

How to implement UPnP (Universal Plug and Play) Port Forwarding in W5200E01-M3

Version 0.9beta



© 2011 WIZnet Co., Inc. All Rights Reserved.

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

Table of Contents

1. Introduction.....	3
1.1 Concept.....	3
1.2 UPnP structure and Steps involved in UPnP.....	3
2. Port Forwarding and UPnP.....	6
3. Port forwarding and W5200.....	7
4. Pre-settings.....	16
4.1 Limitations.....	16
4.2 Settings.....	16
5. Example of Use.....	16
6. Reference.....	19
Document History Information.....	19

Abstract

This application note provides technical information on UPnP support in WIZnet W5200 chip and W5200-based modules. First, a general introduction will explain what UPnP is exactly. Secondly, the Port Forwarding concept as defined by the UPnP working group is made clear. Finally, this application note will show you on how W5200 add port mapping and delete port mapping.

1. Introduction

1.1 The Concept

What is UPnP?

The motivator to develop UPnP came from Plug and Play concept. Plug and play is a term used to describe the characteristic of a computer bus, or device specification, which facilitates the discovery of a hardware component in a system, without the need for physical device configuration, or user intervention in resolving resource conflicts [1][2]. The idea is: just plug in the device and it is ready to use [3].

UPnP now creatively apply the Plug and Play concept to the networking environment. UPnP is designed to support zero-configuration, "invisible" networking, and automatic discovery for a breadth of device categories from a wide range of vendors. This means that a device can dynamically join a network, obtain an IP address, convey its capabilities, and learn about the presence and capabilities of other devices—all automatically; truly enabling zero configuration networks [4].

1.2 UPnP structure and Steps involved in UPnP

UPnP structure

TCP is Client and Server structure. Similarly, the UPnP structure is based on Devices and Control Points:

➔ Devices:

- Offer services

For example, UPnP DVD player is a device which can offer DVD playing service.

- Record their status.

For example, DVD player can record the DVD playing status.

➔ Control Points:

- Control the devices with defined so as to control the service it is offering.

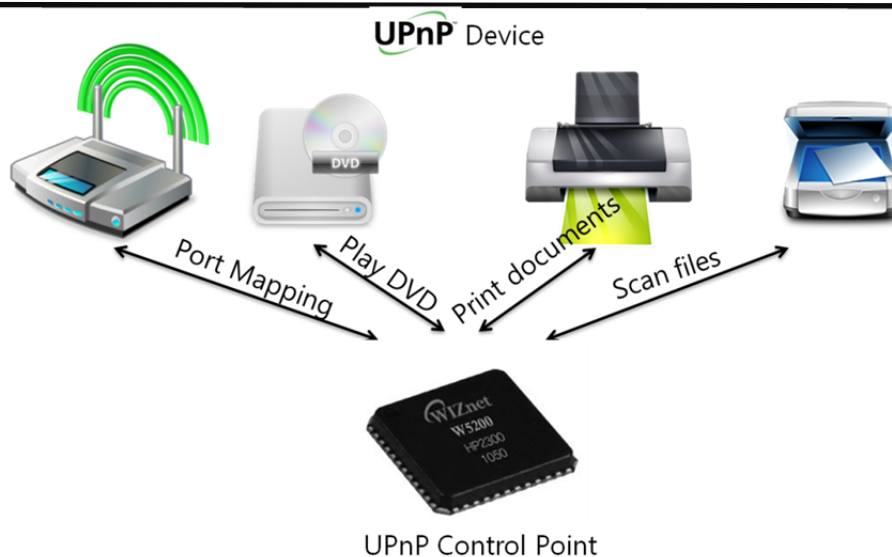


Figure 1. UPnP network

To make all the descriptions come true, Universal Plug and Play mainly uses existing standards such as TCP/IP, DHCP, XML, ..., which makes the concept universal.

Steps involved in UPnP networking

There are 6 different steps in the UPnP mechanism:

① Addressing

Addressing is the process through which control points and devices obtain a network address. They first try to obtain an IP address from a DHCP server; if none is available, an AutoIP address in the 169.254 subnet is randomly chosen.

Note: In Addressing step, both AutoIP and DHCP can help UPnP control point and device to get IP address. But DHCP is more popular and reliable. Thus, in this application note and our W5200-based modules, DHCP is the only way to get IP address.

② Discovery

Discovery allows control points to find devices that are of interest to them.

When control points enter the network, they broadcast search packets to look for devices and/or services, either in general or of a specific type. Devices featuring the appropriate services/subdevices can then respond.

Similarly, UPnP devices will advertise their presence on the network at regular intervals. Control points listen for these advertisement packets or discovery packets to detect new devices and their capabilities as they become present on the network.

UPnP devices leaving the network also send notifications that their services will no longer be available.

Note: In Discovering step, both Searching and Advertising can help UPnP control point to find UPnP device. Given the similarity of two methods, this application note and our W5200-

based modules retained Searching method. Advertising will be ignored.

