How to implement DNS client using 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 Domain Name System 3
3 DNS Demonstration 5
4 Implementation Code 7
4.1 dns_query() Function 7
4.2 dns_makequery() Function 9
4.3 parseMSG() Function 10
4.4 dns_question() Function 11
4.5 dns_answer() Function 12
4.6 parse_name() Function 14

© Copyright 2011 WIZnet Co.,Ltd. All rights reserved.

1 Introduction

This document will explain about the Domain Name System. The document will implement DNS client by using iMCU7100EVB and check all functions by going through demonstrations. All example codes below are based on Keil compiler.

2 Domain Name System

Domain Name System is a system that converts Internet Domain Name (ex: www.wiznet.co.kr) to Internet IP Address (ex: 202.131.29.70) or the other way around. Since the increase in numbers of address as the internet developed, the DNS server became to have a class structure. DNS is composed of Name Server (DNS server), that has a mapping table between domain names, and DNS Resolver (DNS Client), that queries and receives the converted results from the DNS server. DNS Resolver asks the local Name Server for the IP Address to be converted. After being asked, the local Name Server searches DB for it. If there is information regarding the client's request, the converted result will be sent to the Client. If there isn't any information, the local Name Server will query the root Name Server and wait for an answer then give an answer to the client.

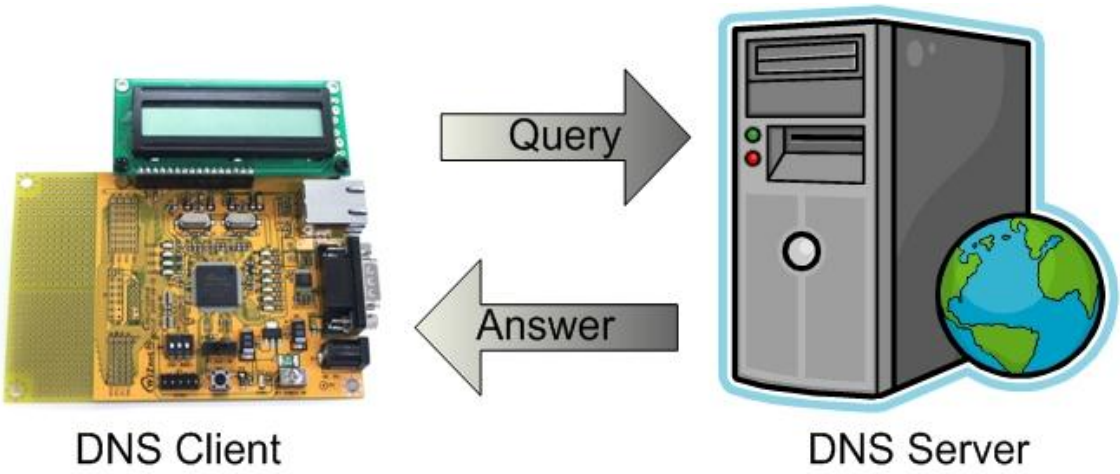


Fig. 2.1 Domain Name System

Below Fig. 2.2 shows the format of the query/answer message between the DNS Client and DNS Server. The messages that are queried and answered can be divided into 5 sections: Header, Question, Answer, Authority, and Additional. The Header section has a fixed size of 12byte, but the other 4 sections don't have a fixed size. Sections Answer, Authority, and Additional can be grouped and named Resource Records (RRs). Each Header, Question, and RRs have a unique format.

Please refer to documents on RFC1034 and RFC1035 for more details on DNS.

© Copyright 2011 WIZnet Co.,Ltd. All rights reserved.

