How to implement TCP for W7100A

version 1.1

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

Table of Contents

1 Introduction

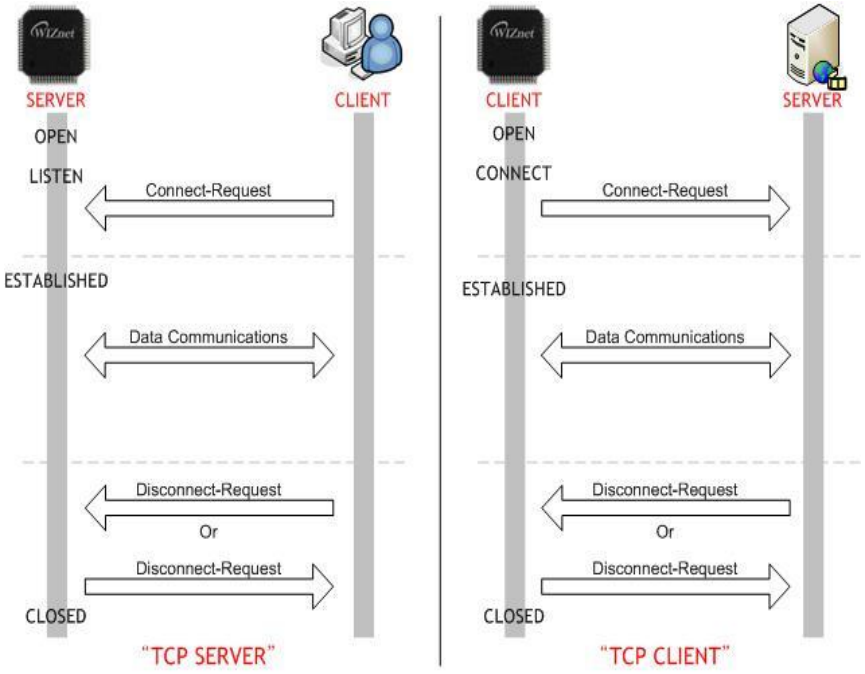
The TCP (Transmission Control Protocol) controls data communication between networks. As one of the main protocols that forms the internet, more details are written in RFC 793 of IETF (Internet Engineering Task Force). The TCP is a protocol that runs above IP, and therefore is written TCP. The TCP guarantees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

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

2 TCP SOCKET

User can use all eight SOCKETs provided by W7100 as TCP protocol. If the user wishes to use W7100 as TCP Protocol, at first the SOCKET that is going to be used should be created. When creating SOCKET, it needs SOCKET number, protocol, port number that is going to be used, and the flag that is going to be set. This document is going to explain about TCP protocol, and the protocol that is going to be used should be set the Sn_MR (SOCKET n Mode Register in TCPIPCore) 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 which supported by WIZnet.

Since the TCP protocol of W7100 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 Fig.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 is running in server mode, the SERVER will wait for CLIENT's connection request in LISTEN status. Otherwise, when TCP protocol is running 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 of the server mode is consisted of OPEN, LISTEN, SEND, RECEIVE, DISCONNECT, and CLOSE. The SOCKET lifecycle of the client mode is consisted of OPEN, CONNECT, SEND, RECEIVE, DISCONNECT, and CLOSE.

2.1 OPEN

The OPEN is the first step of creating a SOCKET for both SERVER and CLIENT mode. To create the SOCKETn (the n-1 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's 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; since there can be a destination port number already in use, it is best to choose a random port number and increase one number at a time until the SOCKET is connected. 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 'TCPIPCore.h'.

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

Method 1 : server mode

```
/* sets Protocol Number */
s = 0; // set SOCKET 0 (From 0 to 7)
/* 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 the SOCKETn, change the SOCKET to LISTEN status so that 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 a CLIENT. Once a 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. Requirements to connect are SOCKET number that is going to be used, destination IP, and destination port number. 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, 1, 2}; // set server(destination) IP
serverport = 0x5000; 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 complete before sending data. Set the SOCKET number, address of the data that is going to be sent, and data size, as user can check it from SEND() function. The address of the data that is going to be sent is usually set by selecting the area, put the data in, and set the area with pointer.

