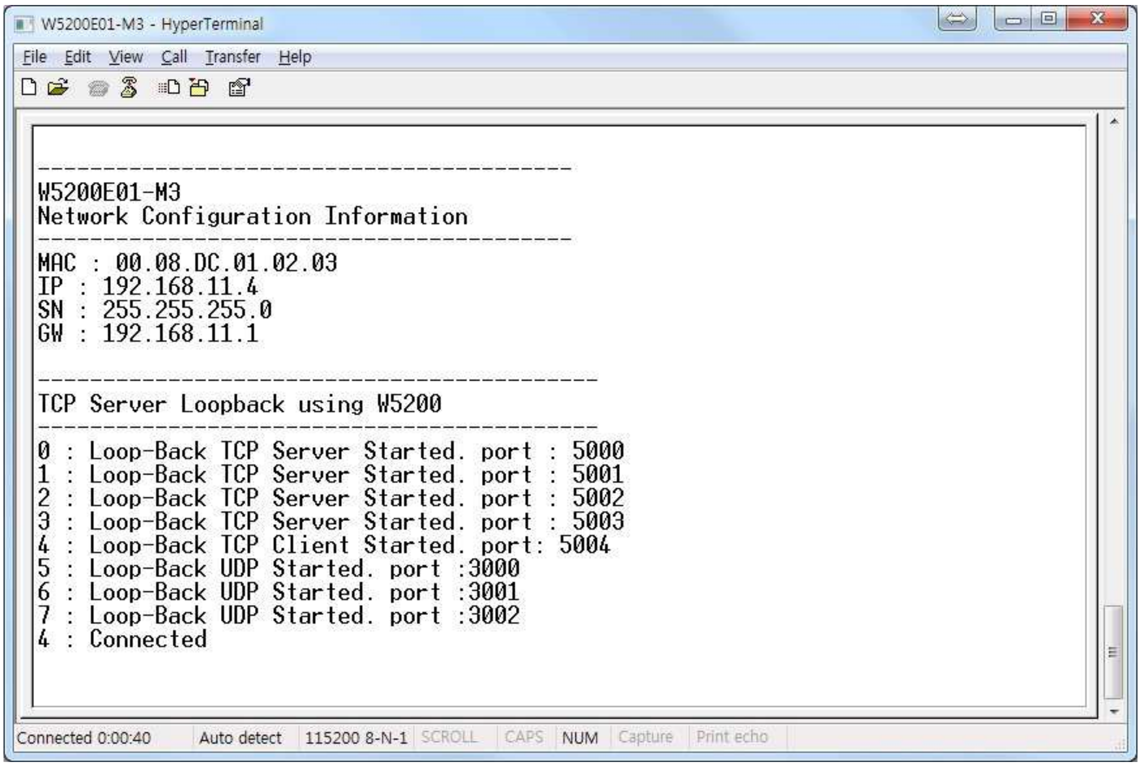


```
[Send Started : 0x61~0xC4 (100 Bytes)]
  {Sent : 100, Remained : 0}
[Send Completed...]
[Receive Started...]
    {Received : 100, Total Received : 100}
[Receive Completed...]
```

Fig.4.16 TCP Server Demonstration (AX1)

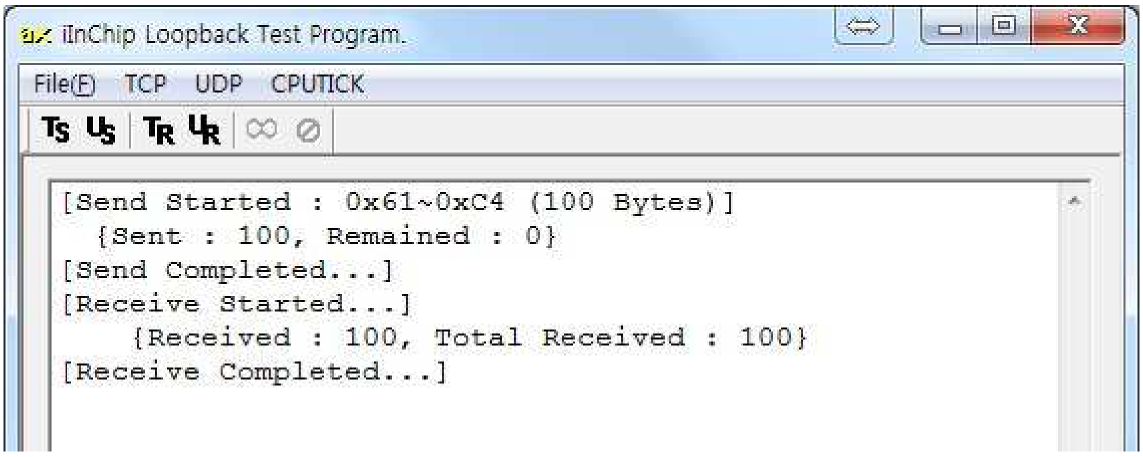
4.3.2 TCP Client Demonstration

After all settings for TCP client, click the TCP send. Then the AX1 shows process of the server PC as in the Fig. 4.18 It shows the size of SEND/RECEIVE data. Also the Hyper Terminal shows the process of the client iMCU7100EVB.



```
W5200E01-M3 - HyperTerminal

-----------------------------------------
W5200E01-M3
Network Configuration Information
-----------------------------------------
MAC : 00.08.DC.01.02.03
IP  : 192.168.11.4
SN  : 255.255.255.0
GW  : 192.168.11.1

-------------------------------------------
TCP Server Loopback using W5200
-------------------------------------------
0 : Loop-Back TCP Server Started. port : 5000
1 : Loop-Back TCP Server Started. port : 5001
2 : Loop-Back TCP Server Started. port : 5002
3 : Loop-Back TCP Server Started. port : 5003
4 : Loop-Back TCP Client Started. port: 5004
5 : Loop-Back UDP Started. port :3000
6 : Loop-Back UDP Started. port :3001
7 : Loop-Back UDP Started. port :3002
4 : Connected

Connected 0:00:40   Auto detect   115200 8-N-1   SCROLL   CAPS   NUM   Capture   Print echo
```

Fig 4.17 TCP Client Demonstration (Hyper terminal)



```
iInChip Loopback Test Program.

[Send Started : 0x61~0xC4 (100 Bytes)]
  {Sent : 100, Remained : 0}
[Send Completed...]
[Receive Started...]
    {Received : 100, Total Received : 100}
[Receive Completed...]
```

Fig. 4.18 TCP Client Demonstration (AX1)

Document History Information

Version	Date	Descriptions
Ver. 1.0	Apr. 2011	Release with W5200 launching