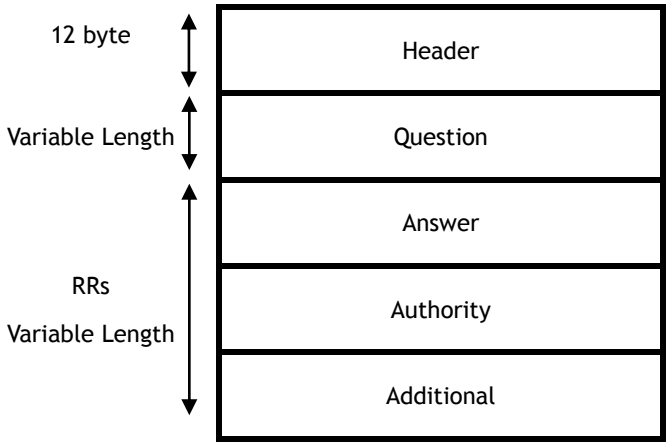


Fig. 2.2 Query/Answer Message Foramt

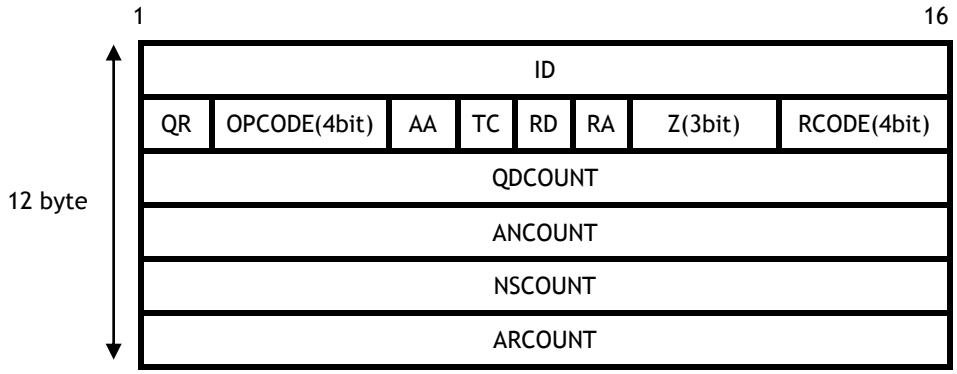


Fig. 2.3 Header Section Format

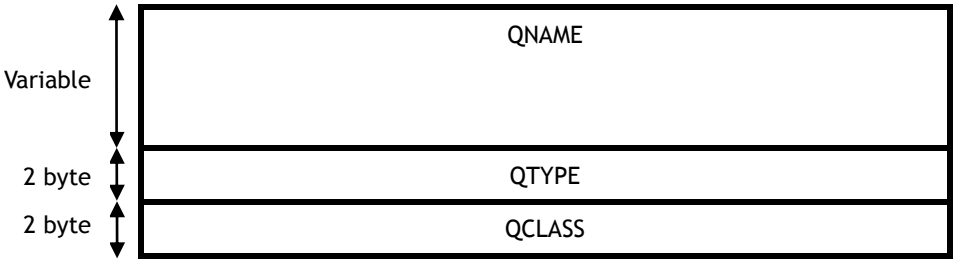


Fig. 2.4 Question Section Format

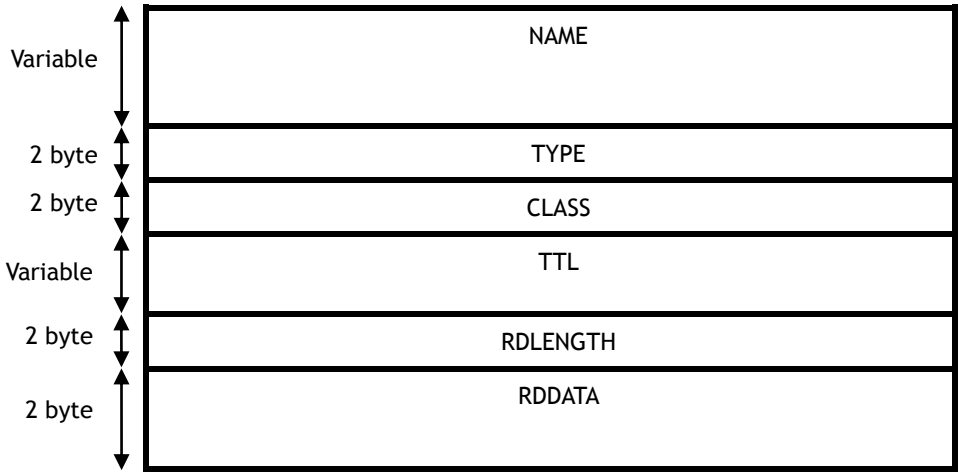


Fig. 2.5 Resource Records (RRs) Section Format

© Copyright 2011 WIZnet Co.,Ltd. All rights reserved.

