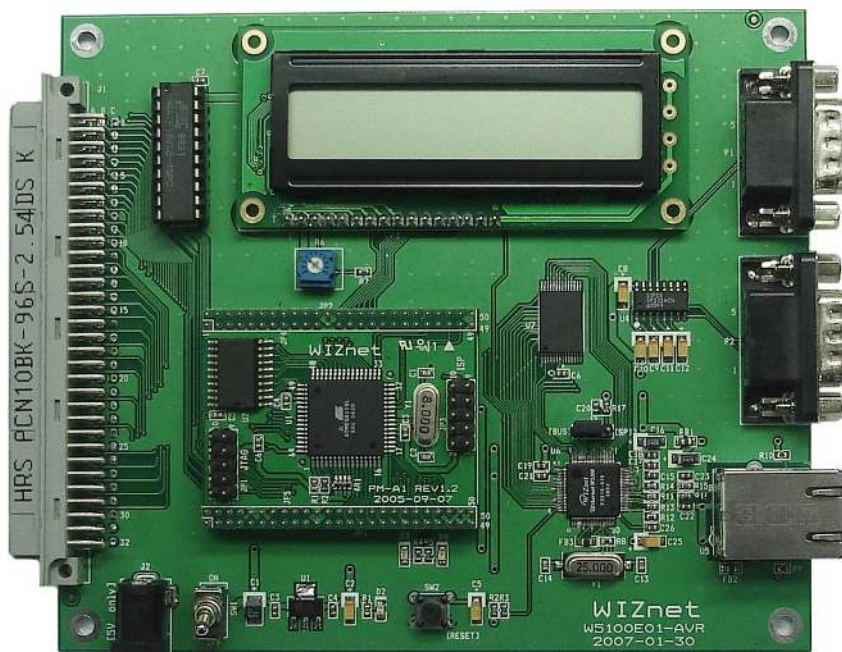


W5100E01-AVR User's Manual

(Version 1.1.0)



©2007 WIZnet Co., Ltd. All Rights Reserved.

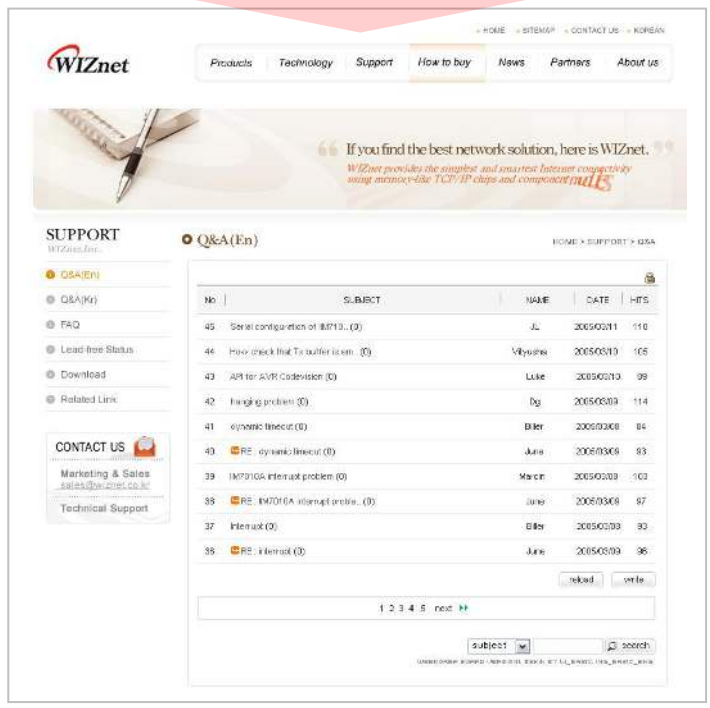
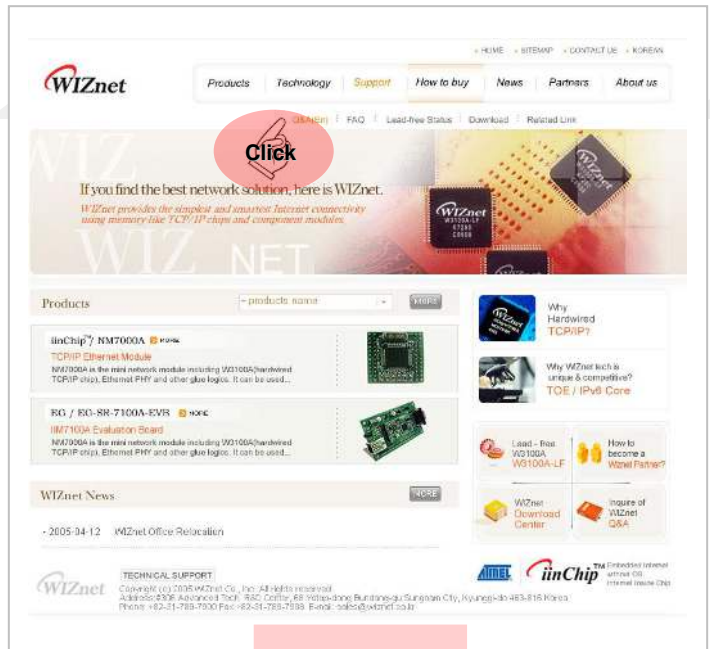
For more information, visit our website at <http://www.wiznet.co.kr>

Document History Information

| Revision | Date | Description |
|------------|------------------|---|
| Ver. 1.0.0 | February 1, 2007 | Original Document |
| Ver. 1.1.0 | June 17, 2013 | The software CD is not provide anymore. For more software contents, please visit our website. (www.wiznet.co.kr) modified the comment about S/W CD. (CH 1.1, 4.3.1, 4.3.2, 4.4.4, 4.5.1, 4.5.2) |

WIZnet's Online Technical Support

If you have something to ask about WIZnet Products, Write down your question on Q&A Board in WIZnet website (www.wiznet.co.kr). WIZnet Engineer will give an answer as soon as possible.



COPYRIGHT NOTICE



Copyright 2007 WIZnet, Ltd. All Rights Reserved.

Technical Support: support@wiznet.co.kr

Sales & Distribution: sales@wiznet.co.kr

General Information: info@wiznet.co.kr

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

Table of Contents

| | | |
|--------|---------------------------------------|----|
| 1. | Overview..... | 1 |
| 1.1. | Package..... | 1 |
| 1.2. | Feature | 2 |
| 1.2.1. | H/W Features..... | 2 |
| 1.2.2. | F/W Feature | 2 |
| 2. | Getting Started..... | 3 |
| 2.1. | System Configuration | 3 |
| 2.1.1. | EVB B/D Layout & Configuration | 3 |
| 2.2. | PC Programs Install | 4 |
| 2.2.1. | Development Program Install..... | 4 |
| 2.2.2. | EVB B/D Test PC Program Install | 4 |
| 2.3. | Quick Start..... | 5 |
| 2.4. | EVB B/D Test..... | 6 |
| 2.4.1. | Manage Program | 7 |
| 2.4.2. | EVB B/D Test Applications | 13 |
| 2.5. | Troubleshooting Guide | 18 |
| 2.5.1. | Ping | 18 |
| 2.5.2. | Misc. | 18 |
| 3. | Programmer's Guide..... | 19 |
| 3.1. | Memory Map | 19 |
| 3.1.1. | Code & Data Memory Map..... | 19 |
| 3.1.2. | AVR Internal EEPROM MAP | 20 |
| 3.2. | EVB B/D Firmware | 26 |
| 3.2.1. | Sources | 27 |
| 3.2.2. | How to Compile | 28 |
| 3.2.3. | How to download | 29 |
| 3.2.4. | EVB B/D's main() | 29 |
| 3.2.5. | Manage Program | 33 |
| 3.2.6. | Applications | 49 |
| 4. | Hardware Designer's Guide | 92 |
| 4.1. | Block Diagram | 92 |
| 4.2. | Block Description..... | 93 |
| 4.2.1. | PM-A1 | 93 |

| | | |
|--------|----------------------------------|-----|
| 4.2.2. | LCD | 97 |
| 4.2.3. | PAL..... | 98 |
| 4.2.4. | SRAM | 98 |
| 4.2.5. | RS232 Port..... | 98 |
| 4.2.6. | Expanded Board Interface | 98 |
| 4.2.7. | Power Regulator..... | 100 |
| 4.2.8. | 3.3V Power On System Reset | 100 |
| 4.3. | Schematic..... | 101 |
| 4.3.1. | W5100E01-AVR | 101 |
| 4.3.2. | PM-A1 | 101 |
| 4.4. | PAL..... | 102 |
| 4.4.1. | IO Define | 102 |
| 4.4.2. | External SRAM Area..... | 103 |
| 4.4.3. | LCD Area | 103 |
| 4.4.4. | W5100 Area..... | 104 |
| 4.5. | Parts List..... | 106 |
| 4.5.1. | W5100E01-AVR Parts List | 106 |
| 4.5.2. | PM-A1 Parts List..... | 106 |
| 4.6. | Physical Specification..... | 107 |
| 4.6.1. | Power Consumption | 107 |

Figures

| | |
|--|-----------|
| <FIG 2.1 : EVB B/D JUMPER SETTING> | 3 |
| <FIG 2.2 : JP3 JUMPER SETTING > | 3 |
| <FIG 2.3 : EVB B/D TEXT LCD DISPLAY > | 5 |
| <FIG 2.4 : OUTPUT OF TERMINAL PROGRAM> | 6 |
| <FIG 2.5 : EVB B/D PING REPLY TEST > | 6 |
| <FIG 2.6 : MANAGE PROGRAM EXECUTION > | 7 |
| <FIG 2.7 : NETWORK CONFIG > | 8 |
| <FIG 2.8 : SOURCE IP ADDRESS SETUP EXAMPLE> | 9 |
| <FIG 2.9 : MAC ADDRESS SETUP EXAMPLE> | 9 |
| <FIG 2.10 : MENU OF CHANNEL CONFIG> | 10 |
| <FIG 2.11 : LOOPBACK TCP CLIENT APPLICATION SETTING EXAMPLE> | 11 |
| <FIG 2.12 : USAGE OF PING APPLICATION > | 12 |
| <FIG 2.13 : PING APPLICATION TEST> | 13 |
| <FIG 2.14 : DHCP CLIENT TEST> | 14 |
| <FIG 2.15 : LOOPBACK TCP SERVER TEST> | 15 |
| <FIG 2.16 : LOOPBACK TCP CLIENT> | 15 |
| <FIG 2.17 : LOOPBACK UDP TEST> | 16 |
| <FIG 2.18 : WEB SERVER TEST> | 16 |
| <FIG 2.19 : DEFAULT WEB PAGE OF EVB B/D> | 17 |
| <FIG 2.20 : WEB PAGE OF EVB B/D CONTROL> | 17 |
| <FIG 3.1: EVB B/D MEMORY MAP> | 19 |
| <FIG 3.2: AVR INTERNAL EEPROM MAP> | 20 |
| <FIG 3.3: EVB B/D's MAIN()> | 32 |
| <FIG 3.4: CHECK_MANAGE()> | 33 |
| <FIG 3.5: MANAGE_CONFIG()> | 34 |
| <FIG 3.6: MANAGE_NETWORK()> | 36 |
| <FIG 3.7: MANAGE_CHANNEL()> | 38 |
| <FIG 3.8: PING_REQUEST()> | 40 |
| <FIG 3.9: PING_REQUEST() – CONTINUE> | 41 |
| <FIG 3.10: ICMP MESSAGE VS PING MESSAGE> | 42 |
| <FIG 3.11: PING()> | 45 |
| <FIG 3.12: DISPLAYPINGSTATISTICS()> | 46 |
| <FIG 3.13: SENDPINGREPLY()> | 47 |
| < FIG 3.14 : LOOPBACK_TCPS() > | 49 |

| | |
|--|----|
| <FIG 3.15: LOOPBACK_TCPC(>..... | 52 |
| <FIG 3.16: LOOPBACK_UDP(>..... | 53 |
| <FIG 3.17: HTTP MESSAGE FLOW>..... | 55 |
| <FIG 3.18: WEB_SERVER(>..... | 58 |
| <FIG 3.19: PROC_HTTP(> | 59 |
| <FIG 3.20: PARSE_HTTP_REQUEST(>..... | 61 |
| <FIG 3.21: FIND_HTTP_URI_TYPE(> | 62 |
| <FIG 3.22: GET_HTTP_URI_NAME() & GET_HTTP_PARSE_VALUE(>..... | 62 |
| <FIG 3.23: NETCONF.CGI PROCESSING> | 63 |
| <FIG 3.24: LCDNLED.CGI PROCESSING>..... | 64 |
| <FIG 3.25: DHCP MESSAGE FLOW>..... | 66 |
| <FIG 3.26: DHCP MESSAGE FORMAT>..... | 67 |
| <FIG 3.27: DHCP MESSAGE'S OPTION FIELD FORMAT>..... | 68 |
| <FIG 3.28: INIT_DHCP_CLIENT(> | 69 |
| <FIG 3.29: GETIP_DHCPS(>..... | 70 |
| <FIG 3.30: DHCP MESSAGE FLOW BY DHCP CLIENT STATE>..... | 72 |
| <FIG 3.31: CHECK_DHCP_STATE(>..... | 73 |
| <FIG 3.32: PARSE_DHCPMSG() & CHECK_DHCP_TIMEOUT(>..... | 74 |
| <FIG 3.33: DOMAIN NAME SYSTEM STRUCTURE & DNS MESSAGE FLOW>..... | 76 |
| <FIG 3.34: DNS MESSAGE FORMAT>..... | 77 |
| <FIG 3.35: HEADER SECTION FORMAT>..... | 77 |
| <FIG 3.36: QUESTION SECTION FORMAT> | 77 |
| <FIG 3.37: RECODE RESOURCES FORMAT>..... | 78 |
| <FIG 3.38: GETHOSTBYADDR() & GETHOSTBYNAME(>..... | 80 |
| <FIG 3.39: DNS_QUERY(>..... | 81 |
| <FIG 3.40: DNS_MAKEQUERY(>..... | 82 |
| <FIG 3.41: EXAMPLE OF QNAME FIELD TRANSFORMATION OF QUESTION SECTION > | 83 |
| <FIG 3.42: DNS_PARSE_RESPONSE(>..... | 85 |
| <FIG 3.43: DNS_PARSE_QUESTION() & DNS_ANSWER(> | 87 |
| <FIG 3.44: PARSE_NAME(> | 88 |
| <FIG 3.45: DNS MESSAGE COMPRESSION SCHEME>..... | 89 |
| <FIG 4.1: EVB B/D BLOCK DIAGRAM> | 92 |
| <FIG 4.2: PM-A1 MODULE DIMENSION>..... | 93 |

Tables

| | |
|--|----|
| <TABLE 1-1 : LIST OF ITEMS CONTAINED IN THE EVB B/D>..... | 1 |
| <TABLE 1-2 : CONTENTS OF SOFTWARE>..... | 1 |
| <TABLE 2-1 : TERMINAL PROPERTIES SETTING>..... | 5 |
| <TABLE 2-2 : EVB B/D DEFAULT NETWORK INFORMATION>..... | 7 |
| <TABLE 2-3 : MENU OF NETWORK CONFIG>..... | 8 |
| <TABLE 2-4 : EVB B/D DEFAULT CHANNEL INFORMATION>..... | 9 |
| <TABLE 2-5 : MENU OF CHANNEL CONFIG>..... | 10 |
| <TABLE 2-6 : W5100 CHANNEL APPLICATION TYPE>..... | 10 |
| <TABLE 2-7 APPLICATION DEFAULT VALUE >..... | 11 |
| <TABLE 3-1 : DEVICE MAP DEFINITION>..... | 20 |
| <TABLE 3-2 : AVR INTERNAL EEPROM MAP DEFINITION>..... | 21 |
| <TABLE 3-3 : SYSTEM INFORMATION>..... | 22 |
| <TABLE 3-4 : SYSINFO DATA TYPE DEFINITION>..... | 22 |
| <TABLE 3-5 : SYSTEM INFORMATION ACCESS FUNCTIONS>..... | 22 |
| <TABLE 3-6 : NETWORK INFORMATION>..... | 23 |
| <TABLE 3-7 : NETCONF DATA TYPE DEFINITION>..... | 23 |
| <TABLE 3-8 : NETWORK INFORMATION ACCESS FUNCTIONS>..... | 23 |
| <TABLE 3-9 : CHANNEL INFORMATION>..... | 24 |
| <TABLE 3-10 : CHANNEL APPLICATION TYPE>..... | 24 |
| <TABLE 3-11 : CHCONF DATA TYPE DEFINITION>..... | 25 |
| <TABLE 3-12 : CHANNEL INFORMATION ACCESS FUNCTION>..... | 25 |
| <TABLE 3-13 : EVB B/D SOURCES>..... | 27 |
| <TABLE 3-14 : W5100's DEFINE OPTION (TYPES.H) >..... | 29 |
| <TABLE 3-15 : REFERENCE FUNCTIONS IN EVB B/D'S MAIN()>..... | 31 |
| <TABLE 3-16 : CALLER FUNCTION AT MANAGE PROGRAM >..... | 35 |
| <TABLE 3-17 : REFERENCE FUNCTIONS IN MANAGE_CONFIG()>..... | 37 |
| <TABLE 3-18 : CONSTRAINT BY APPLICATION TYPES>..... | 38 |
| <TABLE 3-19 : REFERENCE FUNCTIONS IN MANAGE_CHANNEL() >..... | 39 |
| <TABLE 3-20 : PINGMSG DATA TYPE DEFINITION>..... | 43 |
| <TABLE 3-21 : PINGLOG DATA TYPE DEFINITION>..... | 43 |
| <TABLE 3-22 : REFERENCE FUNCTIONS IN PING_REQUEST()>..... | 48 |
| <TABLE 3-23 : REFERENCE FUNCTIONS IN LOOPBACK_TCPS()>..... | 50 |
| <TABLE 3-24 : REFERENCE FUNCTIONS IN LOOPBACK_TCPC()>..... | 52 |
| <TABLE 3-25 : REFERENCE FUNCTIONS IN LOOPBACK_UDP()>..... | 54 |

| | |
|--|-----|
| <TABLE 3-26: WEB BROWSER'S HTTP REQUEST OPERATION PROCEDURE >..... | 55 |
| <TABLE 3-27: HTTP MESSAGE FORMAT>..... | 56 |
| <TABLE 3-28: HTTP MESSAGE BETWEEN EVB B/D AND WEB BROWSER>..... | 57 |
| <TABLE 3-29: SYSTEM ENVIRONMENT VARIABLES USAGE AT "EVBCTRL.HTML" >..... | 60 |
| <TABLE 3-30: "ST_HTTP_REQUEST" DATA>..... | 61 |
| <TABLE 3-31: REFERENCE FUNCTIONS IN WEB_SERVER(>..... | 65 |
| <TABLE 3-32: DHCP MESSAGE DATA TYPE>..... | 67 |
| <TABLE 3-33: DHCP MESSAGE OPTION CODE DEFINITION>..... | 68 |
| <TABLE 3-34: DHCP CLIENT STATE & TIMEOUT DEFINITION>..... | 71 |
| <TABLE 3-35: DHCP MESSAGE FLAG FIELD SETUP>..... | 71 |
| <TABLE 3-36: REFERENCE FUNCTIONS IN DHCP CLIENT>..... | 75 |
| <TABLE 3-37: DNS MESSAGE DATA TYPE>..... | 79 |
| <TABLE 3-38: QUERY TYPE DEFINITION AT DNS_QUERY(>..... | 79 |
| <TABLE 3-39: CONSTANTS AND MACRO USED IN HEADER SECTION>..... | 83 |
| <TABLE 3-40 : CONSTANTS DEFINITION AT QTYPE & QCLASS FIELD>..... | 84 |
| <TABLE 3-41 : CONSTANT DEFINITION AT HEADER SECTION'S RCODE FIELD>..... | 86 |
| <TABLE 3-42 : REFERENCE FUNCTIONS IN DNS CLIENT >..... | 91 |
| <TABLE 4-1: PM-A1 MODULE PIN DESCRIPTION>..... | 94 |
| <TABLE 4-2: ISP PIN DESCRIPTION>..... | 96 |
| <TABLE 4-3: LCD PIN DESCRIPTION>..... | 97 |
| <TABLE 4-4: EXPANDED BOARD INTERFACE PIN DESCRIPTION>..... | 98 |
| < TABLE 4-5 EVB B/D POWER CONSUMPTION >..... | 107 |

1. Overview

W5100E01-AVR is W5100 Evaluation B/D for AVR developers.

1.1. Package

When purchasing W5100E01-AVR B/D, please make sure you have all the following contents.

<Table 1-1: List of Items Contained in the EVB B/D>

| | Item | Quantity |
|------------------|--|---------------|
| EVB B/D | W5100E01-AVR Main Board | 1 |
| | PM-A1 MCU Module (Plugged In W5100E01-AVR) | 1 |
| | Power Adaptor (DC5V / 2A) | 1 |
| Accessory | AVR ISP Internal Flash Programming Tool | Option |
| | UTP Cable | 1 |
| | Serial Cable | 1 |
| | ISP Gender Type I | Option |

<Table 1-2 : Contents of Software>

| Directory | | Contents | |
|--------------|------|------------------|------------------------------------|
| W5100E01-AVR | DOCs | Manual | User's Manual |
| | | Datasheet | All sorts of Datasheet |
| | | Application Note | AVR Tool Guide ISP Gender Guide |
| | HW | Schematics | All sorts of schematics |
| | | Part List | All sorts of Part List |
| | | PAL | Logic Source & JED File |
| | SW | Firmware | EVB B/D Firmware |
| | | PC Utility | All sorts of Tool Program |
| W5100 | | | |

- The contents of Software could be changed by version. Please check the official website of WIZnet.

1.2. Feature

1.2.1. H/W Features

W5100E01-AVR B/D is composed of 2 type B/Ds

- PM-A1
 - MCU : ATmega128, 8MHz
 - RAM : 32KB SRAM (External)
 - ROM : 128KB Flash (Atmega128 Internal Flash)
 - ICE I/F : JTAG, ISP Support
- W5100E01-AVR
 - Power : DC5V, 2A Adaptor
 - UART : Two 232 Serial Port, (default 57600 Baud Rate)
 - LCD Display : 16 X 2 Text LCD
 - PAL : Address Decoder
 - W5100 : Hardwired TCP/IP Chip(embedded PHY chip)
 - MagJack : RD1-125BAG1A (UDE) , Integrated Transformer(1:1)
Link & ACT & FDX LEDs

1.2.2. F/W Feature

The F/W of EVB B/D is made up of two parts.

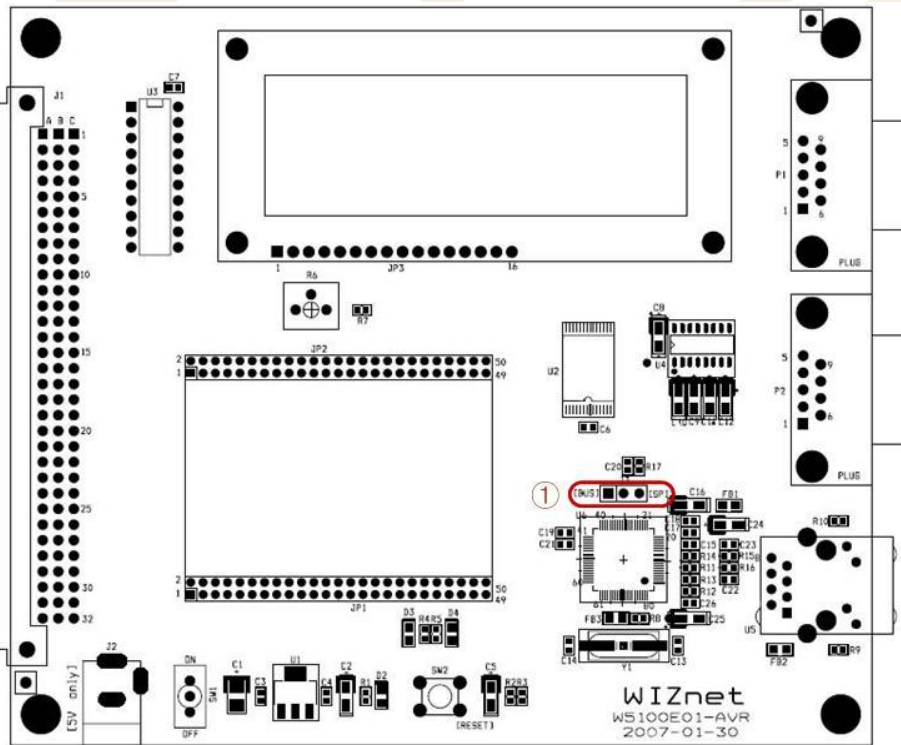
- Manager mode
 - Network Config : MAC, Source IP, G/W IP, S/N, DNS IP Setup
 - Channel Config : W5100 Test Application Setup for each channel
 - Ping Test : Ping Request Test with DNS
- Application mode
 - Loopback TCP Server : TCP Server Mode Test Application
 - Loopback TCP Client : TCP Client Mode Test Application
 - Loopback UDP : UDP Test Application
 - Web Server : Web Server Test Application
 - DHCP Client : Dynamic Network Config using DHCP Server

2. Getting Started

2.1. System Configuration

2.1.1. EVB B/D Layout & Configuration

For testing the functions of the EVB B/D and developing applications, the EVB B/D should be configured as shown below. First, the EVB B/D is connected to the PC using the crossed UTP Cable (for data transmission) and the Serial Cable (for monitoring). Second, the dip switch and jumper should be set as below;



<Fig 2.1 : EVB B/D Jumper Setting>

① SPI Enable : J3

For interfacing W5100 with MCU through SPI mode, the pin of 2 and 3 of JP3 should go short. In case that SPI mode is not used, the pin of 1 and 2 should be shorted.



<Fig 2.2 : JP3 Jumper Setting >

2.2. PC Programs Install

2.2.1. Development Program Install

Please refer to “**AVR Tool Guide Vx.x.pdf**” for more information.

2.2.1.1. Compile Tool Chain

For installation and usage of WinAVR, refer to the related manual.

Firmware of EVB B/D is currently using AVR GCC Version 3.4.6 Compiler and can be changed with compiler version upgrade.

2.2.1.2. ICE Programs

EVB B/D supports JTAG & ISP ICE for development and debugging. For ISP Program, “AVRStudio” program is used. Please refer to “**AVR Tool Guide Vx.x.pdf**” for installation and usage of “AVR Studio” and “**ISP GENDER User's Guide Vx.x.pdf**” for usage of ‘ISP GENDER’.

2.2.1.3. ROM File Maker Program

ROM File Maker Program is a utility program that provides convenience in using simple ‘ROM File System’ for EVB B/D. The reason that ROM File Maker Program is used in EVB B/D is to access Web Pages for Web Server Test Application as ‘ROM File System’. Refer to “**ROM File Maker Manual Vx.x.pdf**” for further instruction on installation and ROM File Maker Program

2.2.2. EVB B/D Test PC Program Install

2.2.2.1. Loopback Test Program (AX1) Install

Loopback Test Program (referred to as “AX1” from here on) is a program to evaluate the performance of W5100 and does the Loopback the file and packet data in connection with EVB B/D channel applications such as Loopback TCP Server/Client and Loopback UDP. Please refer to “**AX1 Manual Vx.x.pdf**” for installation and usage.

2.3. Quick Start

After the confirming the Package of EVB B/D, test EVB B/D in the order shown below.

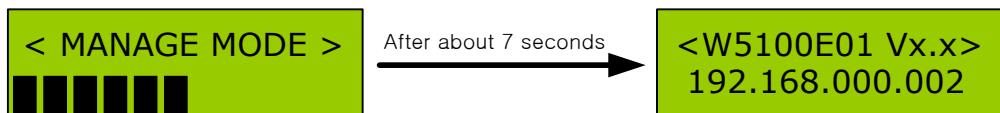
- ① Confirm the testing environment. Refer to [Chapter 2.1](#)
 Connect test PC to EVB B/D using UTP cable directly.
 Connect test PC to EVB B/D using serial cable directly.
 Connect 5V power adaptor to EVB B/D
- ② Confirm the network information of Test PC as the following
 Source IP Address : 192.168.0.3
 Gateway IP Address : 192.168.0.1
 Subnet Mask : 255.255.255.0
- ③ Install AX1 on Test PC. Refer to [Chapter 2.2.2.1](#)
- ④ After the execution of serial terminal program (like Hyperterminal), set up the properties as the following.

<Table 2-1 : Terminal Properties Setting>

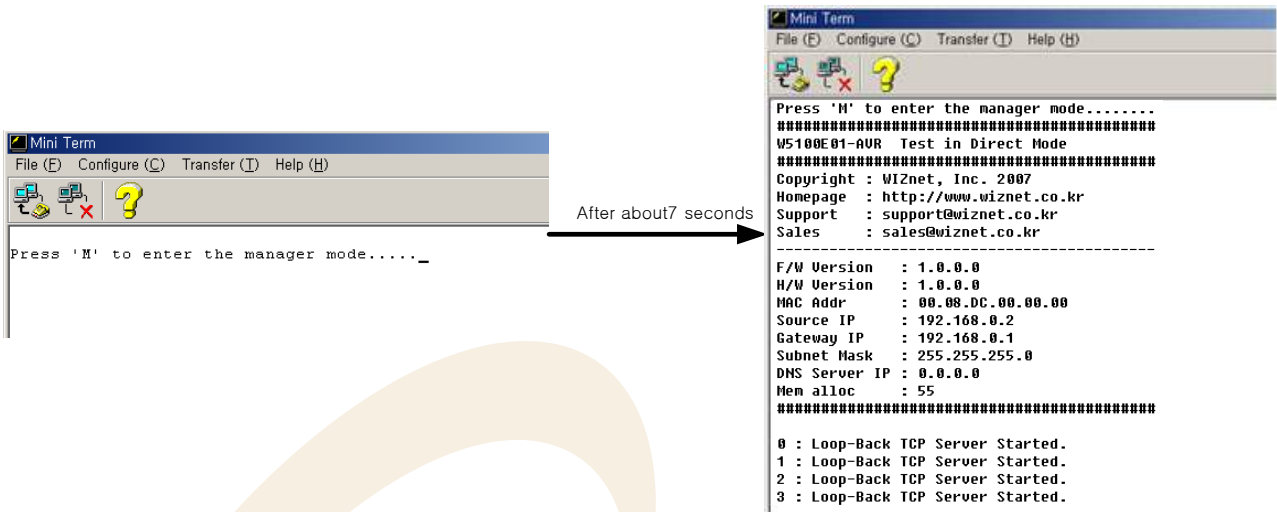
| Properties | Setting Value |
|----------------------------|---------------|
| Bits Per second(Baud Rate) | 57600 bps |
| Data Bits | 8 Bits |
| Stop Bits | 1 Bit |
| Parity | No |
| Flow Control | None |

After the completion of terminal setup, connect to EVB B/D and wait.

- ⑤ Turn on the power switch(SW1) of EVB B/D
 Following items should be checked upon power on
 - Check lighting on power LED(D2) of EVB B/D when powering on
 - Check if LEDs of D3 and D4 blink three times by turns.
 - Check if Text LCD display of EVB B/D outputs in the way shown in <Fig 2.3> and shown in <Fig 2.4> on the Terminal Program



<Fig 2.3 : EVB B/D Text LCD Display >



<Fig 2.4 : Output of Terminal Program>

- ⑥ Execute Ping test with EVB B/D

```

C:\W>ping 192.168.0.2

Pinging 192.168.0.2 with 32 bytes of data:

Reply from 192.168.0.2: bytes=32 time<10ms TTL=64
Reply from 192.168.0.2: bytes=32 time<10ms TTL=64
Reply from 192.168.0.2: bytes=32 time<10ms TTL=64
Reply from 192.168.0.2: bytes=32 time=10ms TTL=64

Ping statistics for 192.168.0.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 10ms, Average = 2ms
    
```

<Fig 2.5 : EVB B/D Ping Reply Test >

- ⑦ Execute "AX1" program. Refer to "AX1 Manual Vx.x.pdf"
- ⑧ Test the operation of "AX1" program with TCP Client. Refer to "AX1 Manual Vx.x.pdf"
- After setting the Server IP Address as "192.168.0.2" and port Number as "5000" by clicking [TCP>>Connect] Menu, then click,[TCP>>Send] Menu or [Ts],[Tr],[∞] Icons.
- ⑨ Test the loopback with any file or packet between "AX1" Program and EVB B/D.

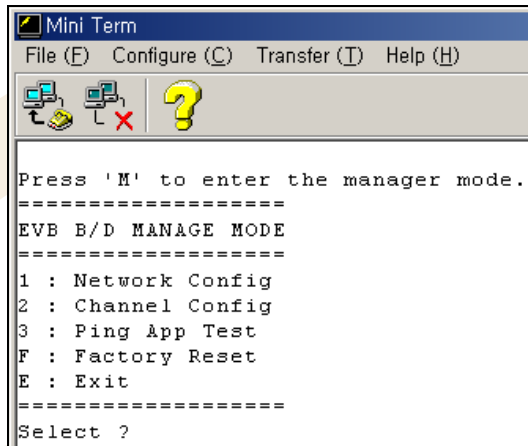
2.4. EVB B/D Test

The firmware of EVB B/D can be divided into Manage Program and EVB B/D Test Application.

Manage Program performs system configuration to run EVB B/D, and EVB B/D Test Application is Network Application Program for W5100 Test.

2.4.1. Manage Program

Manage Program is a program that is executed upon receiving character 'M' or 'm' from the terminal program within 7 seconds when doing the manual reset of EVB B/D and EVB B/D power on. This program sets up the channel application of W5100 to be tested, and perform certain ping request test with DNS server.



<Fig 2.6 : Manage Program Execution >

2.4.1.1. Network Configuration

It selects Network Information that is used in EVB B/D. When choosing '1' at terminal Program of <Fig 2.6>, Network Information of EVB B/D can be set as desired. The default Network Information of EVB B/D is shown in <Table 2-2>.

