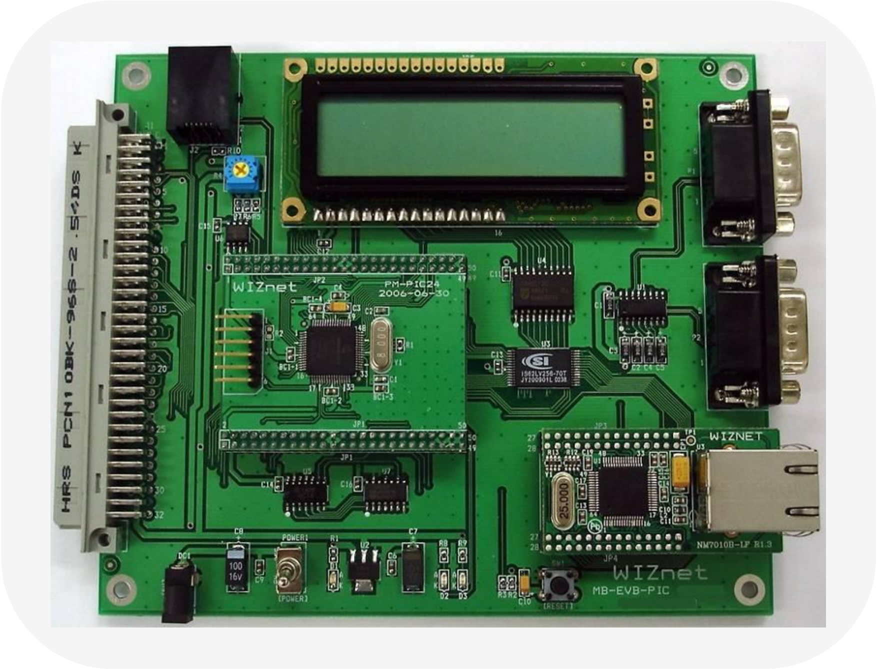
EVB-PIC24 User's Manual

(Version 1.0.0)



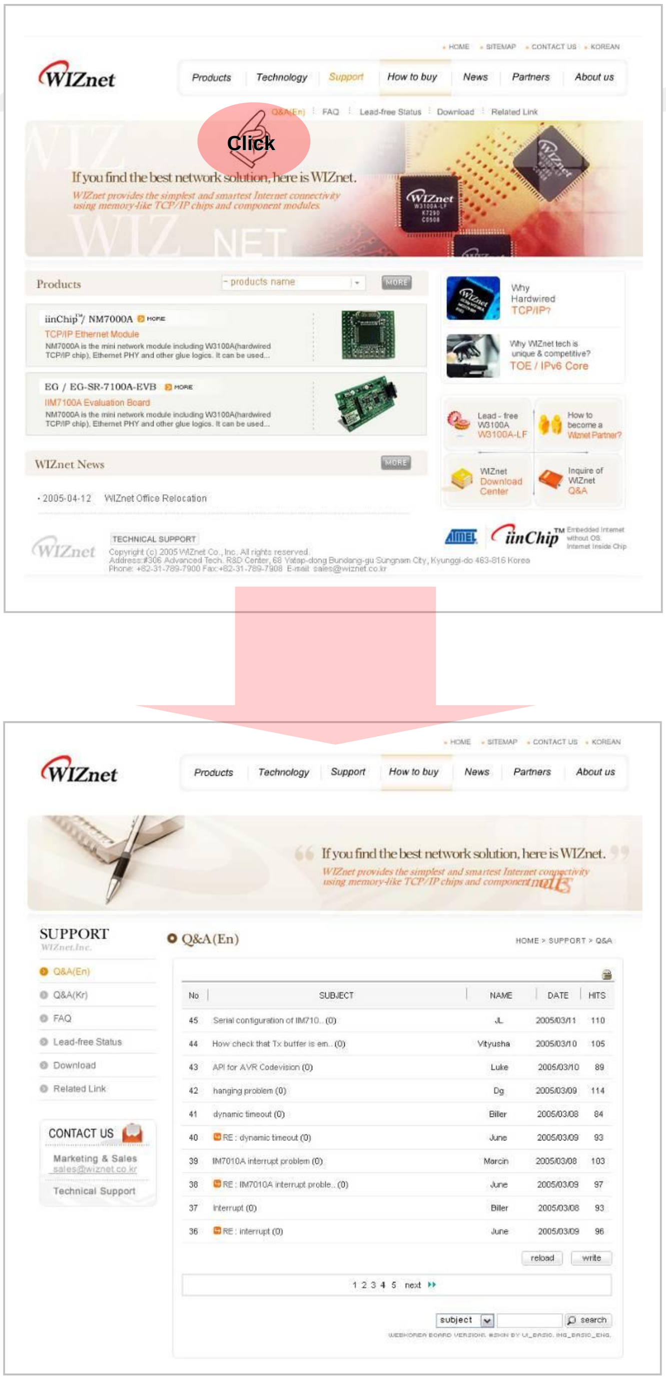
☞ For more information, visit our website at http://www.wiznet.co.kr

Document History Information

Revision	Data	Description
Ver. 1.0	December 7, 2006	Original Document

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.