In this document, iMCU7100EVB works as a DNS Client; it queries the Internet Domain Name to the DNS Server and converts it to Internet IP Address. iMCU7100EVB also creates UDP protocol to send/receive/analyze Answer UDP packet with DNS Server.

3 DNS Demonstration

As mentioned in section 2, the demonstration of DNS by using iMCU7100EVB will be explained in this section. iMCU7100EVB works as DNS Client using UDP protocol. For DNS demonstration, the Hyper terminal is used other than iMCU7100EVB. Hyper terminal is a basic application program from MS Windows and can be used easily by everyone.

Before demonstration, there are important precautions that should be mentioned. **The settings for IP address, GW address, subnet, and etc of iMCU7100EVB in main.c from the DNS code must be set properly according to the user's setting.** For more details on these values, please ask the ISP (Internet Service Provider) or the network manager. Since the values in the sample code are set randomly, the DNS code will not work without changing those values. **Also the settings for DNS server IP address in dns.c must be set properly according to the user's setting.** Usually DNS server is supplied by ISP, so if there are questions related to DNS server, please ask ISP.

To demonstration, connect all power cable, LAN cable, and serial cable to iMCU7100EVB. For details on EVB board, please refer to 'iMCU7100EVB User's Guide.' After modifying the DNS client's IP address and DNS server's IP address properly, compile the attached source files to create HEX file. And load the DNSC HEX file to the iMCU7100EVB board by using the serial cable or Debugger cable. User can use the W7100 debugger program or WizISP program to load. Please refer to 'W7100 Debugger Guide' and 'WizISP Program User Guide for W7100' for directions of iMCU7100 debugger program and WizISP program. Both programs are included in the 'bundle CD,' or can download from WIZnet's homepage.