<Table 2-2 : EVB B/D Default Network Information>

| Network Information | Default Value |
|-----------------------|-------------------|
| MAC Address | 00.08.DC.00.00.00 |
| Source IP Address | 192.168.0.2 |
| Gateway IP Address | 192.168.0.1 |
| Subnet Mask | 255.255.255.0 |
| DNS Server IP Address | 0.0.0.0 |

If "Network Config" menu is selected on Manage Program, menu shown in <Fig 2.7> can be displayed and each function is described in <Table 2-3>.

```

Select ? 1
-----
NETWORK CONFIG
-----
D : Display config
1 : Source IP
2 : Gateway IP
3 : Subnet Mask
4 : DNS Server IP
M : MAC address
A : memory Allocation
F : Factory reset
E : Exit
-----
Select ?
    
```

<Fig 2.7 : Network Config >

<Table 2-3 : Menu of Network Config>

| Menu | Description |
|------------------------|---|
| D : Display Config | Display current Network Information |
| 1 : Source IP Address | Sets up Source IP Address |
| 2 : Gateway IP Address | Sets up Gateway IP Address |
| 3 : Subnet Mask | Sets up Subnet Mask |
| 4 : DNS Server IP | Sets up DNS Server IP Address <Warning> DNS Server is information needed for "Ping Request" test and transformation of Domain Name into IP address. Therefore, it must be set up as Static IP Address. |
| 'A' or 'a' | Sets up Memory Allocation – W5100 Memory Size Register.(RMSR,TMSR) Refer to "W5100 Datasheet.pdf". |
| F : Factory Reset | Initialization of the system with the default value. Refer to <Table 2-2> |
| 'M' or 'm' | Sets up MAC Address. <Warning> This value is not changed when Factory Reset. |
| E : Exit | Exit "Net Config" |

<Fig 2.8> is an example of setting the Source IP of EVB B/D in Network Config

```

-----
NETWORK CONFIG
-----
D : Display config
1 : Source IP
2 : Gateway IP
3 : Subnet Mask
4 : DNS Server IP
M : MAC address
A : memory Allocation
F : Factory reset
E : Exit
-----
Select ? 1
Source IP ? 192.168.0.100
    
```

<Fig 2.8 : Source IP Address Setup Example>

<Fig 2.9> is an example of setting the MAC address of EVB B/D in Network Config

```

-----
NETWORK CONFIG
-----
D : Display config
1 : Source IP
2 : Gateway IP
3 : Subnet Mask
4 : DNS Server IP
M : MAC address
A : memory Allocation
F : Factory reset
E : Exit
-----
Select ? m
MAC Address ? 00.08.dc.00.00.20
    
```

<Fig 2.9 : MAC address Setup Example>

2.4.1.2. Channel Config

It sets up an application that can be operated in 4 channels of W5100. By selecting '2 : Channel Config', each channel can be set up. The default W5100 channel information is shown in <Table 2-4>.

<Table 2-4 : EVB B/D Default Channel Information>

| W5100 Channel | Test Application |
|-----------------|---------------------------------|
| 1 st | Loopback TCP Server (Port 5000) |
| 2 nd | Loopback TCP Server (Port 5000) |
| 3 rd | Loopback TCP Server (Port 5000) |
| 4 th | Loopback TCP Server (Port 5000) |

If "Channel Config" menu is selected in manage program, <Fig 2.10> is displayed and the functionality of each menu is described in <Table 2-5>.

```

Select ? 2
-----
CHANNEL CONFIG
-----
0 : Display Config
1 : 1st Channel
2 : 2nd Channel
3 : 3rd Channel
4 : 4th Channel
F : Factory Reset
E : Exit
-----
Select ?
    
```

<Fig 2.10 : Menu of Channel Config>

<Table 2-5 : Menu of Channel Config>

| Menu | Description |
|-----------------------------|---|
| D : Display Config | Displays current set up Test Application type of each W5100 channel |
| 0 : 1 st Channel | Sets up test application type at W5100 No. "0" channel <Warning> As developing EVB B/D, DHCP Client application setup is possible only at no. "0" channel. |
| 1 : 2 nd Channel | Sets up test application type at W5100 no. "1" channel |
| 2 : 3 rd Channel | Sets up test application type at W5100 no. "2" channel |
| 3 : 4 th Channel | Sets up test application type at W5100 no. "3" channel |
| F : Factory Reset | Initialize into original setup status. Refer to <Table 2-4> |
| E : Exit | Exits "Channel Config" |

Available test application of each W5100 channel is shown as <Table 2-6>

<Table 2-6 : W5100 Channel Application Type>

| Application Type | Description |
|---------------------|---|
| No Use | Not used |
| DHCP Client | Receiving Network Information of EVB B/D from DHCP Server dynamically <Warning> If DHCP Server does not exist in LAN, it sets back to default value after certain amount of time |
| TCP Loopback Server | TCP Server Test Program <Warning> EVB B/D : TCP Server, AX1 : TCP Client |
| TCP Loopback Client | TCP Client Test Program <Warning> EVB B/D : TCP Client, AX1 : TCP Server |
| Loopback UDP | UDP Test Program |
| Web Server | Web Server Test Program |

Other application types except for "DHCP Client" can be repeatedly set up regardless of channel.

<Fig 2.11> shows an example of 2nd channel setting of W5100 as "TCP Loopback Client"

When inputting simply [ENTER] without IP address or port number, the default value is applied. <Table 2-7> shows default values required for each application.

```

Select ? 2
Select the followed APPs type for 1 channel.
    0 : No Use
    2 : Loop-Back TCP Server
    3 : Loop-Back TCP Client
    4 : Loop-Back UDP
    5 : Web Server
Select ? 3
Server IP Address ?
Default Applied. 192.168.0.3
Server Port Num (1~65535) ?
Default Applied. 3000
    
```

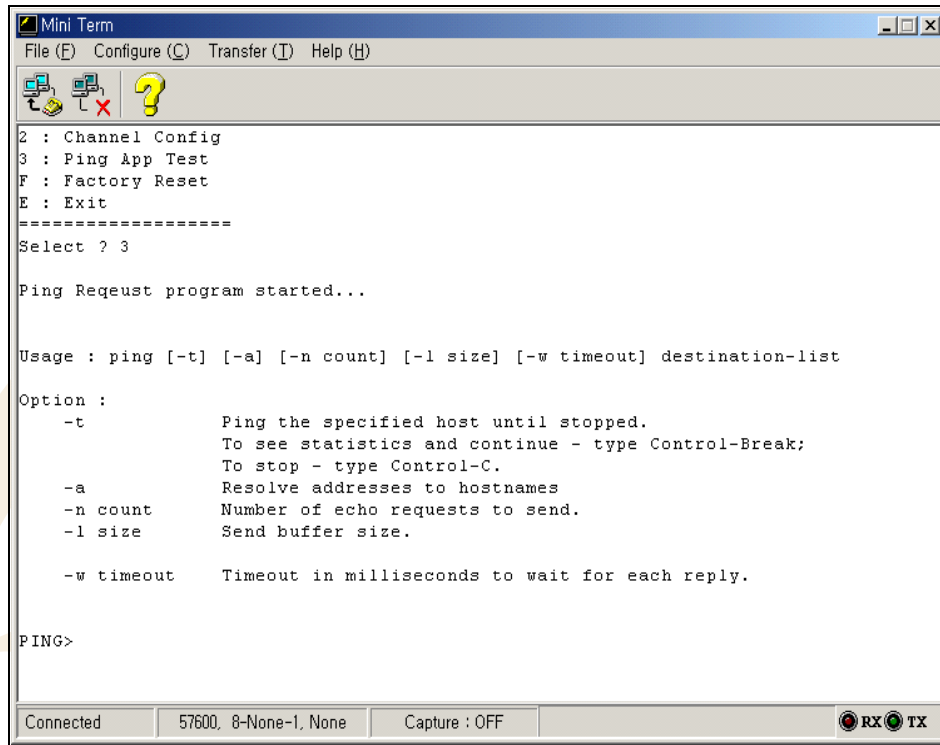
<Fig 2.11 : Loopback TCP Client Application Setting Example>

< Table 2-7 Application Default Value >

| Application Type | Default Value |
|---------------------|--|
| DHCP Client | None |
| TCP Loopback Server | Listen Port Number : 5000 |
| TCP Loopback Client | Server IP Address : 192.168.0.3 Server Port Number : 3000 |
| Loopback UDP | Source Port Number : 3000 |
| Web Server | HTTP Port Number : 80 |

2.4.1.3. Ping Application Test

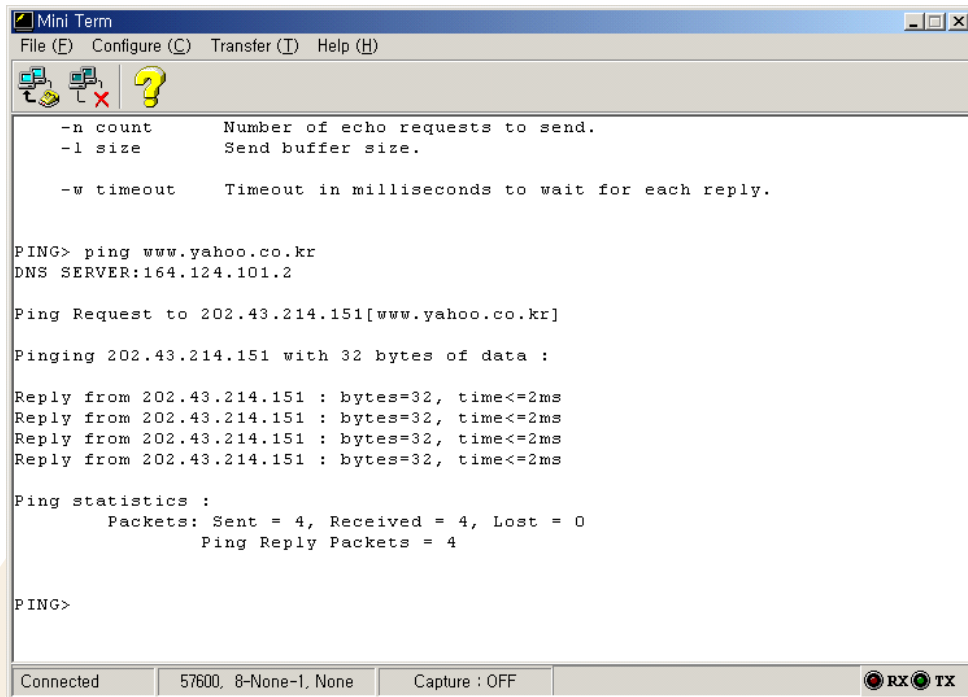
Ping Application Test is a program created for IP RAW channel evaluation of W5100 and sends Ping request to certain peer and receives Ping Reply. This program is set up identically with the ping command in the DOS prompt. It's executed when '3' is chosen <Fig 2.6 : Manage Program Execution>.



<Fig 2.12 : Usage of Ping Application >

<Fig 2.12> displays the execution screen of Ping Application and shows how to use the Ping Application.

<Fig 2.13> shows the real example of sending the Ping Request to the destination and receiving the Ping Reply.



```

Mini Term
File (E)  Configure (C)  Transfer (T)  Help (H)

-n count      Number of echo requests to send.
-l size       Send buffer size.

-w timeout    Timeout in milliseconds to wait for each reply.

PING> ping www.yahoo.co.kr
DNS SERVER:164.124.101.2

Ping Request to 202.43.214.151[www.yahoo.co.kr]

Pinging 202.43.214.151 with 32 bytes of data :

Reply from 202.43.214.151 : bytes=32, time<=2ms
Reply from 202.43.214.151 : bytes=32, time<=2ms
Reply from 202.43.214.151 : bytes=32, time<=2ms
Reply from 202.43.214.151 : bytes=32, time<=2ms

Ping statistics :
    Packets: Sent = 4, Received = 4, Lost = 0
            Ping Reply Packets = 4

PING>

Connected  57600, 8-None-1, None  Capture : OFF  RX TX
    
```

<Fig 2.13 : Ping Application Test>

To terminate the Ping Application type, type “exit” at the “PING>” prompt.

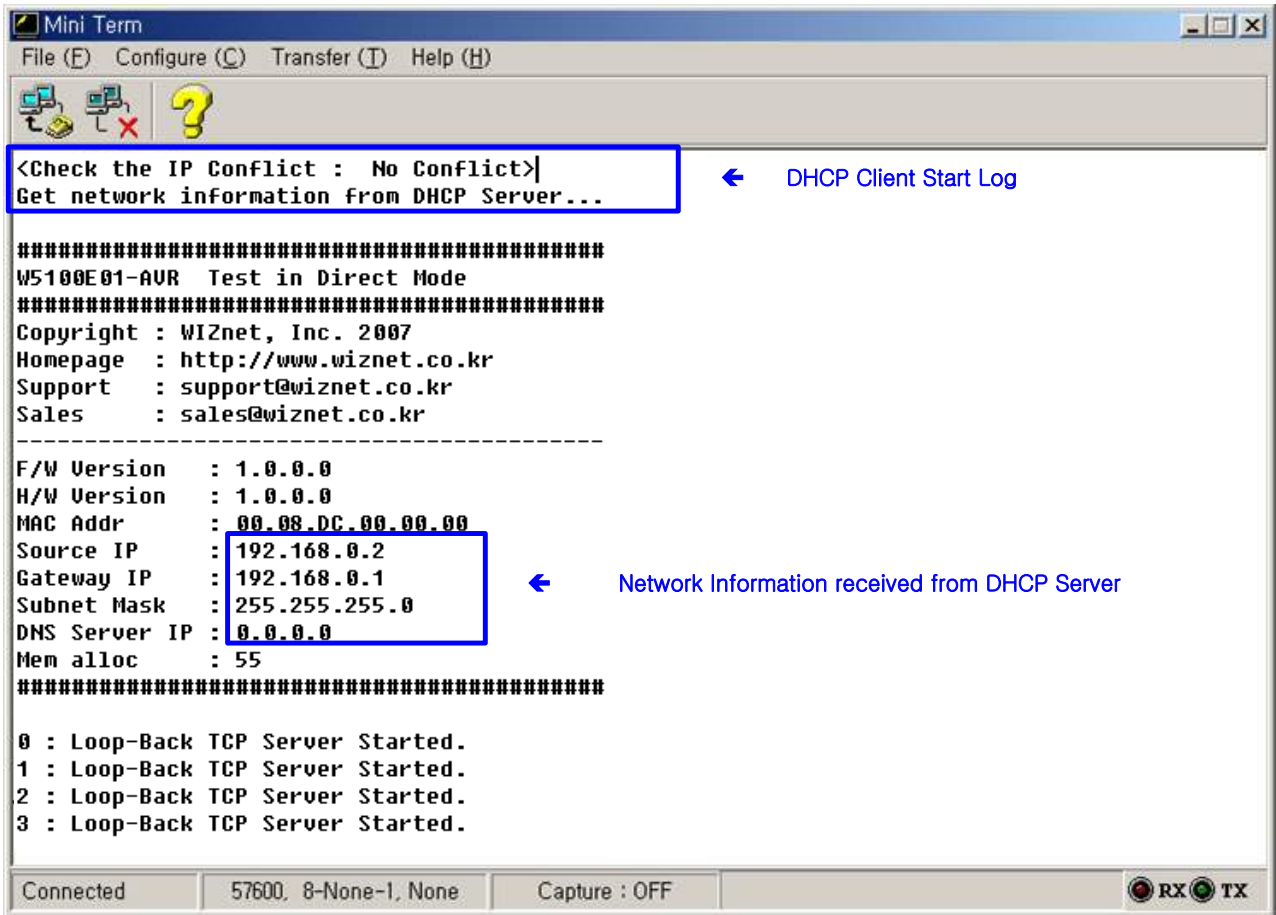
2.4.2. EVB B/D Test Applications

2.4.2.1. DHCP Client

DHCP Client Application is an application that dynamically assigns network information for EVB B/D from DHCP Server. To test DHCP Client, first of all, W5100 1st channel application type must be set up as “DHCP Client” using [Manager>>Channel Config>>0th Channel] Menu.

Refer to [Chapter 2.4.1.2](#)

<Fig 2.14> is the screen that DHCP Client successfully obtains network information. Note that DHCP Client will be set with default network information if DHCP Server does not exist or is not able to obtain network information from DHCP Server.



```

Mini Term
File (F)  Configure (C)  Transfer (T)  Help (H)

<Check the IP Conflict : No Conflict>|
Get network information from DHCP Server...

DHCP Client Start Log

#####
W5100E01-AVR Test in Direct Mode
#####
Copyright : WIZnet, Inc. 2007
Homepage  : http://www.wiznet.co.kr
Support   : support@wiznet.co.kr
Sales     : sales@wiznet.co.kr
-----
F/W Version : 1.0.0.0
H/W Version : 1.0.0.0
MAC Addr    : 00.08.DC.00.00.00
Source IP   : 192.168.0.2
Gateway IP  : 192.168.0.1
Subnet Mask : 255.255.255.0
DNS Server IP : 0.0.0.0
Mem alloc   : 55
#####

0 : Loop-Back TCP Server Started.
1 : Loop-Back TCP Server Started.
2 : Loop-Back TCP Server Started.
3 : Loop-Back TCP Server Started.

Connected  57600, 8-None-1, None  Capture : OFF  RX TX
    
```

<Fig 2.14 : DHCP Client Test>

2.4.2.2. Loopback TCP Server

Loopback TCP Server Application is an application that loops back any file or packet data through TCP channel connected with “AX1” Program of Test PC. First of all, set any channel as “Loopback TCP Server” application type using [Manager>>Channel Config] menu of EVB B/D to test Loopback TCP Server.

When setting up “Loopback TCP Server” application type of EVB B/D, you can set listen port to any value.

Here, it's set as the default value, 5000. Refer to [Chapter 2.4.1.2](#)

After the setup of EVB B/D is complete, run “AX1” at Test PC then try the connection to the IP Address.

When the connection between EVB B/D and “AX1” is successful, loop back the data. Refer to “**AX1 Manual Vx.x.pdf**”


```

Source IP      : 192.168.0.2
Gateway IP    : 192.168.0.1
Subnet Mask   : 255.255.255.0
DNS Server IP : 0.0.0.0
MAC Addr     : 0x00.0x08.0xDC.0x00.0x00.0x35
#####

0 : Loop-Back TCP Server Started.
1 : Loop-Back TCP Server Started.
2 : Loop-Back TCP Server Started.
3 : Loop-Back TCP Server Started.
0 : Connected by 192.168.0.30(2313)
Peer Connection Information in 0 channel of W5100

```

<Fig 2.15 : Loopback TCP Server Test>

2.4.2.3. Loopback TCP Client

Loopback TCP Client Application is an application that loops back any file and packet data through TCP channel connected with “AX1” Program of Test PC

After running the “AX1” on the server, set any channel of W5100 as “Loopback TCP Client” application type using [Manager>>Channel Config] menu of EVB B/D.

When setting up the “Loopback TCP Client” Application type of EVB B/D, set the Server IP as the IP Address of the Test PC and set Server Port as the waiting Server Port Number(3000). Refer to [Chapter 2.4.1.2](#).

After setting up EVB B/D is complete, exit from the manager program and run EVB Test Application. If EVB B/D is connected to “AX1” successfully, loop back the desired data. Refer to **“AX1 Manual Vx.x.pdf”**

```

Source IP      : 192.168.0.2
Gateway IP    : 192.168.0.1
Subnet Mask   : 255.255.255.0
DNS Server IP : 0.0.0.0
MAC Addr     : 0x00.0x08.0xDC.0x00.0x00.0x35
#####

0 : Loop-Back TCP Server Started.
1 : Loop-Back TCP Client Started.
2 : Loop-Back TCP Server Started.
3 : Loop-Back TCP Server Started.
1 : Connected by 192.168.0.30(2827) ← Peer Connection Information
                                     in 1 channel of W5100

```

<Fig 2.16 : Loopback TCP Client>

2.4.2.4. Loopback UDP

Loopback UDP Application is an application that loops back any file and packet data through UDP Channel connected with “AX1” Program of Test PC. First of all, to test Loopback UDP, set up any channel of W5100 as “Loopback UDP” Application Type using [Manager>>Channel Config] Menu of EVB B/D.

In setting up “Loopback UDP” Application type, set Source Port as any value. Here, it's set with 3000. Refer to [Chapter 2.4.1.2](#)

After EVB B/D setup is over, loop back desired data with IP Address and UDP Source Port of EVB B/D using menu or Icon related to UDP.

Refer to “AX1 Manual Vx.x.pdf”.

```

Source IP      : 192.168.0.2
Gateway IP    : 192.168.0.1
Subnet Mask   : 255.255.255.0
DNS Server IP : 0.0.0.0
MAC Addr     : 0x00.0x08.0xDC.0x00.0x00.0x35
#####

0 : Loop-Back TCP Server Started.
1 : Loop-Back TCP Client Started.
2 : Loop-Back UDP Started. ← Loopback UDP Application Log
3 : Loop-Back TCP Server Started.
    
```

<Fig 2.17 : Loopback UDP Test>

2.4.2.5. Web Server

Web Server Application sends and receives web pages and EVB B/D control data through HTTP Channel connected with web browser. For Web Server testing, set up any channel of W5100 as “Web Server” Application Type using [Manager>>Channel Config] menu of EVB B/D.

When setting up “Web Server” Application Type of EVB B/D, set HTTP port as any value. Here, it’s set to 80, the default value. Refer to [Chapter 2.4.1.2](#).

After setup for EVB B/D, run Web browser in the Test PC, type the URL(<http://192.168.0.2/>) of the EVB B/D in the address field and connect to EVB B/D.

```

Source IP      : 192.168.0.2
Gateway IP    : 192.168.0.1
Subnet Mask   : 255.255.255.0
DNS Server IP : 0.0.0.0
MAC Addr     : 0x00.0x08.0xDC.0x00.0x00.0x35
#####

0 : Loop-Back TCP Server Started.
1 : Loop-Back TCP Client Started.
2 : Loop-Back UDP Started.
3 : Web Server Started. ← Web Server Application Log and
3 : Connected by 192.168.0.30(2313) Peer Connection Information
    
```

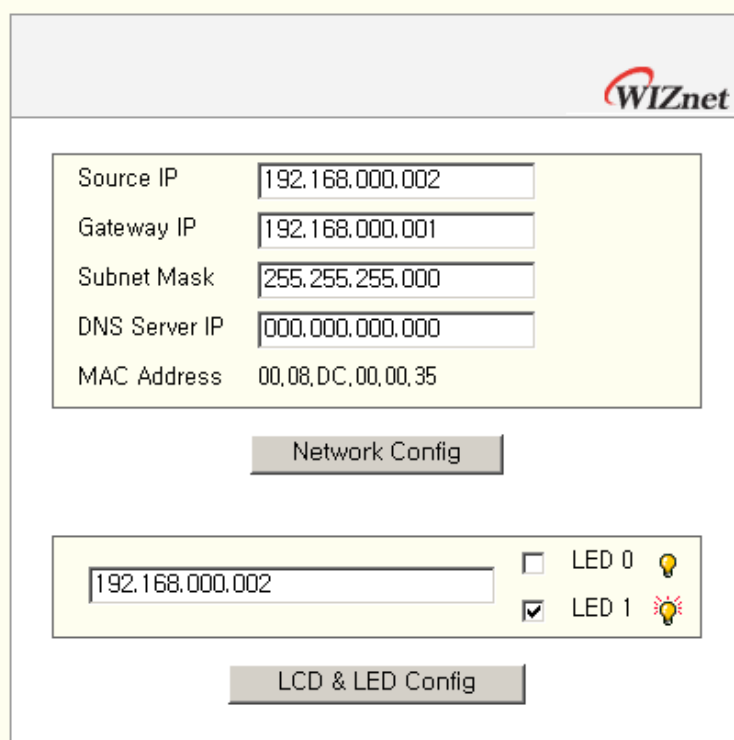
<Fig 2.18 : Web Server Test>

If the web browser is successfully connected to HTTP port of EVB B/D, the Web Page of <Fig 2.19> can be viewed. In case Web Page of <Fig 2.19> is not shown, refresh the screen using the “Refresh” function of the web browser.



<Fig 2.19 : Default Web Page of EVB B/D>

If [Control] button on the Web Page in <Fig 2.19> is clicked, it can set the network information or show the web page that can turn on or off LEDs(D3,D4) and display rows of text on Text LCD display.



<Fig 2.20 : Web Page of EVB B/D Control>

2.5. Troubleshooting Guide

2.5.1. Ping

When you can not reach EVB B/D by Ping command,

Step 1. Check if you connect correctly test PC and EVB B/D with UTP cable.

Step 2. Check if interface jumper of JP3 is correctly set.

JP3 : SPI mode (pin2-3 should be connected), Bus mode(pin1-2 should be connected)

Step 3. Check if you correctly change your test PC's network environment (IP address, Gateway, Subnet)? If not, you should change it as follows:

- IP address: 192.168.0.3
- Gateway address: 192.168.0.1
- Subnet Mask: 255.255.255.0

Step 4. Check if link LED of MAGJACK(left LED from rear view) is on? If it is off, check UTP cable is working properly.

2.5.2. Misc.

When the serial terminal screen remains blank with the power on after a connection is made

Step 1. Check the connection condition of the serial cable.

Step 2. Check the COM Port numbers of the PC and terminal coincide.

Step 3. Check the terminal's baud rate 57600.

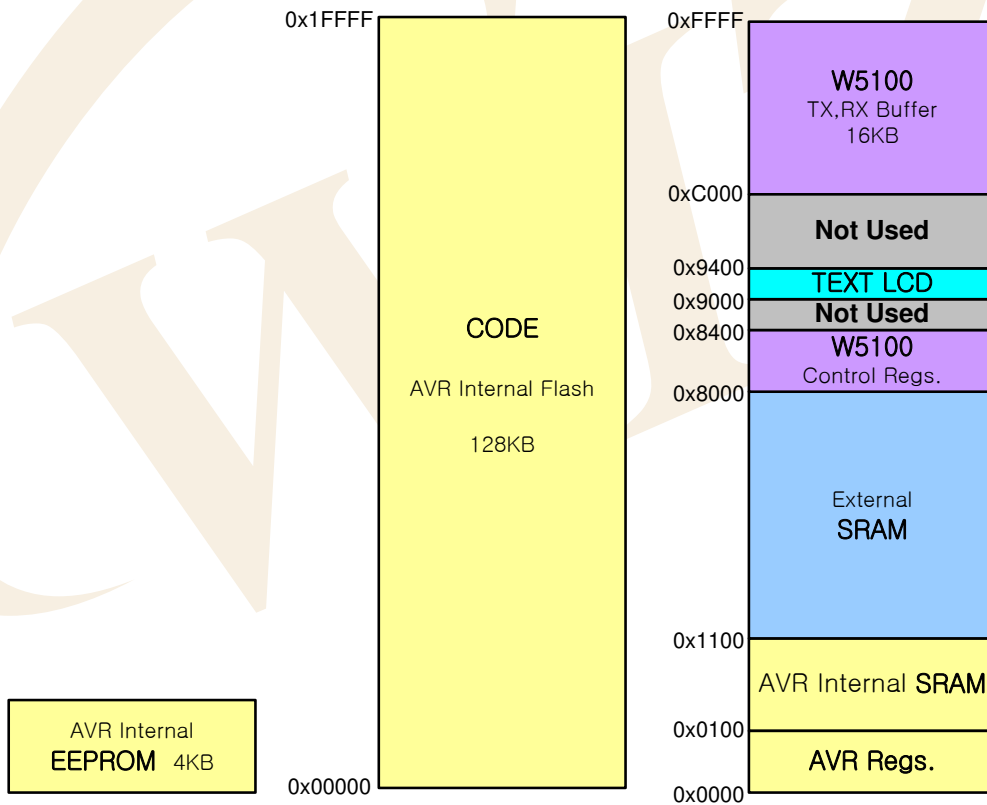
3. Programmer's Guide

3.1. Memory Map

3.1.1. Code & Data Memory Map

Memory Map of EVB B/D is composed of code memory 128 Kbytes and data memory 64Kbytes. Data memory is divided into SRAM, W5100, and Text LCD Area. Other than these, there is 4Kbytes AVR Internal EEPROM. Various types of environmental variables are recorded on this EEPROM.

<Fig 3.1>, <Table 3-1> are representations of System Memory Map of EVB B/D.



<Fig 3.1: EVB B/D Memory Map>

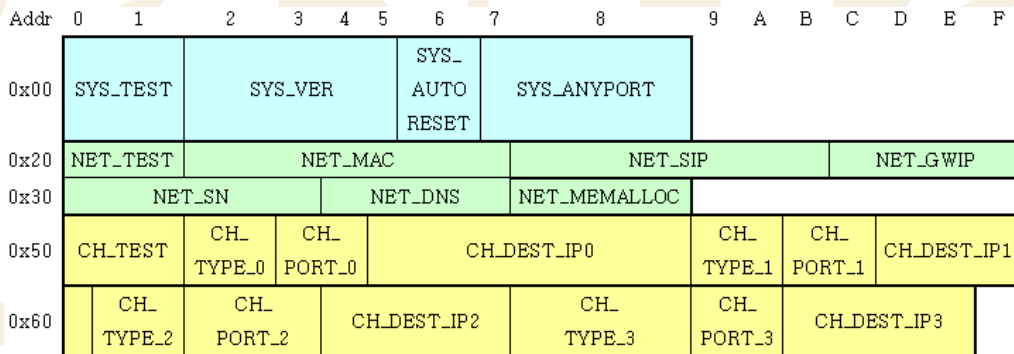
<Table 3-1: Device MAP Definition>

| Device | Map Define | Source Code |
|-----------------|--|-------------|
| W5100 | <pre>#define __DEF_IINCHIP_MAP_BASE__ 0x8000 #if (__DEF_IINCHIP_BUS__ == __DEF_IINCHIP_DIRECT_MODE__) #define COMMON_BASE __DEF_IINCHIP_MAP_BASE__ #else #define COMMON_BASE 0x0000 #endif #define __DEF_IINCHIP_MAP_TXBUF__ (COMMON_BASE + 0x4000) #define __DEF_IINCHIP_MAP_RXBUF__ (COMMON_BASE + 0x6000)</pre> | mcu/types.h |
| Text LCD | <pre>#define LCD_BASEADDR 0x9000</pre> | evb/lcd.h |

3.1.2. AVR Internal EEPROM MAP

<Fig 3.2>, <Table 3.2> are representations of AVR Internal EEPROM Map.

Refer to "evb/config.h" and "evb/config.c."


<Fig 3.2: AVR Internal EEPROM Map>

<Table 3-2: AVR Internal EEPROM MAP Definition>

| | | |
|----------------------------|-----------------------|---------------------|
| System Information | #define SYS_INFO | 0x00 |
| | #define SYS_TEST | (SYS_INFO) |
| | #define SYS_VER | (SYS_TEST + 2) |
| | #define SYS_AUTORESET | (SYS_VER + 4) |
| | #define SYS_ANY_PORT | (SYS_AUTORESET + 1) |
| Network Information | #define NET_CONF | 0x20 |
| | #define NET_TEST | (NET_CONF) |
| | #define NET_MAC | (NET_TEST+2) |
| | #define NET_SIP | (NET_MAC + 6) |
| | #define NET_GWIP | (NET_SIP + 4) |
| | #define NET_SN | (NET_GWIP + 4) |
| | #define NET_DNS | (NET_SN + 4) |
| #define NET_MEMALLOC | (NET_DNS + 4) | |
| Channel Information | #define CH_CONF | 0x50 |
| | #define CH_TEST | (CH_CONF) |
| | #define CH_TYPE_0 | (CH_TEST + 2) |
| | #define CH_PORT_0 | (CH_TYPE_0 + 1) |
| | #define CH_DESTIP_0 | (CH_PORT_0 + 2) |
| | #define CH_TYPE_1 | (CH_DESTIP_0 + 4) |
| | #define CH_PORT_1 | (CH_TYPE_1 + 1) |
| | #define CH_DESTIP_1 | (CH_PORT_1 + 2) |
| | #define CH_TYPE_2 | (CH_DESTIP_1 + 4) |
| | #define CH_PORT_2 | (CH_TYPE_2 + 1) |
| | #define CH_DESTIP_2 | (CH_PORT_2 + 2) |
| | #define CH_TYPE_3 | (CH_DESTIP_2 + 4) |
| | #define CH_PORT_3 | (CH_TYPE_3 + 1) |
| #define CH_DESTIP_3 | (CH_PORT_3 + 2) | |

3.1.2.1. System Information

System Information area is used in recording System Information such as Firmware Version of EVB B/D.

<Table 3-3: System Information>

| Name | Description | Default Value |
|---------------|---|---|
| SYS_TEST | Valid Check of System Information | 0xA5A5 – Valid Others – Invalid |
| SYS_VER | F/W Version | 0xAABBCCDD (AA.BB.CC.DD) |
| SYS_AUTORESET | Auto reset check in case of setting up any environmental variable | 0x01 – System Auto Reset Others – No Reset |
| SYS_ANY_PORT | Using Any Port Number at Socket creation | 1000 ~ 65535 |

System Information is accessed as SYSINFO Data Type.

<Table 3-4: SYSINFO Data Type Definition>

| Type Definition | Instance |
|---|--------------------------|
| <pre>typedef struct _SYSINFO { u_int test; u_long ver; u_char auto_reset; u_int any_port; }SYSINFO;</pre> | SYSINFO SysInfo ; |

<Table 3-5: System Information Access Functions>

| Function | Description |
|-------------------------------------|-----------------------------|
| void set_sysinfo(SYSINFO* pSysInfo) | Save the System Information |
| void get_sysinfo(SYSINFO* pSysInfo) | Get the System Information |

3.1.2.2. Network Information

Network Information is used in recording Network Configuration information to be used for EVB B/D.

<Table 3-6: Network Information>

| Name | Description | Default Value |
|--------------|------------------------------------|------------------------------------|
| NET_TEST | Valid check of Network Information | 0xA5A5 – Valid Others – Invalid |
| NET_SIP | Source IP Address | 0xC0A80002 (192.168.0.2) |
| NET_GWIP | Gateway IP Address | 0xC0A80001 (192.168.0.1) |
| NET_SN | Subnet Mask | 0xFFFFFFFF (255.255.255.0) |
| NET_DNS | DNS Server IP Address | 0x00000000 (0.0.0.0) |
| NET_MEMALLOC | W5100 Memory Allocation | 0x55 |

Network Information is accessed as NETCONF Data Type.

<Table 3-7: NETCONF Data Type Definition>

| Type Definition | Global Instance |
|--|--------------------------|
| <pre>typedef struct _NETCONF { u_int test; u_char mac[6]; u_long sip; u_long gwip; u_long sn; u_long dns; u_char mem_alloc; }NETCONF;</pre> | NETCONF NetConf ; |

<Table 3-8: Network Information Access Functions>

| Function | Description |
|-------------------------------------|------------------------------|
| void set_netconf(NETCONF* pNetConf) | Save the Network Information |
| void get_netconf(NETCONF* pNetConf) | Get the Network Information |

3.1.2.3. Channel Information

Following table introduces applications to be used in 4 channels of W5100.