```
/* Send data to connected peer. */
// max_size_tx_buf must be smaller than the maximum size of the TX buffer
s = 0; //set SOCKET 0
* data_buf[max_size_tx_buf] = (uint8 *)0x7000; // set position of data buffer
```

```
len = 1460; //set length is 1460 Byte
send(s, (uint8 *)data_buf, len);
```

Example 2.4 SEND DATA

2.5 RECEIVE

RECEIVE is similar in usage method to SEND, but it has a checking the Sn_RX_RSR(n). The RECEIVE step is to move the data, that 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 */
s = 0; //set SOCKET 0
/*len indicates the received data size in the RX buffer. It must be smaller than the maximum size of the
RX buffer */
if ( (len = getSn_RX_RSR(s) ) > 0)
/* Received data */
//len is a length included the DATA packet.
* data_buf[max_size_tx_buf] = (uint8 *)0x7000; // set position of data buffer
len = recv(s, (uint8 *)data_buf, 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. 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 closing the SOCKET. When a disconnect request comes in, W7100 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 close 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 has changed 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

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, uint8 xdata * data_buf, uint16 mode)
{
  uint16 len;
  switch (getSn_SR(s))
        {
        case SOCK_ESTABLISHED: //if connection is established
                if ((len = getSn_RX_RSR(s)) > 0)   //check Rx data
                {
                        len = recv(s, data_buf, len);  //read the received data
                        send(s, data_buf, len);        //send the received data
                }
                break;
        case SOCK_CLOSE_WAIT:   //If the client request to close
               if ((len = getSn_RX_RSR(s)) > 0)   //check Rx data
                {
                        len = recv(s, data_buf, len);  //read the received data
                        send(s, data_buf, len);        //send the received data
                }
                disconnect(s);
                break;
        case SOCK_CLOSED:       //if a socket is closed
                close(s);
                socket(s,Sn_MR_TCP,port,0x00);
                break;
        case SOCK_INIT:         //if a socket is initiated
                listen(s);
                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 W7100 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.

3.2 Client mode

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

```
void loopback_tcpc(SOCKET s, uint16 port, uint8 xdata * data_buf, uint16 mode)
{
            uint16 xdata len;
            static uint16 xdata any_port = 1000;
            uint8 destip[4] = {192,168,1,69};            // Set the destination IP address

           switch(getSn_SR(s))
           {
              case SOCK_ESTABLISHED:                    // ESTABLISHED?
               if(getSn_IR(s) & Sn_IR_CON)          // check Sn_IR_CON bit
               {
                  if((len=getSn_RX_RSR(s)) > 0)      // check the size of received data
                  {
                      len = recv(s,data_buf,len);           // Receive the data
                      if(len !=send(s,data_buf,len))       // Send back the received data
                      {
                         PutHTOA(s);PutString(" : Send Fail.len = ");PutLTOA(len);PutString("");
                      }
                  }
               }
              break;
    case SOCK_CLOSE_WAIT:
        disconnect(s);                          // disconnect
```

```
        break;

    case SOCK_CLOSED:
        close(s);                                   // close the SOCKET
        // open the SOCKET with TCP mode and assign any source port number
        socket(s,Sn_MR_TCP,any_port++, mode);
              PutString("socket init OK!\r\n");
        break;

     case SOCK_INIT:                                 // The SOCKET opened with TCP mode
        connect(s, destip, port);                   // Try to connect to "TCP SERVER(destination IP)"
        PutHTOA(s);PutString(" : LOOPBACK_TCPC Started :");
        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.

4 LOOPBACK server Demonstration

In this section, the process for running an example code of TCP loopback will be explained. After downloading the binary file of the TCP loopback application, confirm the package of iMCUW7100EVB for the successful demonstration. Please refer to the 'How to make project in W7100', 'WizISP Program Guide' and 'W7100 Debugger Guide' for more information.

For the TCP loopback, user follows below steps.