If you find the best network solution, here is WIZnet.

WIZnet provides the simplest and smartest Internet connectivity using memory-like TCP/IP chips and component modules.

iinChip™/ NM7000A

TCP/IP Ethernet Module

NM7000A is the mini network module including W3100A(hardwired TCP/IP chip), Ethernet PHY and other glue logics. It can be used...

EG / EG-SR-7100A-EVB

IIM7100A Evaluation Board

NM7000A is the mini network module including W3100A(hardwired TCP/IP chip), Ethernet PHY and other glue logics. It can be used...

· 2005-04-12 WIZnet Office Relocation

Copyright (c) 2005 WIZnet Co., Inc. All rights reserved.
Address:#306 Advanced Tech. R&D Center, 68 Yatap-dong Bundang-gu Sungnam City, Kyunggi-do 463-816 Korea
Phone: +82-31-789-7900 Fax:+82-31-789-7908 E-mail: sales@wiznet.co.kr

If you find the best network solution, here is WIZnet.

WIZnet provides the simplest and smartest Internet connectivity using memory-like TCP/IP chips and component modules

Q&A(En)

No	SUBJECT	NAME	DATE	HITS
45	Serial configuration of IIM710.. (0)	JL	2005/03/11	110
44	How check that Tx buffer is em.. (0)	Vityusha	2005/03/10	105
43	API for AVR Codevision (0)	Luke	2005/03/10	89
42	hanging problem (0)	Dg	2005/03/09	114
41	dynamic timeout (0)	Biller	2005/03/08	84
40	RE : dynamic timeout (0)	June	2005/03/09	93
39	IIM7010A interrupt problem (0)	Marcin	2005/03/08	103
38	RE : IIM7010A interrupt proble.. (0)	June	2005/03/09	97
37	Interrupt (0)	Biller	2005/03/08	93
36	RE : interrupt (0)	June	2005/03/09	96

COPYRIGHT NOTICE

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

Figures

Tables

1. Overview

EVB-PIC24 (referred to as "EVB B/D or EVB" from here on) is iinChip™ Evaluation B/D for PIC developer.

1.1. Package

When purchasing EVB 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	MB-EVB-PIC EVB Main Board	1
	PM-PIC24 MCU Module (Plugged In MB-EVB-PIC)	1
	NM7010B+ (Plugged In MB-EVB-PIC)	1
	Power Adaptor (5V)	1
Accessory	MICROCHIP ICD2 Tool	**Option**
	Software CD	1
	UTP Cable	1
	Serial Cable	1

<Table 1-2 : Contents of Software CD>

Directory			Contents
EVB-PIC24	DOC	Datasheet	All sorts of Datasheet
		Development	ASM, C, Library Manual
		Manual	Users Manual
	HW	Schematics	All sorts of schematics
		Part List	All sorts of Part List
	SW	Firmware	EVB B/D Firmware
		PC Utility	All sorts of Tool Program
NM7010B+			
W3150A+			

- The contents of Software CD could be changed by version. Please check "ReadMe.txt" of CD.

1.2. Feature

1.2.1. H/W Feature

EVB B/D consists of 3 kinds B/D

- ● PM-PIC24
  - MCU : PIC24FJ128GA006, 16MHz
  - RAM : 8KB SRAM (Internal), 32KB SRAM(External)
  - ROM : 128KB Flash (PIC24 Internal Flash)
  - JTAG, ICSP, ICE
- ● MB-EVB-PIC
  - Power : 5V Adaptor
  - UART : RS-232 Serial Port, (1 Port, default 57600 Baud Rate)
  - LCD Display : 16x2 Charactor LCD
- ● Network Module (NM7010B+)
  - iinChip™ : Hardwired TCP/IP Chip W3150A+
  - PHY : RTL8201CP(RealTek), 10/100 BaseT(X) Auto Negotiation
  - MagJack : RD1-125BAG1A (UDE) , Integrated Transformer(1:1)
    Link & ACT LEDs

1.2.2. F/W Feature

The F/W Feature 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 : iinChip™ 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. PC Programs Install

2.1.1. Development Program Install

Please refer to Microchip(www.microchip.com) for more information.

2.1.1.1. MPLAB IDE(Integrated Development Environment)

It included following features – assembler, linker, simulator(with peripheral), VDI(Visual Device Initializer) and full featured debugger.

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

Firmware of EVB B/D is currently using MPLAB IDE Version 7.40 and can be changed with newer IDE.

2.1.1.2. MPLAB C30

It is a full-featured ANSI compliant C compiler for the Microchip 16-bit devices. Also It is fully compatible with MPLAB IDE.

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

Firmware of EVB B/D is currently using MPLAB C30 Version 2.02 and can be changed with newer compiler.

2.1.1.3. 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.

2.1.2. EVB B/D Test PC Program Install

2.1.2.1. Loopback Test Program (AX1) Install

Loopback Test Program (referred to as "AX1" from here on) is a program to evaluate the performance of iinChip™ and does the Loopback on 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.