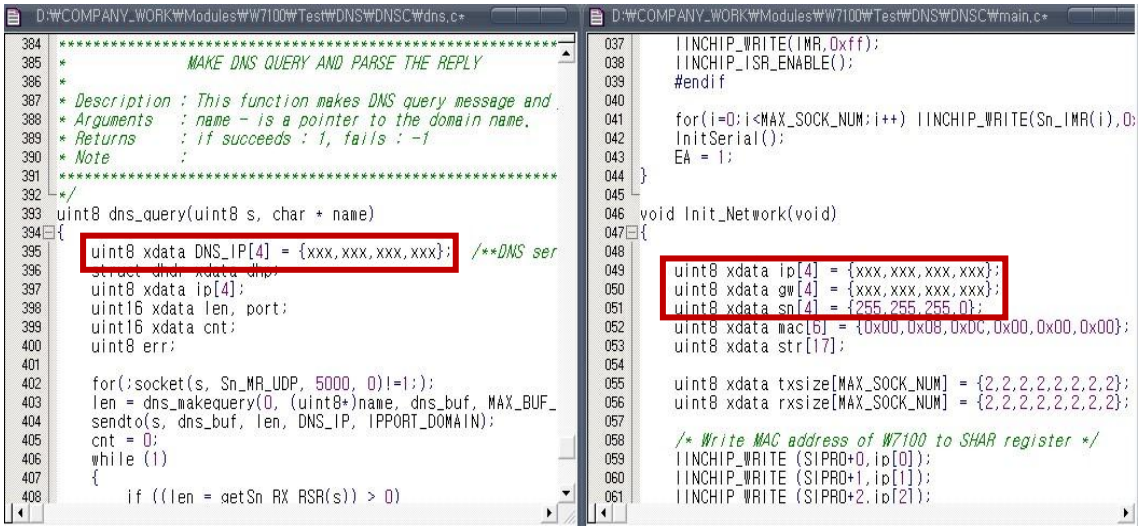
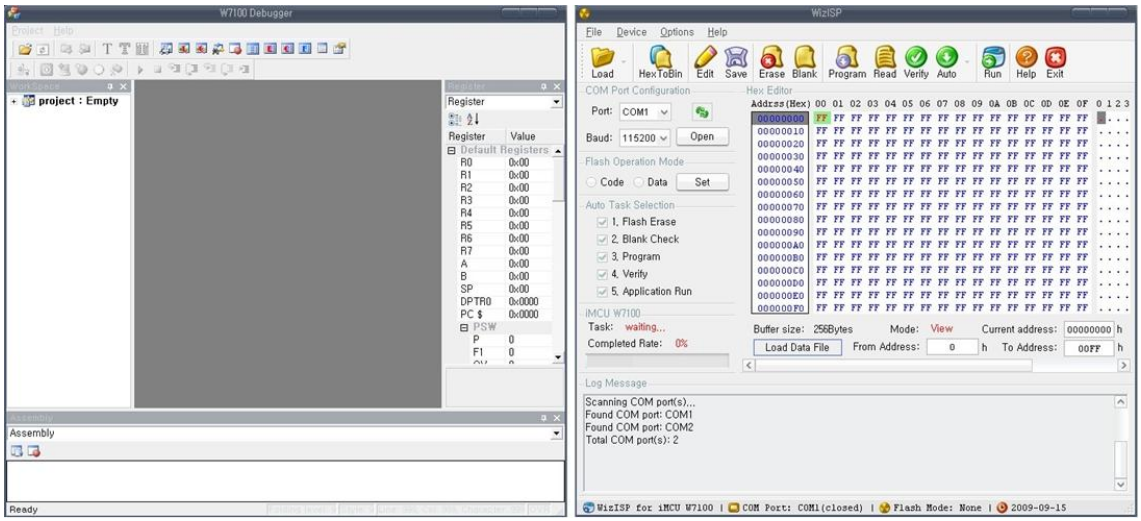


Fig. 3.1 iMCU7100EVB IP, GW, Subnet and DNS Server IP setting

© Copyright 2011 WIZnet Co.,Ltd. All rights reserved.



(a) W7100 Debugger (b) WizISP

Fig. 3.2 W7100 Debugger & WizISP program interface

If the HEX file is successfully loaded, run the Hyper terminal program. If WizISP program is used, turn off the BOOTSEL pin and reset the EVB board. If W7100 Debugger is being used, press the 'run' button or end the Debugger and reset the EVB board.

The reason that Hyper terminal program is being used for DNS demonstration is because the DNS response from iMCU7100EVB is coded to be sent by using serial cable. As shown in Fig. 3.3, Hyper terminal should be set properly according to the user's serial port. Although COM2 port is used as serial port in Fig. 3.3, this can differ depending on the user's PC setting.