③ Description

The discovery packets sent out by the devices refer the control points to a location where they can retrieve a Device Description Document (DDD). This DDD contains:

- ➔ A summary of the device's embedded devices and a list of services.
- ➔ A URL for the so called Service Control Protocol Definition (SCPD). This SCPD describes how the control points can make use of the services offered by the device.
- ➔ Control and Eventing URLs: these are the URLs to which the control points have to send commands in order to configure the UPnP device and make use of the device's services.
- ➔ A URL for presentation (see step 6).
- ➔

④ Control


Control allows control points to send commands to devices. As mentioned before, the URL to which these commands are sent is specified in the DDD.

⑤ Eventing

Eventing allows control points to track state changes in devices. Control points first subscribe to the appropriate service. Subsequently, any state changes in the device's service are sent as events to the subscribed control points to keep them up to date.

⑥ Presentation

Control points can optionally display a user interface for devices. The URL for presentation is specified within the DDD. The presentation page shows an HTML-based user interface so that a user can consult and/or consult the device's status. As such, presentation is complementary to control and eventing.

 **Note: W5200 is playing a role of UPnP control point which not really needs to embed a webserver which is necessary in UPnP device. Thus, in this application note and our W5200-based modules, presentation is not supported.**

2. Port forwarding and UPnP

Port forwarding

Simply speaking, Port forwarding (another saying is NAT Traversal) functionality allows creation of mappings for both TCP and UDP protocols between an external Internet Gateway Device (IGD) port (called ExternalPort) and an internal client address associated with one of its ports (respectively called InternalClient and InternalPort).

Please see below figure to understand the application of Port forwarding:

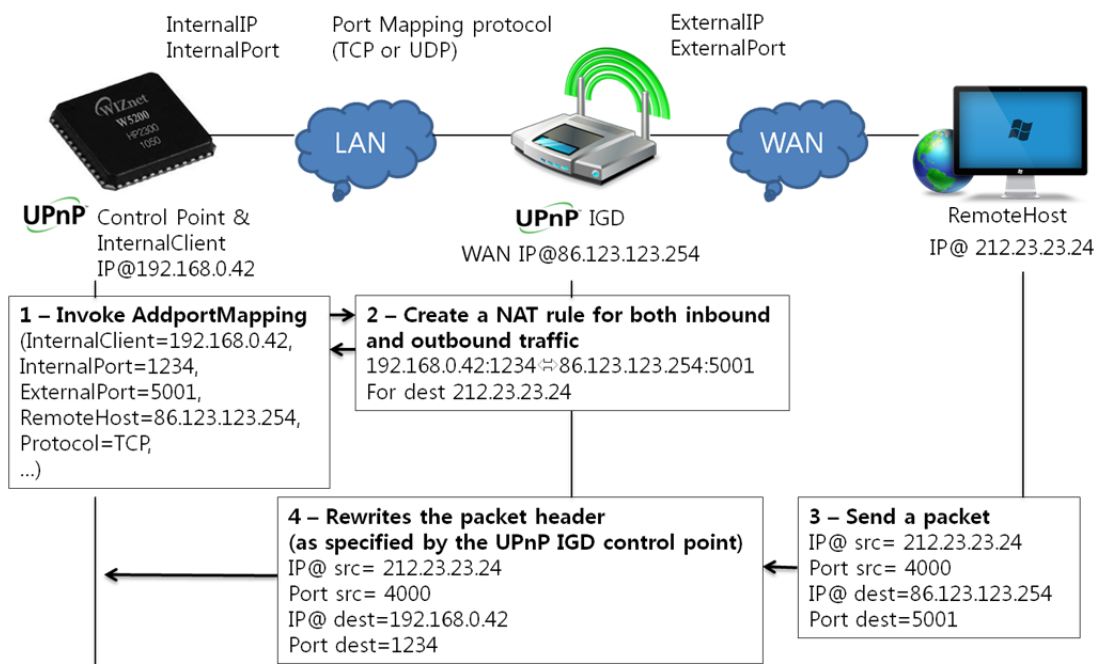


Figure 2. Topology of Port Forwarding application

UPnP and Port Forwarding

Port Forwarding is one of the basic functions of IGD (More standard functions of IGD can be found in UPnP IGD service template). It can be done manually. However, by using UPnP this function can be done silently. We can say UPnP makes Port Forwarding transparent for users.

Currently, many kinds of P2P software support UPnP Port Forwarding, such as Skype, uTorrent and MSN. If you are curious about UPnP, you can login your IGD setting page and find the Port Forwarding list (see Figure 4). You will see all the port mappings. I am sure most of them were not done by you, but by UPnP.