2.2. Quick Start

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

① Connect test PC to EVB B/D using UTP cross 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 2.1.2.1

④ After the execution of serial terminal program(like Hyperterminal), set up the properties as the following.

<Table 2-1 : Terminal Properties Setting>

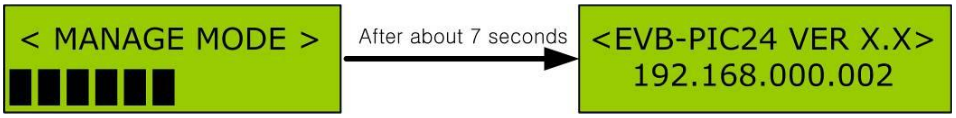
Properties	**Setting Value**
Bits Per second(Baud Rate)	57600 bps
Data Bits	8 Bits
Stop Bits	1 Bit
Parity	No
Flow Control	None

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

⑤ Turn on the power switch of EVB B/D

Following items should be checked upon power on

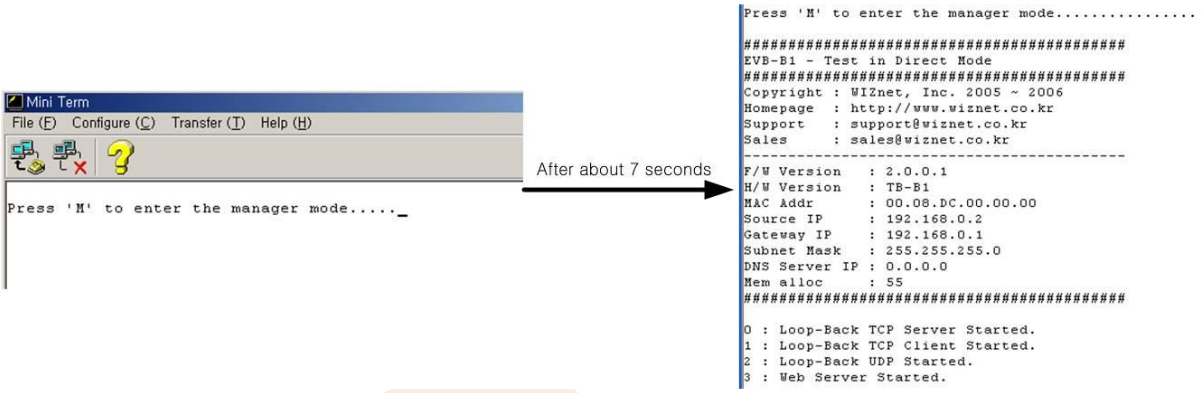
- Check lighting on power LED(D1) of EVB B/D when powering on
- Check if LEDs of D2 and D3 blink three times by turns.
- Check if Text LCD display of EVB B/D outputs in the way shown in <Fig 2.1> and shown in <Fig 2.2> on the Terminal Program



```
< MANAGE MODE >
██████
```

```
<EVB-PIC24 VER X.X>
  192.168.000.002
```

<Fig 2.1 : EVB B/D Text LCD Display>

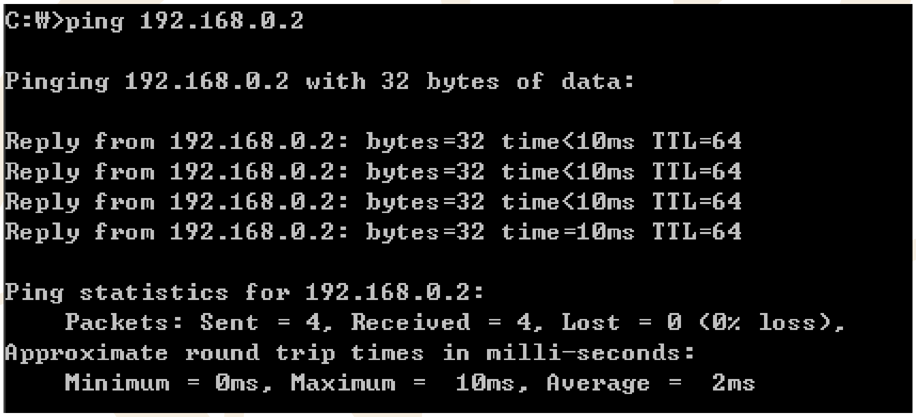


```
Press 'M' to enter the manager mode....._
```

```
Press 'M' to enter the manager mode................

###########################################
EVB-B1 - Test in Direct Mode
###########################################
Copyright : WIZnet, Inc. 2005 ~ 2006
Homepage  : http://www.wiznet.co.kr
Support   : support@wiznet.co.kr
Sales     : sales@wiznet.co.kr
-------------------------------------------
F/W Version    : 2.0.0.1
H/W Version    : TB-B1
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 Client Started.
2 : Loop-Back UDP Started.
3 : Web Server Started.
```

<Fig 2.2: 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 2.3: EVB B/D Ping Reply Test>

⑦ Execute "AX1" program. Refer to "**AX1 Manaul Vx.x.pdf**"

⑧ Test the operation of "AX1" program with TCP Client. Refer to "**AX1 Manaul 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],[∞] Icon.

