

How To Implement a DDNS Client By Using The W7100A

Version 1.1



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

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

Table of Contents

1	Introduction	3
2	Dynamic Domain Name System	3
2.1	DDNS Specifications	3
3	DDNS Demonstration	5
4	Code Implementation	8
4.1	DDNS_Update() Function	9
4.1.1	Retrieve User Input	9
4.1.2	Compare New IP address and Old IP address	9
4.1.3	Perform DNS Query and Connect to Server	10
4.1.4	Base64 Encoding, Prepare Request and Send to Server	11
4.1.5	Parse Out Response Message	12
4.1.6	Base 64 Encoding	12
	Conclusion	15

1 Introduction

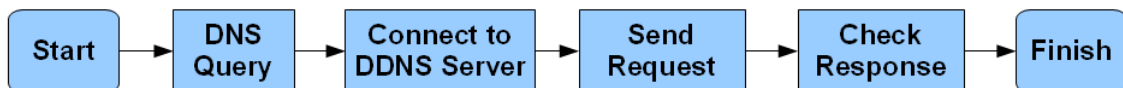
This document provides a step by step procedure of implementing the DDNS (Dynamic Domain Name System) protocol for the W7100. The W7100 evaluation board, iMCU7100EVB, is a total Ethernet solution which is ideally used as a server for automation control. With the constant changing of IP (Internet Protocol) address, the DNS (Domain Name System) is a convenient way to manage IP address of servers. However, once the server's IP address changes, the DNS record needs to be updated. To solve this problem, the Dynamic Domain Name System (DDNS) now comes into play.

2 Dynamic Domain Name System

The DDNS is a protocol which updates the server's IP address in the DNS record. Due to the nature of the ADSL/DSL connection, the IP address is bounded to change by the ISP (internet service provider). To keep the DNS record up to date, the DDNS protocol is used to monitor any changes in the IP address. When a change is detected, the DDNS will send a request to the DNS server to update the new IP address.

Currently, this protocol is compatible with popular DDNS hosts such as DynDNS.com and No-IP.com.

Due to securities reasons, these DDNS hosts do not allow the DDNS client to directly update the DNS record in their DNS servers. Instead, an API (Application Programming Interface) is created for the user to update the record. Their API is easy to use since it utilizes the http protocol. Here is the basic flow of the program.



2.1 DDNS Specifications

The program first performs a DNS query to determine the IP address of the DDNS server. After a connection is established with the server, a request is sent out. The request message is in the raw http get format.

Below shows the raw http get messages for No-IP.com and DynDNS.com:

The client must fill out the items highlighted in red.

No-IP.com Http Request Message Format:

```
GET /nic/update?hostname=mytest.testdomain.com&myip=1.2.3.4 HTTP/1.0
Host: dynupdate.no-ip.com
Authorization: Basic base64-encoded-auth-string
User-Agent: Device Name/Version Number Email
```

DynDNS.com Http Request Message Format:

```
GET /nic/update?hostname=hostname&myip=ipaddress&wildcard=NOCHG&mx=NOCHG&backmx=NOCHG HTTP/1.0
Host: members.dyndns.org
Authorization: Basic base-64-authorization
User-Agent: Company - Device - Version Number
```

Notice that both versions have similar syntax.

The following table describes each parameter in details.

Parameter	Description	Example
Hostname	The website's host name	?hostname= <i>mytest.acb.com</i>
Myip	The IP address which is needed to update	?myip= <i>1.2.3.4</i>
Authorization	User name and password string encoded in Base 64. The string must be in this Format: username:password	Username and Password string: test:test Base64 Encoded String: <i>dGVzdDp0ZXN0</i>
User-Agent	The device's name and version	No - IP.com format: <i>Wiznet W7100/1.0 test@wiznet.co.kr</i> DynDNS.com format: <i>Wiznet - W7100 - 1.0</i>

After a request is sent, the server will reply to the request with the following messages.

DynDNS.com Response Message

Reply Message	Message Type	Description
good <IP_ADDRESS>	Success	Completed Update
nochg <IP_ADDRESS>	Success	Although update is completed, the IP address is not changed. Please make sure update is performed with a new IP address or else the server will block the request and prevent the client from updating.
Badauth	Error	Wrong username and password
!donator	Error	The specified options require a service fee such as Offline URL
Notfqdn	Error	The specified host name is not in the qualified format. (Notice: Hostname should be

		hostname.dyndns.org or hostname.com)
Nohost	Error	The hostname does not exist in the account
Numhost	Error	The request attempted to update more than 20 hosts at once.
Abuse	Error	The hostname is blocked due to abuse
Badagent	Error	Agent sent incorrect Http request format
good 127.0.0.1	Error	Unless the client is intended to update the 127.0.0.1, otherwise this message indicates an error in the http request specifications.
Dnserr	Error	Dns error: Please contact with DynDns Support
911	Error	Service Maintenance is in progress. Please contact DynDns Support