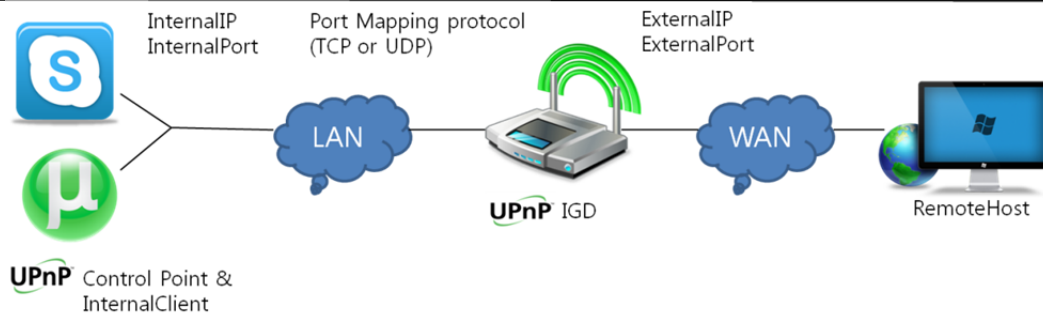
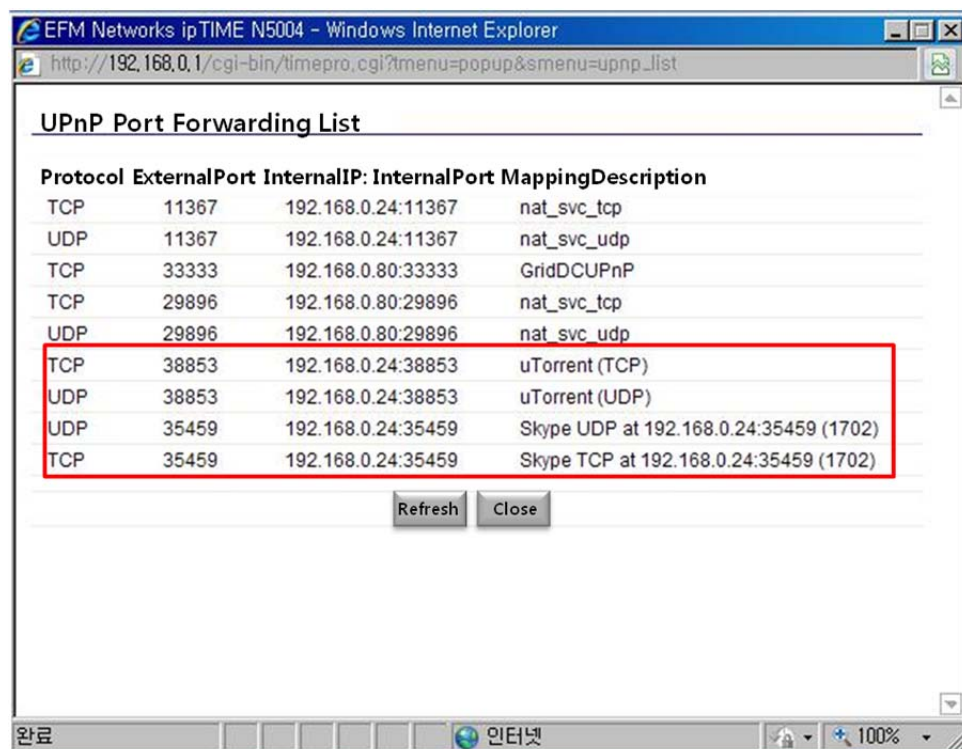


Figure 3. P2P software-based UPnP network



Protocol	ExternalPort	InternalIP: InternalPort	MappingDescription
TCP	11367	192.168.0.24:11367	nat_svc_tcp
UDP	11367	192.168.0.24:11367	nat_svc_udp
TCP	33333	192.168.0.80:33333	GridDCUPnP
TCP	29896	192.168.0.80:29896	nat_svc_tcp
UDP	29896	192.168.0.80:29896	nat_svc_udp
TCP	38853	192.168.0.24:38853	uTorrent (TCP)
UDP	38853	192.168.0.24:38853	uTorrent (UDP)
UDP	35459	192.168.0.24:35459	Skype UDP at 192.168.0.24:35459 (1702)
TCP	35459	192.168.0.24:35459	Skype TCP at 192.168.0.24:35459 (1702)

Figure 4. UPnP port mapping list

3. Port forwarding and W5200

Working flow

This application note focuses on the application of port forwarding via UPnP by using W5200. In port forwarding, W5200 is the control point of IGD, which edits port forwarding. Also, W5200 is the LAN client which needs the port forwarding. IGD executes the port forwarding.

Trough aforementioned we know, control point W5200 can send commands to IGD to implement the port forwarding function. All the commands came from the IGD standard which was defined by the Internet Gateway Working Committee, a committee that was established by the UPnP. Basically, Port forwarding consists of two steps, one is adding port mapping, the other one is deleting port mapping. In addition, during adding and deleting port mapping, there are some exceptions that need W5200 to process. For example:

➔ ConflictInMappingEntry: The port mapping entry specified conflicts with a mapping assigned

previously to another client.

➔ More exceptions can be found in IGD service template.[5]

This application note will show you on how W5200 add port mapping and delete port mapping. Below figure shows the UPnP Port Forwarding process of W5200

Note: For more information about <Step0> Addressing, refer to *How to implement DHCP in W5200*.