⑨ Loopback any file or packet between "AX1" Program and EVB B/D.

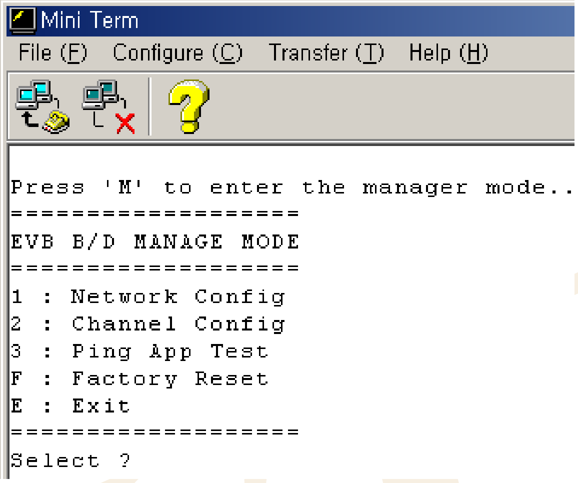
2.3. EVB B/D Test

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 iinChip™ Test.

2.3.1. 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 iinChip™ to be tested. And perform certain ping request test with DNS server.



```
Press 'M' to enter the manager mode..
===================
EVB B/D MANAGE MODE
===================
1 : Network Config
2 : Channel Config
3 : Ping App Test
F : Factory Reset
E : Exit
===================
Select ?
```

<Fig 2.4:Manage Program Execution >

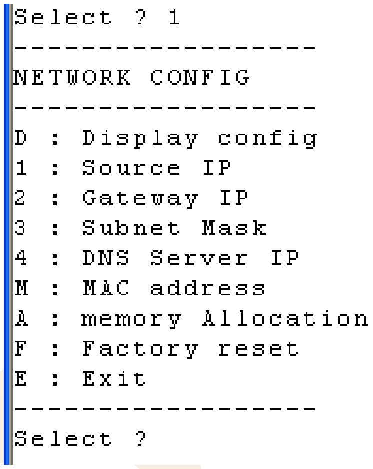
2.3.1.1. Network Configuration

It selects Network Information that is used in EVB B/D. When choosing '1' at terminal Program of <Fig 2.7>, 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.00.00.24
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 2.8> can be displayed and each function is described in <Table 2-3>.



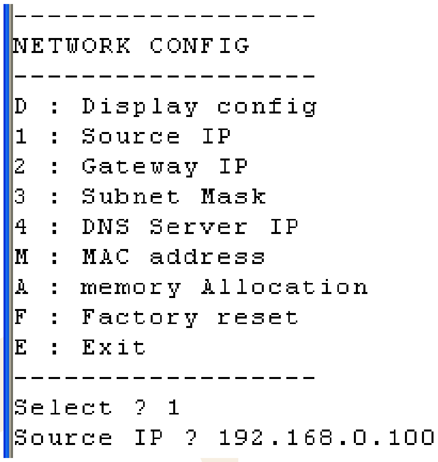
```
Select ? 1
------------------
NETWORK CONFIG
------------------
D : Display config
1 : Source IP
2 : Gateway IP
3 : Subnet Mask
4 : DNS Server IP
M : MAC address
A : memory Allocation
F : Factory reset
E : Exit
------------------
Select ?
```

<Fig 2.5: 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 and so it must be set up as Static IP Address.
‘A’ or ‘a’	Sets up Memory Allocation. iinChip™ Memory Size Register.(RMSR,TMSR) Refer to “W3150A$^+$ 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 change when Factory Reset.
E : Exit	Exit “Net Config”