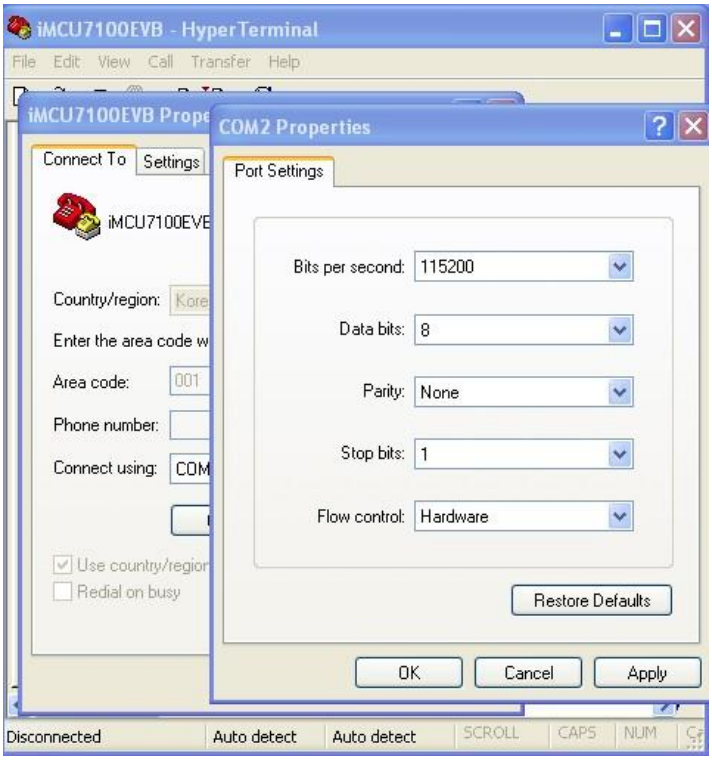


Fig. 3.3 Hyper terminal setting

© Copyright 2011 WIZnet Co.,Ltd. All rights reserved.

Also, the Baud Rate is set as 115200, but the user can manually change by modifying the serial settings of DNSC code. Run the EVB board and Hyper terminal, and enter the proper domain name in the Hyper terminal window. The following results will appear as shown in Fig. 3.4.

```
Input the domain name (ex>www.wiznet.co.kr):www.wiznet.co.kr

Querying DNS server 168.126.63.1 for www.wiznet.co.kr
IP address : 211.110.19.21
Input the domain name (ex>www.wiznet.co.kr):www.google.com

Querying DNS server 168.126.63.1 for www.google.com
IP address : 74.125.53.99
Input the domain name (ex>www.wiznet.co.kr):www.yahoo.co.kr

Querying DNS server 168.126.63.1 for www.yahoo.co.kr
IP address : 119.161.11.206
Input the domain name (ex>www.wiznet.co.kr):www.naver.com

Querying DNS server 168.126.63.1 for www.naver.com
IP address : 222.122.195.6
Input the domain name (ex>www.wiznet.co.kr):
```

(a) success finding

```
Input the domain name (ex>www.wiznet.co.kr):www.google

Cannot find the Domain Name!!, wrong Domain Name or Server is not available now.
..

Input the domain name (ex>www.wiznet.co.kr):www.abcdefg.abc

Cannot find the Domain Name!!, wrong Domain Name or Server is not available now.
..

Input the domain name (ex>www.wiznet.co.kr):
```

(b) fail finding

Fig. 3.4 Results of DNSC example

Part (a) of Fig 3.4 will show when the corresponding IP for domain name is found successfully, and part (b) is when an incorrect domain name is entered, or when the DNS server failed to search the domain name. Other than these two cases, if the DNS Server does not respond after a period of time, iMCU7100EVB will show a Timeout message.

4 Implementation Code

Section 4 will explain the example codes that are loaded into the iMCU7100EVB board which works as DNS Client. UDP is used for communication protocol. For more details on UDP communication, please refer to document 'How to implement UDP for W7100.'

4.1 dns_query() Function

Although the example codes have main() function, serial() function, and socket() function, they operate as additional codes such as UDP and serial communication. Therefore, in this document, the functions in dns.c file, which are directly related to the DNS operation, will be explained.

© Copyright 2011 WIZnet Co.,Ltd. All rights reserved.