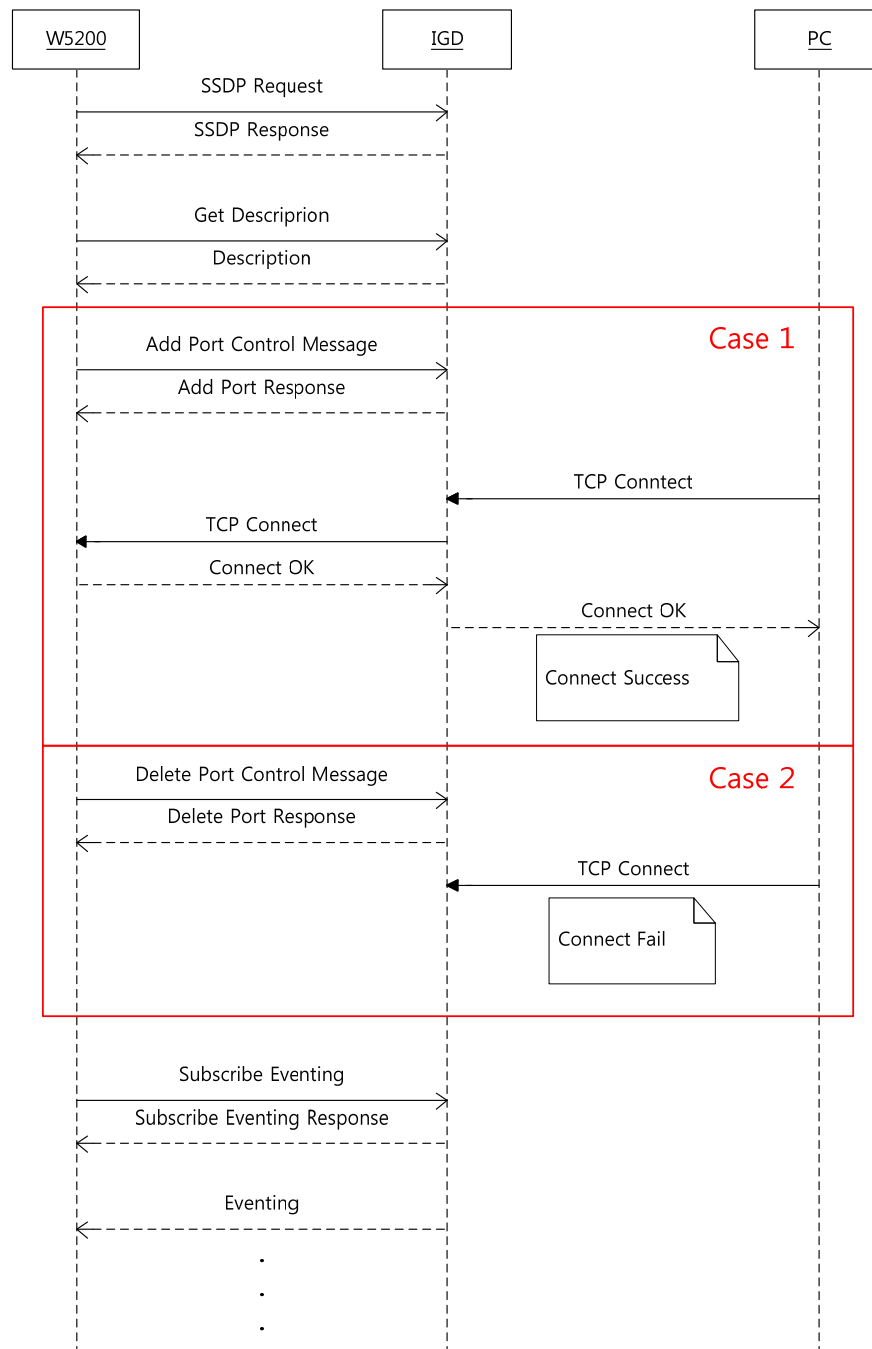


Figure 5. UPnP Port Forwarding process of W5200

1. Phase1

A. SSDP

W5200 must send SSDP M-SEARCH message using UDP multicasting address in order to discover the IGD that is in the same subnet with W5200.

Note: SSDP is just using multicasting address to send data. In terms of socket, it is still using general socket. For example:

1) In real multicasting, W5200 should open socket like this:

```
socket(sock_id, Sn_MR_UDP, MULTICAST_PORT, Sn_MR_MULTICAST);
```

2) In SSDP, W5200 still create general socket:

```
socket(sock_id, Sn_MR_UDP, MULTICAST_PORT, 0)
```

In this application note, all multicasting related description belongs to 2).

SSDP Response will let W5200 know IGD's IP Address, Port Number and the Description URL.

```
M-SEARCH * HTTP/1.1WrWn
Host:239.255.255.250:1900WrWn
ST:urn:schemas-upnp-org:device:InternetGatewayDevice:1WrWn
Man:"ssdp:discover"WrWn
MX:3WrWn
WrWn
```

Below source code is implemented in SSDPProcess() function.

```
{
    Initialize a multicasting socket (UDP, Destination IP is 239.255.255.250, Destination
    Port is 1900, Destination MAC is 0x01:0x00:0x5E:0x7F:0xFF:0xFA)

    // Send SSDP
    sendto(sockfd, SSDP, strlen(SSDP), mcast_addr, 1900);

    // Receive Reply
    while
    {
        if(overtime){
            close a multicasting socket
            return -1;
        } else if(receive buffer is not empty){
            receive the SSDP response from IGD to rcv_buffer
        }
    }
    Close a multicasting socket
}
```

```
// Parse SSDP Message
return parseSSDP(recv_buffer);
}
```

2. Phase2

A. Get Description of IGD service

Complete the HTTP GET Header by using the IGD's IP Address, Port Number and Description URL and then send HTTP Get Header to IGD.