No-IP.com Response Message

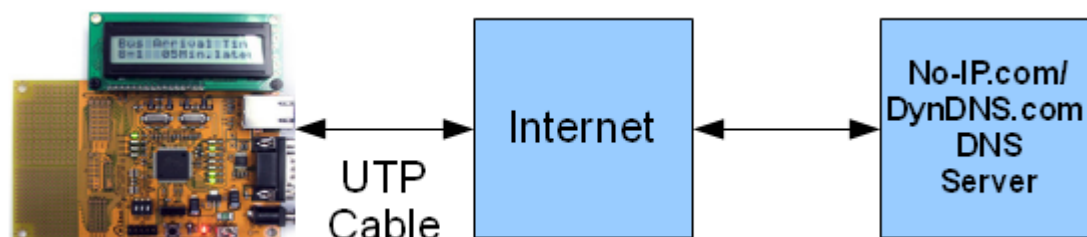
Reply Message	Message Type	Description
good <IP_ADDRESS>	Success	Completed Update
nochg <IP_ADDRESS>	Success	IP address is not changed, update is not performed.
Nohost	Error	The hostname does not exist in the account
Badauth	Error	Wrong username and password
Badagent	Error	Agent sent incorrect Http request format
!donator	Error	The specified options requires a service fee such as Offline URL
Abuse	Error	The hostname is blocked due to abuse
911	Error	Service Maintenance is in progress. Please contact with DynDns Support

Make sure the response message is “good <IP_ADDRESS>”. Otherwise, the client should stop performing updates and resolve the problem first. If these errors are ignored, the client is very likely to get banned by the server.

3 DDNS Demonstration

In this section, a demonstration of the DDNS protocol is shown.

Before testing, please setup the network environment according to the figure below:



The network settings along with the DDNS account settings are located under the title “User Settings” in the file, main.c as highlighted in the figure below.

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "types.h"           // Data Type Definition
#include "delay.h"           // MCU Delay Function
#include "ddns.h"            // DDNS Header
#include "serial.h"          // Serial Functions
#include "TCPIPcore.h"       // TCP/IP Core Definition
#include "w7100.h"           // Define W7100
#include "Wizmemcpy.h"       // Memory Copy Function

//Initialize Functions
void InitMCU();              //Initialize MCU
void InitNetwork();          //Initialize Network

// User Settings
uint8 xdata mac[6] = "\x00\x08\xDC\x11\x22\x33"; //MAC Address
uint8 xdata gw[4] = {192,168,0,1};               //Gateway Address
uint8 xdata source_ip[4] = {192,168,0,2};         //Source IP Address
uint8 xdata sub[4] = {255,255,255,0};             //Subnet Mask
int xdata dns_ip[4] = {0,};                       //DNS Server IP from Internet Provider
uint8 xdata host_name[20] = "test.dnsalias.net";  // Your DynDNS Host's name
uint8 xdata user_pass[75] = "test:test";         // Your "username:password" for DynDNS (format matters!)

uint8 xdata dyndns_ip[4] = {0,};                  //DynDNS Server IP (Provided by DNS Query)

//TX/RX Buffer Memory Settings
uint8 xdata txsize[MAX_SOCK_NUM] = {2,2,2,2,2,2,2,2}; //Set 2 Kbytes of TX memory for each socket
uint8 xdata rxsize[MAX_SOCK_NUM] = {2,2,2,2,2,2,2,2}; //Set 2 Kbytes of RX memory for each socket
    
```

Step 1) Plug in all relevant cable and power up the board

Upon entering the program, the network settings are printed out through the serial port. Make sure these settings match the network. User can use WizISP or W7100 Debugger to download the DDNS example code. Please refer to the “WizISP Program User Guide for W7100” or “W7100 Debugger Guide”.

```

*****Network Settings*****
IP:          192.168.0.2
Gateway:     192.168.0.1
Subnet:      255.255.255.0
MAC:         0.8.dc.11.22.33
DNS:         0.0.0.0
*****DDNS Account Settings*****
Hostname:    test.dnsalias.net
Username & Password:  test:test
User Agent:   WIZnet - W7100 - Version 1.0
    
```

**Step 2) Enter the ISP's DNS server IP address. Please follow this format
xxx.xxx.xxx.xxx**

Enter the ISP's DNS server IP address in this format (255.255.255.255)

—

**Step 3) Enter the new IP address to be updated in the DNS record. Please
follow this format xxx.xxx.xxx.xxx**

Enter the server's new IP address in this format (255.255.255.255)

—

**Step 4) After entering the new IP address, the request is processed by the
DDNS protocol. There are three different responses as listed below:**

Case 1: The update was successful which is indicated by the server's response "good"
followed by the newly updated IP address. Below is an example of a successful update.