<Table 3-9: Channel Information>

| Name | Description | Default Value |
|-------------|--|--|
| CH_TEST | Valid check of channel Information | 0xA5A5 – Valid Others – Invalid |
| CH_TYPE_X | Application type of No."X" Channel | Default - LB_TCPS <div style="border: 1px solid black; padding: 5px;"> NOTUSE : Not Used DHCP_CLIENT : DHCP Client LB_TCPS : Loopback TCP Server LB_TCPC : Loopback TCP Client LB_UDP : Loopback UDP WEB_SEVER : Web Server </div> |
| CH_PORT_X | Source / Destination Port number of No."X" | Little Endian LB_TCPS : Default Source Port, 0x5000 LB_TCPC : Default Destination Port, 0x3000 LB_UDP : Default Source Port, 0x3000 WEB_SERVER : 80 |
| CH_DESTIP_X | Destination IP address of No. "X" channel | 0xC0 A80003 (192.168.0.3) |

Channel Information is used for recording application type for 4 channels of W5100.

Channel application type includes Loopback TCP Server, Loopback TCP Client, Loopback UDP, DHCP Client, Web Server. Channel Information is defined as APPTYPE enumeration type.

<Table 3-10: Channel Application Type>

```
typedef enum _APPTYPE
{
    NOTUSE,
    DHCP_CLIENT,
    LB_TCPS,
    LB_TCPC,
    LB_UDP,
    WEB_SERVER
}APPTYPE;
```

Channel Information is accessed as CHCONF Data Type.

<Table 3-11: CHCONF Data Type Definition>

| Type Definition | Global Instance |
|--|------------------------|
| <pre>typedef struct _CHCONF { u_int test; struct _CH_CONF { u_char type; u_int port; u_long destip; }ch[4]; }CHCONF;</pre> | CHCONF ChConf ; |

<Table 3-12: Channel Information Access Function>

| Function | Description |
|----------------------------------|------------------------------|
| void set_chconf(CHCONF* pChConf) | Save the channel information |
| void get_chconf(CHCONF* pChConf) | Get the channel information |

3.2. EVB B/D Firmware

EVB B/D Firmware -EVB main()- can be divided into two parts. - Manage Program that sets up various environments for running EVB B/D and Loopback Programs that tests W5100 performance. There are Internet Application using Internet Protocols such as DHCP, HTTP, DNS, and ICMP.

Let's look at the source list of which EVB B/D is composed and then look at each application source.



3.2.1. Sources

<Table 3-13: EVB B/D Sources>

| Classification (Directory) | Files | Description |
|----------------------------|----------------------------|---|
| app | ping_app.h, ping_app.c | Ping Request App implementation |
| | loopback.h, loopback.c | TCP, UDP Loopback Apps implementation |
| | webserver.h, webserver.c | Webserver App implementation |
| mcu | delay.h, delay.c | Delay Function – wait_xxx() |
| | serial.h, serial.c | AVR UART control |
| | timer.h, timer.c | AVR Timer enable & disable |
| | types.h | AVR Data Type Definition, & Global Definition |
| evb | channel.h, channel.c | Channel App Handler registration & cancellation |
| | config.h, config.c | EVB B/D Environment |
| | evb.h, evb.c | EVB B/D initialization |
| | lcd.h, lcd.c | EVB B/D Text LCD control |
| | led.h, led.c | EVB B/D LED(D3,D4) control |
| | manage.h, manage.c | Manage App |
| inet | dhcp.h, dhcp.c | DHCP Client Protocol |
| | dns.h, dns.c | DNS Client Protocol |
| | httpd.h, httpd.c | HTTP Protocol |
| | ping.h, ping.c | Ping Protocol |
| main | main.h, main.c | EVB B/D F/W main() |
| rom | [webpage] | EVB B/D Web Pages |
| | romfs.h, romfs.c | EVB B/D Web Pages Image |
| | searchfile.h, searchfile.c | EVB B/D Web Page control |
| util | myprintf.h | printf() for debugging |
| | socketutil.h, socketutil.c | Utilities relating Socket |
| | util.h, util.c | Utilities |
| iinChip | iinchip_conf.h | System Dependant Definition of W5100 |
| | W5100.h, w5100.c | I/O Functions of W5100 |
| | socket.h, socket.c | Socket APIs for W5100 |

3.2.2. How to Compile

Sources of [Chapter 3.2.1](#) compile in bundle after arranging SRC items.

Compiling of W5100E01-AVR B/D firmware can be processed by using WINAVR and AVRSTUDIO. First, install the WINAVR and AVRSTUDIO at the PC. Then, open the firmware file, "~/sw/fw/W5100E01-AVR.aps" through AVRSTUDIO project file to perform the compiling easily.

Be sure to check compile setting detail at the Configuration option of Project menu of AVRSTUDIO. For the setting method, refer to AVR Studio User Guide.

As the firmware provided by WIZnet is based on AVR-GCC 3.4.6, it can not be operated correctly at another version of the comiler.

***만일 이전 AVR-GCC 3.4.3을 사용하실경우, "~/sw/fw/README.txt"파일을 참고하세요**

```

*****
//include "acu.define.h"
//define __MCU_AVR__ 1
#define __MCU_TYPE__ __MCU_AVR__

//--- Refer "Don File Maker Manual Vx.x.pdf"
#include <avr/pgmspace.h>

#define __ENDIAN_LITTLE__ 0 /*<< This must be defined if system is little-endian alignment */
#define __ENDIAN_BIG__ 1
#define __SYSTEM_ENDIAN__ __ENDIAN_LITTLE__

#define MAX_SOCKET_NUM 4 /*<< Maximum number of socket */
#define CLK_CPU 800000 /*<< 8Mhz(for serial) */

/* ## __DEF_IINCHIP_XXX__ : define option for iinchip driver ***** */
#define __DEF_IINCHIP_DBG__ /*<< Involve debug code in driver (socket.c) */
#define __DEF_IINCHIP_INT__ /*<< Involve interrupt service routine (socket.c) */
#define __DEF_IINCHIP_PPP__ /*<< Involve pppoe routine (socket.c) */
/*<< If it is defined, the source files(md5.h,md5.c) must be included in your project.
Otherwise, the source files must be removed in your project. */

#define __DEF_IINCHIP_DIRECT_MODE__ 1
#define __DEF_IINCHIP_INDIRECT_MODE__ 2
#define __DEF_IINCHIP_SPI_MODE__ 3
#define __DEF_IINCHIP_BUS__ __DEF_IINCHIP_DIRECT_MODE__
//define __DEF_IINCHIP_BUS__ __DEF_IINCHIP_INDIRECT_MODE__
//define __DEF_IINCHIP_BUS__ __DEF_IINCHIP_SPI_MODE__ /*Enable SPI_mode*/

/**
__DEF_IINCHIP_MAP_XXX__ : define memory map for iinchip
*/
#define __DEF_IINCHIP_MAP_BASE__ 0x8000
#if (__DEF_IINCHIP_BUS__ == __DEF_IINCHIP_DIRECT_MODE__)
#define COMMON_BASE __DEF_IINCHIP_MAP_BASE__
#else
#define COMMON_BASE 0x0000
#endif
#define __DEF_IINCHIP_MAP_TXBUF__ (COMMON_BASE + 0x4000) /* Internal Tx buffer address of the iinchip */
#define __DEF_IINCHIP_MAP_RXBUF__ (COMMON_BASE + 0x6000) /* Internal Rx buffer address of the iinchip */

Build
avr-objcopy -O ihex -R .eeprom W5100E01-AVR.eif W5100E01-AVR.hex
avr-objcopy -O .eeprom --set-section-flags=.eeprom="alloc,load" --change-section-lma .eeprom=0 -O ihex W5100E01-AVR.eif W5100E01-AVR.eep

AVR Memory Usage
-----
Device: atmega128

Program: 82334 bytes (62.8% Full)
(.text + .data + .bootloader)

Data: 2154 bytes (52.6% Full)
(.data + .bss + .noinit)

Build succeeded with 0 Warnings...
    
```

After compile is completed, a hex file will be created in the folder that user defined before. This file will be programmed in Atmega128.

< Table 3-14 : W5100's DEFINE Option (types.h) >

| | |
|---|--|
| #define <code>_ENDIAN_LITTLE_</code> | <code>0</code> |
| #define <code>_ENDIAN_BIG_</code> | <code>1</code> |
| #define <code>SYSTEM_ENDIAN</code> | <code>_ENDIAN_LITTLE_</code> |
| #define <code>__DEF_IINCHIP_DIRECT_MODE__</code> | <code>1</code> |
| #define <code>__DEF_IINCHIP_INDIRECT_MODE__</code> | <code>2</code> |
| #define <code>__DEF_IINCHIP_SPI_MODE__</code> | <code>3</code> |
| #define <code>__DEF_IINCHIP_BUS__</code> | <code>__DEF_IINCHIP_DIRECT_MODE__</code> |
| //#define <code>__DEF_IINCHIP_BUS__</code> | <code>__DEF_IINCHIP_INDIRECT_MODE__</code> |
| //#define <code>__DEF_IINCHIP_BUS__</code> | <code>__DEF_IINCHIP_SPI_MODE__</code> |

Since EVB B/D is Little-Endian system, `SYSTEM_ENDIAN` should be defined `_ENDIAN_LITTLE_` and used. If the target system is Big-Endian, the defined items should be defined `_ENDIAN_BIG_`. If W5100 is intended to be used as different mode other than Direct Bus Mode, use desired Bus Mode defined as `__DEF_IINCHIP_BUS__` instead of `__DEF_IINCHIP_DIRECT_MODE__`. If DEFINE OPTION of W5100 is changed, the sources must Re-Build. To Re-Build project, do "make clean", then "make". In case of SPI mode, be sure to change the configuration of JP3 in the W5100E01-AVR board. For more detail, refer to [Chapter 2.1.1 EVB B/D Layout & Configuration](#).

3.2.3. How to download

For downloading the hex file, we use AVRStudio and AVRISP Cable.

- 1) Connect AVRISP Cable to JP3 at the PM-A1.
- 2) Supply the power to EVB B/D.
- 3) Run AVRStudio.exe.
- 4) Select "ATmega128" in Device section.
- 5) Select the HEX file in FLASH section.
- 6) Click "Program" button.

Please refer to "AVR Tool Guide.pdf" for more information.

3.2.4. EVB B/D's main()

If we take closer look at `main()`, for certain amount of time, we wait for Manage Program from RS232 Terminal after initialization of board with board reset. At this point, if RS232 terminal displays the Manage Program entering command, EVB B/D environment such as network information and channel Information can be set and ping request program can be run.

If Manage Program is done or there is no entering command from RS232 terminal, the application for each of 4 channels of W5100 is executed and initialized using previously set network information.

<Fig 3.3> process procedure of EVB B/D `main()`. Refer to "main/main.c"

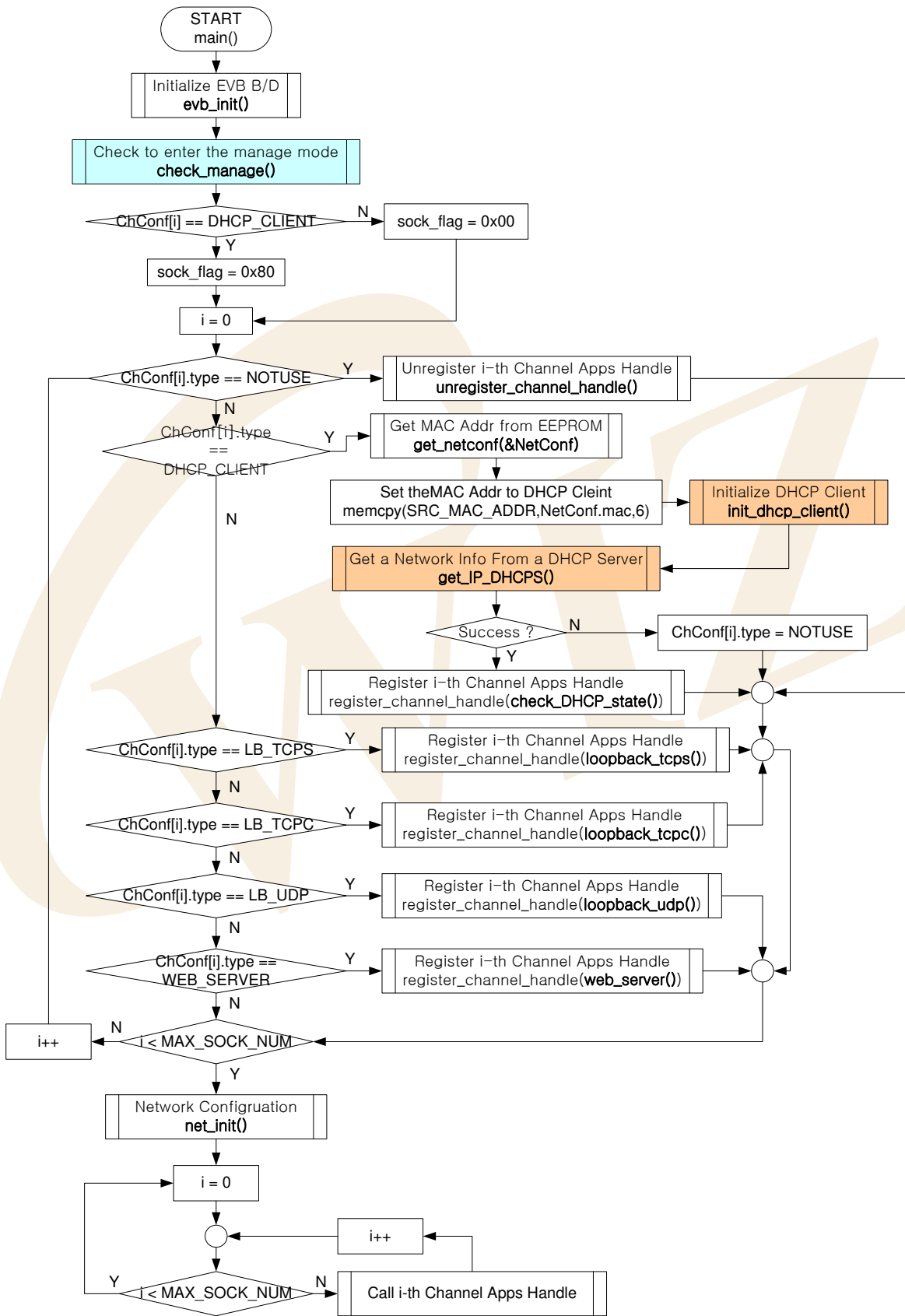
If DHCP client exists in the application, the DHCP client obtains the network information from DHCP server by calling 'get_IP_DHCP()' function. If DHCP client application does not exist or fails to obtain network information from DHCP server, the EVB B/D is initialized with previously-set network information.

After the initialization, it runs test applications of EVB B/D by calling each registered application handler. For further details on DHCP client program, refer to "[Chapter 3.2.6.5 DHCP Client.](#)"



<Table 3-15: Reference Functions in EVB B/D's main()>

| Function Name | Description | Location |
|---|--|-----------------|
| int main(void) | EVB B/D main() | main/main.c |
| void evb_init(void) | AVR, Text LCD, UART initialization | evb/evb.c |
| void net_init(void) | EVB B/D Network initialization | evb/evb.c |
| void check_manage(void) | Manage Program action wait and execution | evb/manage.c |
| void register_channel_handler (u_char ch, void (*handler)(u_char)) | Channel Application Handler registration | evb/channel.c |
| void unregister_channel_handler (u_char ch) | Channel Application Handler cancellation | evb/channel.c |
| void init_dhcp_client(SOCKET s, void (*ip_update)(void), void (*ip_conflict)(void)) | DHCP Client Program initialization | inet/dhcp.c |
| u_int getIP_DHCP(SOCKET s) | Network Information acquisition from DHCP Server | inet/dhcp.c |
| void check_DHCP_state(SOCKET s) | Check to expire the leased time from DHCP server | inet/dhcp.c |
| void loopback_tcps(u_char ch) | Loopback - TCP Server | app/loopback.c |
| void loopback_tcpclient(u_char ch) | Loopback - TCP Client | app/loopback.c |
| void loopback_udp(u_char ch) | Loopback - UDP | app/loopback.c |
| void web_server(u_char ch) | Web Server Program | app/webserver.c |

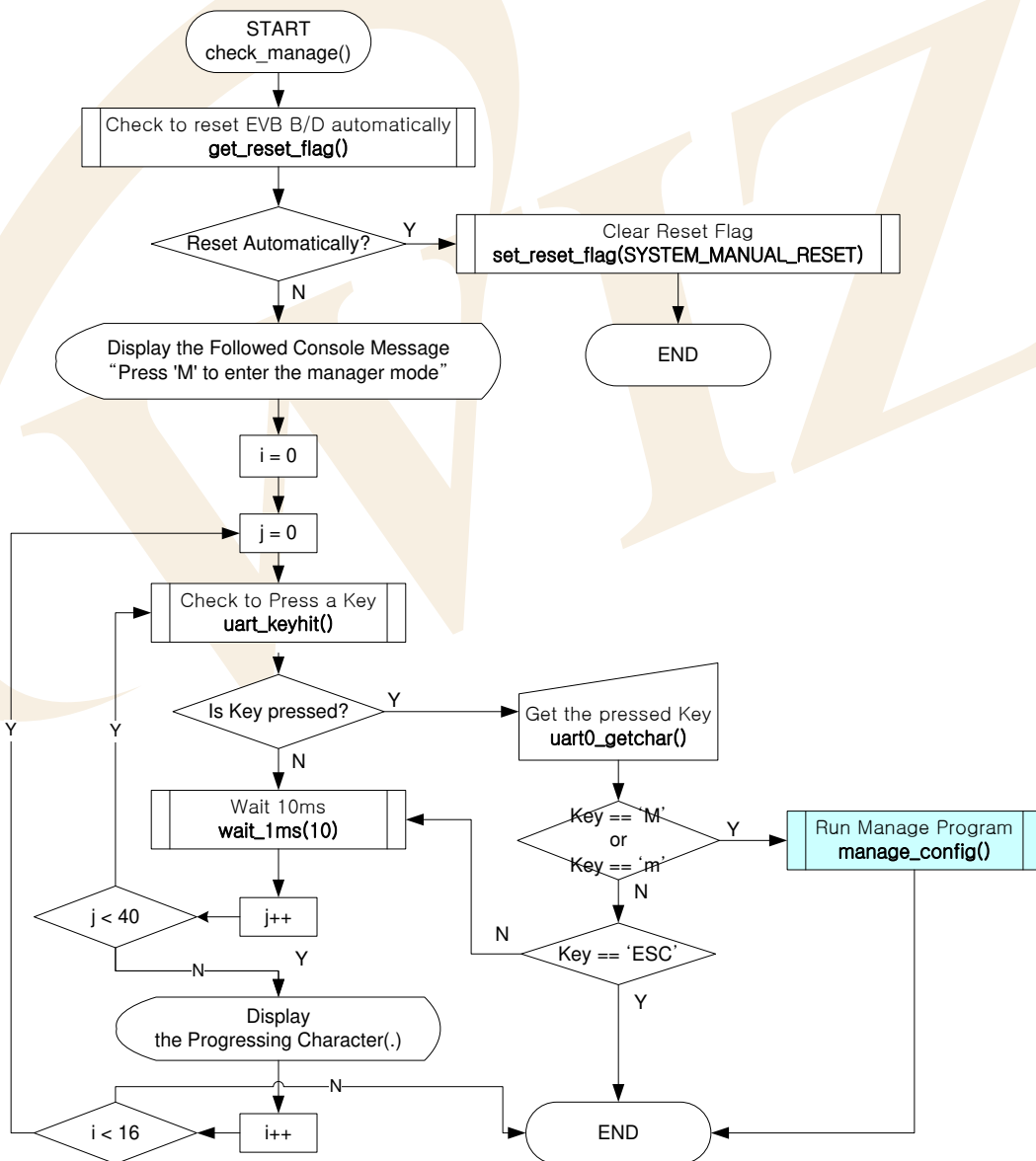


<Fig 3.3: EVB B/D's main()>

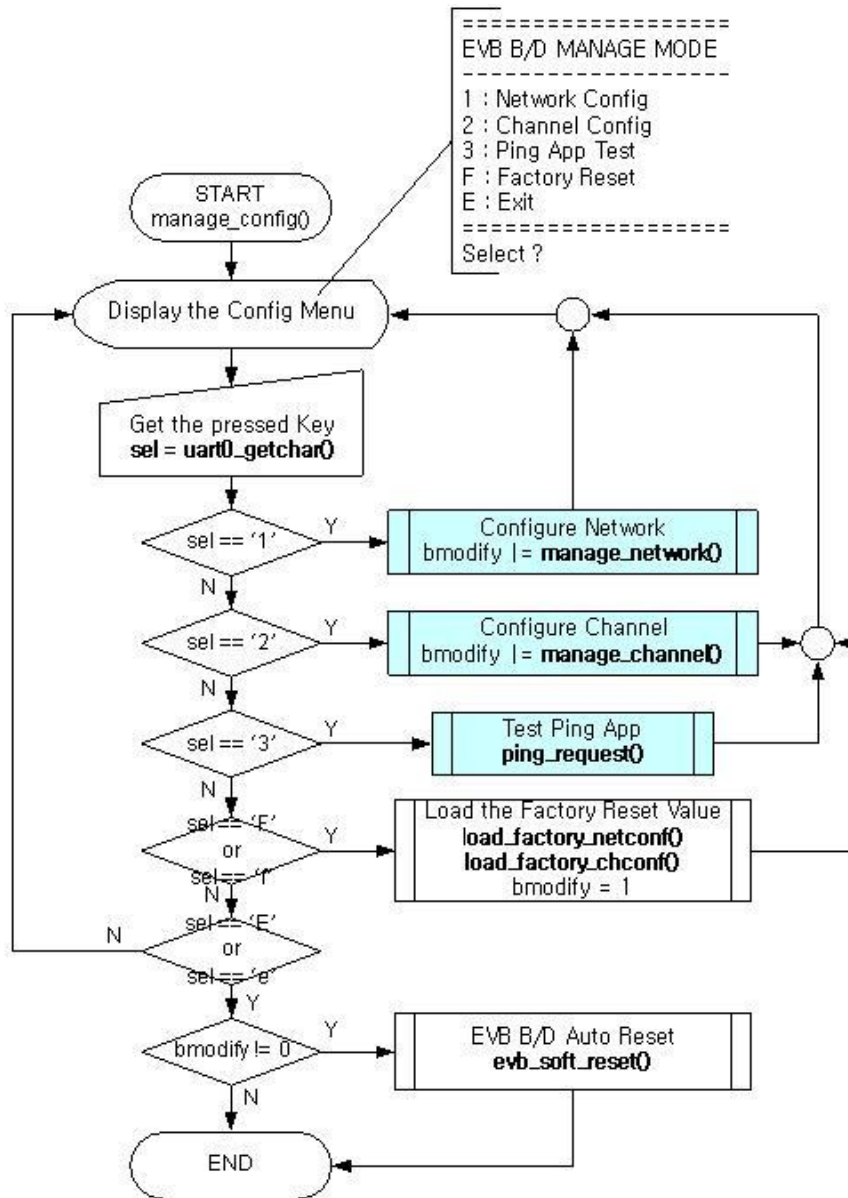
3.2.5. Manage Program

Manage Program is a program that sets up network and channel information through RS232 terminal and tests application by sending Ping Request to certain Destination.

Manage program can be started by calling `check_manage()` from `main()` function. `check_manage()` checks if there is any entering command to Manage Program from RS232 terminal - if character 'M' or 'm' is input or not. And if the command is detected, Manage Program will be entered through `manage_config()`. If the user change the configuration, the EVB B/D automatically reboots and `check_manage()` is skipped.



<Fig 3.4: check_manage()>



<Fig 3.5: manage_config()>

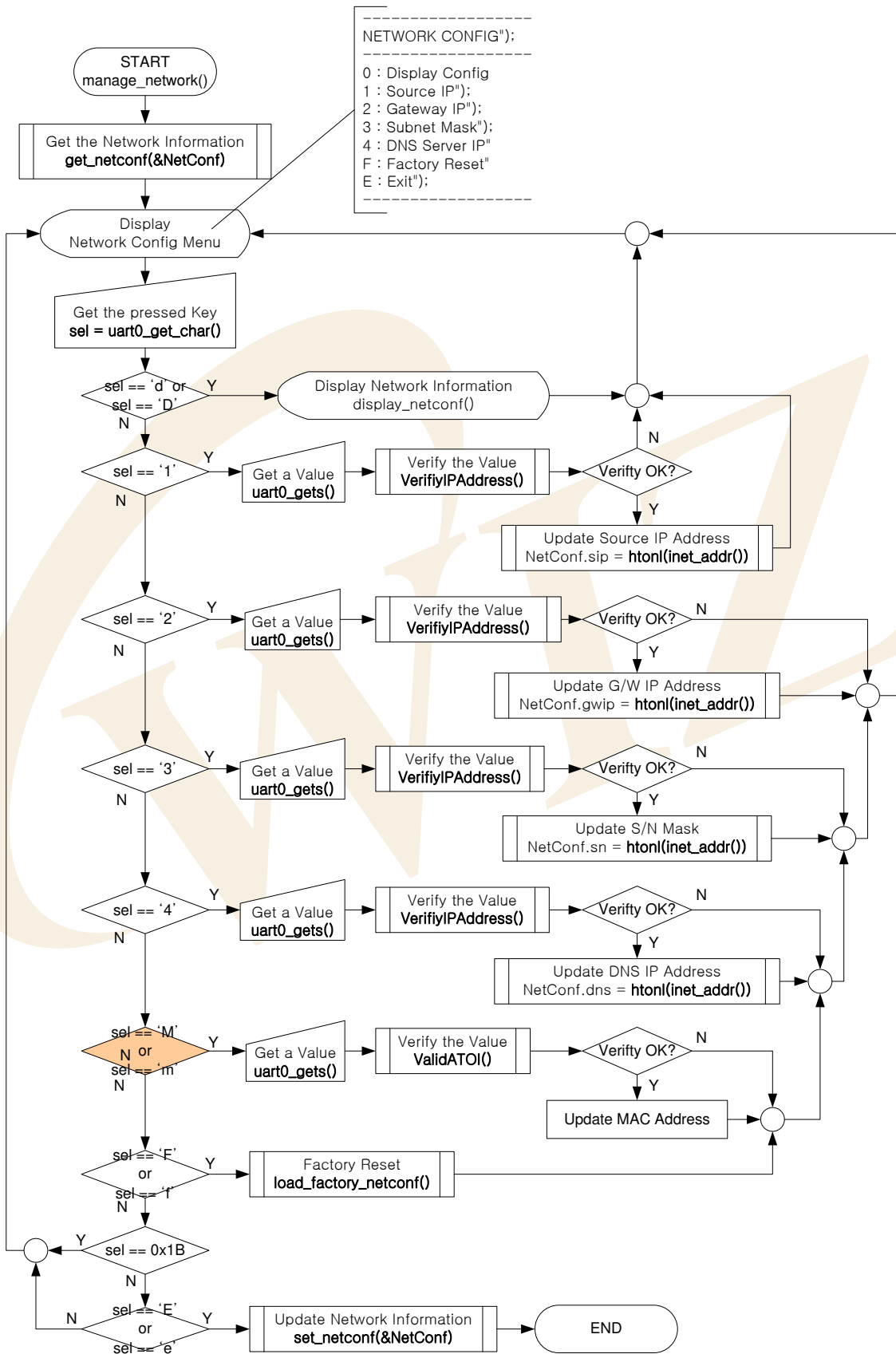
If the EVB B/D is updated, the EVB B/D automatically reboots to apply the updated configuration.

<Table 3-16: Caller Function at Manage Program >

| Function Name | Description | Location |
|----------------------------------|---|------------------------------|
| void check_manage(void) | Decision of Manage Program is executed or not | evb/manage.c |
| void manage_config(void) | Manage Program | evb/manage.c |
| u_char manage_network(void) | Configure Network Information | evb/manage.c |
| u_char manage_channel(void) | Configure Channel Information | evb/manage.c |
| u_char get_reset_flag(void) | EVB B/D's Auto/Manual Reset recognition and confirm Auto : SYSTEM_AUTO_RESET Manual : SYSTEM_MANUAL_RESET | evb/config.h evb/config.c |
| void set_reset_flag(u_char flag) | Copy of EVB B/D Reset status | evb/config.c |
| void load_factory_netconf(void) | Factory Reset Network Information | evb/config.c |
| void load_factory_chconf(void) | Factory Reset Channel Information | evb/config.c |
| u_int uart_keyhit(u_char uart) | Checking the Input from UART(0,1) | mcu/serial.c |
| char uart0_getchar(void) | Read one character from UART0 | mcu/serial.c |
| void wait_1ms(u_int cnt) | Delay Function | mcu/delay.c |
| void ping_request(void) | Ping Request Test Program | app/ping_app.c |

3.2.5.1. Network Configuration

Network Configuration is a sub-program of Manage Program and built with manage_network(). And it's the program that sets up Network Information of EVB B/D. In general, MAC Address of Network Information is hardly updated after the initial setup. Accordingly, MAC Address setup does not provide Configuration Menu such as Source IP, Gateway IP, or Subnet Mask but it provides hidden menu. Also, MAC Address is not changed at the time of Factory Reset. MAC Address is updated using 'M' or 'm'.



<Fig 3.6: manage_network()>

<Table 3-17: Reference Functions in manage_config(>

| Function Name | Description | Location |
|---|--|-----------------|
| u_char manage_network(void) | Configure Network Information | evb/manage.c |
| void get_netconf(NETCONF* pNetConf) | Get the Network Information that is previously set | evb/config.c |
| void set_netconf(NETCONF* pNetConf) | Update the Network Information | evb/config.c |
| void display_netconf(NETCONF* pNetConf) | Outputs the Network Information to the terminal | evb/config.c |
| Void load_factory_netconf(void) | Load Factory Reset Network Information | evb/config.c |
| char uart0_getchar(void) | Read one character from UART0 | mcu/serial.c |
| int uart_gets(u_char uart, char * str, char bpasswordtype, int max_len) | Read text lines from UART(0,1) | mcu/serial.c |
| char VerifyIPAddress(char* src) | Check if the string is IP Address | util/sockutil.c |
| Unsigned long htonl (unsigned long hostlong) | Transforms ordering of Long Type Data | util/sockutil.c |
| Unsigned long inet_addr (unsigned char* addr) | Transforms IP string into long type | util/sockutil.c |

3.2.5.2. Channel Configuration

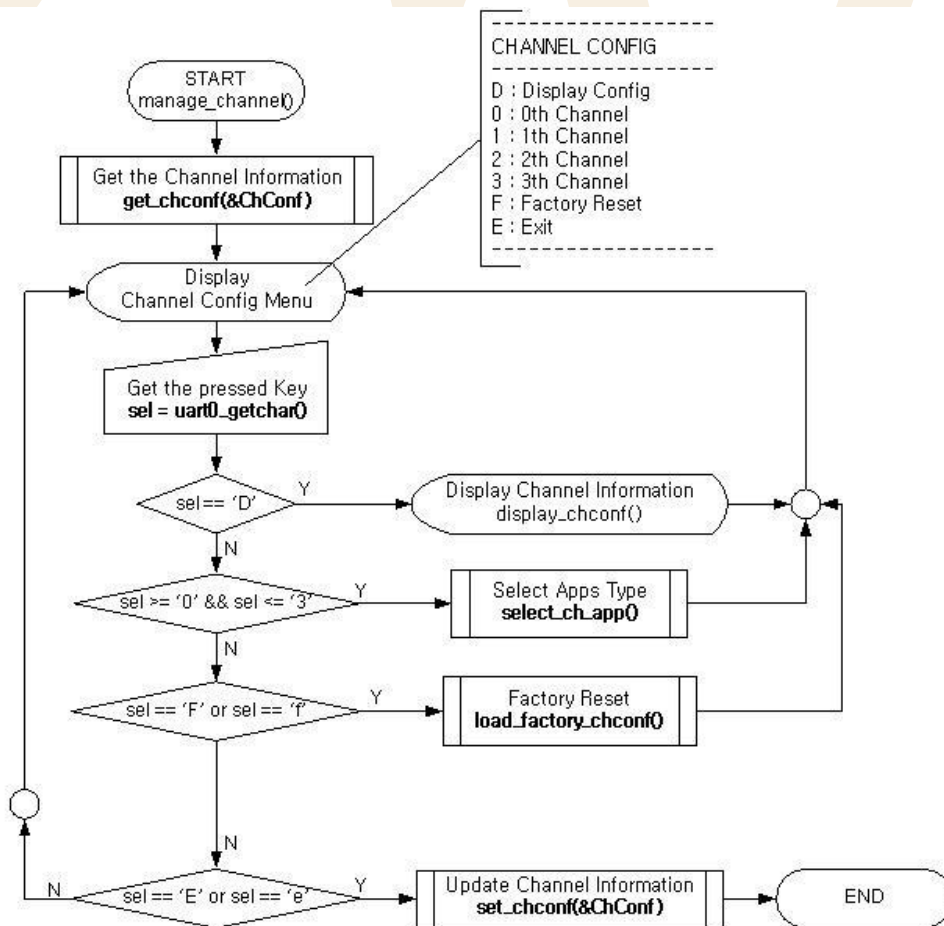
Channel Configuration, a sub-program of Manage Program is made of manage_config() and decides which application to be applied for each of 4 channels of W5100.

The application types that can be set up, are DHCP Client, Loopback TCP Server/Client, Loopback UDP, and Web Server Program. Each channel can be set up with any one of the applications above. However, the DHCP Client can only be supported by the first channel and the setting cannot be repeated on other channels.

TCP Server Program (LB_TCPS,WEB_SERVER) can be set repeatedly by all channels. In such case, the same port can be used. Here, the number of clients is as many as the same port number. Other applications can be set repeatedly by channels, but the same port number cannot be used.