When IGD received the HTTP GET Header, IGD will let W5200 know its Description.

This Description will let W5200 know its Control URL and Eventing Subscription URL.

```
GET destURL HTTP/1.1\r\n
Accept: text/xml, application/xml\r\n
User-Agent: Mozilla/4.0 (compatible; UPnP/1.0; Windows NT/5.1)\r\n";
Host: destIP:destPORT\r\n
Connection: Keep-Alive\r\n
Cache-Control: no-cache\r\n
Pragma: no-cache\r\n
\r\n
```

Below source code is implemented in GetDescriptionProcess() function.

```
{
    // Make HTTP GET Header
    MakeGETHeader(send_buffer);

    Initialize a unicasting socket

    // Connect to IGD(Internet Gateway Device)
    connect(sockfd, ipaddr of IGD, port of IGD);

    wait while connecting to IGD

    // Send Get Discription Message
    send(sockfd, send_buffer, strlen(send_buffer), FALSE);

    // Receive Reply
    while
```

```

{
    if(overtime){
        close a unicasting socket
        return -1;
    } else if(receive buffer is not empty){
        receive the SSDP response from IGD to recv_buffer
    }
}
close a unicasting socket

// Parse Discription Message
return parseDescription(recv_buffer);
}

```

3. Phase3

A. Case 1: Add Port Control

Complete the XML by using the IGD's IP Address, Port Number and Control URL and then to implement the AddPortMapping action by using HTTP POST method-based SOAP.

Below source code is implemented in AddPortProcess() function.

```

{
    // Make "Add Port" XML(SOAP)
    MakeSOAPAddControl(content, protocol, external_port, internal_ip, internal_port,
description);

    // Make HTTP POST Header
    MakePOSTHeader(send_buffer, strlen(content), ADD_PORT);
    strcat(send_buffer, content);

    Initialize a unicasting socket

    // Connect to IGD(Internet Gateway Device)
    connect(sockfd, ipaddr of IGD, port of IGD);

    wait while connecting to IGD

    // Send "Add Port" Message
    send(sockfd, send_buffer, strlen(send_buffer), FALSE);
}

```

```

// Receive Reply
while
{
    if(overtime){
        close a unicasting socket
        return -1;
    } else if(receive buffer is not empty){
        receive the SSDP response from IGD to recv_buffer
    }
}
close a unicasting socket

// Parse Replied Message
return parseAddPort(recv_buffer);
}

```

B. Case 2: Delete Port Control

Complete the XML by using the IGD's IP Address, Port Number and Control URL and then to implement the DeletePortMapping action by using HTTP POST method-based SOAP.

Below source code is implemented in DeletePortProcess() function.

```

{
    // Make "Delete Port" XML(SOAP)
    MakeSOAPDeleteControl(content, protocol, external_port);

    // Make HTTP POST Header
    MakePOSTHeader(send_buffer, strlen(content), DELETE_PORT);
    strcat(send_buffer, content);

    Initialize a unicasting socket

    // Connect to IGD(Internet Gateway Device)
    connect(sockfd, ipaddr of IGD, port of IGD);

    wait while connecting to IGD

    // Send "Delete Port" Message

```

```

send(sockfd, send_buffer, strlen(send_buffer), FALSE);

// Receive Reply
while
{
    if(overtime){
        close a unicasting socket
        return -1;
    } else if(receive buffer is not empty){
        receive the SSDP response from IGD to recv_buffer
    }
}
close a unicasting socket

// Parse Replied Message
return parseDeletePort(recv_buffer);
}

```

C. Test Case 1 and Case 2

In order to check whether UPnP Port Forwarding can normally work or not, users can run a TCP client in remote PC and then let that TCP client connect to W5200 (TCP server). Users can refer to *section 5 Example of use* to learn how to verify AddPortMapping and DeletePortMapping.

Below source code is implemented in tcp_test() function.

```

{
    switch (getSn_SR(sockfd))
    {
        case SOCK_ESTABLISHED: /* if connection is established */
            if(receive buffer is not empty){
                recv(sockfd, recv_buffer, len);
            }
            break;

        case SOCK_CLOSE_WAIT: /* If the client request to close */
            if (receive buffer is not empty)
            {
                recv(sockfd, recv_buffer, len);
            }
    }
}

```

```

        disconnect(sockfd);

        break;

        case SOCK_CLOSED: /* if a socket is closed */
            reinitialize a TCP socket
            listen
            break;

    }
}