All DNS Client example codes work by the dns_query() function. All other lower functions work under the dns_query() function. Code 4.1 below are codes under the dns_query() function.

```
uint8 dns_query(uint8 s, char * name)
{
  uint8 xdata DNS_IP[4] = {xxx,xxx,xxx,xxx};   //DNS server IP, User must change it(please ask your ISP)
  struct dhdr xdata dhp;   //Structure variable for DNS header section
  uint8 xdata ip[4];
  uint16 xdata len, port, cnt;
  uint8 xdata err;
  for(;socket(s, Sn_MR_UDP, 5000, 0)!=1;);   //Open the UDP socket
  len = dns_makequery(0, (uint8*)name, dns_buf, MAX_BUF_SIZE);   //Make DNS query
  sendto(s, dns_buf, len, DNS_IP, IPPORT_DOMAIN);   //Send DNS query to DNS server
  cnt = 0;
  while (1){
      if ((len = getSn_RX_RSR(s)) > 0){
        if (len > MAX_BUF_SIZE) len = MAX_BUF_SIZE;
        len = recvfrom(s, dns_buf, len, ip, &port);   //Receive DNS answer from DNS server
        close(s);
        break;
      }
      wait_10ms(1);
      if (cnt++ == 0x100){      // Timeout check
         return -1;
         break;
      }
  }
  err = parseMSG(&dhp, dns_buf);       //Analyze the DNS answer
  if(err == 1)   return 1;     //Success to find
  else return -1;   //Fail to find
}
```

Code 4.1 dns_query() fucntion

Among the variables that are used in the dns_query() function, the structure variable dhp is included in the DNS header section. The DNS_IP variable is set to xxx.xxx.xxx.xxx; this value is the IP address of the DNS server and must be entered correctly. Please ask ISP or the network manager for the IP address of DNS server.

First, open the UDP socket. In Code 4.2, port number is set to 5000 (port number can be modified what user want). Then, create a query message; use dns_makequery() function (There will be more details on this later). After creating the message, use sento()function to send the message to the UDP packet.

© Copyright 2011 WIZnet Co.,Ltd. All rights reserved.

After the message is sent, iMCU7100EVB waits for a response from DNS server. Use getSn_RX_RSR() function to check whether the message was received properly; and if there are no response after a period of time, think of it as Timeout. When there is any response, use recvfrom() function to receive DNS answer message. The received answer message is analyzed by using pareseMSG() function. The parseMSG() function checks the rcode (DNS message header section) from the DNS answer; and it returns 1 or 0. When the the parseMSG() returns 1 (rcode ==0), it means the IP address for the domain name is found successfully. Other values of rcode other than 0 mean different errors. But in the example code, parselMSG() function is coded to return 0 for all other errors.

4.2 dns_makequery() Function

Dns_makequery() function is called by dns_query() function, it makes a DNS query message. Refer to section 2 for the configuration of DNS query message. For the return value, it returns the message pointer of DNS buffer.

```
int dns_makequery(uint16 op, uint8 * name, uint8 * buf, uint16 len)
{
  uint8 xdata *cp;
  char xdata *cp1, *dname;
  char xdata sname[MAX_BUF_SIZE];
  uint16 xdata p, dlen;
  cp = buf;
  MSG_ID++;                          //MSG_ID = 0x1122;
  cp = put16(cp, MSG_ID);
  p = (op << 11) | 0x0100;           //Recursion desired
  cp = put16(cp, p);                 //put16() copies uint16 variable to network buffer
  cp = put16(cp, 1);
  cp = put16(cp, 0);
  cp = put16(cp, 0);
  cp = put16(cp, 0);
  strcpy(sname, name);
  dname = sname;
  dlen = strlen(dname);
  for (;;){                          //Look for next dot
    cp1 = strchr(dname, '.');
    if (cp1 != NULL) len = cp1 - dname;        //More to come
    else len = dlen;                           // Last component
    *cp++ = len;                               //Write length of component
    if (len == 0) break;
    strncpy((char *)cp, dname, len);           //Copy component up to (but not including) dot
```

© Copyright 2011 WIZnet Co.,Ltd. All rights reserved.

```
        cp += len;
        if (cp1 == NULL){
            *cp++ = 0;                          //Last one; write null and finish
            break;
        }
        dname += len+1;
        dlen -= len+1;
    }
    cp = put16(cp, 0x0001);                     //Type component: 0x0001 = Type A
    cp = put16(cp, 0x0001);                     //Class component: 0x0001 =   Internet class
    return ((int)((uint16)(cp) - (uint16)(buf)));       //Return the DNS message pointer
}

uint8 * put16(uint8 * s, uint16 i)
{
    *s++ = i >> 8;        //s = network buffer pointer
    *s++ = i;
    return s;
}
```

Code 4.2 dns_makequery() function

The query message created from the dns_makequery() function, it has the message formant as explained in the section 2. Put16_function is used when entering DNS message in the buffer; this function saves the 16 bit input into the 8 bit buffer.