<Table 3-18: Constraint by Application Types>

| APPTYPE | Repeat Setups | Port Repeat | Destination IP Setup |
|-------------|---------------|--|----------------------|
| DHCP_CLIENT | X | X | X |
| LB_TCPS | O | O, supports all the simultaneously connected clients as many as the number of repeated ports | X |
| LB_TCPC | O | X | O |
| LB_UDP | O | X | X |
| WEB_SERVER | O | O, supports all the simultaneously connected clients as many as the number of repeated ports | X |


<Fig 3.7: manage_channel()>

< Table 3-19: Reference Functions in manage_channel() >

| Function Name | Description | Location |
|--|--|--------------|
| u_char manage_channel(void) | Configure Channel Information | evb/manage.c |
| void select_ch_app (CHCONF* pChConf, u_char ch) | Select available Application Type and Setup required factors | evb/manage.c |
| void get_chconf (CHCONF* pChConf) | Get Channel Information | evb/config.c |
| void set_chconf (CHCONF* pChConf) | Update Channel Information | evb/config.c |
| void display_chconf (CHCONF * pChConf) | Output Channel Information through Terminal | evb/config.c |
| void load_factory_chconf(void) | Factory Reset Channel Information | evb/config.c |
| char uart0_getchar(void) | Read one character from UART0 | mcu/serial.c |

3.2.5.3. Ping Request Program

Ping Request Program is a program that sends Ping Request to a certain destination. It uses ICMP protocol message on IP protocol and made with ping_request().

ping_request() is created with the form similar to Ping program in DOS command prompt. It sends Ping request to a destination after analyzing and processing the options.

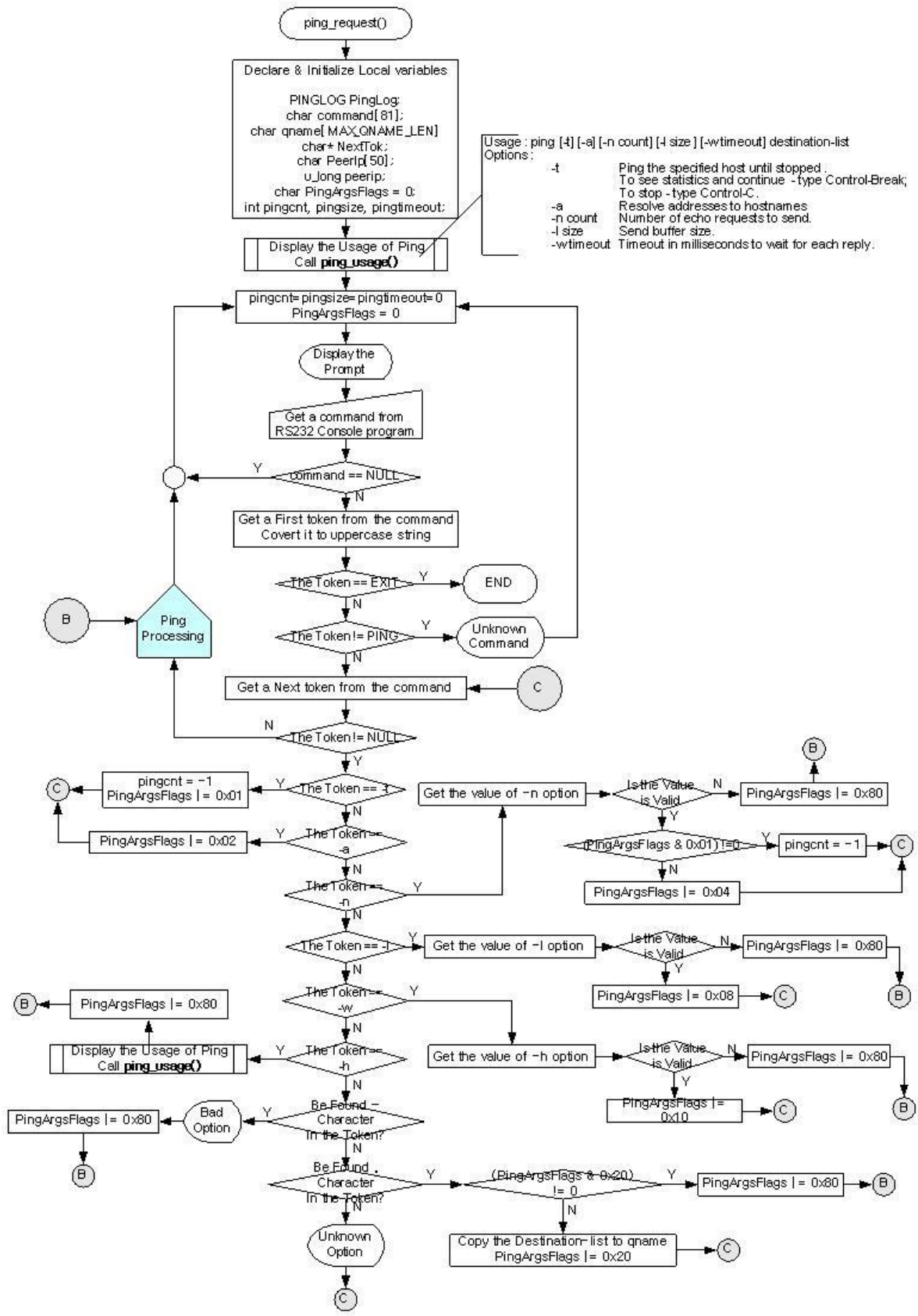
Both domain name and IP address can be used for destination address to the Ping request. In case of using domain name, domain name is changed into IP address using gethostbyname() or DNS. With the changed IP address, the Ping request is sent.

When IP address is used with '-a' option, domain name can be obtained through gethostbyaddr() from DNS Server and the Ping request is sent to the IP address. When IP address is used without the '-a' option, Ping request is sent to input IP address without the connection with DNS.

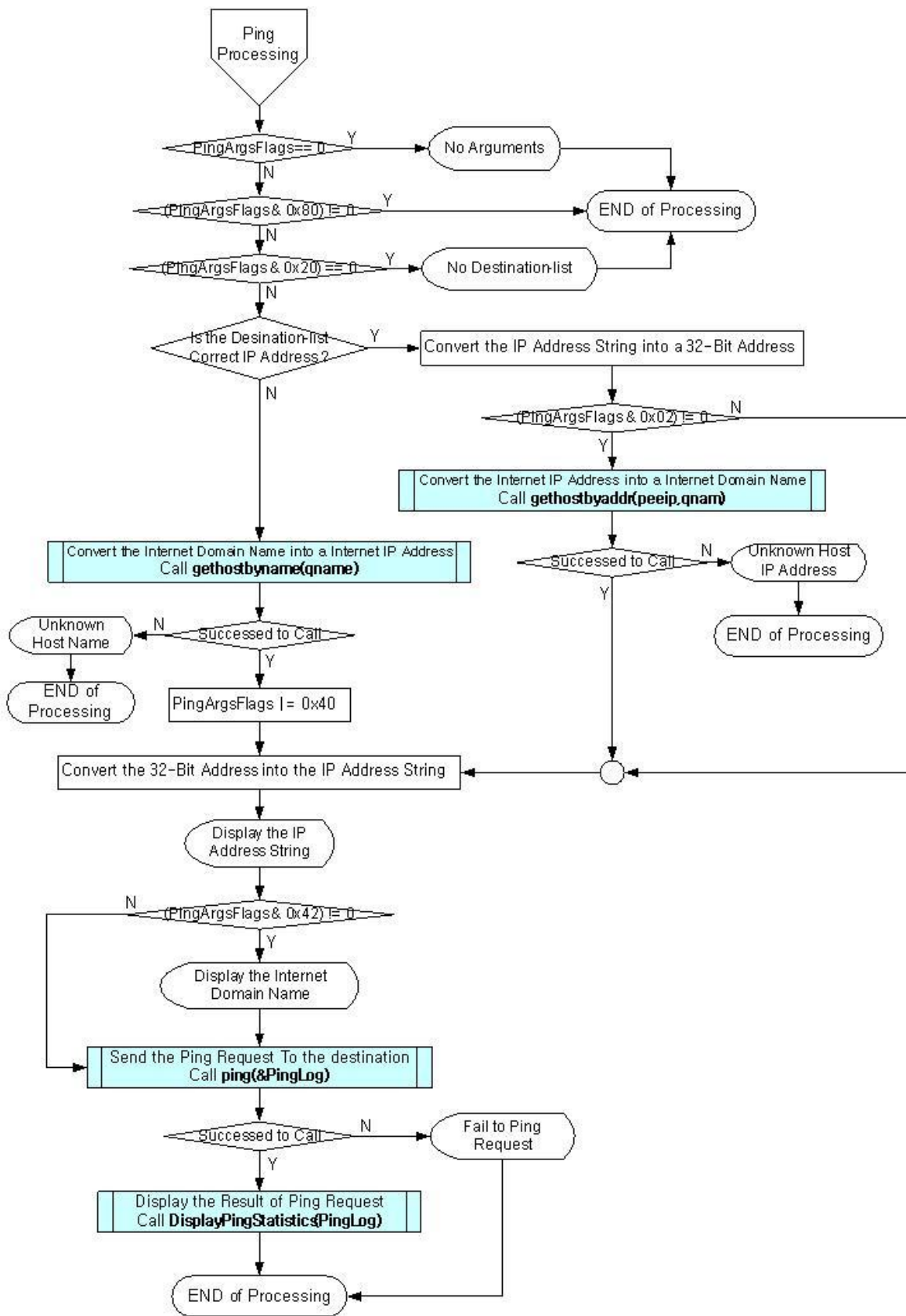
gethostbyname(), gethostbyaddr() is DNS-related functions. For further information, refer to [Chapter 3.2.6.6 DNS Client](#). <Fig 3.8> and <Fig 3.9> are processing procedures of ping_request().

<Fig 3.8> describes how tokens of inputs of Command, Option, and Option Value are created and the related Bit of Argument Flag(PingArgsFlags) is decided.

<Fig 3.9> calls ping() based on relevant option and option after checking the validity of command, option, and option value with bits of argument flag. ping() sends Ping request message to a certain destination and processes the ICMP message which is received from any destination.



<Fig 3.8: ping_request(>



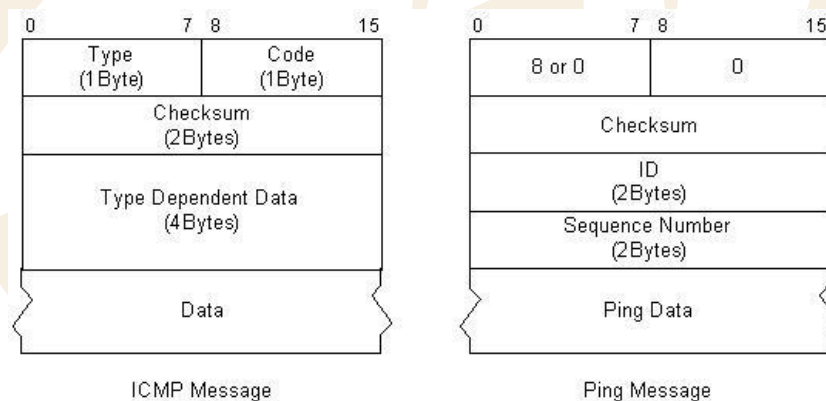
<Fig 3.9: ping_request() – Continue>

Let's take a brief look at Ping message before we proceed to Ping program.

Ping message has the value of '0'(Ping Reply) or '8'(Ping Request) at Type Field. The Code Field of ICMP Message has 0. Type Dependant Data Field(4Bytes) of ICMP Message can be re-defined as ID Field(2Bytes), Sequence Number Field(2Bytes) respectively. Data Field of ICMP Message is filled with the Ping data to be looped back.

Finally, it calculates the checksum of ICMP header and Ping data of which the checksum fields are 0. After the calculation, it replaces 0 checksum fields with the newly calculated values.

<Fig 3.10> is a diagramming representation of the relationship between the ICMP message format and the Ping message.



<Fig 3.10: ICMP Message VS Ping Message>

Checking the Ping reply to the Ping request can be processed by checking if the values of ID, sequence number and ping data field are same or not. In case the Ping reply does not come back in wait time, the ping can be sent again. In such case, the Ping request is sent with the sequence number incremented by 1. Transmitting Ping request message and checking the Ping reply message were done by ping(). The elements of ping() are of destination IP address, Ping reply wait time, number of Ping requests. Ping data size and received Ping Replies are analyzed and processed to fit the elements.

<Fig 3.11> is the process of ping() and Ping message is defined and used as the data type of <Table 3-21>. Refer to "inet/ping.h"

<Table 3-20: PINGMSG Data Type Definition>

```

typedef struct _PINGMSG
{
    char    Type;                // 0 - Ping Reply, 8 - Ping Request
    char    Code;                // Always 0
    u_short CheckSum;           // Check sum
    u_short ID;                 // Identification
    u_short SeqNum;             // Sequence Number
    char    Data[PINGBUF_LEN];  // Ping Data
}PINGMSG;
    
```

Data field size of PINGMSG is of 'PINGBUF_LEN' Byte. PINGBUF_LEN is defined as '32'. However, data field max size may be '1472'. This is because the sending MTU of W5100 is 1480 bytes, and the sum of Code, CheckSum, ID, and SeqNum Field Size is 8 Bytes. If we subtract 8 from 1480 we get 1472. Hence, the size is 1472 bytes.

The results from ping() are saved in Data Type defined in <Table 3-22>.

<Table 3-21: PINGLOG Data Type Definition>

```

typedef struct __PINGLOG
{
    u_short CheckSumErr;
    u_short UnreachableMSG;
    u_short TimeExceedMSG;
    u_short UnknownMSG;
    u_short ARPError;
    u_short PingRequest;
    u_short PingReply;
    u_short Loss;
}PINGLOG;
    
```

The saved Ping log can be output with RS232 terminal through DisplayPingStatistics() function. <Fig 3.12> shows the process procedures of DisplayPingStatistics().

CheckSumErr field is incremented by 1 whenever the checksum of Ping Reply from peer is not correctly received.

Unreachable MSG field and TimeExceedMSG field are incremented by 1 in case of receiving Unreachable Message or Time Exceeded Message from peer or gateway.

UnknownMSG field is incremented by 1 when the unknown message is received.

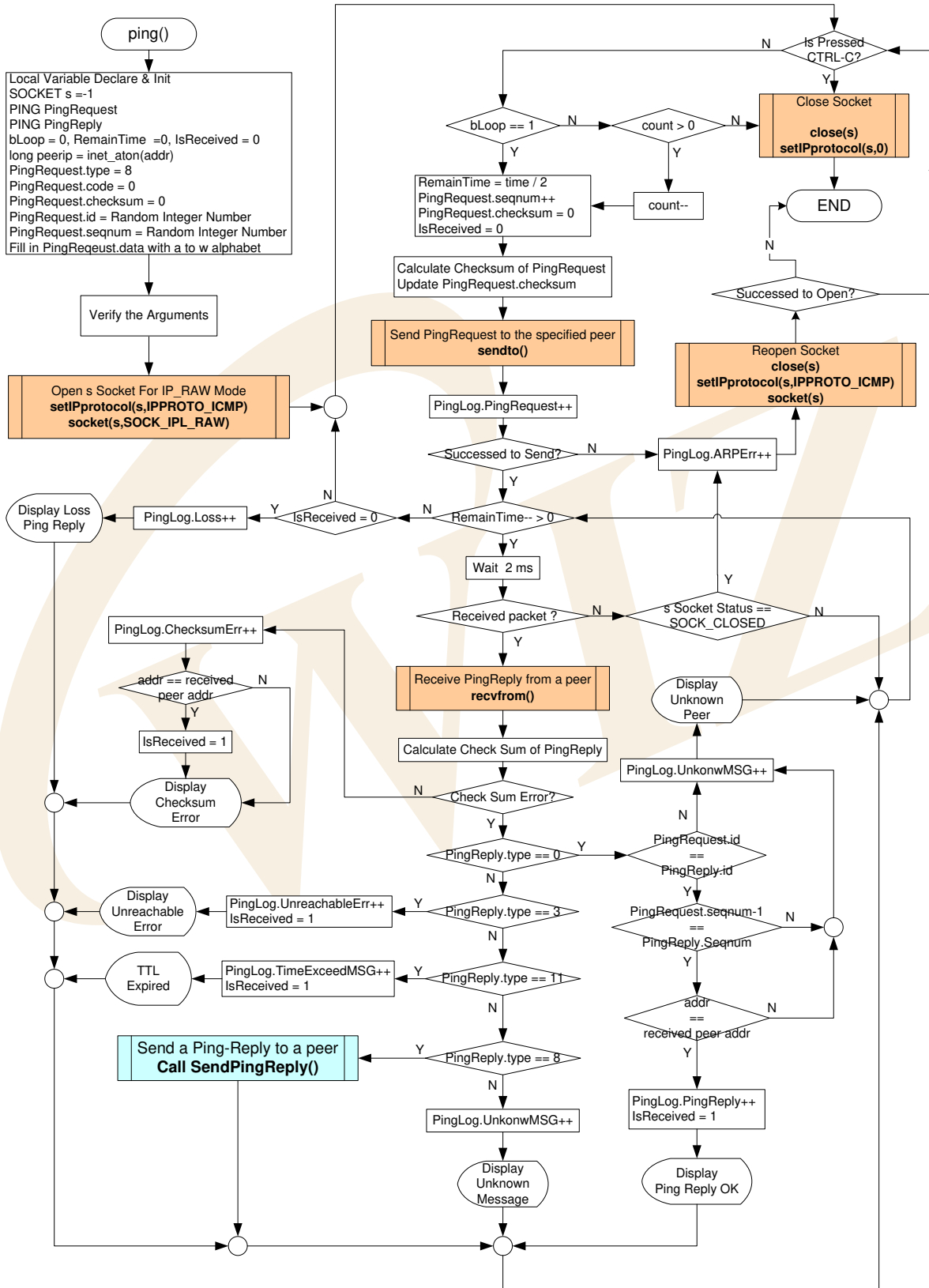
ARPErr field is incremented by 1 whenever ARP reply is not received upon ARP request to get the Hardware address(MAC Address) of the peer.

PingRequest field is incremented by 1 whenever ping() sends Ping request.

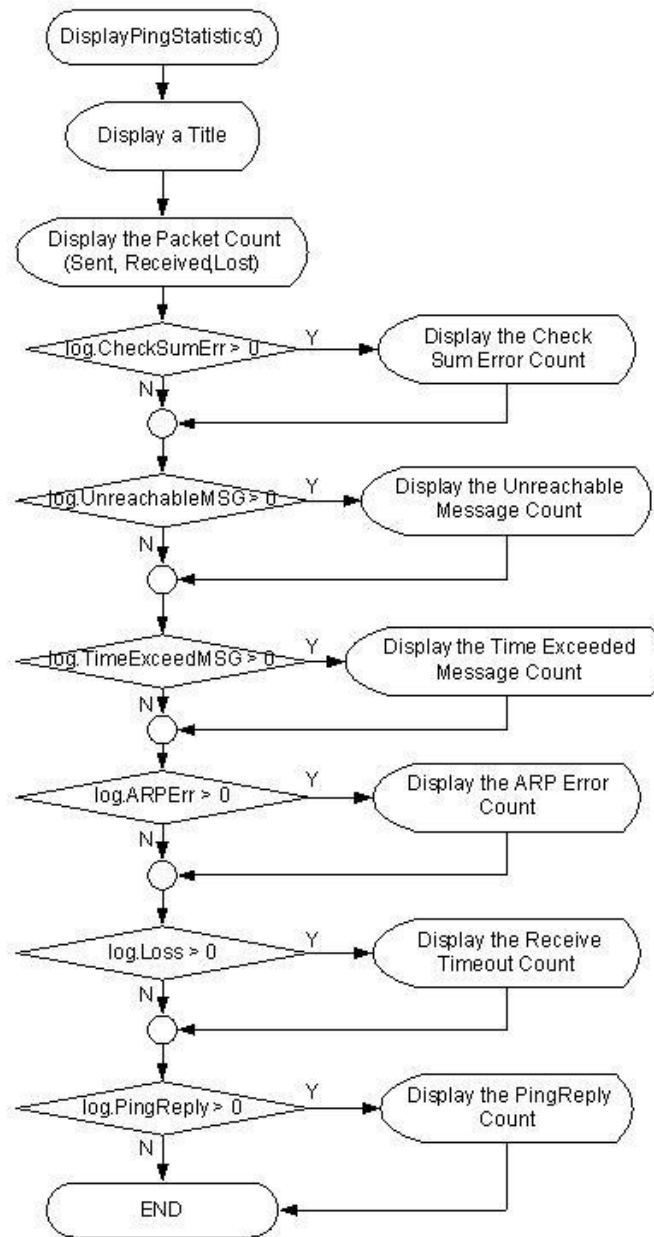
PingReply field is incremented by 1 whenever Ping reply for Ping request from the peer is received.

Loss field is incremented by 1 whenever Wait Timeout is occurred because nothing is replied to the peer in certain period of time after sending Ping request.



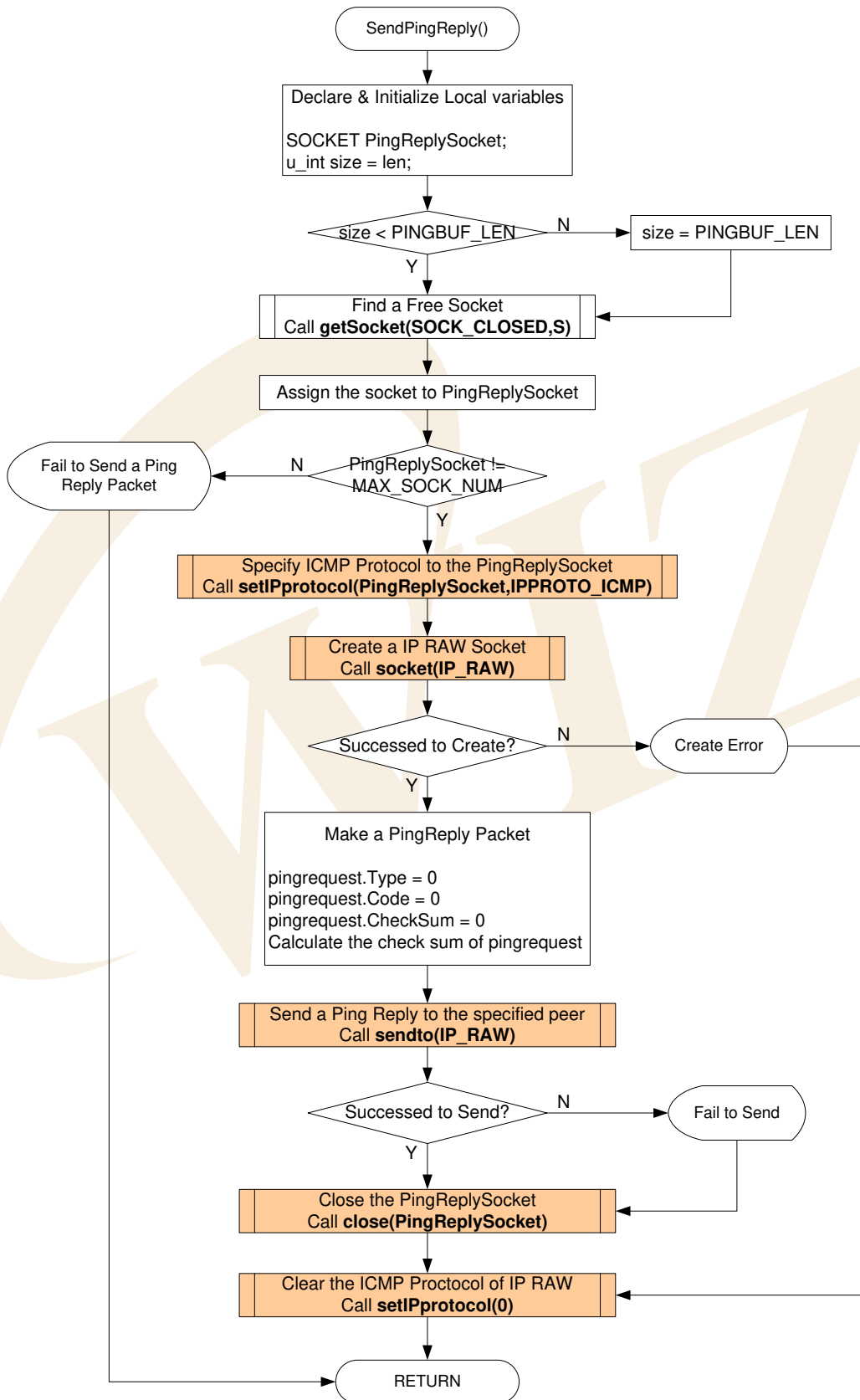


<Fig 3.11: ping(>



<Fig 3.12: DisplayPingStatistics(>

Ping request program is, as explained previously, a program that uses ICMP protocol which is running on IP Protocol. In case of using ICMP channel at W5100, as shown in <Fig 3.11> and <Fig 3.13>, IP protocol to be used must be decided. The socket must be created after calling `setIPProtocol(s, IPPROTO_ICMP)`. `IP_RAW` channel must be created by calling `socket(s, SOCK_IP_RAW, port, flag)` when creating the socket. In case of closing ICMP Socket, `setIPProtocol(s, 0x00)` should be called after `close(s)` and clear the ICMP Flag which was previously set.



<Fig 3.13: SendPingReply(>

<Table 3-22: Reference Functions in ping_request(>

| Function Name | Description | Location |
|--|---|------------------|
| void ping_request(void) | Ping Request program | app/ping_app.c |
| void ping_usage(void) | Outputs the instruction of Ping Request program | app/ping_app.c |
| char ping (int count, u_int size, u_int time, u_char* addr, PINGLOG* log) | Sends Ping Request to specific destination, and processes ICMP message received from any destination. | inet/ping.c |
| void DisplayPingStatistics (PINGLOG log) | Outputs the results from ping() calling | inet/ping.c |
| void setIPprotocol (SOCKET s, u_char ipprotocol) | Assigns IP protocol of the related socket | iinChip/w5100.c |
| char socket(SOCKET s, u_char protocol, u_int port, u_char flag) | Creates sockets related to as TCP/UDP/IP | iinChip/socket.c |
| void close(SOCKET s); | Closes the related socket | iinChip/socket.c |
| int sendto(SOCKET s, const u_char * buf, u_int len, u_char * addr, u_int port) | Sends Datagram packet to specific destination. | iinChip/socket.c |
| int recvfrom(SOCKET s, u_char * buf, u_int len, u_char * addr, u_int * port) | Receives Datagram packet from any destination | iinChip/socket.c |
| SOCKET getSocket(unsigned char status, SOCKET start) | Searches for socket has the designated status | util/sockutil.c |

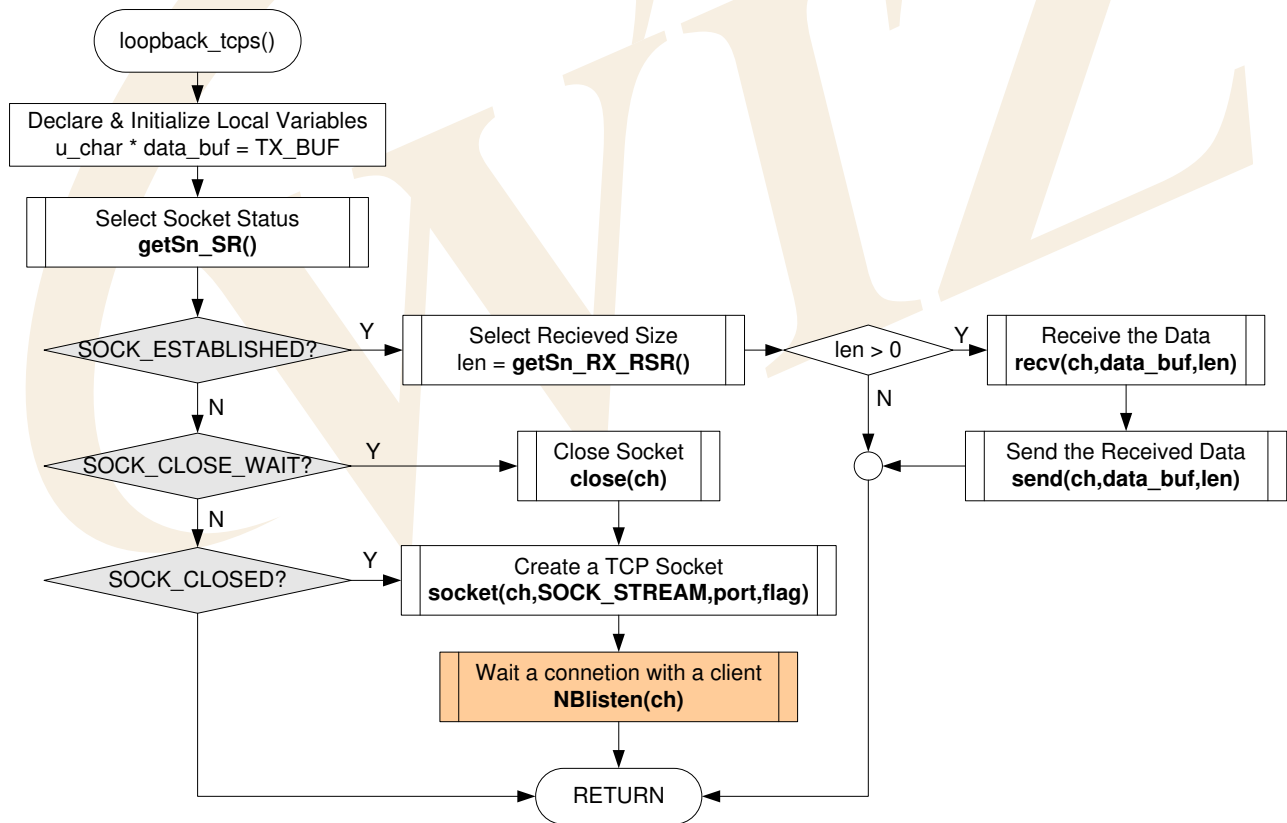
3.2.6. Applications

It's a network application using W5100. It includes Loopback program, Web Server, and DHCP Client. Application is selected by Manager Program.

3.2.6.1. Loopback TCP Server

The Loopback TCP Server program of EVB B/D works as server mode, and AX1 program of the testing PC works as client mode. AX1 tries to connect to EVB B/D and if the connection is successful, AX1 transmits the data stream through the TCP channel. EVB B/D returns back the data stream from AX1 without processing through the TCP channel.

Loopback TCP Server Program uses `loopback_tcps()` and <Fig 3.14> shows the process procedure of `loopback_tcps()`.



< Fig 3.14 : `loopback_tcps()` >

<Table 3-23: Reference Functions in loopback_tcps(>

| Function Name | Description | Location |
|---|--|------------------|
| void loopback_tcps(u_char ch) | Loopback TCP Server program | app/loopback.c |
| uint8 getSn_SR(SOCKET s) | Get the socket status | iinChip/w5100.c |
| uint16 getSn_RX_RSR(SOCKET s) | size of data transmittable, and received data | iinChip/w5100.c |
| u_char socket(SOCKET s, u_char protocol, u_int port, u_char flag) | Create the socket | iinChip/socket.c |
| u_char listen(SOCKET s) | It sets related socket as server mode | iinChip/socket.c |
| u_int send(SOCKET s, const u_char * buf, u_int len) | Transfer the data to the connected socket. | iinChip/socket.c |
| u_int recv(SOCKET s, u_char * buf, u_int len) | Receive the data to the connected socket. | iinChip/socket.c |
| void disconnect(SOCKET s); | Close the connection of the socket. | iinChip/socket.c |

If the server socket is in SOCK_CLOSED status, loopback_tcps() calls socket() with the elements of SOCK_STREAM, Listen Port Number, and Option Flag to create TCP server socket.

The socket() function changes the socket status to SOCK_INIT regardless of the previous socket status. If the server socket is created successfully, it's put in TCP Server mode after calling listen() with the server socket as the parameter. listen() makes the server socket status as SOCK_LISTEN status and maintains SOCK_LISTEN status until any client's connection.

At this point, when any client tries to connect to the server socket, the server socket status is changed from SOCK_LISTEN to SOCK_ESTABLISHED. This is when the connection between Client and Server is complete and data transfer is possible in SOCK_ESTABLISHED status.

Data is transferred using recv() and send() at the SOCK_ESTABLISHED. The data transfer here is 1-on-1 transfer between EVB B/D(The server) and AX1(The client).

In the SOCK_ESTABLISHED status, if the client requests closing of the connection, the server socket status is changed from SOCK_ESTABLISHED to SOCK_CLOSE_WAIT. In SOCK_CLOSE_WAIT status, data communication is not available and the server socket must be closed. In SOCK_CLOSE_WAIT status, disconnect() is called to close socket. disconnect() changes the socket status to SOCK_CLOSED regardless of previous socket status.

3.2.6.2. Loopback TCP Client

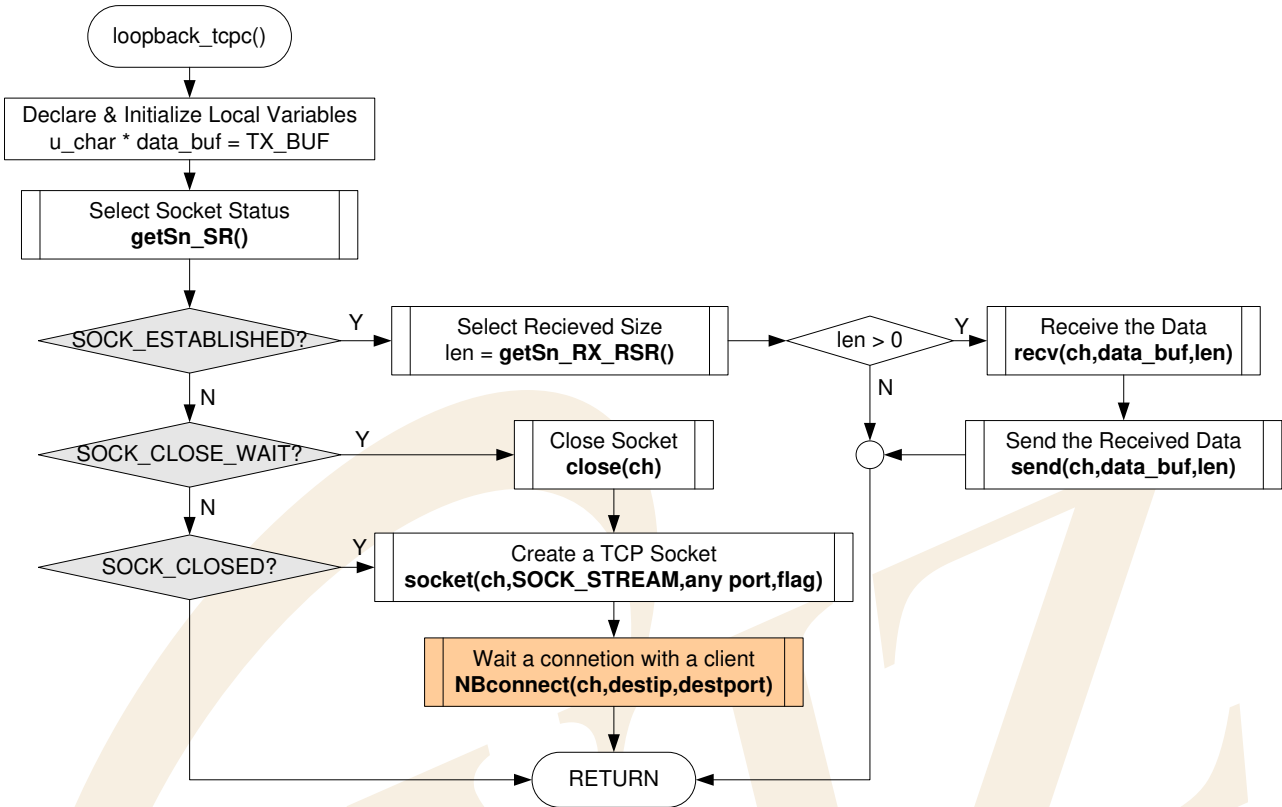
At Loopback TCP Client program, EVB B/D works in client mode and AX1, PC test program works in server mode. EVB B/D tries to connect to AX1 which is waiting as the server, if the connection is successful EVB B/D receives data stream through TCP channel and then EVB B/D sends back the received data stream to AX1.