```
Created Socket 7
Connected to 204.13.248.112 to update DYNDNS record
Sent Request Message
Received 168 Bytes of Data from Server
The Server Respond is: good 10.0.0.1
Successfully Updated with the new IP
Exit DDNS Update
```

Case 2: The update was skipped due to the fact that the new IP address is the same
as the previously updated IP address. Below is an example of skipping update.

```
No Change in IP address detected. Skip update
Exit DDNS Update
```

Case 3: The update failed. When the update fails, the server will provide an error
code for the user to diagnose with the DDNS host.

```
Connected to 204.13.248.112 to update DYNDNS record
Sent Request Message
Received 162 Bytes of Data from Server
The Server Respond is : badauth
Failed to update with the new IP. Please check this website
```

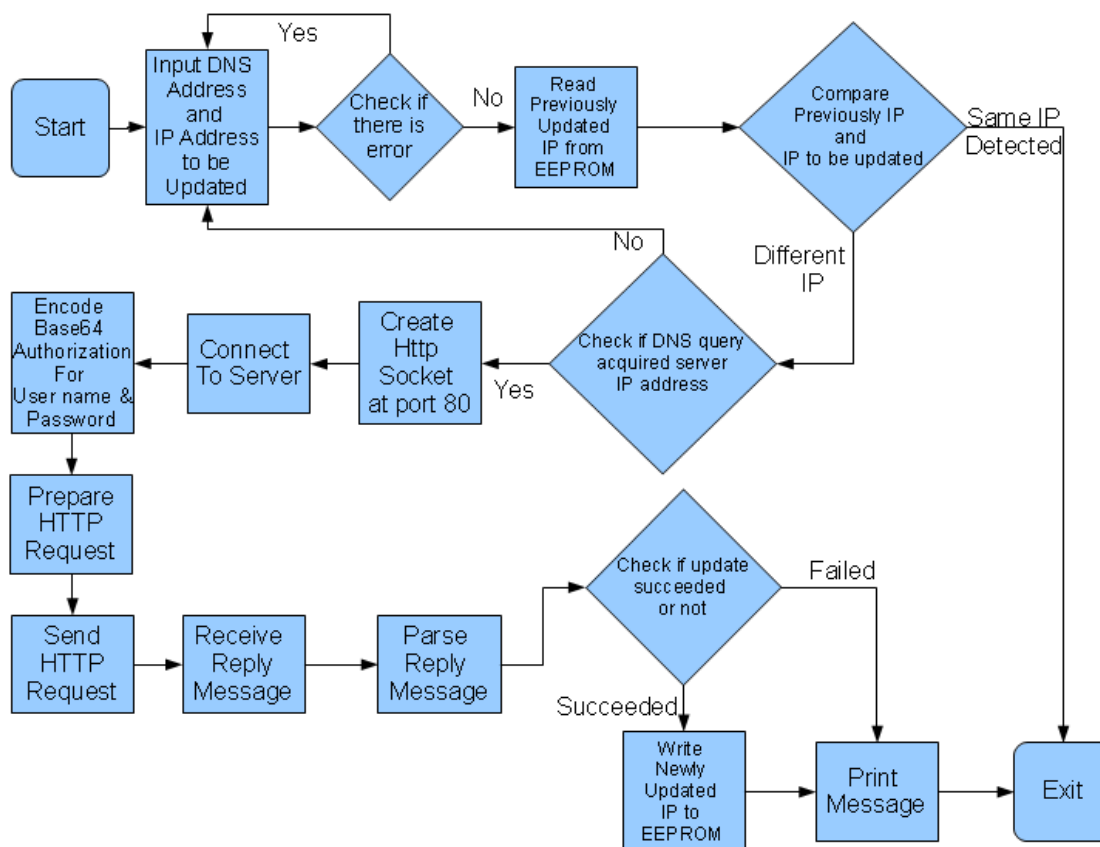
(<http://www.dyndns.com/developers/specs/return.html>)for more details about the error message
Exit DDNS Update

This concludes the procedure to operate the DDNS protocol.

4 Code Implementation

This section will go over all codes which have been used in the DDNS protocol. The DDNS protocol's code is stored in the DDNS.c and DDNS.h. In order to perform the DDNS update, the main program must call the function DDNS_update().

The DDNS_update() function follows the flowchart below:



Although the DDNS is a simple request and respond protocol, there are many steps required to prepare the DDNS request message. The following section will go through all the different stages of the DDNS protocol.

4.1 DDNS_Update() Function

4.1.1 Retrieve User Input

In this part of the code, the user's DNS address, which is provided by the ISP, and the new IP address of the web server are retrieved.