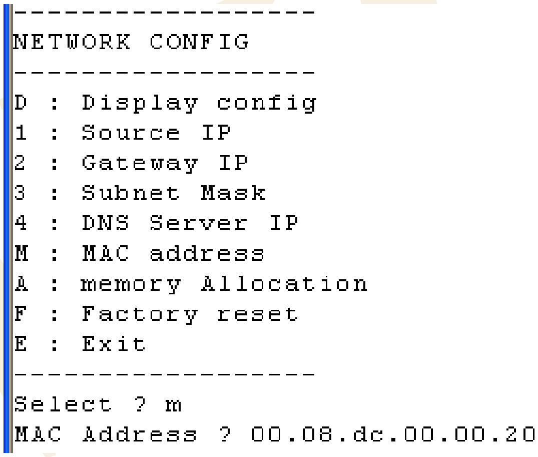
<Fig 2.9> is an example of setting the Source IP of EVB B/D in Network Config



```
-------------------
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 2.6: Source IP Address Setup Example>

<Fig 2.10> 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 2.7: MAC address Setup Example>

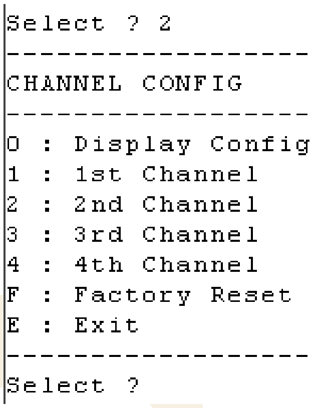
2.3.1.2. Channel Config

It set up Test Application that can be operated in iinChip™ 4 channel of EVB B/D. If '2' is selected, each channel can be set up. The default iinChip™ Channel Information is shown in <Table 2-4>.

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

iinChip™ Channel	Test Application
1st	Loopback TCP Server (Port 5000)
2nd	Loopback TCP Server (Port 5000)
3rd	Loopback TCP Server (Port 5000)
4th	Loopback TCP Server (Port 5000)

If "Channel Config" menu is selected in manage program, <Fig 2.11> is displayed and the functionality of each menu is same as <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 2.8: Menu of Channel Config>

<Table 2-5 : Menu of Channel Config>

Menu	Description
D : Display Config	Output of current Set up Test Application Type by each iinChip™ Channel
0 : 1st Channel	Test Application Type Setup at iinChip™ No. "0" Channel <Warning> As developing EVB B/D, DHCP Client Application Setup is possible at only No. "0" Channel.
1 : 2nd Channel	Test Application Type Setup at iinChip™ No. "1" Channel
2 : 3rd Channel	Test Application Type Setup at iinChip™ No. "2" Channel
3 : 4th Channel	Test Application Type Setup at iinChip™ No. "3" Channel
F : Factory Reset	Initialization into original Setup status. Refer to <Table 2-4>
E : Exit	Exit "Channel Config"

Available Setup Test Application by each iinChip™ Channel is shown as <Table 2-6>

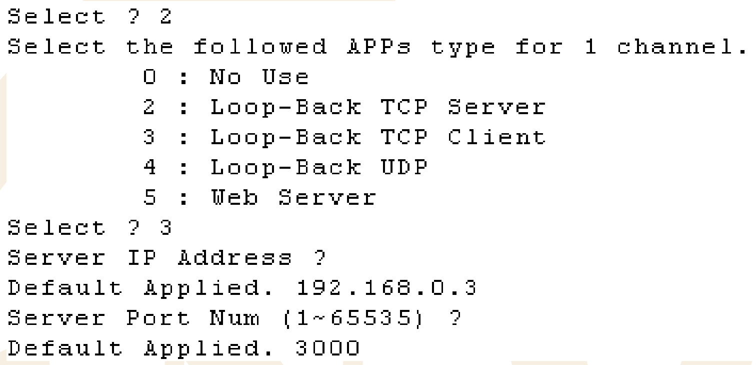
<Table 2-6 : iinChip™ Channel Application Type>

Application Type	Description
No Use	Not used
DHCP Client	Receiving Network Information of EVB B/D from DHCP Server dynamically <Warning> If DHCP Server is not existing 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 2.12> shows an example of setting the 2nd Channel of iinChip™ 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
```

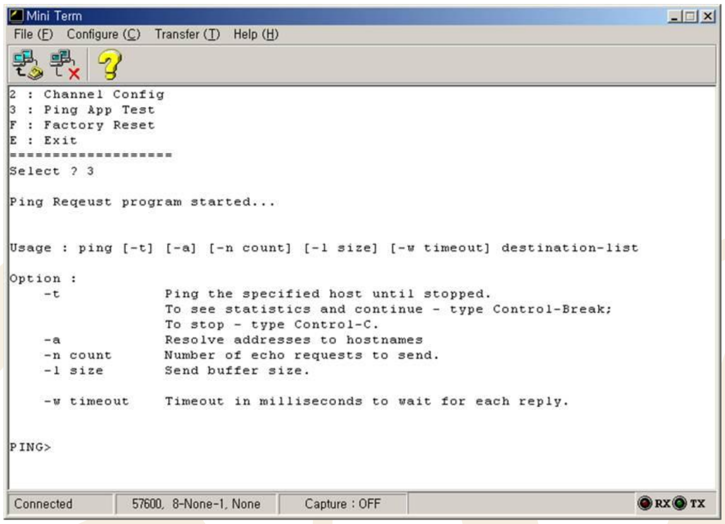
<Fig 2.9: Loopback TCP Client Application Setting Example>

< 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

2.3.1.3. Ping Application Test

Ping Application Test is a program created for IP RAW Channel Evaluation of iinChip™ 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 2.4:Manage Program Execution >