4.3 parseMSG() Function

parseMSD()function is called by dns_query()function, it analyzes the received DNS message. First the header section is analyzed, and then dns_question()function is called to analyze the question section. After that, dns_answer()function is called to analyze the answer section. Among the analyzed answer messages, the IP address, that map with the domain name, saves into a variable to output using printf() function.

```
uint8 parseMSG(struct dhdr * dhdr, uint8 * buf)
{
    uint16 xdata tmp, i;
    uint8 xdata * msg, *cp;
    msg = buf;
    memset(dhdr, 0, sizeof(*dhdr));
    dhdr->id = get16(&msg[0]);
    tmp = get16(&msg[2]);
```

© Copyright 2011 WIZnet Co.,Ltd. All rights reserved.

```
  if (tmp & 0x8000) dhdr->qr = 1;       //Check the QR value of DNS header message
  dhdr->opcode = (tmp >> 11) & 0xf;     // Check the Opcode value of DNS header message
  if (tmp & 0x0400) dhdr->aa = 1;       // Check the AA value of DNS header message
  if (tmp & 0x0200) dhdr->tc = 1;       // Check the TC value of DNS header message
  if (tmp & 0x0100) dhdr->rd = 1;       // Check the RD value of DNS header message
  if (tmp & 0x0080) dhdr->ra = 1;       // Check the RA value of DNS header message
  dhdr->rcode = tmp & 0xf;            // Check the RCODE value of DNS header message
  dhdr->qdcount = get16(&msg[4]);       // Check the QDCOUNT value of DNS header message
  dhdr->ancount = get16(&msg[6]);       // Check the ANCOUNT value of DNS header message
  dhdr->nscount = get16(&msg[8]);       // Check the NSCOUNT value of DNS header message
  dhdr->arcount = get16(&msg[10]);      // Check the ARCOUNT value of DNS header message
  cp = &msg[12];                        // Now parse the variable length sections
  for (i = 0; i < dhdr->qdcount; i++){      // Question section
      cp = dns_question(msg, cp);
  }
  for (i = 0; i < dhdr->ancount; i++){    // Answer section
      cp = dns_answer(msg, cp);
  }
  for (i = 0; i < dhdr->nscount; i++){    // Name server (authority) section
      ;
  }
  for (i = 0; i < dhdr->arcount; i++){    // Additional section
      ;
  }
  if(dhdr -> rcode == 0) return 1;        //Means No Error
  else return 0;
}
```

Code 4.3 parseMSG() function

4.4 dns_question() Function

dns_question() function is called by parseMSG()function, it analyzes the question section (Qname, Qtype, Qclass). Please refer to documents of FC1034 and RFC1035 for more details about question section values.

```
uint8 * dns_question(uint8 * msg, uint8 * cp)
{
  int xdata len;
  char xdata name[MAX_BUF_SIZE];
  len = parse_name(msg, cp, name, MAX_BUF_SIZE);
```

© Copyright 2011 WIZnet Co.,Ltd. All rights reserved.

```
    if (len == -1) return 0;
    cp += len;                    // Qname
    cp += 2;                      // Qtype
    cp += 2;                      // Qclass
    return cp;
}
```

Code 4.4 dns_question() function

4.5 dns_answer() Function

dns_answer()function is called by parseMSG()function, it analyzes the resource records section ((Name, Type, Class, TTL, Rdlength, Rddata). After the message type is analyzed, the message is analyzed only when the message type is TypeA, TypePTR, TypeHINFO, TypeMX, TypeSOA, or TypeTXT.