- Confirm the testing environment. Refer to 'iMCU7100EVB User's Guide'

  Connect test PC to iMCU7100EVB by directly using UTP cable

  Connect test PC to iMCU7100EVB by directly using Serial cable

  Enable 5V power adapter to iMCU7100EVB

- Add the network information of Test PC as the following

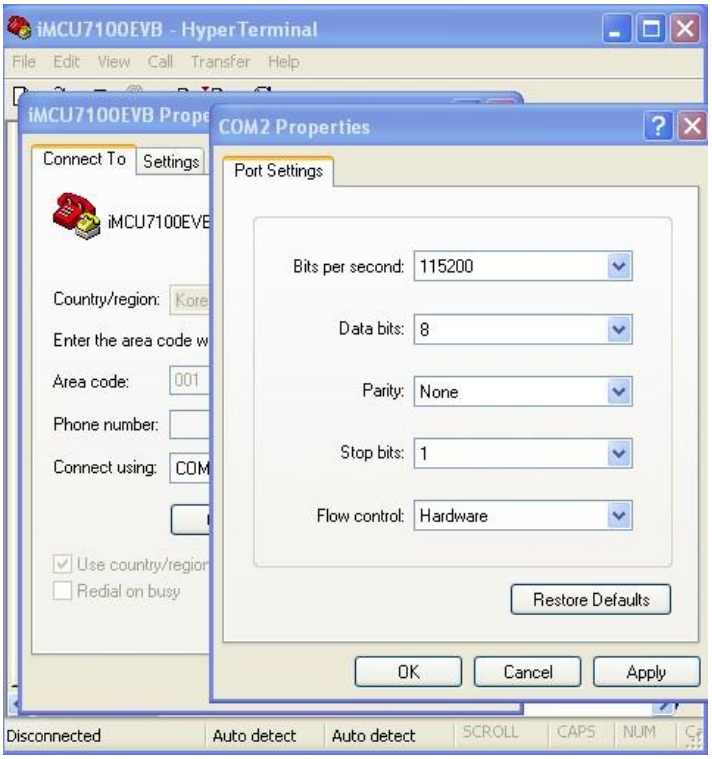
  - IP Address : 192.168.1.xxx ('xxx' is used for user's PC IP address, in the client mode, this IP address will be server's IP address which we set before to destip[4] in TCP client code), In this document, the IP sets 192.168.1.69 but user can modify it.

  - Gatway IP Address : 192.168.1.1

  - Subnet Mask : 255.255.255.0

- Run the Hyper Terminal and AX1 program

4.1 HyperTerminal



<Fig.4.1> Hyper terminal options

After running Hyper Terminal, set options for Serial communication as shown in Fig4.1. Hyper Terminal will show the running status of iMCU7100EVB using serial communication.