```

4. Phase4

A. Subscribe Eventing (Optional)

In order to subscribe the eventing, first of all, W5200 should complete the GENA message by using the IGD's IP Address, Port Number and Eventing Subscription URL and then send the GENA message to IGD.

```

SUBSCRIBE eventSubURL HTTP/1.1WrWn
Host: destIP:destPORTWrWn
USER-AGENT: Mozilla/4.0 (compatible; UPnP/1.1; Windows NT/5.1)WrWn
CALLBACK: <http://myIP:listen_port/>WrWn
NT: upnp:eventWrWn
TIMEOUT: Second-1800WrWn
WrWn

```

Below source code is implemented in SetEventing() function.

```

{

    // Make Subscription message
    MakeSubscribe(send_buffer, listen_port);

    Initialize a unicasting socket

    // Connect to IGD(Internet Gateway Device)
    connect(sockfd, ipaddr of IGD, port of IGD);

    wait while connecting to IGD

    // Send Subscription Message
    send(sockfd, send_buffer, strlen(send_buffer), FALSE);
}

```

```

// Receive Reply
while
{
    if(overtime){
        close a unicasting socket
        return -1;
    } else if(receive buffer is not empty){
        receive the SSDP response from IGD to rcv_buffer
    }
}
close a unicasting socket

return 0;
}

```

B. Receive Eventing

In order to receive the Eventing from IGD, W5200 should create a TCP server port which is responsible for listening the Eventing. When W5200 received the Eventing message, the function named `parseEventing()` will make the message human readable and display it through serial port.

Below source code is implemented in `eventing_listener()` function.

```

{
    switch (getSn_SR(sockfd))
    {
        case SOCK_ESTABLISHED: /* if connection is established */
            if(receive buffer is not empty){
                rcv(sockfd, rcv_buffer, len);
                send(sockfd, HTTP_OK, strlen(HTTP_OK), FALSE);
                // Parse Replied Message
                parseEventing(rcv_buffer);
            }
            break;

        case SOCK_CLOSE_WAIT: /* If the client request to close */
            if (receive buffer is not empty)
            {

```

```

                                recv(sockfd, recv_buffer, len);
                                }
                                disconnect(sockfd);
                                break;

                                case SOCK_CLOSED: /* if a socket is closed */
                                    reinitialize a TCP socket
                                    listen
                                    break;
                                }
        }
    }
}

```

4. Pre-Settings

4.1 Limitations

- We don't support AutoIP. In this application note and our W5200-based modules, DHCP is the only way to get IP address.
- We don't support device advertising. In this application note and our W5200-based modules, broadcast searching is the only way to discover UPnP device. Also, leaving notification cannot be recognized.
- W5200 is playing a role of UPnP control point which not really needs to embed a webserver which is necessary in UPnP device. In this application note and our W5200-based modules, presentation is not supported.

4.2 Settings

IGD must enable DHCP and UPnP functions in advance.

5. Example of use

Users must follow below network structure to implement UPnP Port Forwarding.

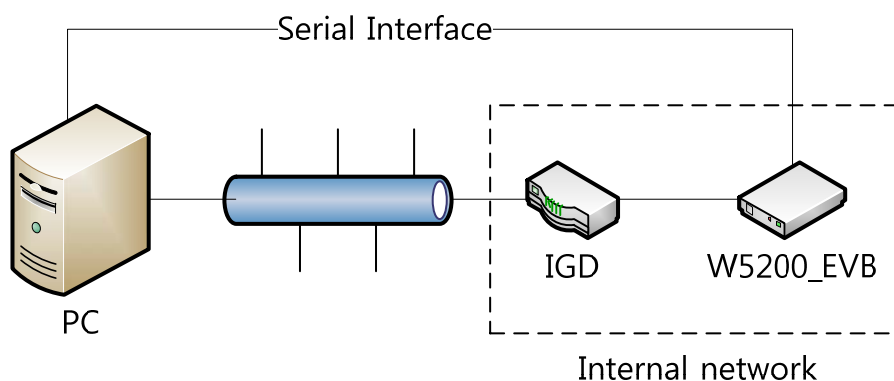


Figure 6. Network topology to implement UPnP Port Forwarding

Note: By default, this app note adds the port mapping of Internal IP: 4321 ↔ External IP:

1234.

From Figure 5 we can see that the process of this application is as follows:

(10 sec)

DHCP → SSDP → Get Description → AddPortMapping----->DeletePortMapping

“10 sec” means that after AddPortMapping, users have 10sec available to do testing. 10 sec later, DeletePortMapping will be done automatically, which is followed by related Eventing information. As said before, for this testing, W5200 (TCP server) keeps listening a socket (internal port 4321). Thus, if connection with that added external port can be established, AddPortMapping is successful. 10sec later, if connection failed, DeletePortMapping is successful.

Follow below step 1~3 orderly to implement UPnP Port Forwarding.

1. Upload the firmware of App Note to W5200_EVB (for uploading method, refer to W5200_EVB User Guide)
2. Run Serial Terminal and then open the COM that is connected with W5200_EVB.
(Baud Rate: 115200, Data Bit: 8bit, Parity Bit: none, Stop Bit: 1bit, Flow Control: XON/XOFF)
3. Reset W5200_EVB and then Serial Terminal will let users know the running status of app note. Message “AddPort Success!!” means that action of AddPortMapping is successfully done.