Loopback TCP client program is created with `loopback_tcpcli()` and <Fig 3.15> is processing procedure of `loopback_tcpcli()`.

If the client socket is in `SOCK_CLOSED` status, `loopback_tcpcli()` calls `socket()` with the elements of `SOCK_STREAM`, any Port Number, and Option Flag to create TCP client socket.

In creating socket here, any port number is used for `get_system_any_port()`. This is because connection may be failed if it tries to connect to the same server with same port number. After successfully creating the socket, call `connect()` with the elements of the client socket to connect to the AX1 server.

`connect()` makes the socket status into `SOCK_SYNSENT` and keeps the status as `SOCK_SYNSENT` until it receives the authorization for connection from the server. If the connection is successful the socket status is changed from `SOCK_SYNSENT` to `SOCK_ESTABLISHED`. In `SOCK_ESTABLISHED` status, the operation is same as explained in `loopback_tcpcli()`.



<Fig 3.15: loopback_tcpcli()>

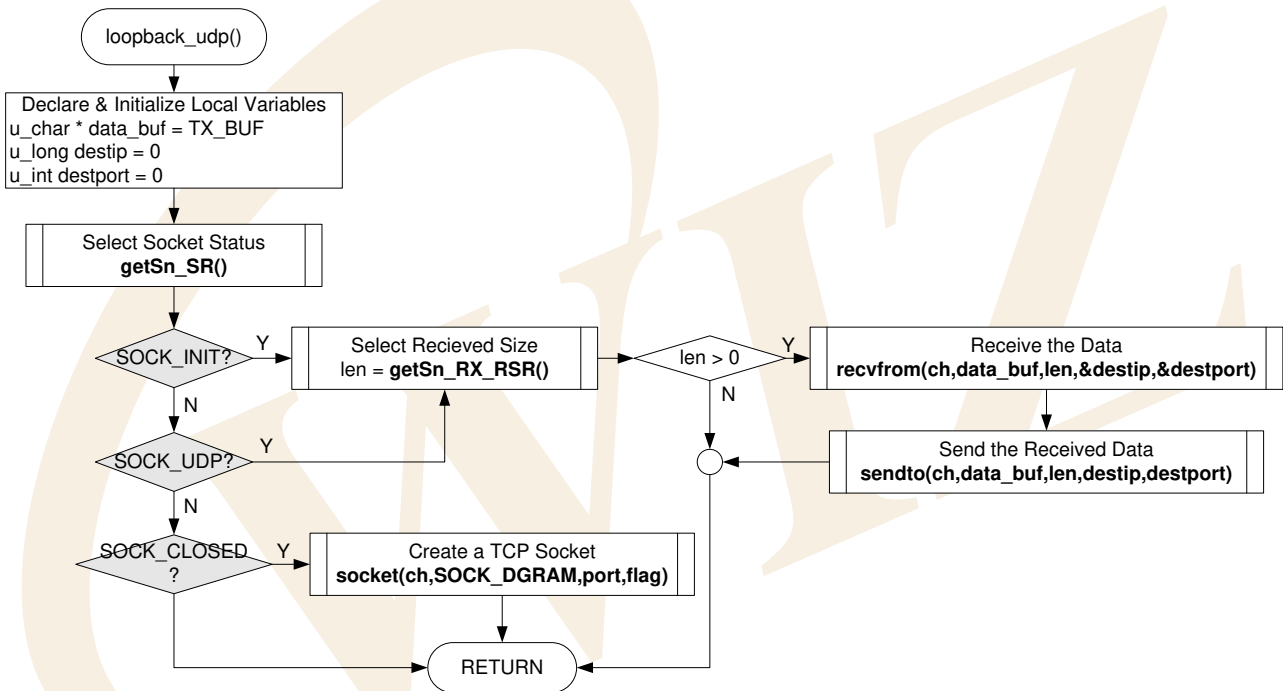
<Table 3-24: Reference Functions in loopback_tcpcli()>

| Function Name | Description | Location |
|---|--|------------------|
| void loopback_tcpcli(u_char ch) | Loopback TCP Client Program | app/loopback.c |
| uint8 getSn_SR(SOCKET s) | Get the socket status | iinChip/w5100.c |
| uint16 getSn_RX_RSR(SOCKET s) | size of data transmittable, and received data | iinChip/w5100.c |
| u_char socket(SOCKET s, u_char protocol, u_int port, u_char flag) | Related socket can be created as TCP/UDP/IP | iinChip/socket.c |
| u_char connect(SOCKET s, u_char * addr, u_int port) | Attempts to connect to the specific server with related socket | iinChip/socket.c |
| u_int send(SOCKET s, const u_char * buf, u_int len) | Sends the data to related socket that is in connection | iinChip/socket.c |
| u_int recv(SOCKET s, u_char * buf, u_int len) | Receives the data to related socket that is in connection | iinChip/socket.c |
| void disconnect(SOCKET s); | Close the related socket | iinChip/socket.c |
| u_int get_system_any_port(void) | Get any port number. | evb/config.c |

3.2.6.3. Loopback UDP

Loopback UDP Program is a program that uses unicast datagram communication of UDP protocol. It operates same as Loopback TCP Server/Client program does. UDP communication includes unicast datagram communication and broadcast datagram communication, and basically supports 1-to-many communication that is used for many destinations with one channel.

Loopback UDP program uses `loopback_udp()` and <Fig 3-16> shows processing procedure of `loopback_udp()`.



<Fig 3.16: loopback_udp(>

<Table 3-25: Reference Functions in loopback_udp(>

| Function Name | Description | Location |
|--|--|------------------|
| void loopback_udp(u_char ch) | Loopback udp program | app/loopback.c |
| uint8 getSn_SR(SOCKET s) | Gets the socket status | iinChip/w5100.c |
| uint16 getSn_RX_RSR(SOCKET s) | size of data transmittable, and received data | iinChip/w5100.c |
| u_char socket(SOCKET s, u_char protocol, u_int port, u_char flag) | Creates related socket as TCP/UDP/IP. | iinChip/socket.c |
| u_int sendto(SOCKET s, const u_char * buf, u_int len, u_char * addr, u_int port) | Sends data to specific port of specific destination related socket | iinChip/socket.c |
| u_int recvfrom(SOCKET s, u_char * buf, u_int len, u_char * addr, u_int * port) | Sends data to any port of any destination related socket | iinChip/socket.c |
| void close(SOCKET s) | Close related socket | iinChip/socket.c |

If the udp socket is in SOCK_CLOSED status, socket() is called using SOCK_DGRAM, Port Number, and Option Flag as the elements to create the UDP socket.

UDP communication, as opposed to TCP, is a datagram communication without the requirement of connection process. So, direct data communication is possible immediately after socket creation. After creation of UDP socket, the udp socket status will be changed from SOCK_CLOSED to SOCK_UDP.

Here, not like TCP for data communication which uses send() and recv(), sendto() and recvfrom() are used.

This is because TCP is 1-to-1 communication method of which destination is known but UDP is 1-to-many communication without connection procedure. sendto() sends data to specific port of specific destination that is sent as an element, recvfrom() is used to receive the incoming data from temporary port. Destination information from recvfrom() is informed to user using destip and destport which are sent as elements.

In loopback_udp(), there is no example of using close(), but in case that the UDP communication is not needed anymore, close() can be always called to close the udp socket.

3.2.6.4. Web Server

Web Server program is a TCP server program using HTTP protocol which is used on TCP protocol. Before building Web server program, message structure of HTTP protocol that is transmitted between Web server and Web client(Web browser) are needed to be understood.

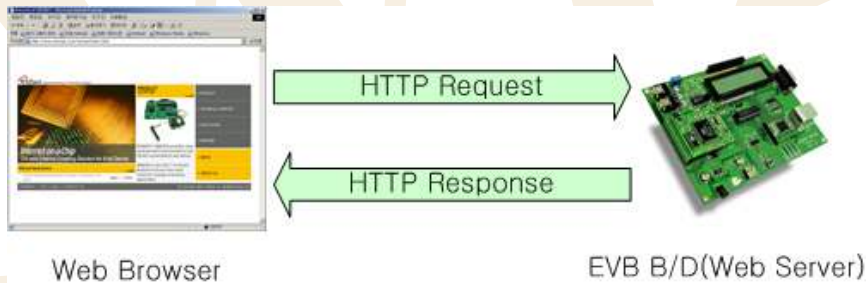
HTTP, which stands for Hyper Text Transfer Protocol, is a protocol used in Internet for transferring between Web server and client browsers.

<Table 3-26: Web Browser's HTTP Request Operation Procedure >

| |
|--|
| Request of Client(Web Browser) --> URL Analysis(Transforming Domain Name to IP Address at DNS) --> Connection to server at the other end --> Client(Web Browser) requests document wanted from URL --> Sending Document(Server)/Receiving Document (Client) --> Displays received document on the browser |
|--|

Web Server program analyzes method and URI(Uniform Resource Identifier) of HTTP Request message received from web browser. In case the related URI simply requests for web page, the page will be sent. If it requests an action such as CGI(Common Gateway Interface), it takes the action and the result is informed in web page.

<Fig 3.17> shows HTTP message flow between web server and web client. <Table 3-28> shows structure of HTTP message.



<Fig 3.17: HTTP Message Flow>

<Table 3-27: HTTP Message Format>

| | |
|----------------------|---|
| HTTP-message | = Simple-Request Simple-Response Full-Request Full-Response |
| Full-Request | = Request-Line *(General-Header Request-Header Entity-Header) CRLF [Entity-Body] |
| Full-Response | = Status-Line *((General-Header Response-Header Entity-Header) CRLF) CRLF [Entity-Body] |
| Request-Line | = Method SP Request-URI SP HTTP-Version CRLF |
| Status-Line | = HTTP-Version SP Status-Code SP Reason-Phrase CRLF |
| Entity-Header | = Allow Content-Encoding Content-Length Content-Type Expires Last-Modified extension-header |
| Entity-Body | = *OCTET |
| Method | = "GET" "HEAD" "POST" extension-method |

For further information on HTTP message, refer to RFC2616. HTTP request message varies according to web browser type. <Table 3-29> shows the examples of HTTP message communication between Internet Explores on Windows 2000 and EVB B/D.

<Table 3-28: HTTP MESSAGE BETWEEN EVB B/D AND WEB BROWSER>
HTTP Request Message

```
Ex1> GET wiz_log.gif HTTP/1.1CRCF
      Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.-ms-
           powerpoint, application/vnd.-ms-excel, application/ms-word, /*CRCF
      Accept Language: koCRCF
      Accept Encoding: gzip, deflateCRCF
      User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0; .NET CLR
           1.3705)CRCF
      Host: 192.168.0.2CRCF
      Connection: Keep-AliveCRCF
      CRCF
```

```
Ex2> GET http://192.168.0.2/LCDNLED.CGI?lcd=hi.+EVB B/D&led0=on HTTP/1.1CRCF
      Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.-ms-
           powerpoint, application/vnd.-ms-excel, application/ms-word, /*CRCF
      Accept Language: koCRCF
      Accept Encoding: gzip, deflateCRCF
      User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0; .NET CLR
           1.3705)CRCF
      Host: 192.168.0.2CRCF
      Connection: Keep-AliveCRCF
      CRCF
```

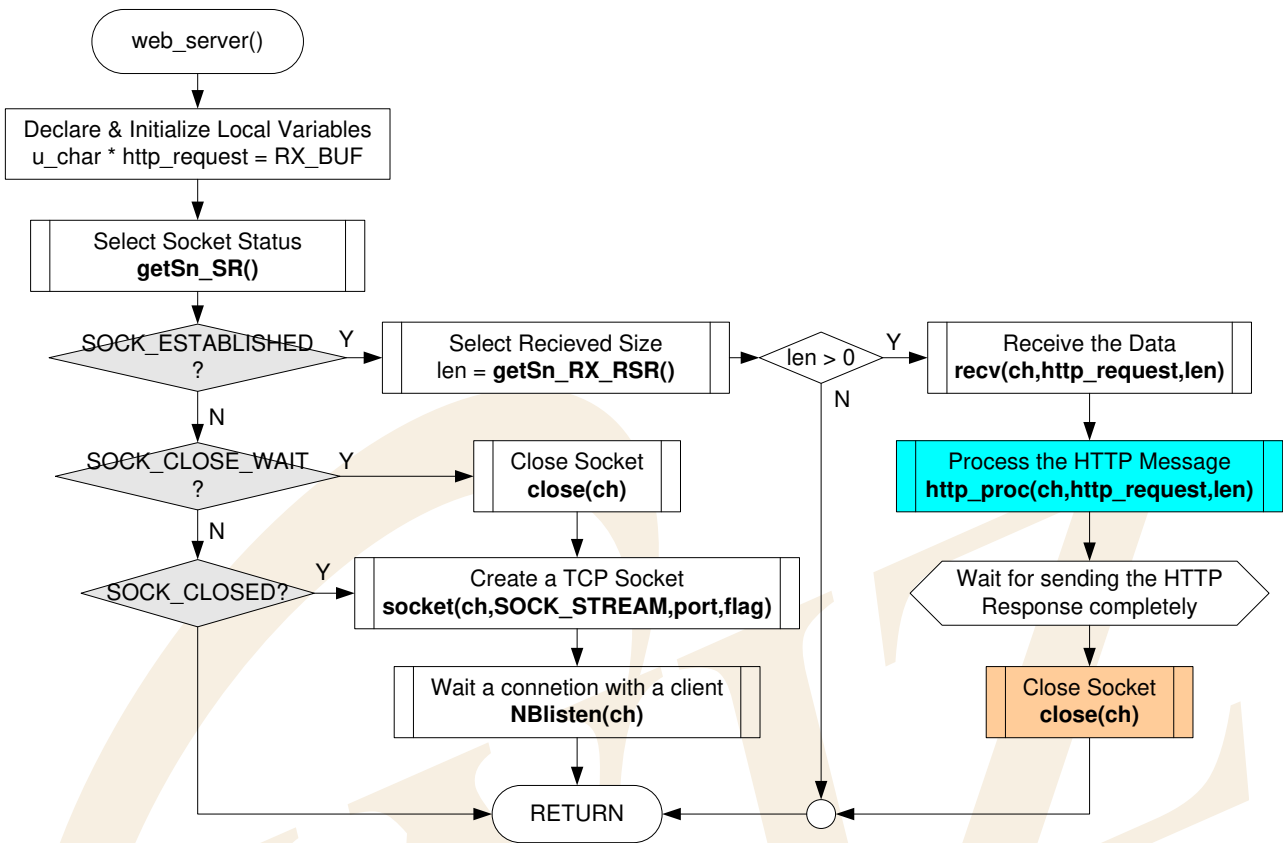
HTTP Response Message

```
Ex1> HTTP/1.1 200 OK CRCF
      Content-Type: text/htmlCRCF
      Content-Length: 1451CRCF
      [Html Document]
```

```
Ex2> HTTP/1.1 200 OKCRCF
      Content-Type: gif/imageCRCF
      Content-Length: 613CRCF
      [GIF IMAGE]
```

Web Server program is composed of web_server() to manage HTTP server socket and proc_http() to manage HTTP message.

<Fig 3.18> is processing procedure.

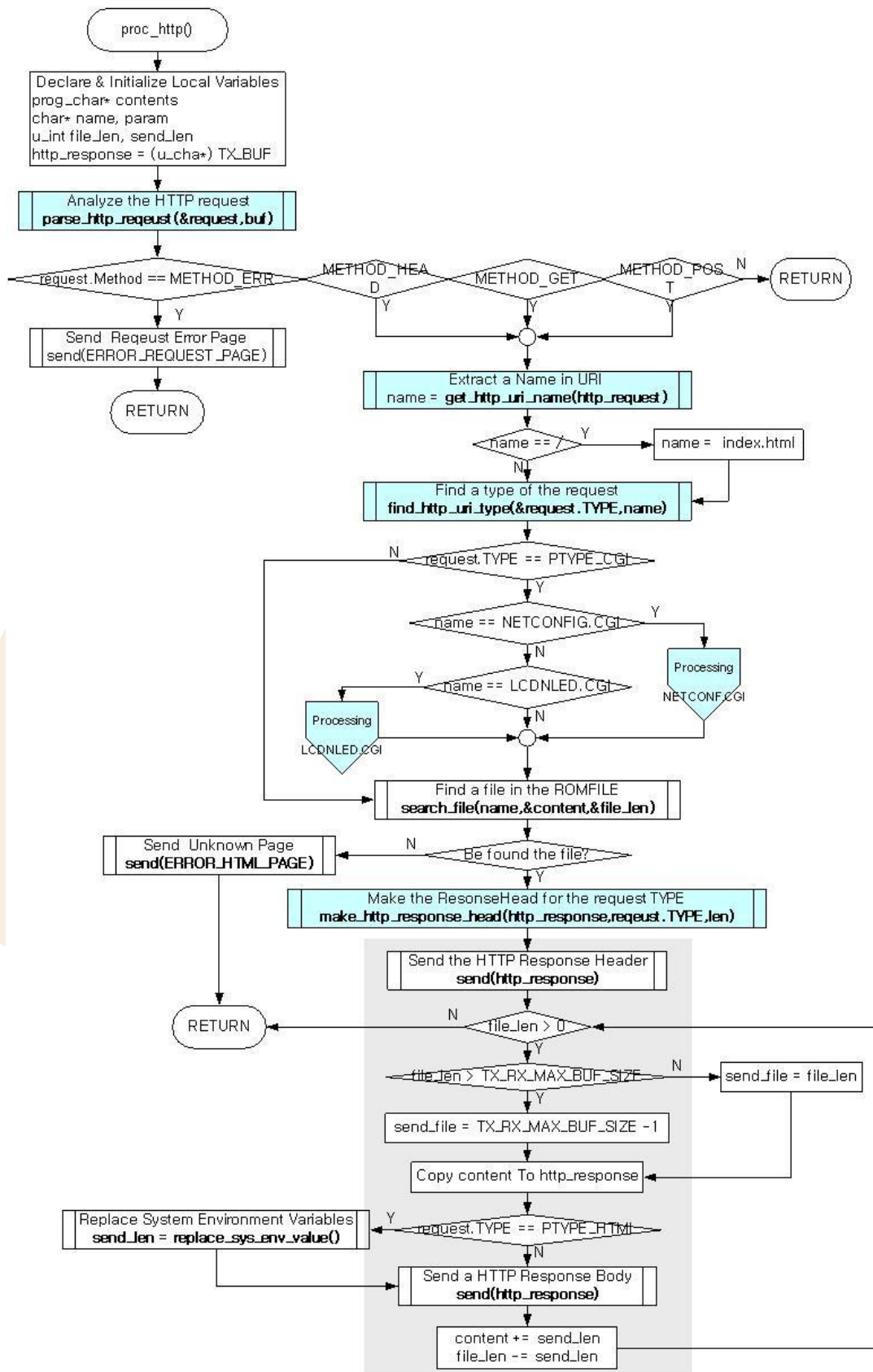


<Fig 3.18: web_server()>

Since web_server() is TCP server program, it is built in the similar way as loopback_tcps() as explained in [Chapter 3.2.6.1](#). Difference between web_server() and loopback_tcps() is in the data communication codes. web_server() calls proc_http() that processes HTTP request message from web browser at SOCK_ESTABLISHED of the http socket.

After calling function proc_http(), it waits until the HTTP response message to HTTP request from web browser, and then calls disconnect() to close the http socket.

This socket close is called Active Close and, in the case, EVB B/D requests the close to the client first. For your reference, Passive Close is where client requests disconnection first. The reason why web server program supports Active Close is that EVB B/D supports the connection with other clients.



<Fig 3.19: proc_http(>

proc_http() calls parse_http_request() to analyse the HTTP request message received from web browser. If the METHOD of analyzed HTTP request message is "GET", "HEAD", or "POST", get_http_uri_name() is called and URI Name is extracted from HTTP Request message. If extracted URI Name is "/", replace URI Name "/" to "index.html" which is web server default page of EVB B/D, because this means that web browser is requesting default page of web server.

After getting the HTTP request type of HTTP request message by calling find_http_uri_type(), if HTTP request type is "CGI", it performs the related CGI command process.

After processing CGI commands or in case that HTTP request type is not the CGI, search file with URI Name from ROM File Image which is built in EVB B/D.

If the file is found, create HTTP response message and send it.

HTTP Response message is composed of HTTP response header transmission and HTTP response body transmission. For transmission of HTTP response header, it calls make_http_response_head() using HTTP request type as the element to create HTTP response header. After transmitting the created HTTP response header, the HTTP response body is transmitted. For example, if the HTTP response body is any file in ROM File Image, the files are much bigger than the MTU of W5100. Hence it has to be divided into maximum size of W5100 before transmission. At this point, if system environment variables that are defined in EVB B/D in HTTP response body exist, it calls replace_sys_env_value() and replaces system environment variables to system environment value stored in EVB B/D.

<Table 3-29: System Environment Variables Usage at "evbctrl.html" >

```

<tr>
  <td width="110" height="22"><font color="#FEFEEF">...</font>Source IP</td>
  <td width="240" height="27"><input name="sip" type="text" size="20" value="$SRC_IP_ADDRESS$" ></td>
</tr>
<tr>
  <td width="110" height="22"><font color="#FEFEEF">...</font>Gateway IP</td>
  <td height="27"><input name="gwip" type="text" size="20" value="$GW_IP_ADDRESS$" ></td>
</tr>
<tr>
  <td width="110" height="22"><font color="#FEFEEF">...</font>Subnet Mask</td>
  <td height="27"><input name="sn" type="text" size="20" value="$SUB_NET_MASK$" ></td>
</tr>
<tr>
  <td width="110" height="22"><font color="#FEFEEF">...</font>DNS Server IP</td>
  <td height="27"><input name="dns" type="text" size="20" value="$DNS_SERVER_IP$" ></td>
</tr>
<tr>
  <td width="110" height="22"><font color="#FEFEEF">...</font>MAC Address</td>
  <td height="27">$SRC_MAC_ADDRESS$ </td>
</tr>
    
```

<Table 3-30> is a part of "evbctrl.html" in ROM File Image of EVB B/D.

The length of the system environment variables is defined to fit the length of system environment value to be replaced. For example, if Source IP Address of EVB is expressed in string, the maximum is 16. Hence, the length of \$SRC_IP_ADDRESS\$ is 16 as well. 'ROM File System' of EVB B/D can be created with

“ROMFileMaker.exe” provided by WIZnet. Refer to “ROM File Maker Manual Vx.x.pdf” for further information.

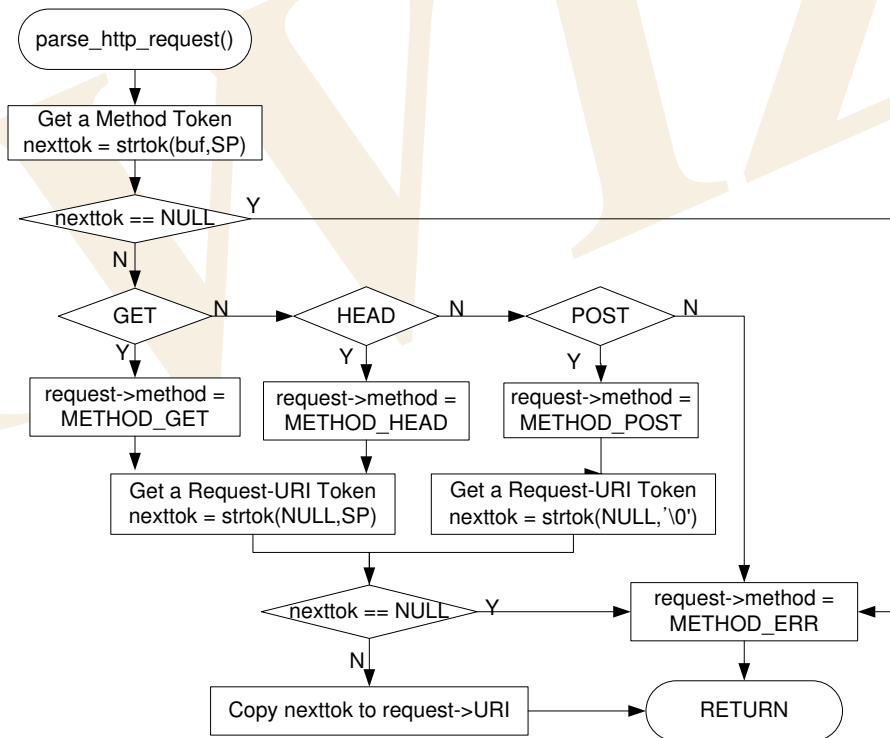
HTTP Request message can be divided into Method and Request-URI by parse_http_request() and stored in ‘st_http_request’ Date Type which is defined in <Table 3-31>. It gets the requested URI Type with get_http_uri_type().

<Table 3-30: “st_http_request” Data>

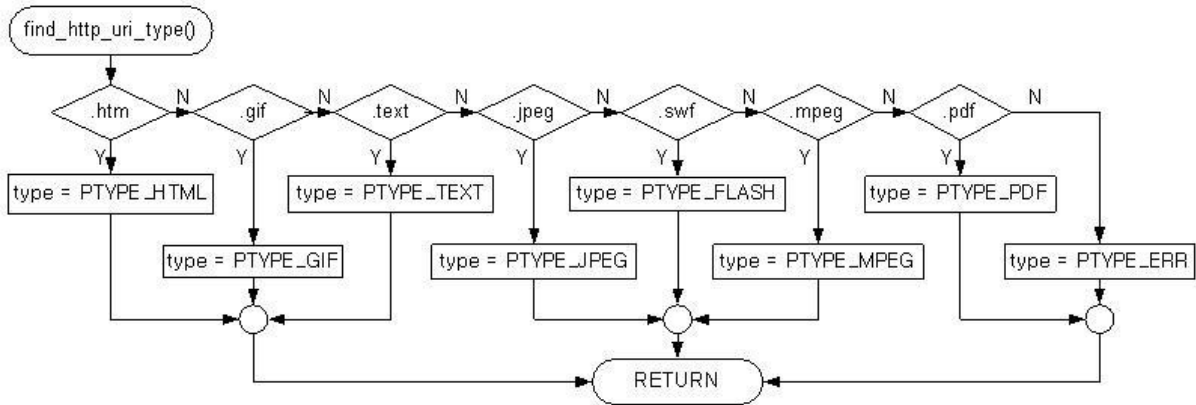
```

#define MAX_URI_SIZE (2048 - sizeof(char)*2)

typedef struct _st_http_request
{
    u_char METHOD;          /* request method(METHOD_GET...). */
    u_char TYPE;          /* request type(PTYPE_HTML...). */
    char URI[MAX_URI_SIZE]; /* request file name. */
}st_http_request;
    
```

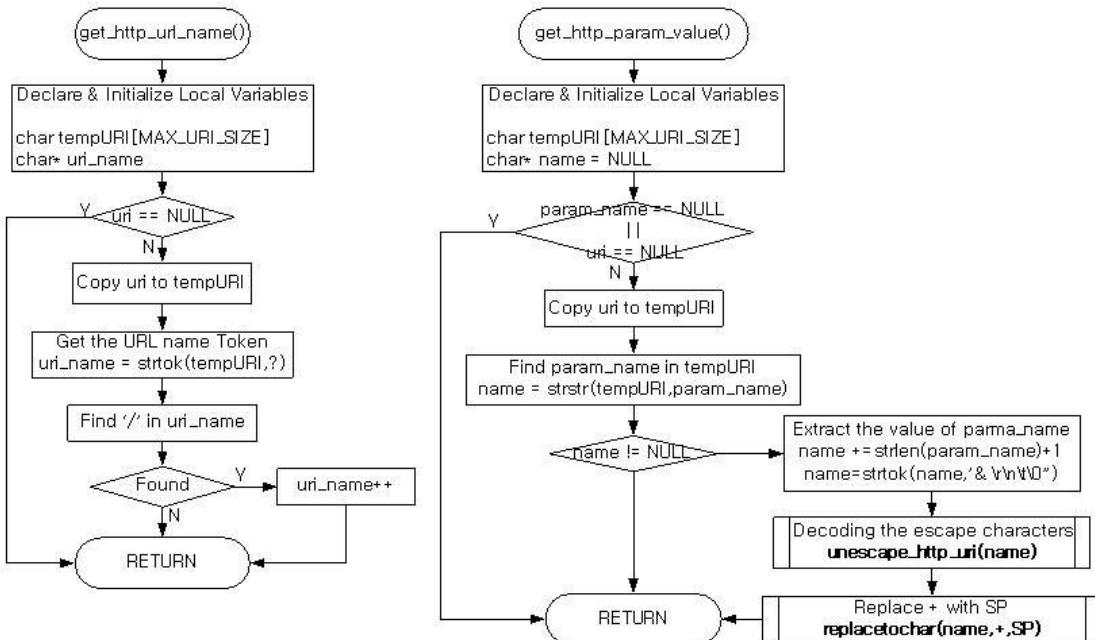


<Fig 3.20: parse_http_request()>



<Fig 3.21: find_http_uri_type(>

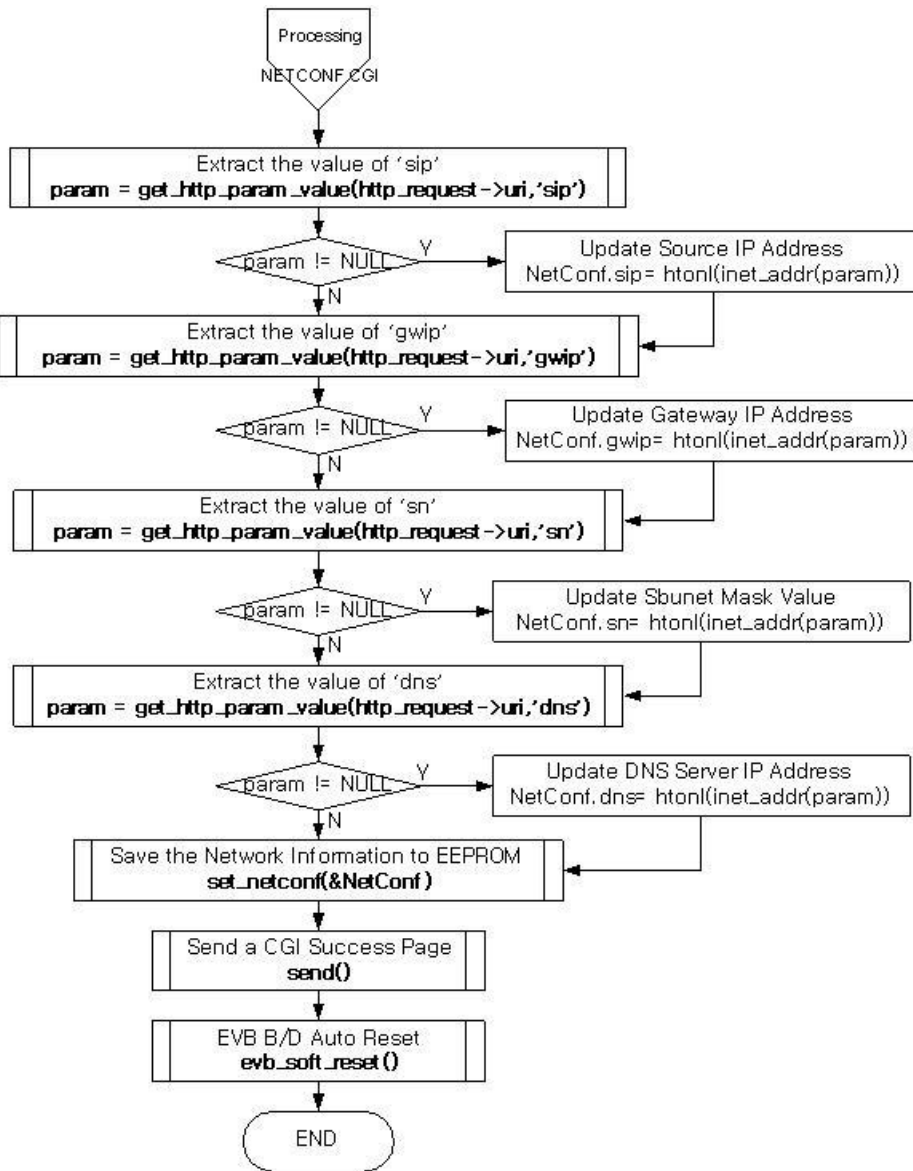
Request-URI which is saved in URI [MAX_URI_SIZE] of st_http_request has URI Name before “?” symbol and Query String after “?” sign. When Request-URI is transferred from Web Browser to Web Server, SP (Space) text is transmitted in the form of ‘+’ and, other Reserved Texts are transmitted in the form of “%HEXHEX.” Accordingly, Reserved Texts in Request-URI needs to be decoded to the previous value, from ‘+’ to SP and from %HEXHEX to related ASCII vales. For the details of Request-URI decoding, refer to RFC1738. URI name of Request-URI is extracted with get_http_uri_name().Query String of Request-URI can include one or more “variable=value” pair that has “&” as a separator. Through function get_http_param_value(), it can extract the wanted variable value in Query String.