```
start:
// Input the ISP's DNS IP address
for(x = 0; x<4; x++)
{
    // Split the string into an integer array
    if(x == 0)
        split = strtok(tmp_str, ".");
    else
    {
        split = strtok(NULL, ".");
    }
    dns_ip[x] = atoi(split);
    //Check if IP is valid or not
    if(dns_ip[x] >255 || dns_ip[x] < 0)
    {
        // Invalid IP detected, re enter IP address
        goto start;
    }
}

// Print the DNS IP address
// Input new IP address of the web server and check if IP is valid or not (same code as above)
//Print the Web Server's IP address
```

4.1.2 Compare New IP address and Old IP address

The DDNS program will read the IP address which was stored in the flash during the last update. Then, a comparison is performed to determine whether an update is required or not.

```
for(x = 0; x<4; x++)
{
    //Read Previously Updated IP address from flash
    flash_ip[x] = EEPROM_Read(x);
```

```

    }

    // Compare previously IP address with current IP address
    if(memcmp(flash_ip,current_ip,4)==0)
    {
        // Same IP address is detected, inform user and exit
        goto exit; // Exit Program
    }

    // Continue DDNS protocol

```

4.1.3 Perform DNS Query and Connect to Server

The DNS query attempts to find out the real IP address from the DDNS server's hostname. After the IP address is found, the program connects to the server. For more information about the dns_query() function, please refer to the "How to implement DNS client using W7100" document.

```

dns_query(s,"members.dyndns.org"); // find the IP address of members.dyndns.org with DNS Query

// Check if we got IP address from DNS query
if( dyndns_ip[0] == 0 && dyndns_ip[1] == 0 && dyndns_ip[2] == 0 && dyndns_ip[3] == 0)
{
    // failed to get IP address from Hostname
    goto start; // Ask user to re enter the DNS IP address information
}

//Use while loop to send a request and receive a response from server

switch(getSn_SR(s)) // check socket state
{
case SOCK_CLOSED:
port++;
if(socket(s,Sn_MR_TCP, port, 0x00) == 1) // Open Socket in TCP mode
{
    //Successfully Created a socket
    sent_flag = 0;
}
else
{
    // Failed to create a socket
    close(s);
}
}

```

```

    }
break;
case SOCK_INIT:
if(connect(s,dynDNS_ip,80) == 1)      // Connect to DynDNS Server
    {
        //Successfully Connected to server and print the IP address connected to
    }
else
    {
        //Failed to connect to server
    }
break;

```

4.1.4 Base64 Encoding, Prepare Request and Send to Server

Base64_encode() encodes the user and password string into a base64 encoding format. After the encoded user and password string is ready, the HTTP GET request message is prepared. Please make sure there is no space after each `\r\n` or else the request message will not be interpreted as a http packet. Once the message is prepared, it is sent out to the server. Then, the program waits for a reply message from the server.

```

case SOCK_ESTABLISHED:
    // Encode user and password into base64
    base64_encode(user_pass,base64_user_pass,strlen(user_pass));

    // create HTTP GET request message
    sprintf(request_msg,"GET /nic/update?hostname=%s&myip=%s&wildcard=NOCHG&mx=NOCHG&
backm x=NOCHG HTTP/1.1 \r\nHost: members.dyndns.org \r\nAuthorization: Basic %s \r\nUser-
Agent:%s \r\n\r\n",host_name ,ip_addr_str ,base64_user_pass,user_agent);

    //Send out Request Messages
    send(s,request_msg,strlen(request_msg));

    //Check if there is data in receive buffer
    len = getSn_RX_RSR(s);
    if (len > 0)
    {
        //check if received data is bigger than buffer or not
        if(len > MAX_LENGTH) len = MAX_LENGTH;
        len = recv(s, reply_msg, len); //Place Data into reply_msg
    }

```

```

    }
    break;

```

4.1.5 Parse Out Response Message

The program examines the reply message and prints out the result.