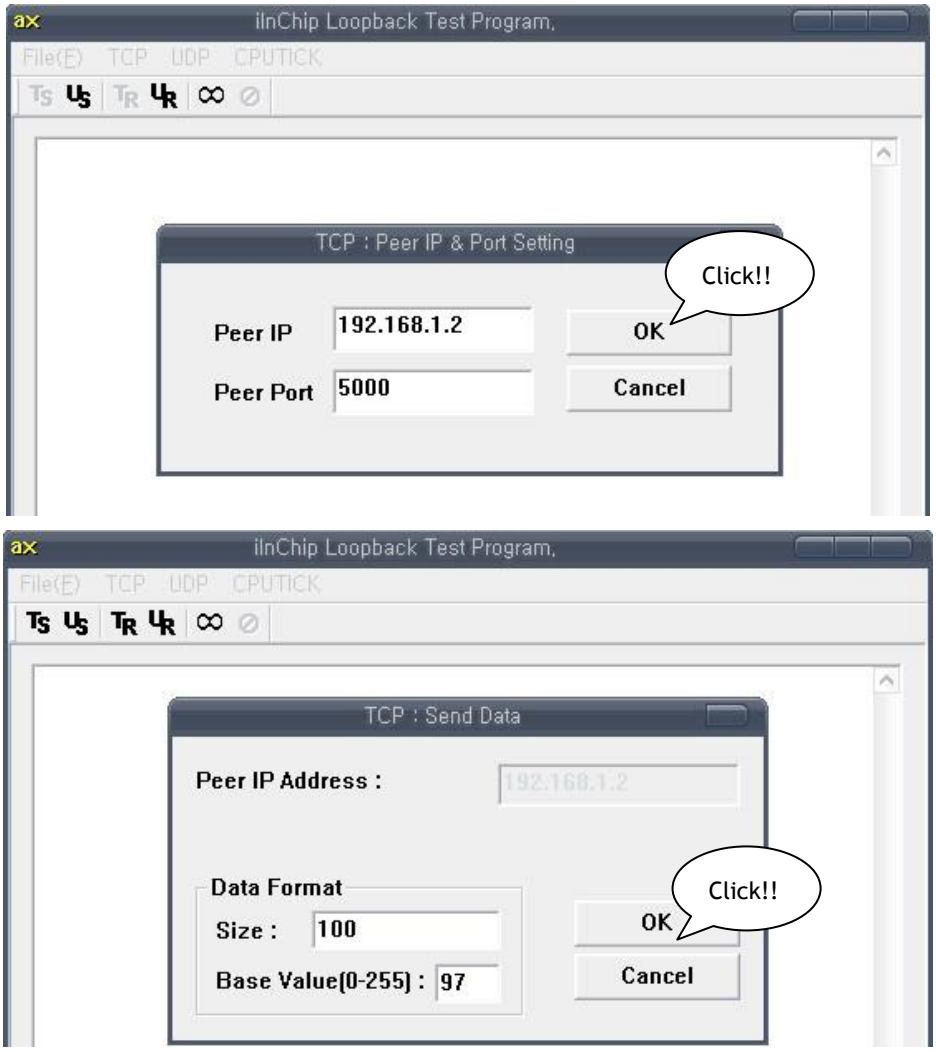
4.2 AX1 setting

For detail information about the AX1 program, please refer to the AX1 manual.

4.2.1 iMCU7100EVB in TCP server

When iMCU7100EVB runs in server mode, use AX1 program in PC to connect to iMCU7100EVB as a TCP client. In the AX1, select the TCP => CONNCET menu for connecting to the iMCU7100EVB, using peer IP address 192.168.1.2 and peer port 5000.

Once connection is successful, select TCP => SEND menu and send data as shown in Figure 4.2. Since iMCU7100EVB is the server, the AX1 program window will show the client's status.

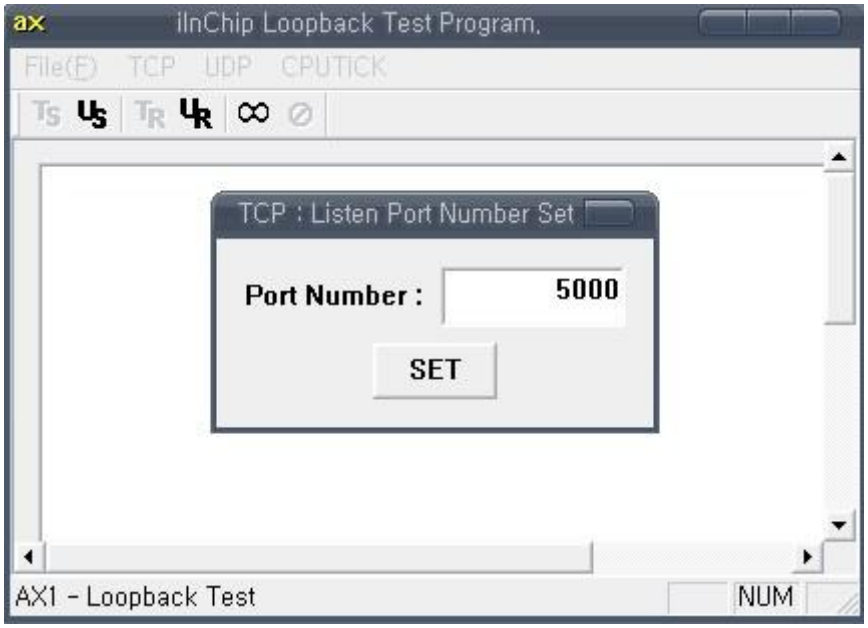


<Fig.4.2> AX1 Send data

4.2.2 iMCU7100EVB in TCP client

When iMCU7100EVB runs in client mode, use AX1 program in PC as a TCP server. Then connect iMCU7100EVB to the server port that is set by the AX1 program. If the server port number is opened as 5000 in AX1 (listen state), iMCU7100EVB will connect to server IP address and server port (192.168.1.xx, 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 4.2. Since iMCU7100EVB is the client, the AX1 program window will show the server's status.