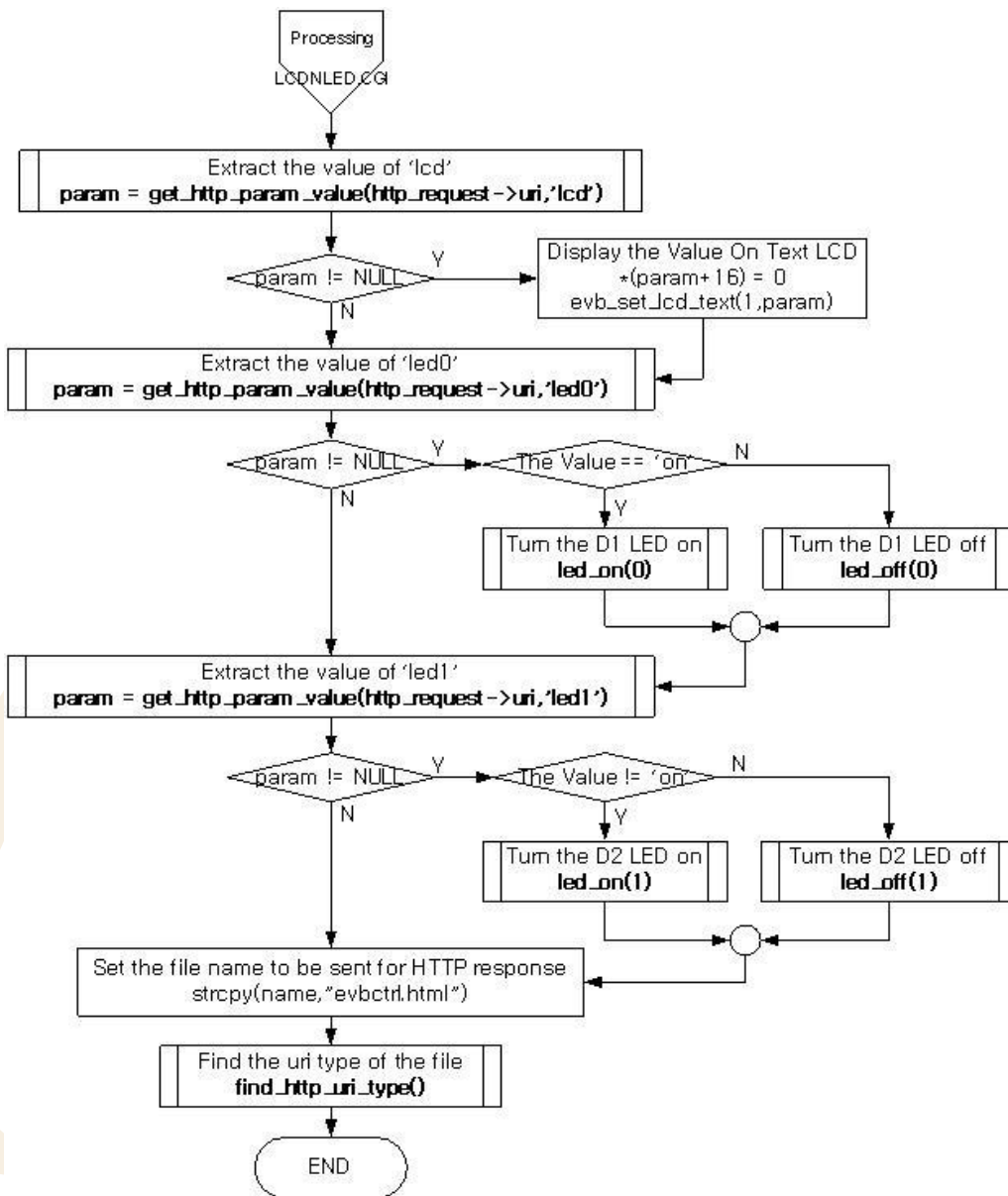
<Fig 3.22: get_http_uri_name() & get_http_parse_value(>

CGI processing of Web Server Program at EVB B/D is different from general Web Server Program which is

based on OS. Web Server Program which is based on OS creates separate process to take care of communication between processes independently. However, Web Server of EVB B/D is OS-less, so, instead of making independent process, it calls relevant functions to deal directly with CGI processing. EVB B/D supports "NETCONF.CGI" which updates Network Information and "LCDNLED.CGI" which controls text LCD, D1/D2 LED of EVB B/D. <Fig 3.23> and <Fig 3.24> shows both CGI processing.



<Fig 3.23: NETCONF.CGI Processing>



<Fig 3.24: LCDNLED.CGI Processing>

<FORM> of NETCONF.CGI is submitted in "POST" Method. <FORM> submitted using "POST" Method is not submitted in Query String but submitted in Entity Body of HTTP Request Message. Such value of parameter for NETCONF.CGI, also, is used to extract related parameter value using `get_http_param_value()`.

<FORM>of LCDNLED.CGI is submitted in "GET" Method and <FORM> submitted as "GET" Method is submitted in Query String of Request-URI. Parameters submitted by Query String of Request-URI can also extract parameter value using `get_http_param_value()`.

<Table 3-31: Reference Functions in web_server(>

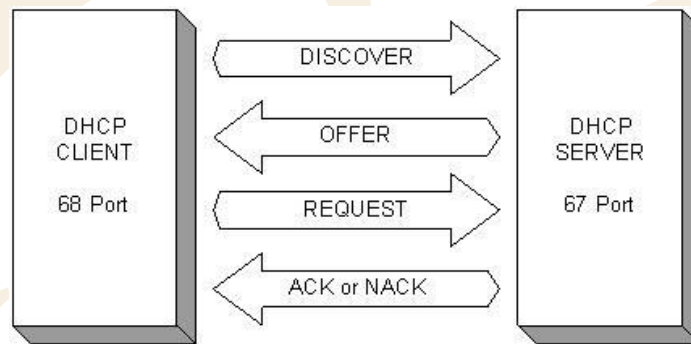
| Function Name | Description | Location |
|---|--|------------------|
| void web_server(u_char ch) | Web Server Program | app/webserver.c |
| void proc_http(SOCKET s, u_char * buf, int length) | Processes HTTP Message using related socket | app/webserver.c |
| u_int replace_sys_env_value (u_char* base, u_int len) | Change Pre-defined System Environment Variables in HTTP Response Message to Real Values. | app/webserver.c |
| void parse_http_request (st_http_request *, u_char *) | Analyzes and processes HTTP Request Message and saves it in st_http_request structure. | inet/httpd.c |
| void find_http_uri_type (u_char *, char *) | Gets MIME Type of HTTP Request Message. | inet/httpd.c |
| char* get_http_uri_name (char* uri) | Gets Request-URI Name of HTTP Request Message. | inet/httpd.c |
| char* get_http_param_value (char* uri, char* param_name) | Gets Relevant Parameter Value in Query String of Request-URI | inet/httpd.c |
| void unescape_http_uri(char * url) | Transforms Escape Character | inet/httpd.c |
| void make_http_response_head (char *, char, u_long) | Creates header of HTTP Response Message | inet/httpd.c |
| uint8 getSn_SR(SOCKET s) | Informs the socket status | iinChip/w5100.c |
| uint16 getSn_RX_RSR(SOCKET s) | size of data transmittable, and received data | iinChip/w5100.c |
| u_char socket(SOCKET s, u_char protocol, u_int port, u_char flag) | Creates related socket as TCP/UDP/IP | iinChip/socket.c |
| void listen(SOCKET s) | Puts the related socket in Server Mode | iinChip/socket.c |
| u_int send(SOCKET s, const u_char * buf, u_int len) | Sends data using connected socket | iinChip/socket.c |
| u_int recv(SOCKET s, u_char * buf, u_int len) | Receives data from the data from the connected socket | iinChip/socket.c |
| void disconnect(SOCKET s) | Closes the connection of the socket | iinChip/socket.c |
| void replacetochar(char * str, char oldchar, char newchar) | Changes the special characters in text rows into new characters. | util/util.c |

3.2.6.5. DHCP Client

DHCP Client program is a program that assigns the network information from DHCP server in the network. Note that DHCP Client program must be started prior to other programs because it manages Network Information setup. First, review basic facts on DHCP(Dynamic Host Configuration Protocol) and get further into the usage of DHCP Client program.

DHCP uses UDP protocol in Transport Layer and communicates with DHCP server using broadcast of UDP. The reason why it uses broadcast is because it has no IP address and the IP address of server is unknown. When UDP broadcast at W5100, destination IP address needs to be set '255.255.255.255' for broadcast packet transmission.

<Fig 3.25> is a Message Flow between DHCP Server and Client.

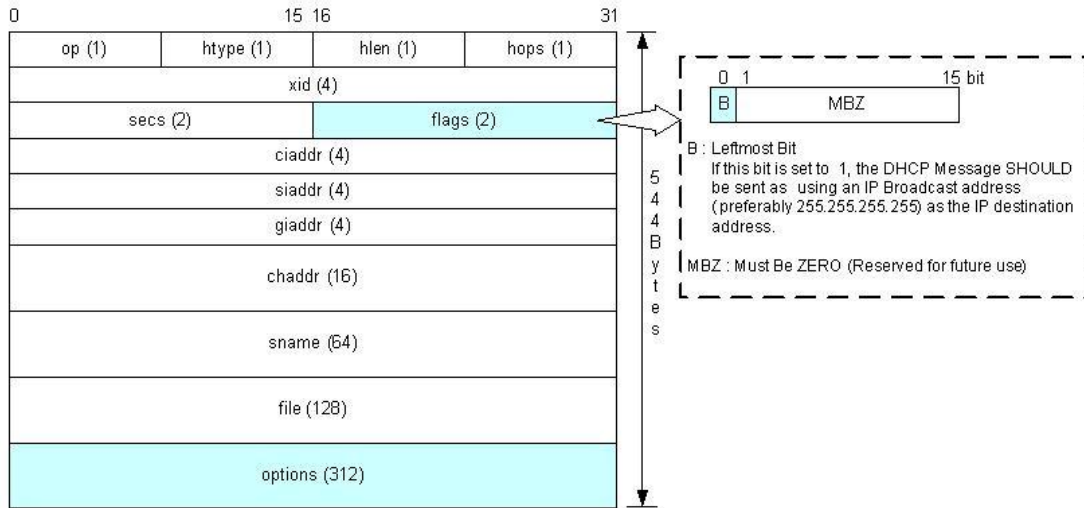


<Fig 3.25: DHCP Message Flow>

First of all, DHCP client broadcasts DISCOVERY message to the local Network. If DHCP server exists at the network then DHCP server receives Discovery message and provides network Information such as IP, G/W IP, Subnet Mask, and DNS sever IP which can be used by DHCP Client, and information such as Lease Time to the DHCP Client as OFFER message. DHCP Client can detect DHCP server by receiving the OFFER message and then it sends REQUEST message to DHCP server to use the information suggested by DHCP server. After receiving REQUEST message from DHCP Client, DHCP server finds out if the requested network information is usable. If it is, it sends ACK message, if not, NACK message is sent to DHCP Client. After receiving ACK message from DHCP server, DHCP Client uses the offered network information. The network information is valid only for the Lease Time suggested by DHCP server. Hence, if DHCP Client wants to keep using the network information, it retransmits REQUEST message to DHCP server to maintain network information usually after half of the Lease Time. In this process, DHCP client can get same or new network information from DHCP server. In case that it receives new network information, the new one must be used.

Message between DHCP server and client has the format as in <Fig 3.26> with the size of 544 Bytes. Refer to document 'RFC1541' for detailed explanation for each field of DHCP message Format. op Field of the first

byte decided Request/Reply, and fields after ciaddr is used to deliver network information, and options field of 312 byte is used to transmit message type or the information such as Client Identifier.



<Fig 3.26: DHCP Message Format>

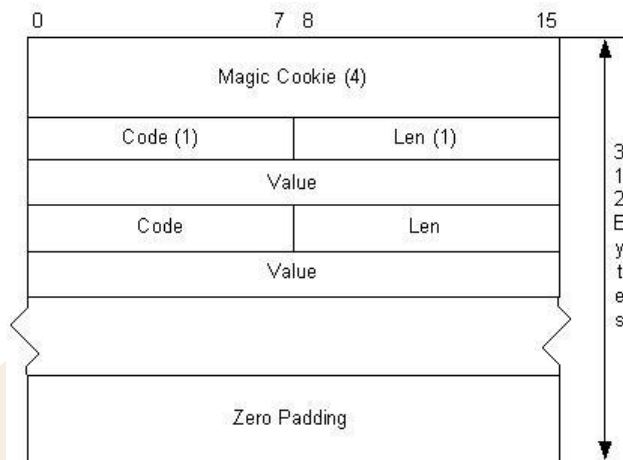
<Table 3-32: DHCP Message Data Type>

```
typedef struct _RIP_MSG
{
    u_char  op;           // DHCP_BOOTREQUEST or DHCP_BOOTREPLY
    u_char  htype;       // DHCP_HTYPE10MB
    u_char  hlen;        // DHCP_HLENETHERNET
    u_char  hops;        // DHCP_HOPS
    u_long  xid;         // DHCP_XID
    u_int   secs;        // DHCP_SECS
    u_int   flags;       // DHCP_FLAGSBROADCAST
    u_char  ciaddr[4];
    u_char  yiaddr[4];
    u_char  siaddr[4];
    u_char  giaddr[4];
    u_char  chaddr[16];
    u_char  sname[64];
    u_char  file[128];
    u_char  OPT[312];
}RIP_MSG;
```

DHCP Message of <Fig 3.26> is managed by RIP_MSG Data Type defined in <Table 3-33>. Refer to "inet/dhcp.h"

To take a brief look at the Option Field of DHCP Message, Option Field has the format of <Fig 3.27>, it contains Magic Cookie Field, a Lease Identification Cookie with the size of 4 Byte and Code Set ranged from Code 0 to Code 255. From Code1 to Code 254, codes are composed of pairs of {Code, Len, Value}, and

Code 0 and Code 255 are composed of {Code} only. For further explanation of each Code of Option Field, refer to RFC1533.



<Fig 3.27: DHCP Message's Option Field Format>

<Table 3-33: DHCP Message Option Code Definition>

| Code | Enumeration Type | Description |
|------|----------------------|--|
| 0 | padOption | used to cause subsequent fields to align on word boundaries |
| 1 | subnetMask | specifies the client's subnet mask |
| 3 | routersOnSubnet | a list of IP addresses for routers on the client's subnet |
| 6 | dns | specifies a list of DNS servers available to the client |
| 12 | hostName | specifies the name of the client |
| 50 | dhcpRequestedIPAddr | request that a particular IP address be assigned by the server |
| 51 | dhcpIPAddrLeaseTime | a lease time for the IP address |
| 53 | dhcpMessageType | used to convey the type of the DHCP message |
| 54 | dhcpServerIdentifier | the IP address of the selected server |
| 55 | dhcpParamRequest | request values for specified configuration parameters |
| 61 | dhcpClientIdentifier | specify client unique identifier |
| 255 | endOption | marks the end of valid information |

In the Option Field of 312 Bytes, the unused bytes are denoted with 0's padding.

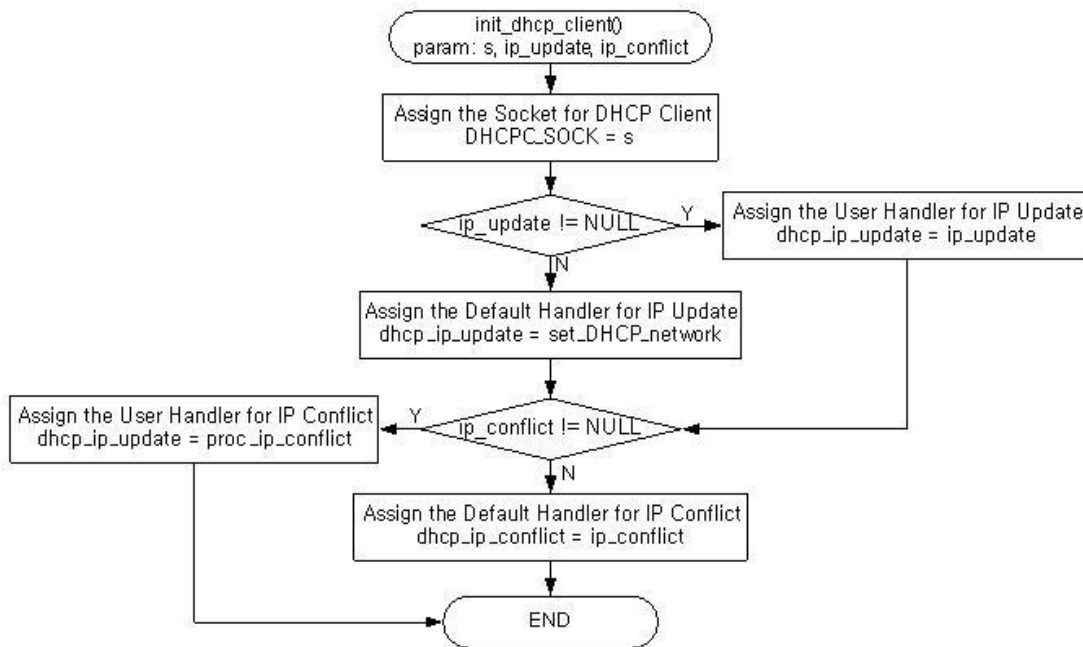
<Table 3-34> is defined as enumeration data type in "inet/dhcp.h" and shows most common Option Codes that are used in DHCP Client Program.

Other codes that are not defined in <Table 3-34> are skipped from DHCP Client Program.

The operation of DHCP Client Program is displayed in EVB B/D's main(). Refer to <Fig 3.3>.

First, set up the MAC address to be used by DHCP Client at the initialization. MAC address is unique address for all the devices in the network. MAC address is most basic address in Network communication and necessary information to recognize DHCP Clients in DHCP Server. For MAC Address of DHCP Client program, it sets up SRC_MAC_ADDR which is global variable of DHCP client using the MAC Address of EVB B/D. By calling `init_dhcp_client()` after setup of SRC_MAC_ADDR, it can register two functions to be called in case of collision of the IP received from DHCP Server and in case of renewal the IP from DHCP Server.

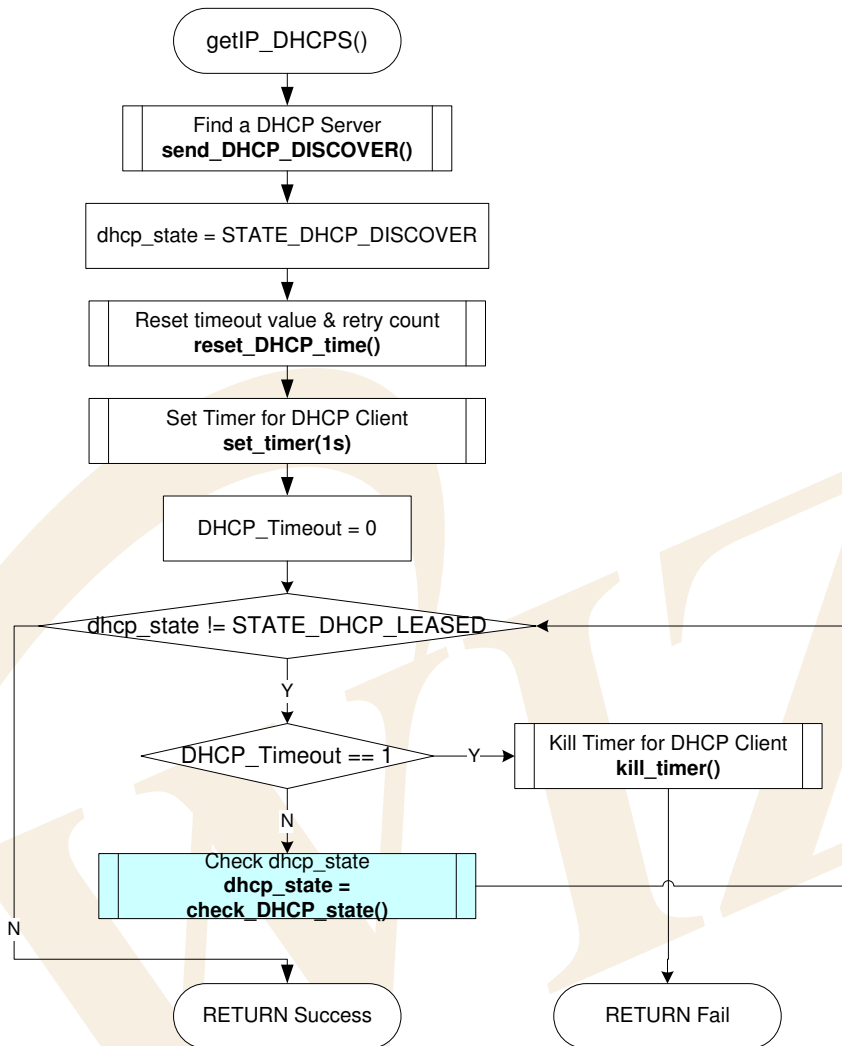
When calling `init_dhcp_client()`, if each function is not specified, `set_DHCP_network()` and `proc_ip_conflict()` of DHCP Client Program respectively.



<Fig 3.28: `init_dhcp_client()`>

When network information is renewed or IP collision occurs, register `evb_soft_reset()` to run auto reset for EVB B/D.

Second, Network Information acquirement can be done through `getIP_DHCP()`.



<Fig 3.29: getIP_DHCP()>

getIP_DHCP() initializes W5100 using setIP(), setMACAddr(), and etc, and it initializes 'dhcp_state' variable as DHCP client program state to 'STATE_DHCP_DISCOVER'. After the initialization, it calls send_DHCP_DISCOVER() to transfer a DHCP DISCOVERY message to DHCP server.

After transmitting DISCOVERY DHCP message, it initializes timer variables which are the leased time of network information received from DHCP server by calling reset_DHCP_time() and uses 'DHCP Timer' for 1-sec interval using set_timer(). After the initialization of DHCP_Timeout with 0, it waits for DHCP message to be received from DHCP server as long as the 'DHCP_WAIT_TIME' defines and as many as the 'MAX_DHCP_RETRY' defines. While waiting for 'DHCP_WAIT_TIME & MAX_DHCP_RETRY' time, it continuously checks if dhcp_state is changed to STATE_DHCP_LEASED through check_DHCP_state().

STATE_DHCP_LEASED state represents the network information and means that getIP_DHCP() is done successfully. If network information is not obtained from DHCP Server during the waiting time for 'DHCP_WAIT_TIME & MAX_DHCP_RETRY', check_DHCP_state() sets DHCP_Timeout to 1. When

DHCP_Timeout is 1, getIP_DHCP() returns failure after releasing the DHCP Timer.

When it failed to obtain network information from DHCP server, EVB B/D sets network configuration using default network information or previously obtained network information.

<Table 3-35> is a definition of State, Timeout, and Retry Count of DHCP Client.

<Table 3-34: DHCP Client State & Timeout Definition>

| Define | Description |
|--------------------------------|---|
| #define STATE_DHCP_DISCOVER 1 | DISCOVERY Transmission |
| #define STATE_DHCP_REQUEST 2 | OFFER Receiving & REQUEST Transmission |
| #define STATE_DHCP_LEASED 3 | ACK Receiving, Acquiring Network Information |
| #define STATE_DHCP_REREQUEST 4 | After obtaining Network Information, REQUEST Retransmission |
| #define STATE_DHCP_RELEASE 5 | RELEASE Transmission |
| #define MAX_DHCP_RETRY 3 | Number of Same DHCP Message Transmission, 3 times |
| #define DHCP_WAIT_TIME 5 | Waiting time for receiving DHCP Message, 5 sec. |

At getIP_DHCP(), 'DHCP_XID' is variable to set up xid Field of DHCP message in <Fig 3.26: DHCP Message Format>, it must be unique and maintain the same value until Lease Time of network information is expired. DHCP_XID is fixed with '0x12345678' on here, but it's recommended to use the random value.

Be advised to set source IP address as '0.0.0.0' when initializing W5100 for communication with DHCP server. You can use any IP address to set Source IP address of W5100, but using '0.0.0.0' is better because '0.0.0.0' corresponds to Class A in IPv4 addressing and it's a Null IP address that is not actually used. For this reason, there is no chance for collision with other network.

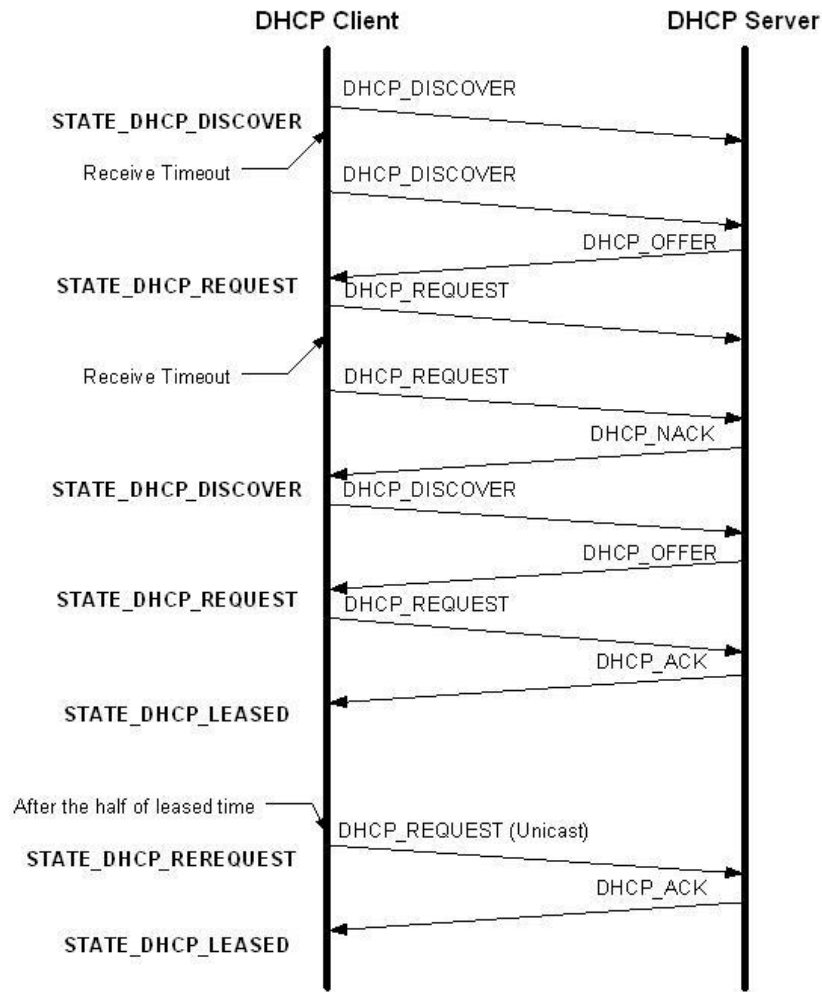
For DHCP server to transmit UDP broadcast packet, note that Flag field MSB of DHCP message must be set 1. Refer to <Fig 3.26: DHCP Message Format>.

<Table 3-36> is a part of code that sets up Flag field

<Table 3-35: DHCP Message Flag Field Setup>

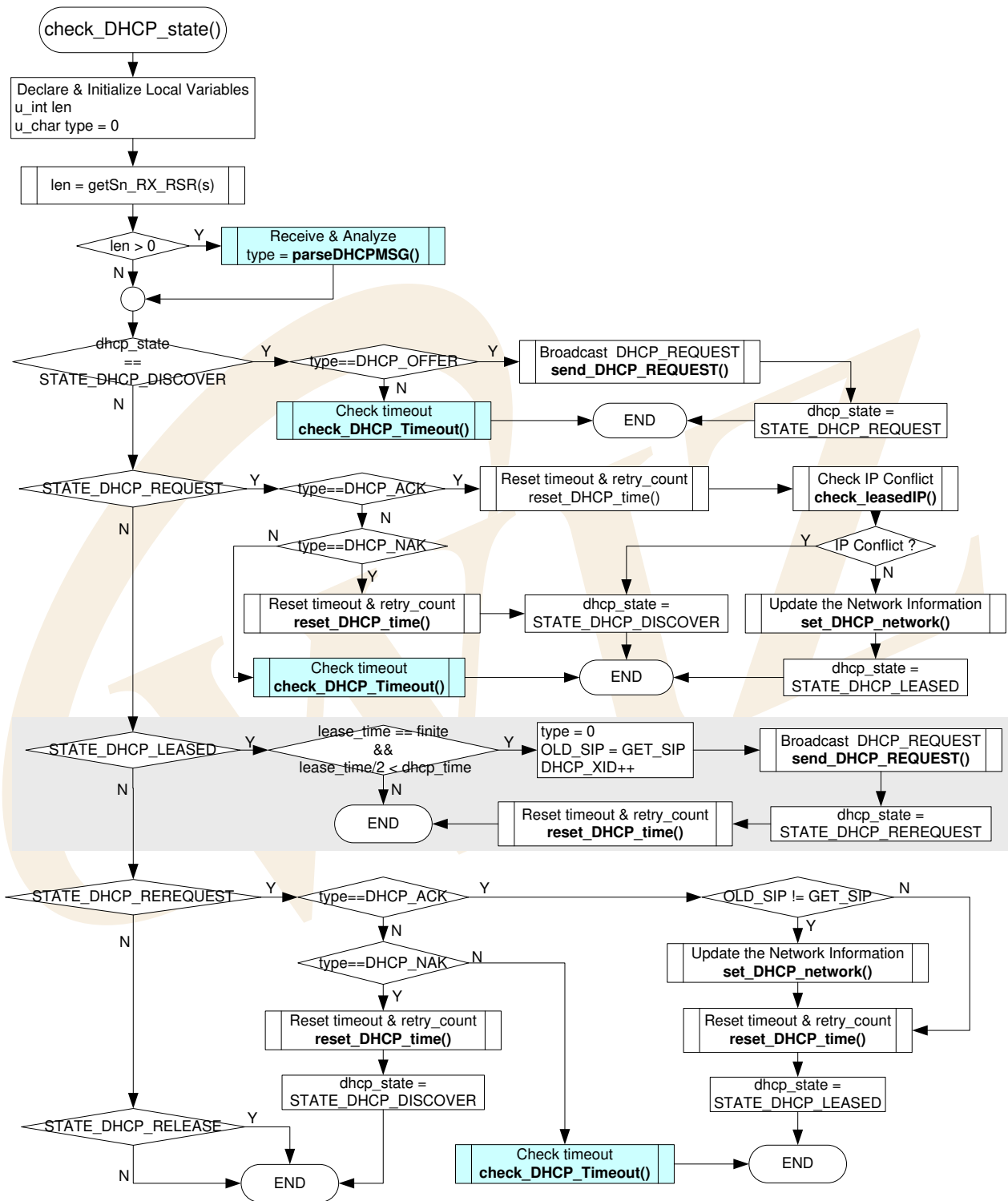
| | |
|--|--------|
| #define DHCP_FLAGSBROADCAST | 0x8000 |
| pRIPMSG->flags = htons(DHCP_FLAGSBROADCAST); | |

Third, management of network information obtained from DHCP server can be performed by check_DHCP_state(). <Fig 3.30> shows DHCP message flow due to DHCP client state change in the check_DHCP_state() process.



<Fig 3.30: DHCP Message Flow by DHCP Client State>

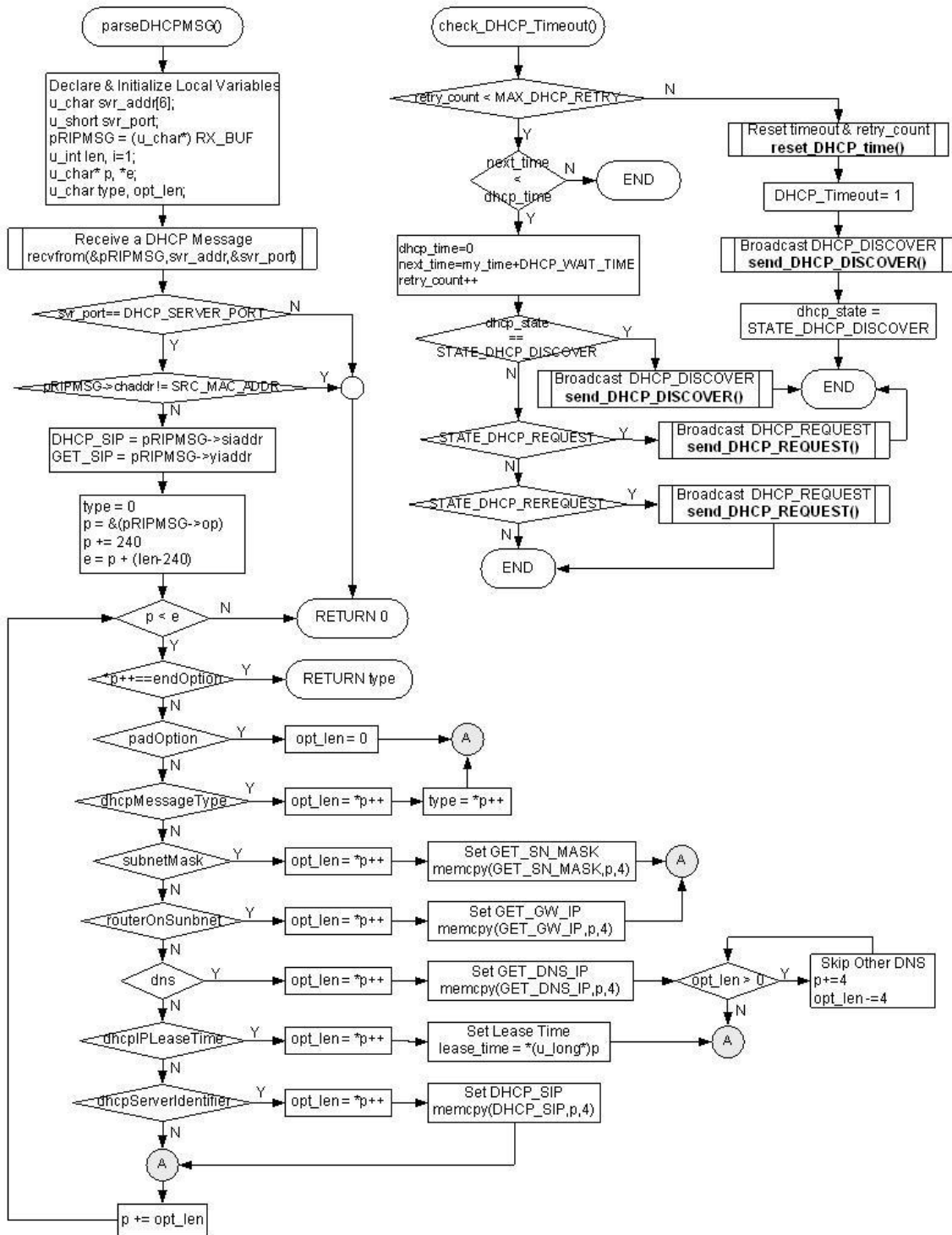
check_dhcp_state() checks if there is DHCP message from DHCP server. It receives and analyzes DHCP message. According to the types of analyzed DHCP message, if it's DHCP message that can be receivable, it changes to next state after it changes DHCP Client State as shown DHCP Message Flow of <Fig 3.30>.