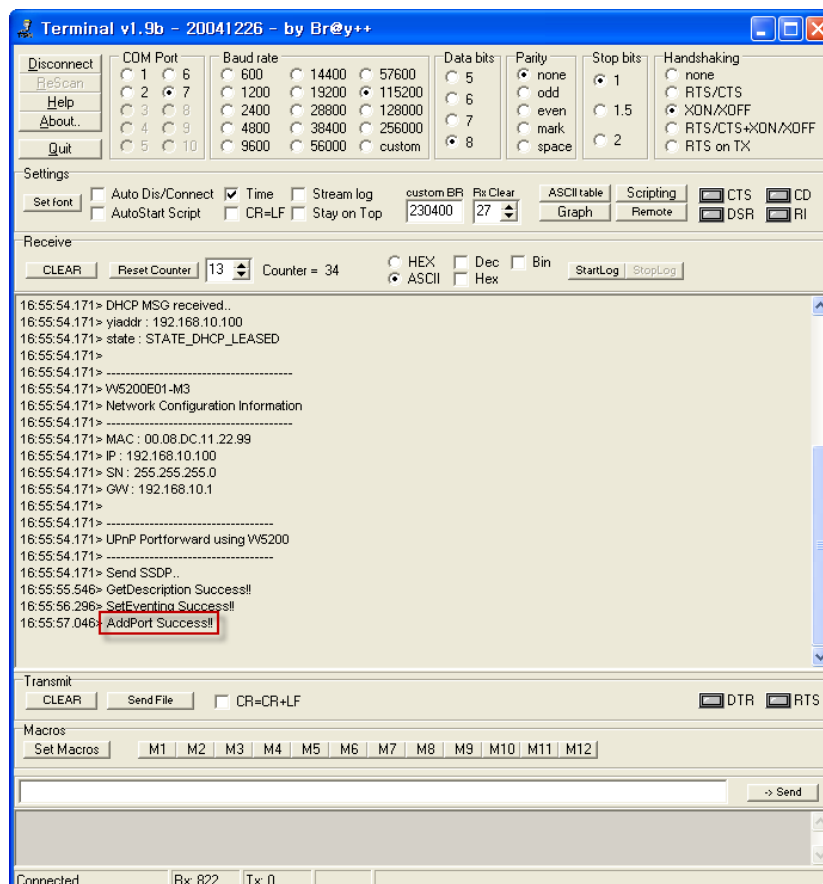


Figure 7. AddPortMapping OK

- By default, this app note adds the port mapping of **Internal IP: 4321** ↔ **External IP: 1234**. Users can verify this point using AX1.

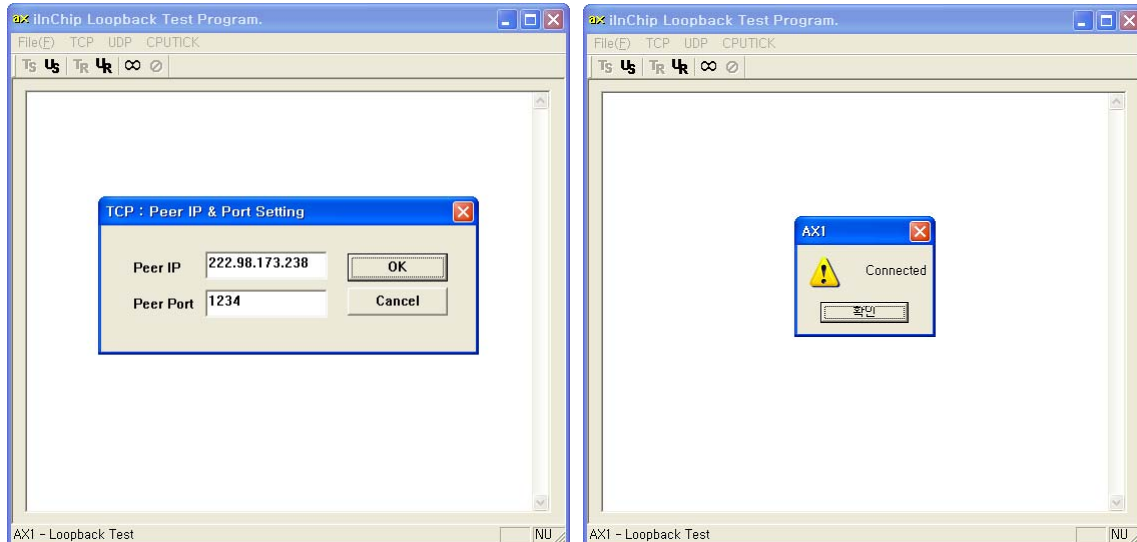


Figure 8. Run TCP client in remote host and let it connect to IGD (added external port)

- By default, the added mapping entry will be automatically deleted 10 seconds later. At that moment, serial message "DeletePort Success!!" will be seen.