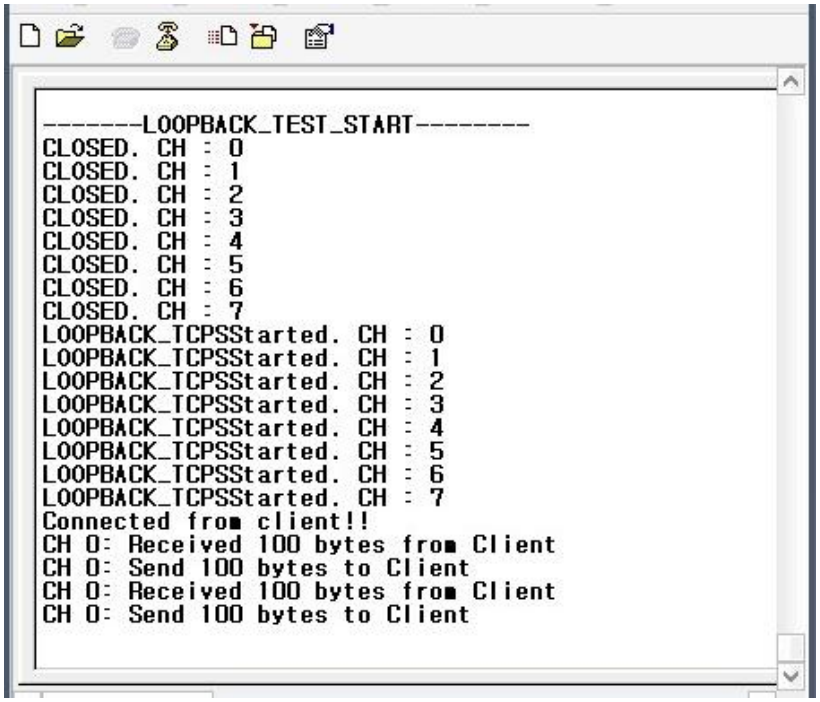


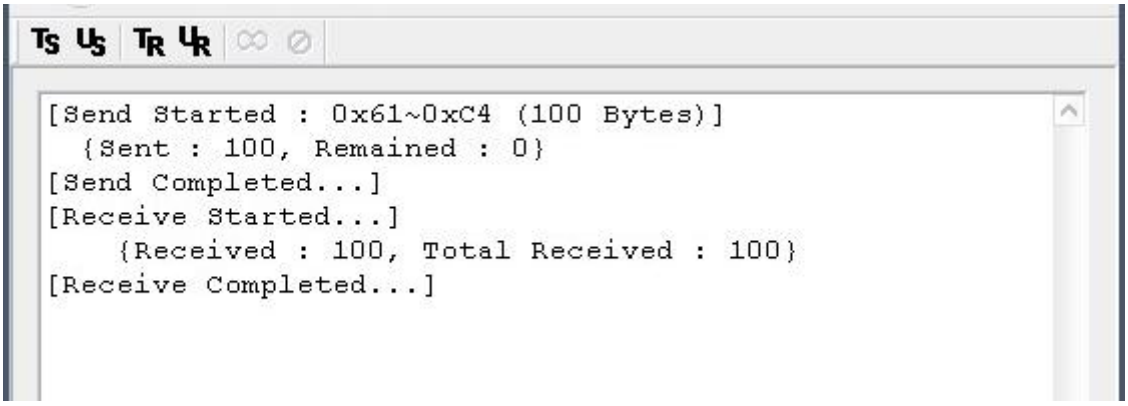
<Fig.4.3> AX1 Listen

4.3 TCP loopback Results

4.3.1 Result for TCP server

After all settings are done for TCP server, click TCP send. Then AX1 shows the process of the client PC as in the Figure 4.4. It shows the size of SEND/RECEIVE data. The Hyper Terminal shows the process of server iMCU7100EVB.