<Fig 3.31: check_DHCP_state(>

check_DHCP_state() processes correspondingly with DHCP client state through the series of processes shown in <Fig 3.31>. If we take a look at DHCP_STATE_LEASED state at check_DHCP_state(), the Lease

Time received from DHCP server is finite, in case that half of the Lease Time passed, it sends DHCP_REQUEST Message to DHCP Server and changes it as DHCP_STATE_REREQUEST after it backs up the source IP. As it continuously transmits DHCP_REQUEST to the server, network information is maintained.



<Fig 3.32: parse_DHCPMSG() & check_DHCP_Timeout(>

parseDHCPMSG() receives DHCP message from DHCP server, categorizes Type of DHCP Message, and saves network information. When performing check_DHCP_state(), check_DHCP_Timeout() is called in case that DHCP message is not received during the DHCP_WAIT_TIME or received DHCP message from DHCP server is not expected, to retransmit DHCP message to DHCP server. If the retransmission of DHCP message is repeated as much as MAX_DHCP_RETRY, it transmits DHCP_DISCOVER message to DHCP server after it initializes all the variables to start the connection of DHCP server and DHCP message.

<Table 3-36: Reference Functions in DHCP Client>

| Function Name | Description | Location |
|---|--|-------------|
| void init_dhcp_client(SOCKET s, void (*ip_update)(void), void (*ip_conflict)(void)) | Initializes DHCP Client | inet/dhcp.c |
| u_int getIP_DHCP(SOCKET s) | Obtains network information from the server | inet/dhcp.c |
| void check_DHCP_state(SOCKET s) | Manages network information obtained from DHCP Server | inet/dhcp.c |
| void set_DHCP_network(void) | Applies network information obtained from DHCP server to W3150A ⁺ . | inet/dhcp.c |
| char parseDHCPMSG(SOCKET s, u_int length) | Analyzes and processes DHCP message | inet/dhcp.c |
| void check_DHCP_Timeout(void) | Retransmits the DHCP message when DHCP connection Timeout occurs | inet/dhcp.c |
| char check_leasedIP(void) | Check if the IP obtained from DHCP server is faced with collision. | inet/dhcp.c |
| void reset_DHCP_time(void) | Initializes DHCP Timer related variables. | inet/dhcp.c |
| void DHCP_timer_handler(void) | DHCP Timer Handler | inet/dhcp.c |
| void send_DHCP_DISCOVER(SOCKET s) | Transmits DHCP_DISCOVER message to DHCP server. | inet/dhcp.c |
| void send_DHCP_REQUEST(SOCKET s) | Transmits DHCP_REQUEST message to DHCP server. | inet/dhcp.c |
| void send_DHCP_RELEASE_DECLINE(SOCKET s, char msgtype) | Transmits DHCP_DISCOVER/DHCP_DECLINE message to DHCP server | inet/dhcp.c |
| u_int init_dhcp_ch(SOCKET s) | Creates DHCP client socket. | inet/dhcp.c |

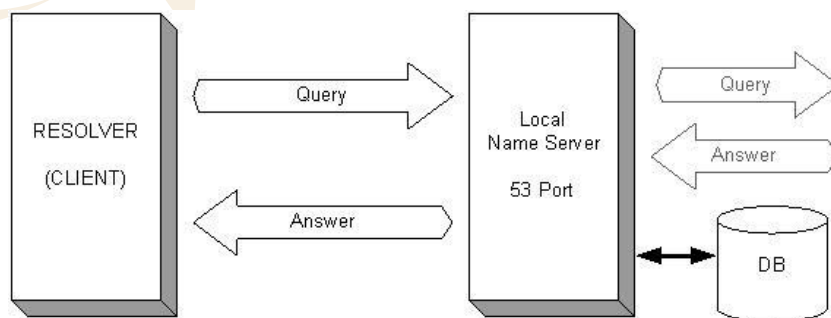
| | | |
|--|--|------------------|
| uint8 getSn_SR(SOCKET s) | Informs status of socket | iinChip/w5100.c |
| uint16 getSn_RX_RSR(SOCKET s) | size of data transmittable, and received data | iinChip/w5100.c |
| u_char socket(SOCKET s, u_char protocol, u_int port, u_char flag) | Creates sockets as TCP/UDP/IP | iinChip/socket.c |
| u_int sendto(SOCKET s, const u_char * buf, u_int len, u_char * addr, u_int port) | Transmits data through specific port of specific Destination | iinChip/socket.c |
| u_int recvfrom(SOCKET s, u_char * buf, u_int len, u_char * addr, u_int * port) | Receives data through any port of any destination. | iinChip/socket.c |
| void close(SOCKET s) | Closes the Socket | iinChip/socket.c |

3.2.6.6. DNS Client

Let's take a brief look at the DNS(Domain Name System) before DNS Client setup is introduced.

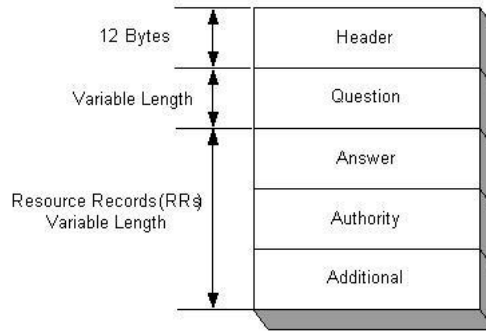
DNS is a system that transforms Internet Domain Name to Internet IP Address or Internet IP Address to Internet Domain Name. DNS is composed of Name Server that contains mapping table between IP Address and Domain Name, and DNS resolver that receives query results by transmitting query to Name Server.

DNS resolver queries IP address or Domain Name to be transformed to local Name Server. Local Name Server which received the Query searches its DB and answers back to the Resolver. If Resolver cannot find the information it looks up, Local Name Server sends the received query to Name Server at higher layer and the received answer can be sent to the Resolver.



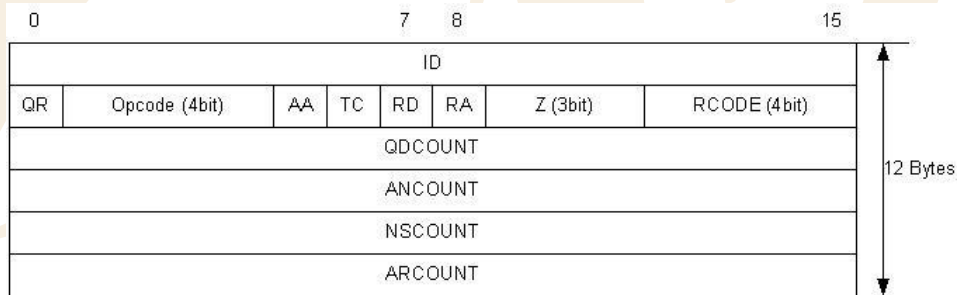
<Fig 3.33: Domain Name System Structure & DNS Message Flow>

As seen in <Fig 3.33>, DNS Query and DNS Answer Message transmittable between DNS Resolver and Name Server are composed of 5 Sections in <Fig 3.34>.

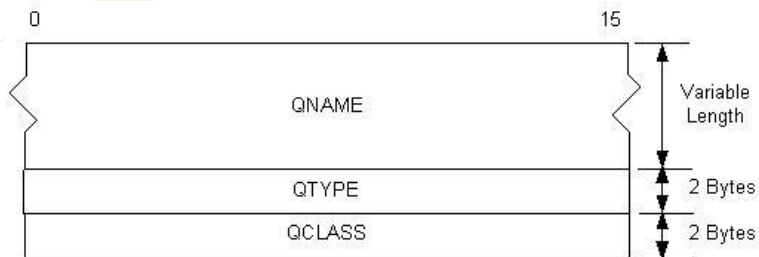


<Fig 3.34: DNS Message Format>

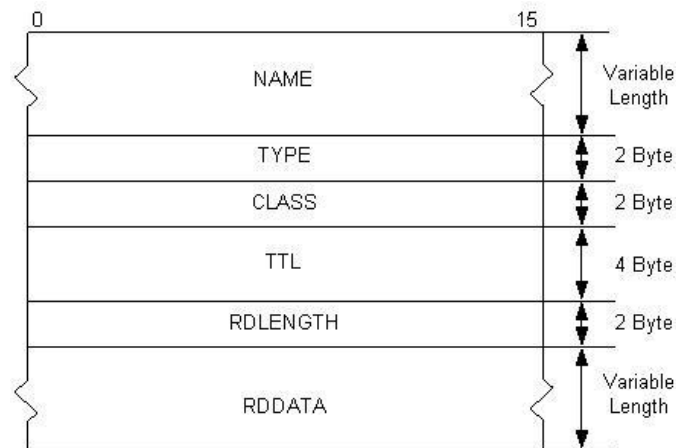
Header Section has fixed 12 Bytes length and the other 4 sections have variable lengths. Answer, Authority, Additional Section other than Header and Question Section are called Resource Records(RRs). Each of Header, Question, and RRs has different format.



<Fig 3.35: Header Section Format>



<Fig 3.36: Question Section Format>



<Fig 3.37: Recode Resources Format>

Header Section of DNS Message holds type of Message, DNS Query type, and count information on variable length section.

In <Fig 3.35: Header Section Format>, QR field gets 0 when DNS Message is a request from Resolver to Name Server and gets 1 when it's from Name Server to Resolver. Opcode Field gets 0 when it queries Domain Name as IP Address and gets 2 when it queries Name Server status.

QDCOUNT, ANCOUNT, NSCOUNT, and ARCOUNT Field, count information for variable length, represent Block Count that is composed of Question, Answer, Authority, and additional section. Question section is made of blocks shown in <Fig 3.36: Question Section Format>. Answer, Authority, and Additional Sections are composed of blocks shown in <Fig 3.37>.

For example, if QDCOUNT is 1, ANCOUNT is 10, NSCOUNT is 10, and ARCOUNT is 10 then Question Section is composed of block 1 of <Fig 3.36: Question Section Format>. Answer, Authority, and Additional Section are composed of 10 blocks shown in <Fig 3.37>.

NAME of <Fig 3.37>, QNAME Filed of <Fig 3.36> and RDDATA Field also get variable lengths. QNAME and NAME are variable length fields which are composed of <Fig 3.36> Format and they process each field. RDDATA, variable length field, processes using the data length of RDLENGTH Field.

For further details, refer to RFC1034 and RFC1035

DNS Message is operated by Data Type defined in <Table 3-38>. Refer to "inet/dns.h"

<Table 3-37: DNS Message Data Type>

```

/* Header Section */
typedef struct _DHDR
{
    u_int    id;                               /* Identification */
    u_char  flag0;
    u_char  flag1;
    u_int    qdcount; /* Question count */
    u_int    ancount; /* Answer count */
    u_int    nscount; /* Authority (name server) count */
    u_int    arcount; /* Additional record count */
}DHDR;

/* Question Section */
typedef struct _QUESTION
{
    // char* qname;                          // Variable length data
    u_int  qtype;
    u_int  qclass;
}DQST;

/* Resource Records */
typedef struct _RESOURCE_RECORD
{
    // char*   _name;                          // Variable length data
    u_int    _type;
    u_int    _class;
    u_long   _ttl;
    u_int    _rdlen;
    // char*   _rdata;                          // Variable length data
}DRR;
    
```

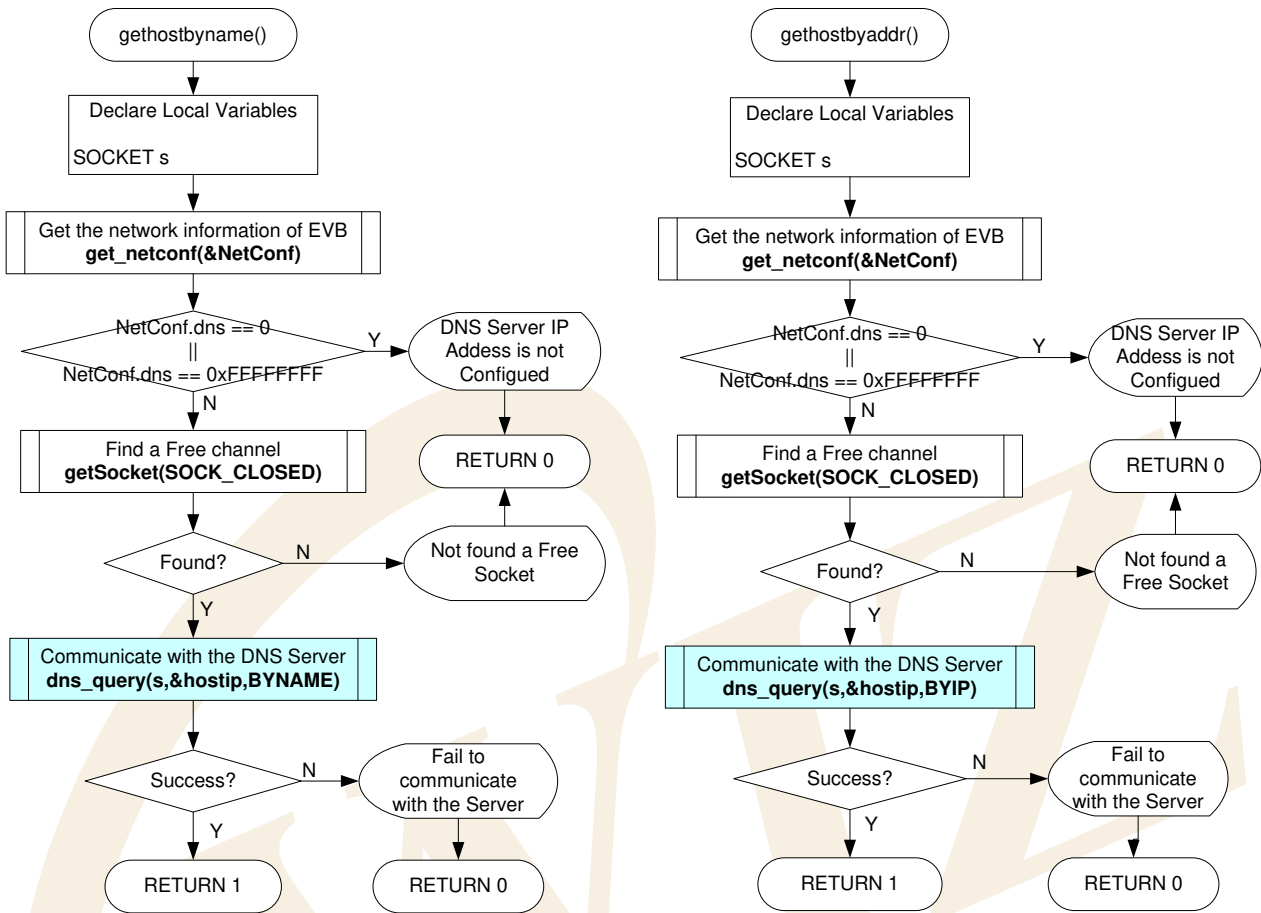
DNS Resolver works based on `gethostbyaddr()` and `gethostbyname()`. `gethostbyaddr()` transforms Internet IP Address to Internet Domain Name and `gethostbyname()` transforms Internet Domain Name to Internet IP Address. `gethostbyaddr()` and `gethostbyname()` test the setup of DNS Name Server IP Address and search free channels of W5100 needed for connection with DNS Name Server. If a free channel of W5100 exists, `gethostbyaddr()` and `gethostbyname()` call `dns_query()` with 'BYNAME' or 'BYIP' as the elements.

For examples of `gethostbyaddr()` and `gethostbyname()`, refer to [Chapter 3.2.5.3](#) Ping Request Program.

Actual connection with DNS Name Server is performed through `dns_query()`, and `gethostbyaddr()` and `gethostbyname()` are reporting only the result of `dns_query()`.

<Table 3-38: Query Type Definition at `dns_query()`>

```
typedef enum _QUERYDATA{BYNAME,BYIP}QUERYDATA; /* Query type */
```



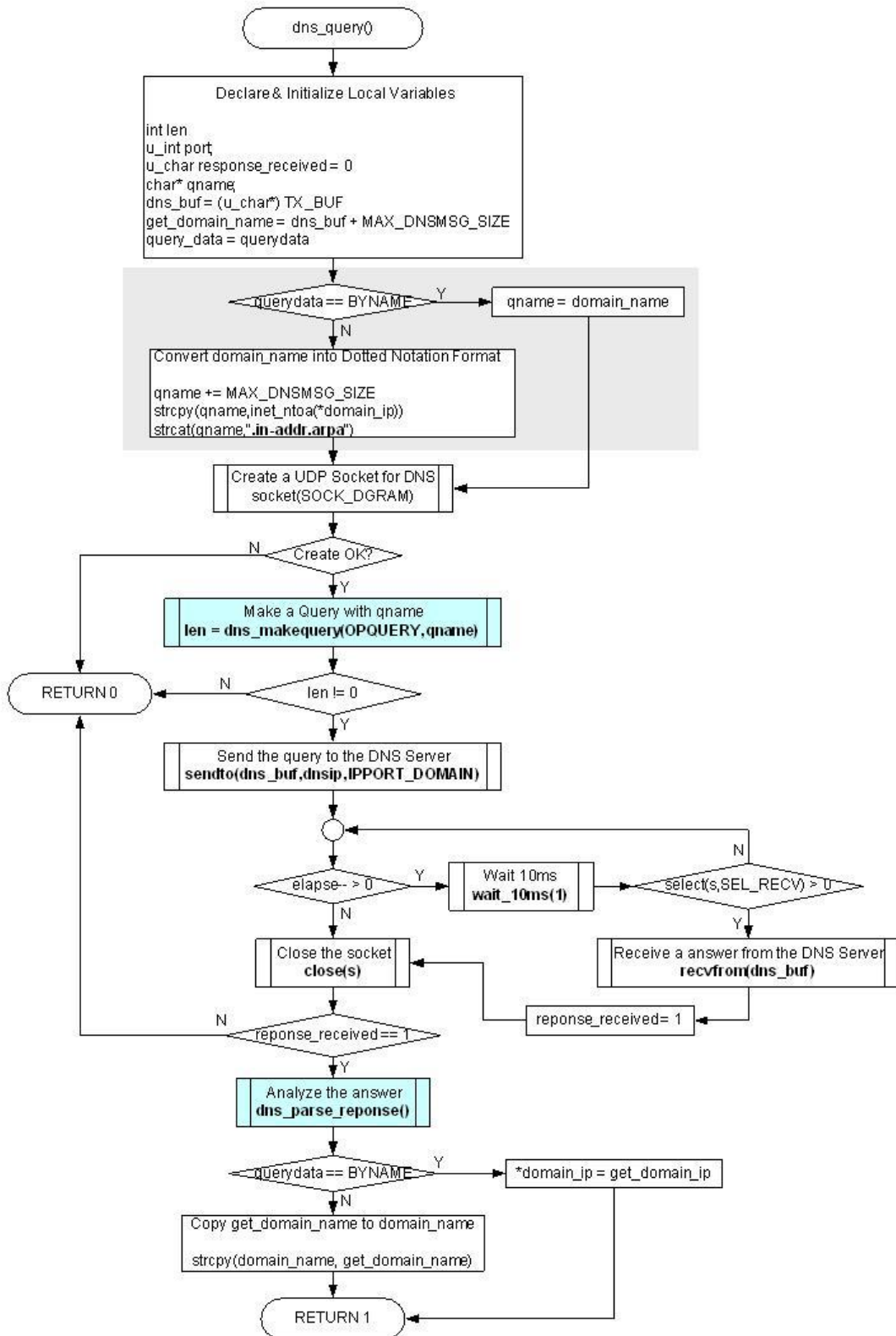
<Fig 3.38: gethostbyaddr() & gethostbyname()>

dns_query() initializes the buffer that is needed for DNS inter-working and creates QNAME of Question Section based on Query Type 'BYNAME', and 'BYIP.' If the Query Type is 'BYNAME,' that is, when querying the Domain Name with IP Address, Domain Name can be used as QNAME without transformation.

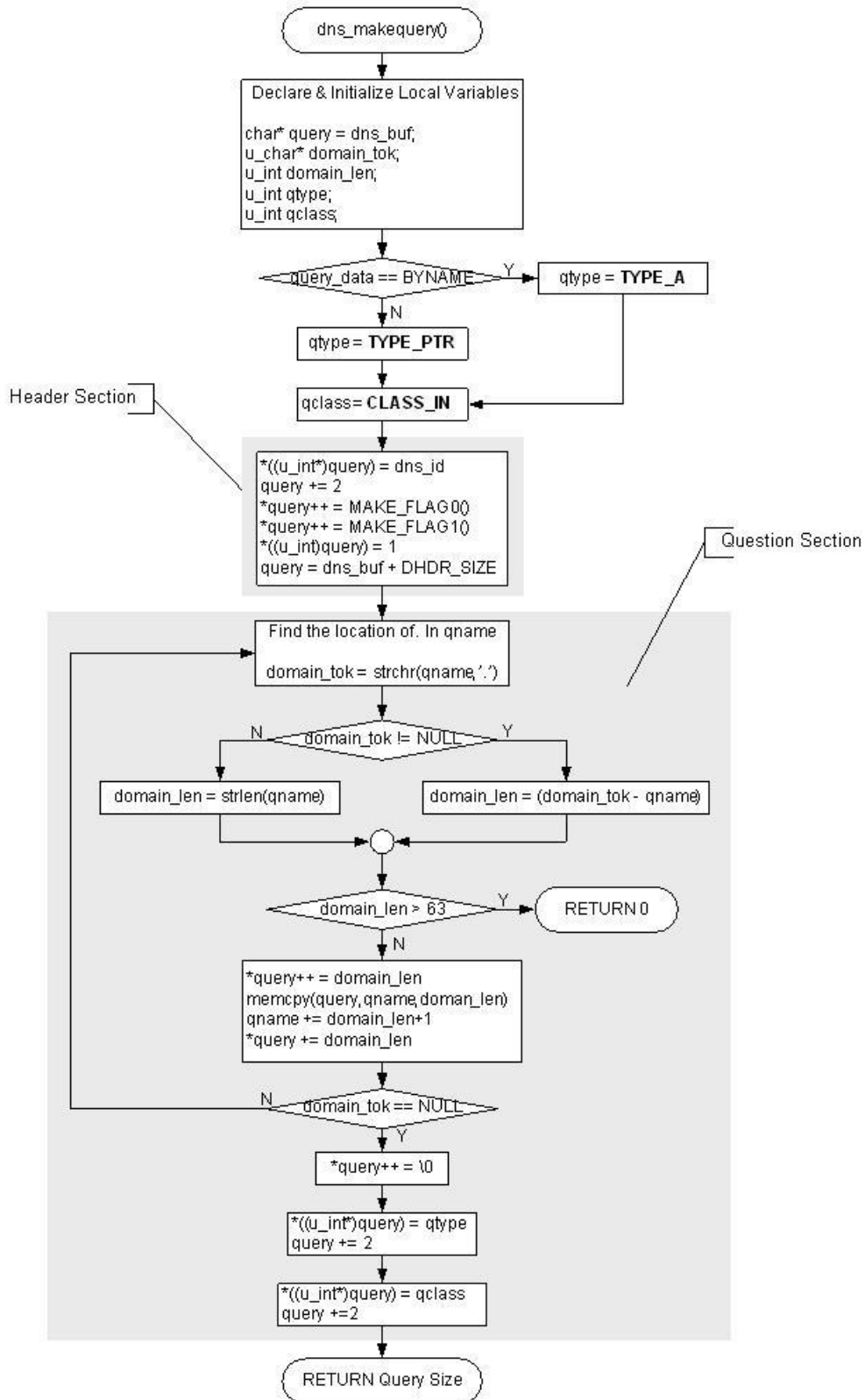
When Query Type is 'BYIP,' that is, when querying the Domain Name with IP Address, change IP Address to IP Address String and QNAME is used after adding "in-addr.arpa" to the changed IP Address String. After the creation of QNAME, UDP Socket is created for DNS inter-working and DNS Request Message is created by calling dns_make_query(). If DNS Request Message is created successfully DNS Request Message is sent to DNS Name Server through UDP Socket. After sending DNS Request Message it receives DNS Response Message or waits until the waiting time is expired.

If DNS response message is received from DNS name server during the waiting time, it analyzes received DNS response message using dns_parse_response(). dns_query() returns IP Address or Domain Name depending on Query Type.

<Fig 3.39> is dns_query()'s process map



<Fig 3.39: dns_query(>



<Fig 3.40: dns_makequery()>

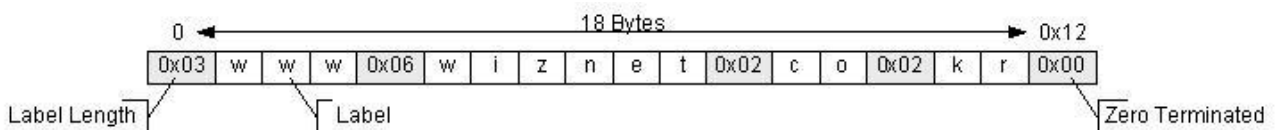
dns_makequery() creates DNS Request message to be sent to DNS Name Server. Since DNS Request Message can query only with Header, Question Section, RRs Sections is not needed to be created. If you examine the header section creation at dns_makequery(), first, it sets ID Field values as any value in DNS Message inter-working. On here, ID is set with 0x1122, and for further inter-working, the value is incremented by 1. QR, Opcode, AA, TC, RD Field are set as QR_QUERY, OP_QUERY/OP_IQUERY, 0, 0, 1 respectively through MAKE_FLAG0(), and RA, Z, RCODE Field are set as 0, 0, 0 respectively through MAKE_FLAG1().

<Table 3-39: Constants and MACRO used in Header Section>

| | | |
|---|--|--|
| #define QR_QUERY | 0 | |
| #define QR_RESPONSE | 1 | |
| #define OP_QUERY | 0 | /* a standard query (QUERY) */ |
| #define OP_IQUERY | 1 | /* an inverse query (IQUERY) */ |
| #define OP_STATUS | 2 | /* a server status request (STATUS) */ |
| #define MAKE_FLAG0(qr, op, aa, tc, rd) | $((qr \& 0x01) \ll 7) + ((op \& 0x0F) \ll 3) + ((aa \& 0x01) \ll 2) + ((tc \& 0x01) \ll 1) + (rd \& 0x01)$ | |
| #define MAKE_FLAG1(ra, z, rcode) | $((ra \& 0x01) \ll 7) + ((z \& 0x07) \ll 4) + (rcode \& 0x0F)$ | |

Since the count fields, QDCOUNT, ANCOUNT, NSCOUNT, and ARCOUNT, have only one question, each is set as 1, 0, 0, 0 respectively.

Let's look at Question Section. QNAME Field is the field that sets IP Address string. Domain Name and IP Address string are composed of label length of 1 byte and label of MAX 63 Byte. The end of QNAME is always set with 0 to find out the variable length of QNAME. <Fig 3.41> is actual example of transformation of Domain Name "www.wiznet.co.kr" in QNAME field.



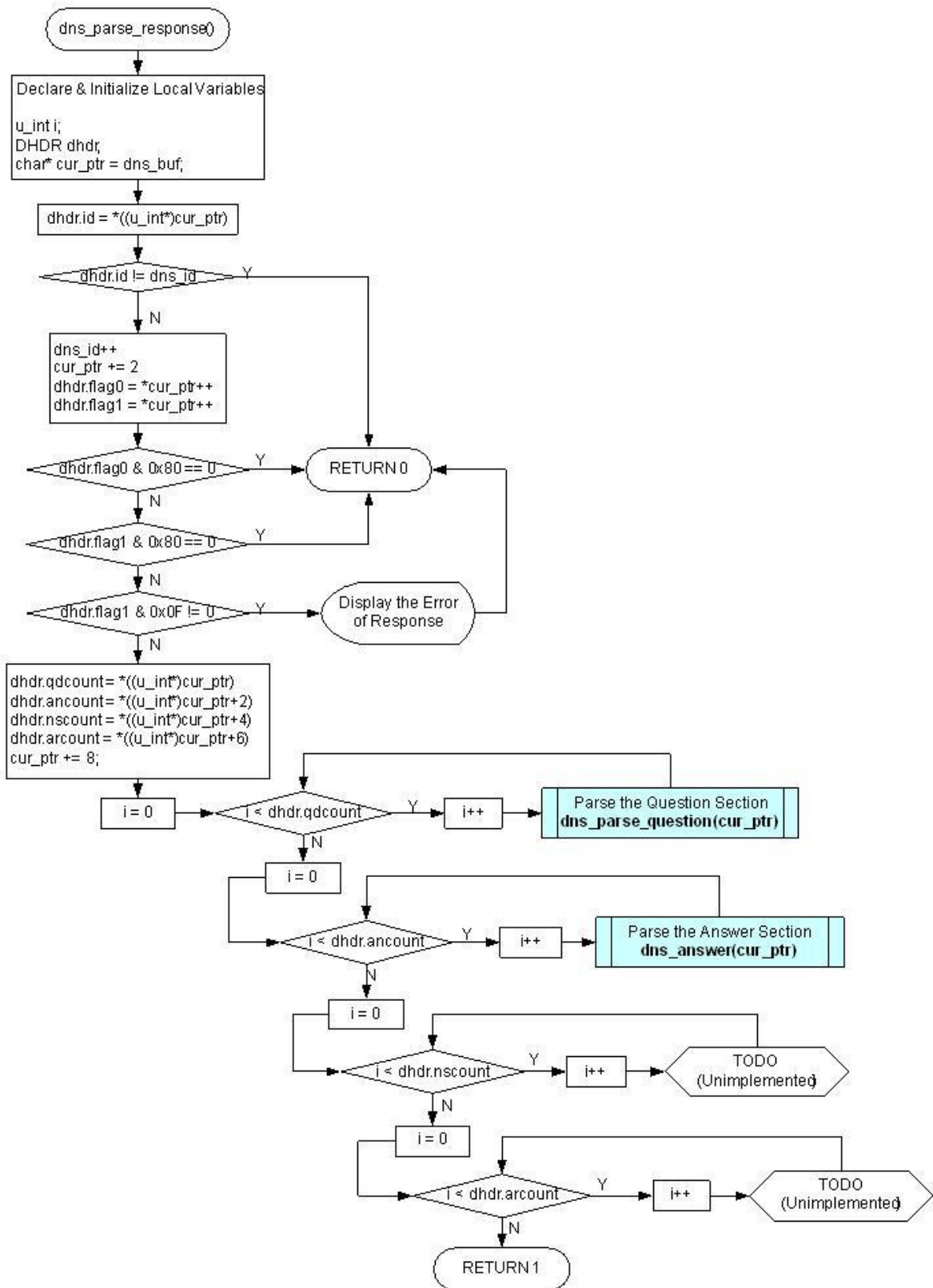
<Fig 3.41: Example of QNAME Field transformation of Question Section >

QTYPE Field of Question Section is set 'TYPE_PTR', when it holds Domain Name as QNAME. When it's IP address, it's set as 'TYPE_A', and QCLASS field is set as 'CLASS_IN' since it is included in Internet.

Table 3-41 is definition of constants that are used in QTYPE & QCLASS Fields.

<Table 3-40 : Constants Definition at QTYPE & QCLASS Field>

| Definition | | Description |
|-------------------------|-----|--|
| #define TYPE_A | 1 | The ARPA Internet |
| #define TYPE_NS | 2 | an authoritative name server |
| #define TYPE_MD | 3 | a mail destination (Obsolete - use MX) |
| #define TYPE_MF | 4 | a mail forwarder (Obsolete - use MX) |
| #define TYPE_CNAME | 5 | the canonical name for an alias |
| #define TYPE_SOA | 6 | marks the start of a zone of authority |
| #define TYPE_MB | 7 | a mailbox domain name |
| #define TYPE_MG | 8 | a mail group member |
| #define TYPE_MR | 9 | a mail rename domain name |
| #define TYPE_NULL | 10 | a null RR |
| #define TYPE_WKS | 11 | a well known service description |
| #define TYPE_PTR | 12 | a domain name pointer |
| #define TYPE_HINFO | 13 | host information |
| #define TYPE_MINFO | 14 | mailbox or mail list information |
| #define TYPE_MX | 15 | mail exchange |
| #define TYPE_TXT | 16 | text strings |
| #define QTYPE_AXFR | 252 | A request for a transfer of an entire zone |
| #define QTYPE_MAILB | 253 | A request for mailbox-related records |
| #define QTYPE_MAILA | 254 | A request for mail agent RRs |
| #define QTYPE_TYPE_ALL | 255 | A request for all records |
| #define CLASS_IN | 1 | Internet |
| #define CLASS_CS | 2 | CSNET class |
| #define CLASS_CH | 3 | CHAOS class |
| #define CLASS_HS | 4 | Hesiod [Dyer 87] |
| #define QCLASS_ANY | 255 | Any class |



<Fig 3.42: dns_parse_response()>

dns_parse_response() of <Fig 3.42> analyzes Response Message received by DNS Name Server. dns_parse_response() checks if it's same as Request Message ID that was sent to DNS Name Server and it also checks if the message received is a response message by checking QR Field of Header Section. If the received message is response from DNS Name Server, the success of change is decided by checking the RCODE Field value of Header Section.