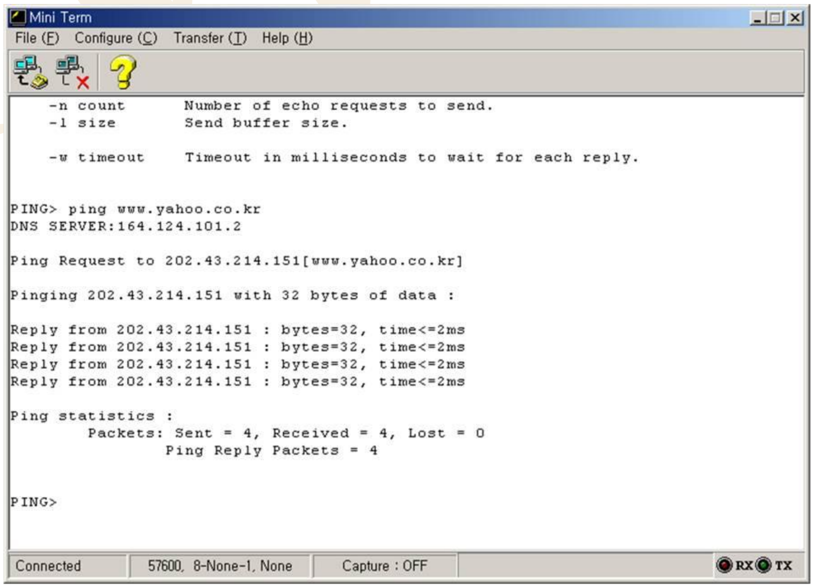


```
2 : Channel Config
3 : Ping App Test
F : Factory Reset
E : Exit
===================
Select ? 3

Ping Reqeust program started...


Usage : ping [-t] [-a] [-n count] [-l size] [-w timeout] destination-list

Option :
    -t              Ping the specified host until stopped.
                    To see statistics and continue - type Control-Break;
                    To stop - type Control-C.
    -a              Resolve addresses to hostnames
    -n count        Number of echo requests to send.
    -l size         Send buffer size.

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


PING>
```

<Fig 2.10: Usage of Ping Application >

<Fig 2.13> displays the execution screen of Ping Application and shows how to use the Ping Application.

<Fig 2.14> shows the real example of sending the Ping Request to the destination and receiving the Ping Reply.



```
    -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>
```

<Fig 2.11: Ping Application Test>

To terminate the Ping Application type, "exit" in "PING>" prompt.

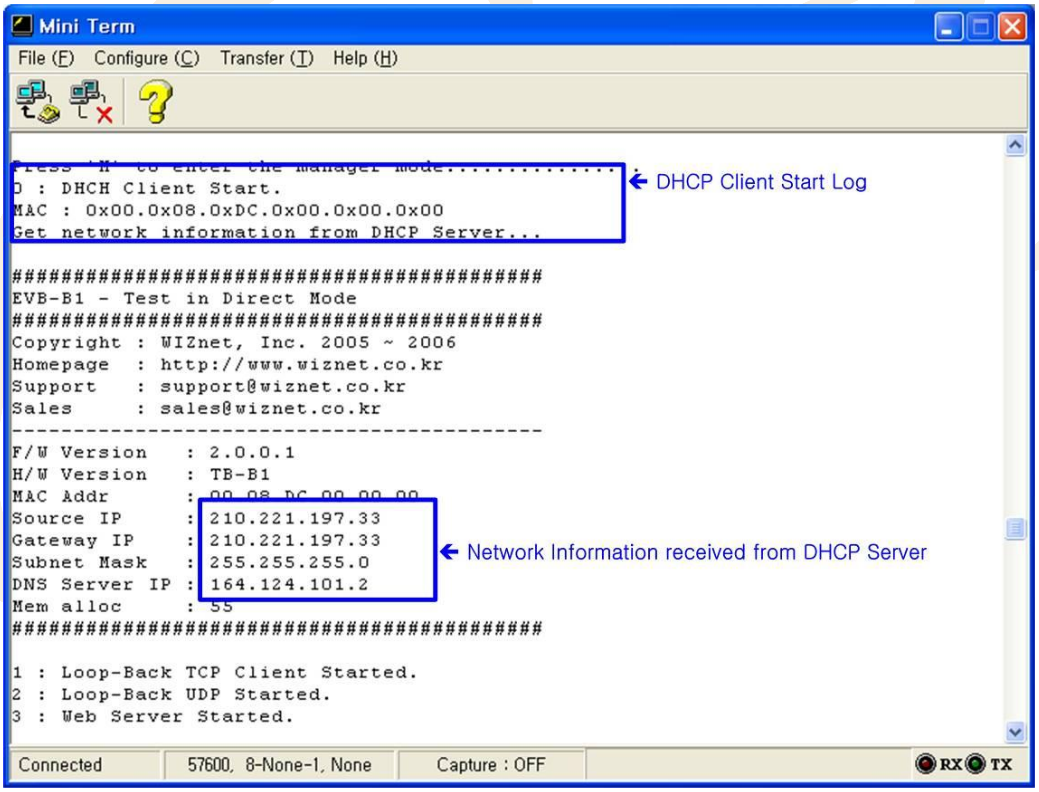
2.3.2. EVB B/D Test Applications

2.3.2.1. 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, iinChip™ 1st channel application type must set up as "DHCP Client" using [Manager>>Channel Config>>0th Channel] Menu.