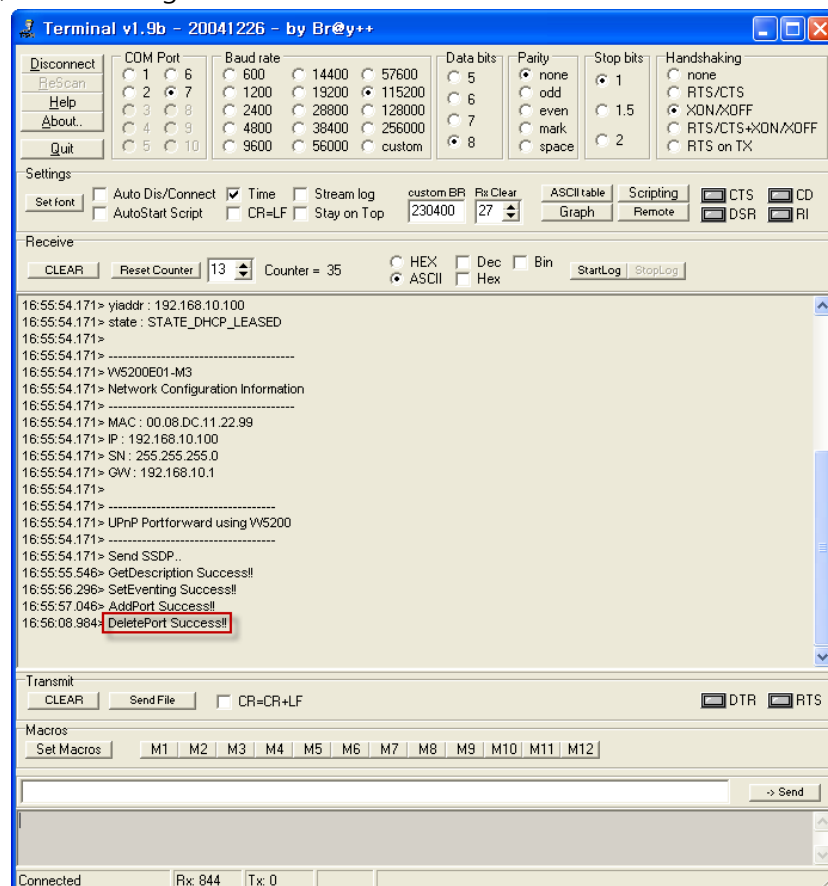


Figure 9. DeletePortMapping is OK

6. Run AX1 again to verify whether mapping entry is deleted.

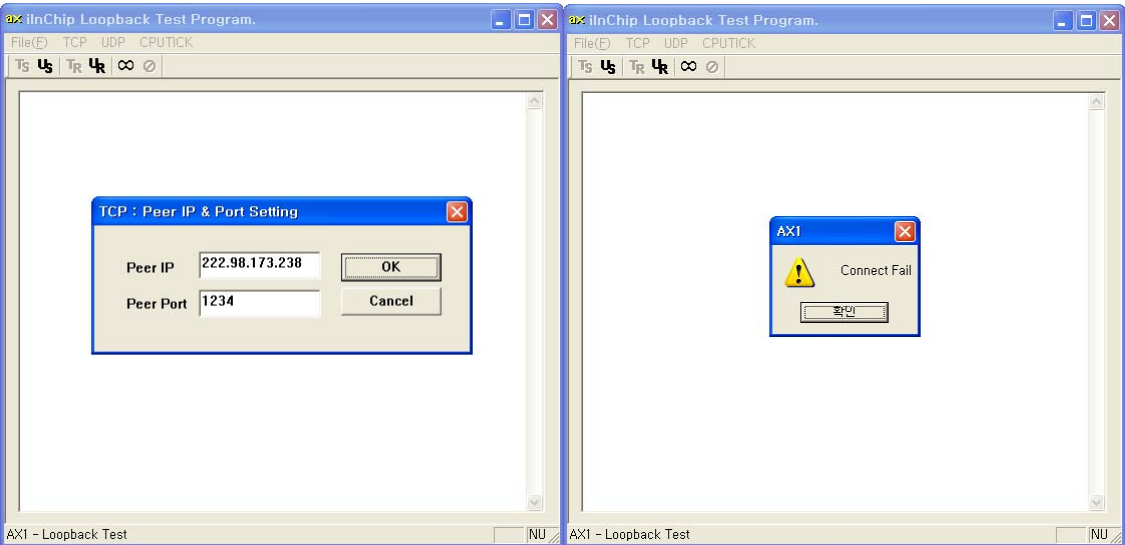


Figure 10. Run TCP client in remote host and let it connect to IGD (deleted external port)

6. Reference

[1] <http://www.pcguide.com/ref/mbsys/res/pnp-c.html>
[2] http://www.pcmag.com/encyclopedia_term/0,2542,t=plug+and+play&i=49389,00.asp
[3] http://ipstore.us/documentation/application_notes/AppNote_UPnP.pdf
[4] http://www.upnp.org/download/UPNP_UnderstandingUPNP.doc
[5] <http://www.upnp.org>

Document History Information

Version	Date	Descriptions
Ver. 0.9beta	Sep,2011	