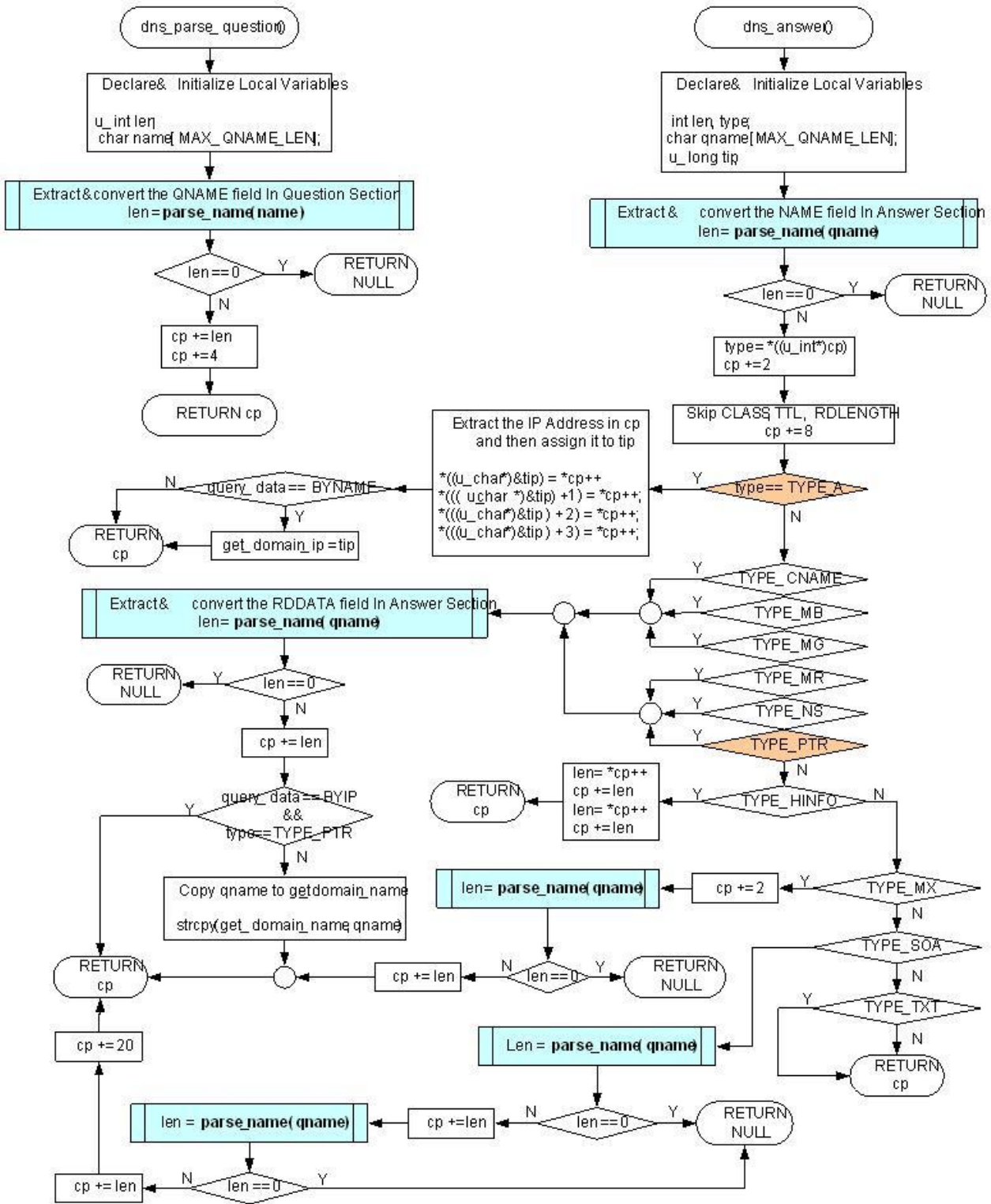
<Table 3-42> is definition of constants that are used in RCODE Field.

<Table 3-41 : Constant Definition at Header Section's RCODE Field>

| Definition | Description |
|---------------------------|--|
| #define RC_NO_ERROR 0 | No error condition |
| #define RC_FORMAT_ERROR 1 | Format error - The name server was unable to interpret the query |
| #define RC_SERVER_FAIL 2 | Server failure - The name server was unable to process this query due to a problem with the name server |
| #define RC_NAME_ERROR 3 | Name Error - Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist. |
| #define RC_NOT_IMPL 4 | Not Implemented - The name server does not support the requested kind of query. |
| #define RC_REFUSED 5 | Refused - The name server refuses to perform the specified operation for policy reasons. |

If the RCODE is RC_NO_ERROR, variable length sections such as Question, Answer, Authority, and Additional Section are analyzed. Since the necessary information is set in Answer Section, it's analyzed and processed, and other section analysis and process are not performed. If you need information on Authority and Additional Section, you can get them easily on your own.

Question Section is processed as many as QDCOUNT of Header Section by calling dns_parse_question(). Answer Section is processed as many as ANCOUNT of Header Section by calling dns_parse_question().



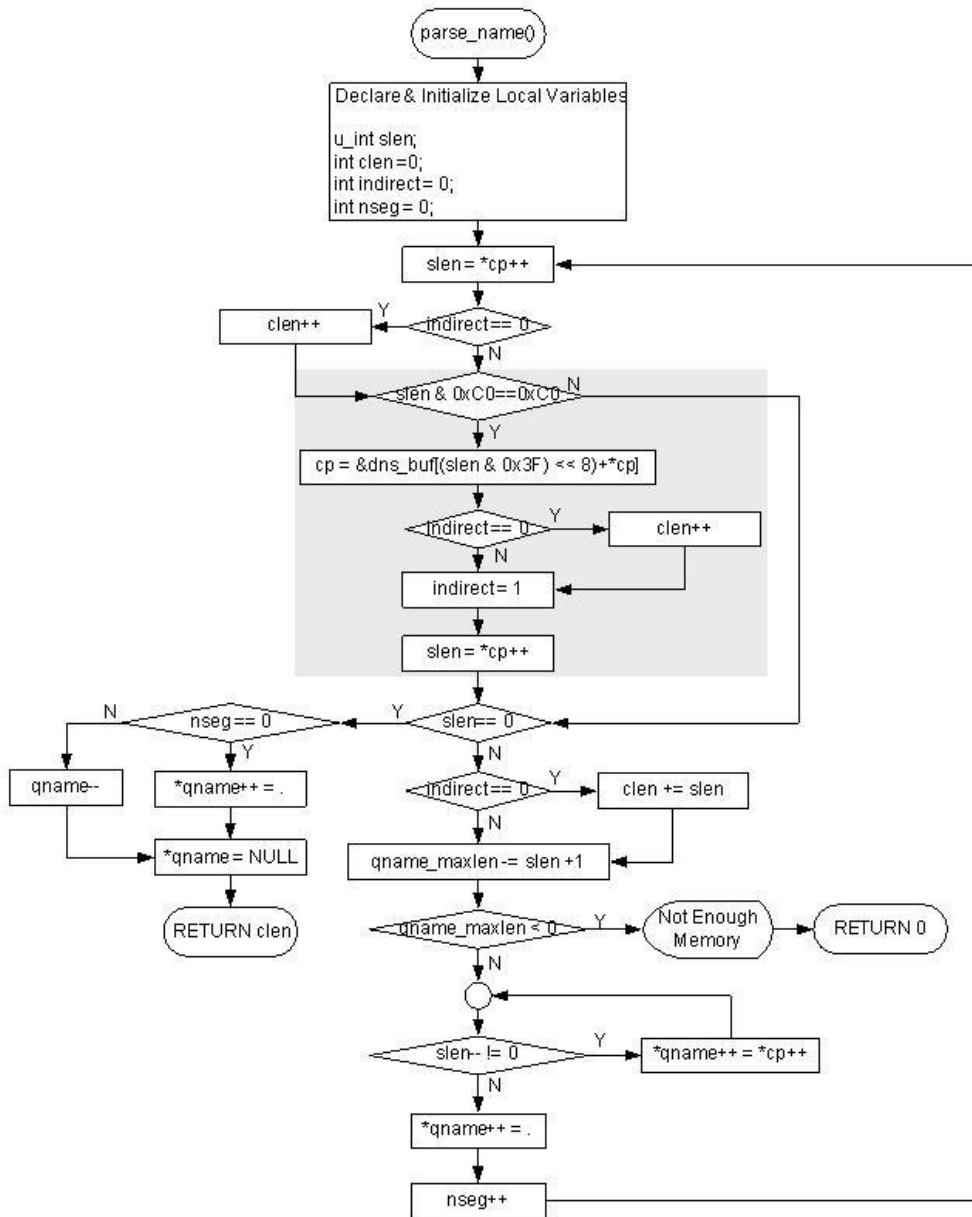
<Fig 3.43: dns_parse_question() & dns_answer()>

dns_parse_question() analyses and processes Question Section. There is no information that actually used in the Question Section of DNS Request Message, but it must be processed to get the starting position of Answer Section. Since QNAME Field of Question Section gets variable length, parse_name() processes

QNAME Field to process the variable length processes and QTYPE, and QCLASS Field are skipped.

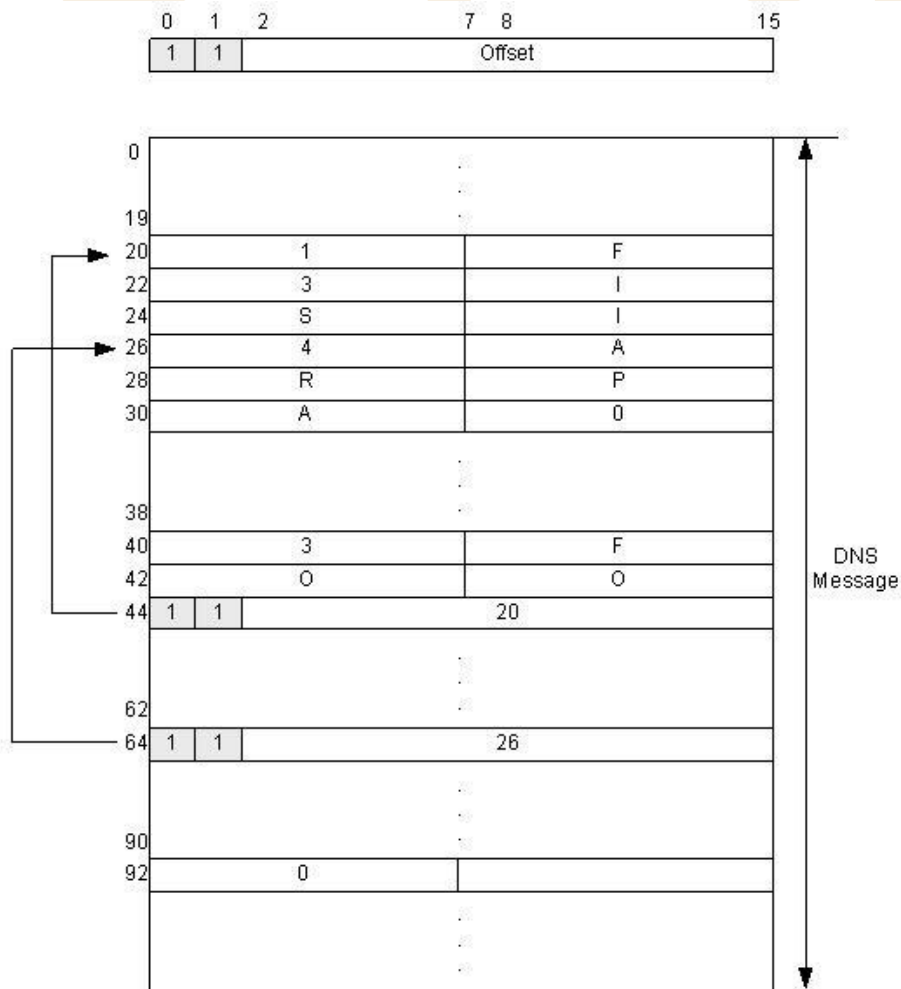
dns_answer() analyzes and processes Answer Section. Answer Section is a section where transformation actually takes effects and it performs appropriate process to TYPE Field of Answer Section.

TYPE of Answer Section has one of values from <Table 3-41 : Constants Definition at QTYPE & QCLASS Field> and the value comes from either TYPE_A or TYPE_PTR. In case that the Domain Name is changed to IP Address, it can get the changed IP Address from TYPE_A and if the IP Address is changed to Domain Name, Domain Name can be obtained from TYPE_PTR. Changed Domain Name or IP Address are also processed and extracted by parse_name().



<Fig 3.44: parse_name()>

parse_name() processes QNAME Field of Question Section or NAME, RRDATA Field of RRs Section. QNAME, NAME, RRDATA Field are mostly composed as in <Fig 3.41: Example of QNAME Field transformation of Question Section >. However, it can be compressed to reduce DNS Message Size. Compression scheme is expressed in 2 Byte. If the first byte - the upper 2 bits are '11,' it means the Label is compressed. It has the offset that is composed of 1ST Byte excluding upper 2 bits and 2nd Byte. This offset is Offset of DNS Message and means the actual value of Label is located by the offset from the starting point of DNS message. When Compress Scheme tries to reuse Domain Name that was already used in DNS Message, relevant Domain Name sets the offset that is located in DNS Message as Indirect so that it can reduce the size of DNS Message. <Fig 3.45> is an example of Compress Scheme of DNS Message and its application.



<Fig 3.45: DNS Message Compression Scheme>

The example of Compression Scheme of <Fig 3.45> shows DNS Message in case of "F.ISI.ARPA", "FOO.F.ISI.ARPA", "ARPA", and ROOT. "F.ISI.ARPA" is processed in the format of <Fig 3.41: Example of

QNAME Field transformation of Question Section > with Offset 20 of DNS Message without compression.

In "FOO.F.ISI.ARPA," since the rest except for "FOO" is same as Name which is previously processed, "FOO" is processed with <Fig 3.41: Example of QNAME Field transformation of Question Section > Format without compression and the rest of names is processed by Offset 26. ROOT is the highest Domain and it's processed with Label Length Field of 0.

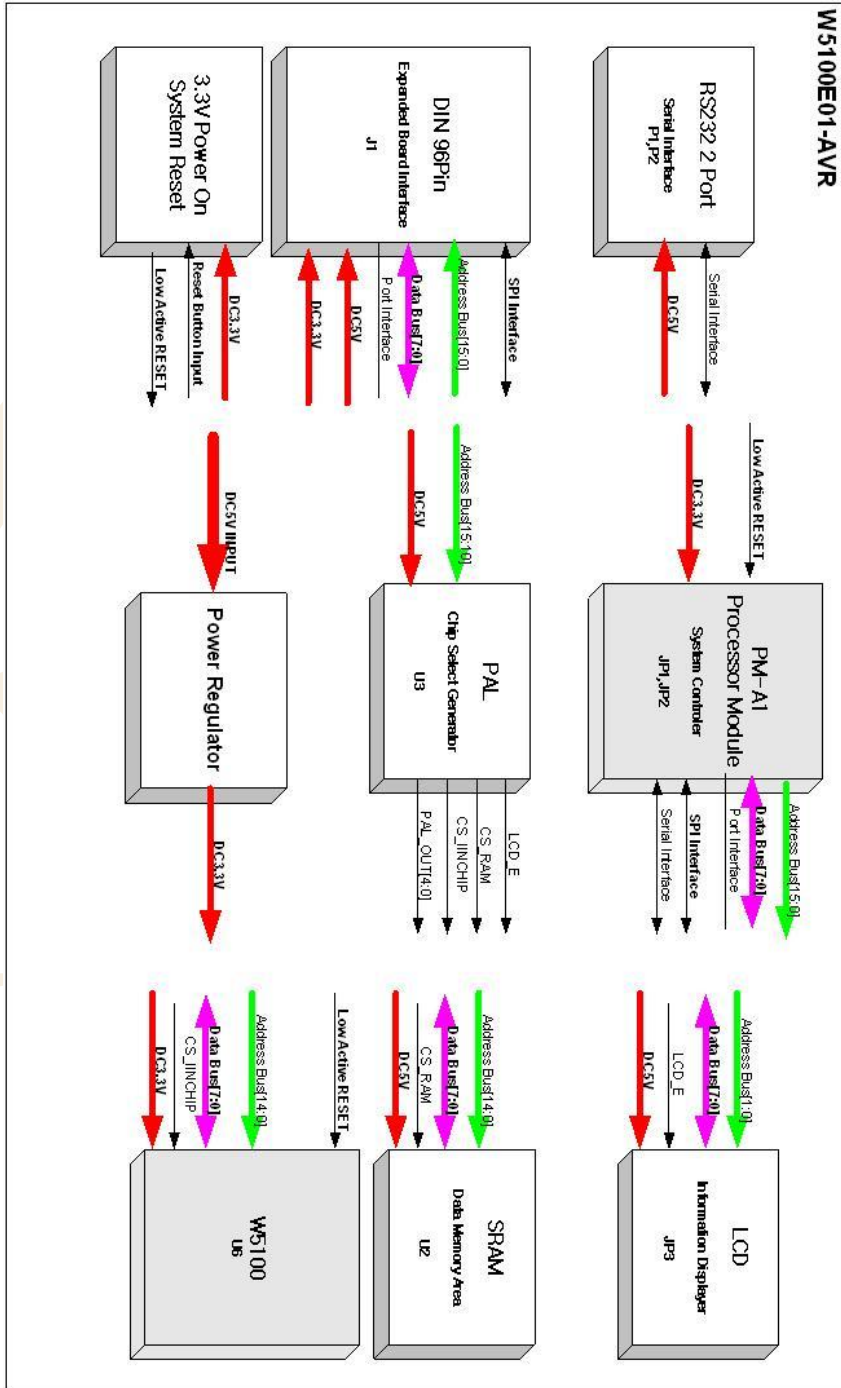
parse_name(), before analysis of Name, checks if upper 2 bits of Label Length Byte are 11, if it's '11' the related Label analyzes the Label at the offset of DNS Message where the Label is located. If it's no '11' then the Label is analyzed and processed like as <Fig 3.41: Example of QNAME Field transformation of Question Section >.

<Table 3-42 : Reference Functions in DNS Client >

| Function Name | Description | Location |
|--|--|------------------|
| int gethostbyaddr (u_long ipaddr, char* domain) | Changes IP Address to Domain Name | inet/dns.c |
| u_long gethostbyname (char* hostname) | Changes Domain Name to IP Address | inet/dns.c |
| u_char dns_query (SOCKET s, u_long dnsip, u_char * domain_name, u_long* domain_ip, QUERYDATA querydata, u_int elapse) | DNS Message Processing | inet/dns.c |
| int dns_make_query (u_char op, char * qname) | Creates DNS Request Message | inet/dns.c |
| Int dns_parse_reponse(void) | Analyzes DNS Response Message | inet/dns.c |
| u_char * dns_parse_question (u_char * cp) | Analyzes Question Section of DNS Response Message | inet/dns.c |
| u_char * dns_answer (u_char *cp) | Answer Section of DNS Response Message | inet/dns.c |
| int parse_name(char* cp, char* qname, u_int qname_maxlen) | Analyzes NAME Field of Question, RRs Section | inet/dns.c |
| uint16 getSn_RX_RSR(SOCKET s) | size of data transmittable, and received data | iinChip/w5100.c |
| u_char socket(SOCKET s, u_char protocol, u_int port, u_char flag) | Creates sockets as TCP/UDP/IP | iinChip/socket.c |
| u_int sendto(SOCKET s, const u_char * buf, u_int len, u_char * addr, u_int port) | Transmits data through specific port of specific Destination | iinChip/socket.c |
| u_int recvfrom(SOCKET s, u_char * buf, u_int len, u_char * addr, u_int * port) | Receives data through any port of any destination. | iinChip/socket.c |
| void close(SOCKET s) | Closes the related Socket | iinChip/socket.c |

4. Hardware Designer's Guide

4.1. Block Diagram



<Fig 4.1: EVB B/D Block Diagram>

4.2. Block Description

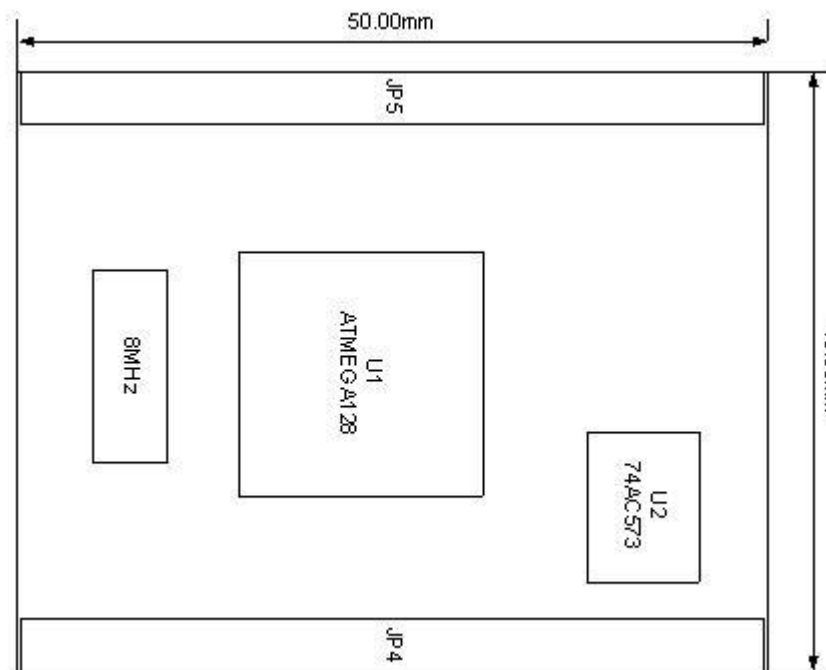
EVB B/D is composed of W5100E01-AVR(EVB Base Board) and PM-A1(AVR MODULE).

Following 9 blocks are components of EVB B/D.

- PM-A1
- LCD
- PAL
- SRAM
- RS232 Port
- Expanded Board Interface
- Power Regulator
- 3.3V Power On System Reset

4.2.1. PM-A1

PM-A1(AVR MODULE) is composed of Atmega128 Processor, 74HC573 for address latch, 8MHz external crystal and header for interfacing to Base board(JP4,JP5), and ISP(JP3) & JTAG(JP1) Interface.



<Fig 4.2: PM-A1 MODULE Dimension>

For easy development using EVB Board, all the port pin except for /ALE(PG2) are connected to MB-EVB-X2 through module Interface(JP4,JP5). Pin description of Interface is shown in <Table 4-1: PM-A1 MODULE Pin

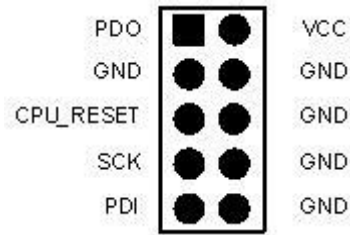
Description>.

<Table 4-1: PM-A1 MODULE Pin Description>

| PM-A1 MODULE Header # | Pin # | Pin Name | Dir. | Description |
|--|--|--|------|--|
| JP4 | 25 ~ 32 | D0(PA0) ~ D7(PA7) | I/O | Databus[0:7] or PA[0:7] |
| JP5 | 26 ~ 33 | PB0 ~ PB7 | I/O | PB[0:7] |
| JP4 | 3 ~ 10 | A0 ~ A7 | I/O | Address bus[0:7] |
| JP4 | 11 ~ 18 | A8(PC0) ~ A15(PC7) | I/O | Address bus[8:15] / PC[0:7] |
| JP5 JP5 JP4 JP4 JP5 JP5 JP5 JP5 | 42 43 47 45 34 35 36 37 | PD0/SCL PD1/SDA PD2/RXD1 PD3/TXD1 PD4 PD5 PD6 PD7 | I/O | PD[0:7] |
| JP4 JP4 JP5 JP5 JP5 JP5 JP5 JP5 | 48 46 38 44 23 46 6 8 | RXD0 PE1/TXD0 PE2 PE3 PE4/I2CHIP_IRQ PE5 PE6 PE7 | I/O | <i>RXD0 is connected with PE0 through 1K ohm resistor.</i> PE[1:7] |
| JP5 | 13 ~ 20 | PF0 ~ PF7 | I/O | PF[0:7] |

| | | | | |
|-----|--------------|-----------|-----|--|
| JP4 | 41 | /WR(PG0) | I/O | PG[0:4] without ALE(PG2) |
| JP4 | 42 | /RD(PG1) | | |
| JP5 | 40 | PG3/LED_0 | | |
| JP5 | 41 | PG4/LED_1 | | |
| JP5 | 4 | CPU_RESET | I | Reset Signal Input process generated by EVB B/D's Reset Switch(SW3). |
| JP5 | 1,2 | 3.3V | I | 3.3V Power Input. |
| JP4 | 1,2 | 5V | I | 5V Power Input Not Used. |
| JP5 | 10,12,21, | GND | | Signal Ground |
| JP5 | 22,45,47, | | | |
| JP5 | 48,49,50 | | | |
| JP4 | 23,24,49,50 | | | |
| JP5 | 3,5,7,9,11, | RES0 | | RESERVED LINE |
| JP4 | 19,20,21,22 | ~ | | |
| | 33,34,35,36, | RES18 | | |
| | 37,38,39,40, | | | |
| | 43,44 | | | |
| JP5 | 24 | NC | | |
| | 25 | NC | | |
| | 39 | NC | | |

AVR ISP (JP3) Pin Mapping



<Table 4-2: ISP Pin Description>

| SIGNAL | Pin Number | I/O | Description |
|-----------|------------|--------|--|
| VCC | 2 | - | Power is delivered to the AVRISP |
| GND | 3,4,6,8,10 | - | Ground |
| PDO | 1 | Input | Commands and data from AVRISP to EVB B/D |
| PDI | 9 | Output | Data from EVB B/D to AVRISP |
| SCK | 7 | Input | Serial Clock, Controlled by AVRISP |
| CPU_RESET | 5 | Input | Reset. Controlled by AVRISP |

4.2.2. LCD

LCD is used for debugging and system status display.

Pin Description of LCD Interface (JP3) is as follows.

<Table 4-3: LCD PIN Description>

| PIN# | EVb B/D PIN NAME/ LCD PIN NAME | DIR. | Description |
|------|-----------------------------------|------|-------------------------------------|
| 1 | GND/VSS | | Signal Ground |
| 2 | 5V/VDD | I | LCD Power Supply |
| 3 | V0/V0 | I | Voltage for LCD drive |
| 4 | A1/RS | I | Data/Instruction register select |
| 5 | A0/RW | I | Read/Write |
| 6 | LDC_E/E | I | Enable signal,start data read/write |
| 7 | D0/DB0 | I/O | Data Bus Line |
| ~ | ~ | | |
| 14 | D7/DB7 | | |
| 15 | NC1/LED A | O | LED Anode, power supply+ |
| 16 | NC2/LED K | O | LED Cathode,ground 0V |

It uses minimum -0.3V and maximum 13V of VDD-V0 at Specification Document of LC1624. To fit the data, R6(5V Pull Up maximum 10K) and R7(Gnd Pull Down 820R) are used and, in real application, LCD Display became clear when R6 was adjusted. For details on LC1624, refer to "LC1624 Specifications" document.

4.2.3. PAL

PAL is used to make enable signal of various chip or module that are used for EVB B/D. The PAL element that is used in the product is ATF16V8B-15PL from ATMEL co. It uses 10 input pins and 8 I/O Pins. It makes Chip Select or Enable Signal about SRAM(/CS_RAM), LCD(LCD_E), and W5100(/CS_IINCHIP). The output, PAL_OUT_0~PAL_OUT_4, are set aside for expansion through Expanded Interface.

4.2.4. SRAM

SRAM, with the size of 32Kbytes, is used as external data memory of Atmega128.

4.2.5. RS232 Port

It's a interface for Dual Serial USARTs that is supported by Atmega128. EVB B/D uses 9Pin DSUB male Type(P1,P2) connector.

4.2.6. Expanded Board Interface

Expanded board interface is designed to be developed easily using EVB B/D. Most of the port pin of Atmega128, output sinal of PAL (PAL_OUT_0~PAL_OUT_4), power and many reserved pin are connected to Expanded Board Interface.

The Signals of Atmega128 that are not connected to Expanded Board Interface are 7 RXD1(PD2), TXD1(PD3), RXD0(PE0), TXD0(PE1), LED0(PG3), LED1(PG4), /I2CHIP_IRQ(PE4).

<Table 4-4: Expanded Board Interface Pin Description>

| Pin # | Pin Name | Dir. | Description |
|---|--|------|---|
| Bus Interface | | | |
| 66,34,67,35, 68,36,69,37, 70,38,71,39, 73,40,74,41 | A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10,A11 A12,A13,A14,A15 | O | Parallel Address Bus[0:15] |
| 77,45,78,46 79,47,80,48 | D0, D1, D2, D3, D4, D5, D6, D7 | I/O | Parallel Data Bus[0:7] |
| 53 86 | /RD /WR | O | Parallel Bus Read Strobe Parallel Bus Write Strobe |
| 25 ~ 29 | PAL_OUT_0 ~ PAL_OUT_4 | O | Reserved Parallel Bus Chip Select / Enable |
| 18 | SDA/PD0 | I/O | I2C Bus Data Line/ Port D0 |

| | | | |
|--|--|-----|--|
| 19 | SCL/PD1 | O | I2C Bus Clock Line/Port D1 |
| Atmega128 Port Interface | | | |
| 20 | PB0 | I/O | Port B[0:7] |
| 21 | PB1 | | |
| 56 | PB2 | | |
| 57 | PB3 | | |
| 58 | PB4 | | |
| 59 | PB5 | | |
| 60 | PB6 | | |
| 61 | PB7 | | |
| 92 | PD4 | I/O | Port D[4:7] |
| 93 | PD5 | | |
| 89 | PD6 | | |
| 90 | PD7 | | |
| 91 | PE2 | I/O | Port E[2:3], Port E[5:7] |
| 22 | PE3 | | |
| 23 | PE5 | | |
| 3 | PE6 | | |
| 5 | PE7 | | |
| 1,2,4,6, 7,75,42,76 43,81,49,83, 50,84,51,85, 52,54,87 | RES0~RES3 RES4~RES7 RES8~RES11 RES12~RES15 RES16~RES18 | | Not Available |
| Power Interface | | | |
| 31,32 | 5V | O | 5V Power Supply |
| 63,64 | 3.3V | O | 3.3V Power Supply |
| 8,9,24,30,44, 55,62,65,72, 82,88,94 | GND | | Ground No. 8 Pin and GND became Short in AVR Module. |

Expanded Board Interface Connector, which is "PCN10BK-96S-2.54DS" of Hirose co., is a Din Connector 96Pin Female Rightangle Type. Connector of Male Type that is mated here is "PCN10-96P-2.54DS."

4.2.7. Power Regulator

EVb B/D gets 5V DC power through power adaptor. The powers used inside the board are 5V and 3.3 V. The regulator is LT1963EST-3.3(U1). To shut down the regulator, Toggle Switch(SW1) is used.

4.2.8. 3.3V Power On System Reset

Manual reset and Power On Reset is implemented using RC analog circuit.



4.3. Schematic

4.3.1. W5100E01-AVR

Please refer to "**W5100E01-AVR.DSN**" in the official website of WIZnet (www.wiznet.co.kr).

4.3.2. PM-A1

Please refer to "**PM-A1.DSN**" in the official website of WIZnet (www.wiznet.co.kr).



4.4. PAL

In EVB B/D, PAL creates Chip Select (Module Enable).

The address map of EVB B/D is same as <Fig 3.1: EVB B/D Memory Map>.

The EVB B/D supports 3 enable signal(Chip Select) as shown in the address map of EVB B/D.

EVB B/D provides VHDL Code. For developer who uses PAL element, CUPL is recommended since it is a freeware PAL Compiler. WINCUPL of ATMEL co. can be used after simple registration.

Use it with "AWINCUPLEXE" that is downloadable from ATMEL Homepage.

Refer to "**AVR Tool Guide.pdf**" for usage.

4.4.1. IO Define

The following is VHDL Source code.

```
entity evb_pal is
  port(
    Addr      : in std_logic_vector(15 downto 10);
    nRD       : in std_logic;
    nWR       : in std_logic;
    nRAMCS    : out std_logic;
    nCS_IINCHIP : out std_logic;
    LCDCS     : out std_logic
  );
```

The following is CUPL Source code.


```

/* ***** INPUT PINS ***** */
PIN [1..6] = [A10..15]; /* address upper 6bits */
PIN 7 = nRD; /* read signal */
PIN 8 = nWR; /* write signal */
/* ***** OUTPUT PINS ***** */
PIN 12 = nCS_RAM; /* External SRAM CS */
PIN 13 = LCD_E; /* LCD CS */
PIN 14 = nCS_IINCHIP; /* iinChip CS */
    
```

4.4.2. External SRAM Area

External SRAM area is ranged from 0x0000 to 0x7fff.

The following is a VHDL Source Code that makes SRAM CS.

```

--nRAMCS (0x0000 - 0x7fff) :
process(Addr)
begin
    if (Addr < "100000") then
        nRAMCS <= '0';
    else
        nRAMCS <= '1';
    end if;
end process;
    
```

The following is a CUPL Source Code that makes SRAM CS.

```

/* < 0x8000 */
!nCS_RAM = !A15;
    
```

4.4.3. LCD Area

LCD is ranged 0x9000 ~ 0x9400.

WR and RD Signal are used together to control the timing.

```
--LCDCS (0x9000 - 0x93ff)
process(Addr, nRD, nWR)
begin
  if (((Addr >= "100100") and (Addr < "100101")) and (nRD = '0' or nWR = '0')) then
    LCDCS <= '1';
  else
    LCDCS <= '0';
  end if;
end process;
```

```
/* 0x9000 <= < 0x9400 */
LCD_E = (A15 & !A14 & !A13 & A12 & !A11 & !A10) & (!nRD # !nWR);
```

LCD is High Active Enable Signal.

4.4.4. W5100 Area

In case of W5100, the address is divided into 2 parts about same Chip.

For more details, refer to **“W5100 Datasheet”**

```
-- IINCHIP (0x8000 - 0x8800, 0xC000 - 0xFFFF)
process(Addr)
begin
  if (((Addr >= "100000") and (Addr < "100010")) or (Addr >= "110000")) then
    nCS_IINCHIP <= '0';
  else
    nCS_IINCHIP <= '1';
  end if;
end process;
```

```
/* 0x8000 <= < 0x8800 OR > 0xC000 */
!nCS_IINCHIP = (A15 & !A14 & !A13 & !A12 & !A11) # (A15 & A14);
```

For VHDL Source Code, refer to “**EVb_PAL.VHD**” in the official website of WIZnet (www.wiznet.co.kr).

For CUPL Source Code, refer to “**EVb_PAL.PLD**” in the official website of WIZnet (www.wiznet.co.kr).

Please refer to “AVR Tool Guide.pdf” for compiling.

4.5. Parts List

4.5.1. W5100E01-AVR Parts List

Please refer to "**W5100E01-AVR_PARTLIST.PDF**" in the official website of WIZnet (www.wiznet.co.kr).

4.5.2. PM-A1 Parts List

Please refer to "**PM-A1_PARTLIST.PDF**" in the official website of WIZnet (www.wiznet.co.kr).



4.6. Physical Specification

4.6.1. Power Consumption

Power consumption of each component of EVB B/D is as in the following table.

< Table 4-5 EVB B/D Power Consumption >

| Power Level | MIN | TYP | MAX | UNIT |
|-------------|-----|-----|-----|------|
| 5V | - | 243 | - | mA |
| 3.3V | - | 198 | - | mA |

Total Power consumption is $243\text{mA} \times 5\text{V} = 1215\text{mW}$.