Refer to Chapter 2.4.1.2

<Fig 2.15> 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)
DHCP Client Start Log
Network Information received from DHCP Server
Connected
57600, 8-None-1, None
Capture : OFF
RX
TX

```
Press 'M' to enter the manager mode...............
0 : DHCH Client Start.
MAC : 0x00.0x08.0xDC.0x00.0x00.0x00
Get network information from DHCP Server...

############################################
EVB-B1 - Test in Direct Mode
############################################
Copyright : WIZnet, Inc. 2005 ~ 2006
Homepage  : http://www.wiznet.co.kr
Support   : support@wiznet.co.kr
Sales     : sales@wiznet.co.kr
--------------------------------------------
F/W Version   : 2.0.0.1
H/W Version   : TB-B1
MAC Addr      : 00.08.DC.00.00.00
Source IP     : 210.221.197.33
Gateway IP    : 210.221.197.33
Subnet Mask   : 255.255.255.0
DNS Server IP : 164.124.101.2
Mem alloc     : 55
############################################

1 : Loop-Back TCP Client Started.
2 : Loop-Back UDP Started.
3 : Web Server Started.
```

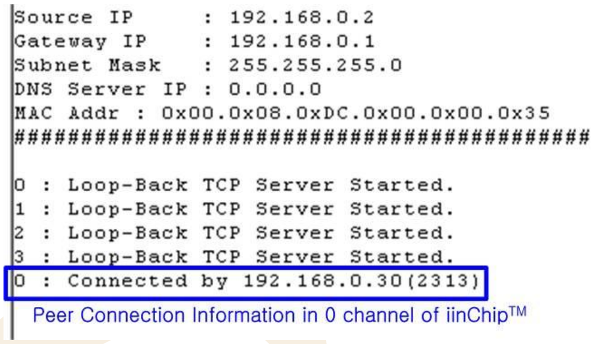
<Fig 2.12: DHCP Client Test>

2.3.2.2. 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 Value 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 iinChip™

<Fig 2.13: Loopback TCP Server Test>

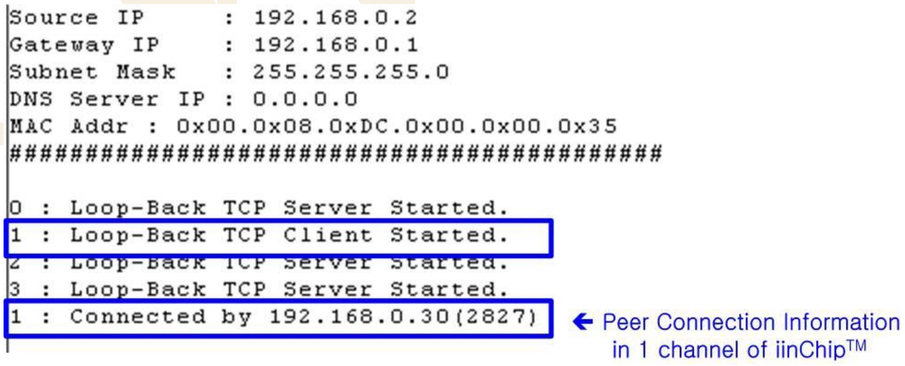
2.3.2.3. Loopback TCP Client

Loopback TCP Client Application is an application that loop backs 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 iinChip™ 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 iinChip™

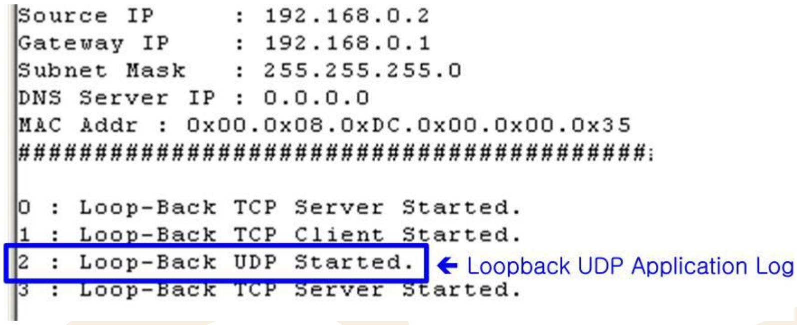
<Fig 2.14: Loopback TCP Client>

2.3.2.4. 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 iinChip™ 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.
3 : Loop-Back TCP Server Started.
```

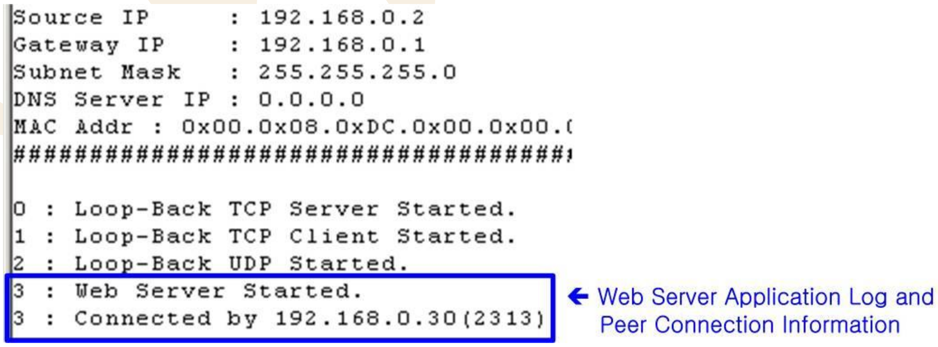
← Loopback UDP Application Log

<Fig 2.15: Loopback UDP Test>

2.3.2.5. Web Server

Web Server Application sends and receives web pages and EVB B/D control data etc. through HTTP Channel connected with web browser. For Web Server testing, set up any channel of iinChip™ 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.

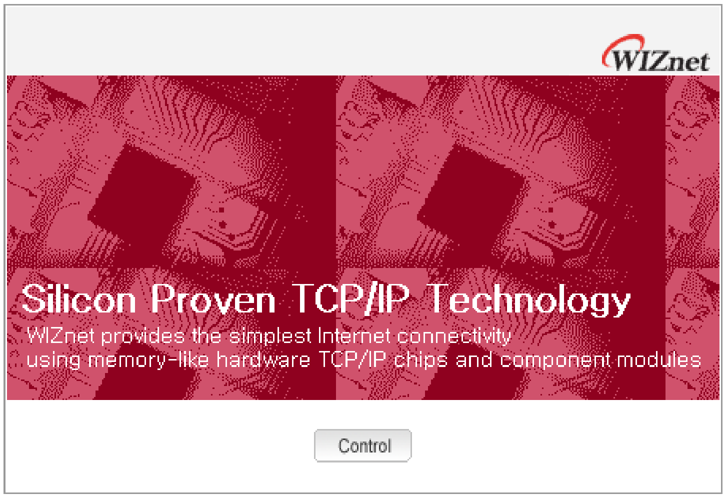