```

// search for update success in reply string
split = strstr(reply_msg, "good");
if(split != NULL)
{
    //Update succeed parsing data from reply message
    split_end = strstr(split, "\r\n0\r\n\r\n");
    len = split_end - split;

    // write newly updated IP address into flash
    EEP_Block_Write(0, current_ip, 4);

    // Print Response Message
}
else
{
    //failed to update parsing data from reply message
    split = strstr(reply_msg, "\r\n\r\n");
    split = strstr(split+4, "\r\n");
    split += 2;
    split_end = strstr(split, "\r\n0\r\n\r\n");
    len = split_end - split;

    //print response message
}

```

4.1.6 Base 64 Encoding

The function `base64_encode()` uses an encoding table to convert a string into base64 format. For example, if the character 'a' is encoded, the result will be "YQ==". The character 'a' has 8 bits ("01100001"). The first 6 bits are extracted ("011000") and inserted into the encoding table output 'Y.' The same process applies to the last two bits ("010000"). Notice the last two bits are shifted to the first and second position. After inserting these 6 bits binary into the encoding table, the result is 'Q.' Please note that this encoding function always output characters in multiples of 4. As a result, the characters "==" are added into the encoded string for blank characters.

```

Void base64_encode(uint8* usrpwd, uint8* base64_userpwd, int len)

```

```

{
    // Encoding Table
    static char xdata encodingTable [64] =
    { 'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','a','b','c','d','e','f','g','h','i','j','k','l',
      'm','n','o','p','q','r','s','t','u','v','w','x','y','z','0','1','2','3','4','5','6','7','8','9','+','/' };

    uint8 xdata in_idx = 0;          //input buffer index
    uint8 xdata out_idx = 0;         //output buffer index
    uint8 xdata remaining = 0;       //remaining characters to be processed
    uint8 xdata num = 0;              //number of characters to be processed
    uint8 xdata tmp[100] = {0,};    // tmp string

    num = (len%3);                   //determine how many characters there are
    if(num == 0)
    {
        remaining = len + (len/3);   // compute length of the encoded string
    }
    else
    {
        switch(num)
        {
            case 1: remaining = ((len *4)/3)+ 3; // compute length of the encoded string
                    break;
            case 2: remaining = ((len *4)/3)+ 2; // compute length of the encoded string
                    break;
            default:
                break;
        }
    }

    while(remaining > 0)
    {
        // Extract first 6 bits and insert into encoding table
        tmp[out_idx] = (usrpwd[in_idx]&0xFC) >> 2;
        base64_userpwd[out_idx] = encodingTable[tmp[out_idx]];

        // Extract next set of 6 bits and insert into encoding table
        tmp[out_idx + 1] = (((usrpwd[in_idx]&0x03) <<4) | ((usrpwd[in_idx+1]&0xF0)>>4));
        base64_userpwd[out_idx + 1] = encodingTable[tmp[out_idx + 1]];
    }
}
    
```

```

// Extract next set of 6 bits and insert into encoding table
tmp[out_idx + 2] = (((usrpwd[in_idx+1]&0x0F) <<2) | ((usrpwd[in_idx+2]&0xC0)>>6));
base64_userpwd[out_idx + 2] = encodingTable[tmp[out_idx + 2]];

// Extract next set of 6 bits and insert into encoding table
tmp[out_idx + 3] = (usrpwd[in_idx+2]&0x3F);
base64_userpwd[out_idx + 3] = encodingTable[tmp[out_idx + 3]];

// Special Case Applies for the last 4 characters
if(remaining == 4)
{
switch(num)
{
case 1: // add '=' characters to fill in the empty encoded string
memcpy(base64_userpwd+out_idx + 2, "=", 2);
break;

case 2: // add a '=' character to fill in the empty encoded string
tmp[out_idx + 2] = (((usrpwd[in_idx+1]&0x0F) <<2) |
((usrpwd[in_idx+2]&0xC0)>>6));
base64_userpwd[out_idx + 2] = encodingTable[tmp[out_idx + 2]];
memcpy(base64_userpwd+out_idx + 3, "=", 1);
break;

default:
break;
}
}

out_idx += 4; // increment index
in_idx += 3; // increment index
remaining -= 4; // decrease the remaining characters counter
}
}

```

Conclusion

The DDNS protocol is a good method to manage the web server's IP address. Be sure to follow all specifications of the DDNS host or else the client program might get blocked by the server for abuse. For questions/comments/ or bug report, please feel free to contact with me.

Victor Tsang (victortsang@wiznet.co.kr) Thank you

Document History Information

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

Copyright Notice

Copyright 2011 WIZnet Co.,Ltd. All Rights Reserved.

Technical Support: support@wiznet.co.kr

Sales & Distribution: sales@wiznet.co.kr

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