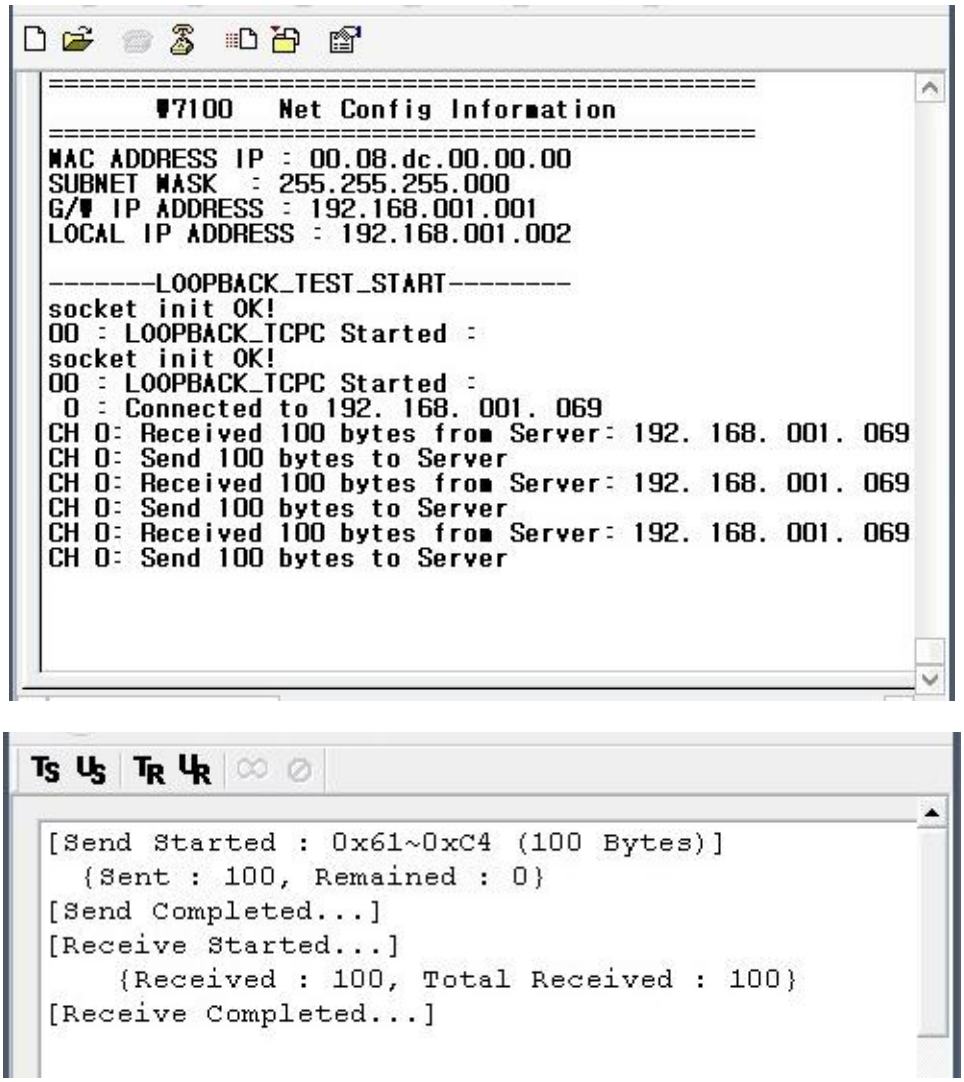




<Fig.4.4> Result of TCP server

4.3.2 Result for TCP client

After all settings are done for TCP client, click TCP send. Then AX1 shows process of the server PC as in the Figure 4.5. It shows the size of SEND/RECEIVE data. The Hyper Terminal shows the process of the client iMCU7100EVB



<Fig.4.5> Result of TCP client

Document History Information

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