Please refer to documents on RFC1034 and RFC1035 for more details on each type.

```
uint8 * dns_answer(uint8 * msg, uint8 * cp)
{
    int xdata len, type;
    char xdata name[MAX_BUF_SIZE];
    len = parse_name(msg, cp, name, MAX_BUF_SIZE);
    if (len == -1) return 0;
    cp += len;                    // Name
    type = get16(cp);
    cp += 2;                      // Type
    cp += 2;                      // Class
    cp += 4;                      // TTL
    cp += 2;                      // Rdlength
    switch (type){
        case TYPE_A:              //Just read the address directly into the structure
            SIP[0] = *cp++;
            SIP[1] = *cp++;
            SIP[2] = *cp++;
            SIP[3] = *cp++;
            break;
        case TYPE_CNAME:
        case TYPE_MB:
        case TYPE_MG:
        case TYPE_MR:
        case TYPE_NS:
        case TYPE_PTR:            //These types all consist of a single domain name
```

© Copyright 2011 WIZnet Co.,Ltd. All rights reserved.

```
            len = parse_name(msg, cp, name, MAX_BUF_SIZE);     //convert it to ASCII format
            if(len == -1) return 0;
            cp += len;
            break;
        case TYPE_HINFO:
            len = *cp++;
            cp += len;
            len = *cp++;
            cp += len;
            break;
        case TYPE_MX:
            cp += 2;              //Get domain name of exchanger
            len = parse_name(msg, cp, name, MAX_BUF_SIZE);
            if(len == -1) return 0;
            cp += len;
            break;
        case TYPE_SOA:            //Get domain name of name server
            len = parse_name(msg, cp, name, MAX_BUF_SIZE);
            if(len == -1) return 0;
            cp += len;            //Get domain name of responsible person
            len = parse_name(msg, cp, name, MAX_BUF_SIZE);
            if(len == -1) return 0;
            cp += len;
            cp += 4;
            cp += 4;
            cp += 4;
            cp += 4;
            cp += 4;
            break;
        case TYPE_TXT:            //Just stash
            break;
            default:              //Ignore
            break;
        }
        return cp;
}
```

Code 4.5 dns_answer() function

© Copyright 2011 WIZnet Co.,Ltd. All rights reserved.

4.6 parse_name() Function

Parse_name()function analyzes only the name section from the DNS answer message. Parse_name()function also changes the DNS message format so that the user can handle easier. The length of Name is returned after the analysis.

```
int parse_name(uint8 * msg, uint8 * compressed, char * buf, uint16 len)
{
   uint16 xdata slen;                    // Length of current segment
   uint8 xdata * cp;
   int xdata clen = 0;                   // Total length of compressed name
   int xdata indirect = 0;               // Set if indirection encountered
   int xdata nseg = 0;                   // Total number of segments in name
   cp = compressed;
   for (;;)
   {
      slen = *cp++;              // Length of this segment
      if (!indirect) clen++;
      if ((slen & 0xc0) == 0xc0)
      {
         if(!indirect) clen++;
         indirect = 1;           // Follow indirection
         cp = &msg[((slen & 0x3f)<<8) + *cp];
         slen = *cp++;
      }
      if (slen == 0)  break;       // Zero length == all done
      len -= slen + 1;
      if (len < 0) return -1;
      if (!indirect) clen += slen;
      while (slen-- != 0) *buf++ = (char)*cp++;
      *buf++ = '.';
      nseg++;
   }
   if (nseg == 0)
   {                 // Root name; represent as single dot
      *buf++ = '.';
      len--;
   }
   *buf++ = '\0';
   len--;
```

© Copyright 2011 WIZnet Co.,Ltd. All rights reserved.

```
    return clen;        //Length of compressed message
}
```

Code 4.6 parse_name() function

© Copyright 2011 WIZnet Co.,Ltd. All rights reserved.

Document History Information

Version	Date	Descriptions
Ver. 0.9 Beta	Sep, 2009	Release with W7100 launching
Ver. 1.0	Mar, 2011	Modified 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