```
Source IP      : 192.168.0.2
Gateway IP     : 192.168.0.1
Subnet Mask    : 255.255.255.0
DNS Server IP  : 0.0.0.0
MAC Addr : 0x00.0x08.0xDC.0x00.0x00.(
####################################

0 : Loop-Back TCP Server Started.
1 : Loop-Back TCP Client Started.
2 : Loop-Back UDP Started.
3 : Web Server Started.
3 : Connected by 192.168.0.30(2313)
```

← Web Server Application Log and Peer Connection Information

<Fig 2.16: Web Server Test>

If the web browser successfully connected to HTTP port of EVB B/D, the Web Page of <Fig 2.20> can be viewed. In case Web Page of <Fig 2.20> is not shown, refresh the screen using the “Refresh” function of the web browser.

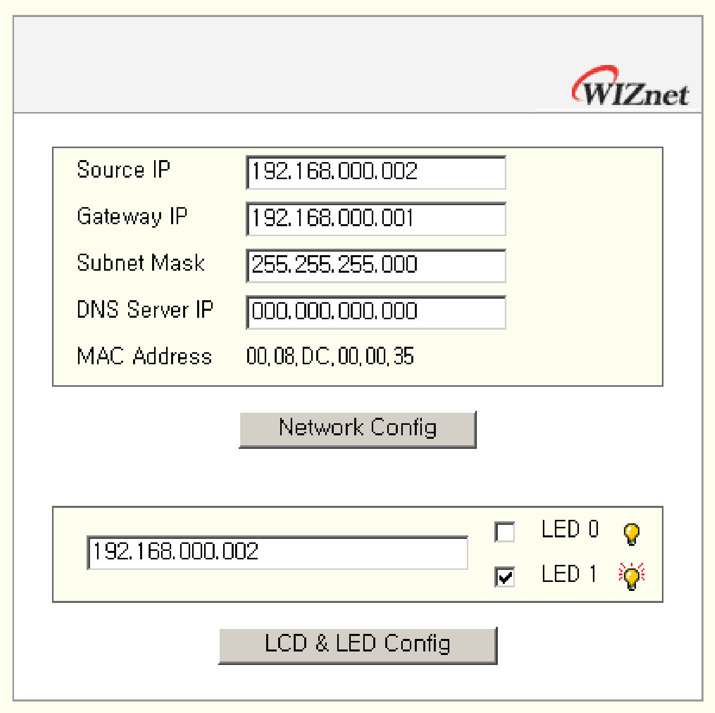


Silicon Proven TCP/IP Technology

WIZnet provides the simplest Internet connectivity using memory-like hardware TCP/IP chips and component modules

<Fig 2.17: Default Web Page of EVB B/D>

If [Control] button on the Web Page in <Fig 2.20> is clicked, it can set the network information or show the web page that can turn on/off LEDs(D1,D2) and display rows of text on Text LCD display.



Source IP 192.168.000.002

Gateway IP 192.168.000.001

Subnet Mask 255.255.255.000

DNS Server IP 000.000.000.000

MAC Address 00,08,DC,00,00,35

192.168.000.002

□ LED 0

☑ LED 1

<Fig 2.18: Web Page of EVB B/D Control>

2.4. Troubleshooting Guide

2.4.1. Ping

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

Step 1. Did you connect correctly between test PC and EVB B/D with UTP cable?

Step 2. Did you change your test PC's network environment (IP address, Gateway, Subnet)? If no, you should change it first as follows:

IP address: 192.168.0.3

Gateway address: 192.168.0.1

Subnet Mask: 255.255.255.0

Step 3. Whether NM7010B^{+}'s Link LED(left LED from rear view) is on?

If it is off, you'd better check whether the UTP cable works correctly.

2.4.2. 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 if the COM Port numbers of the PC and terminal coincide.

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

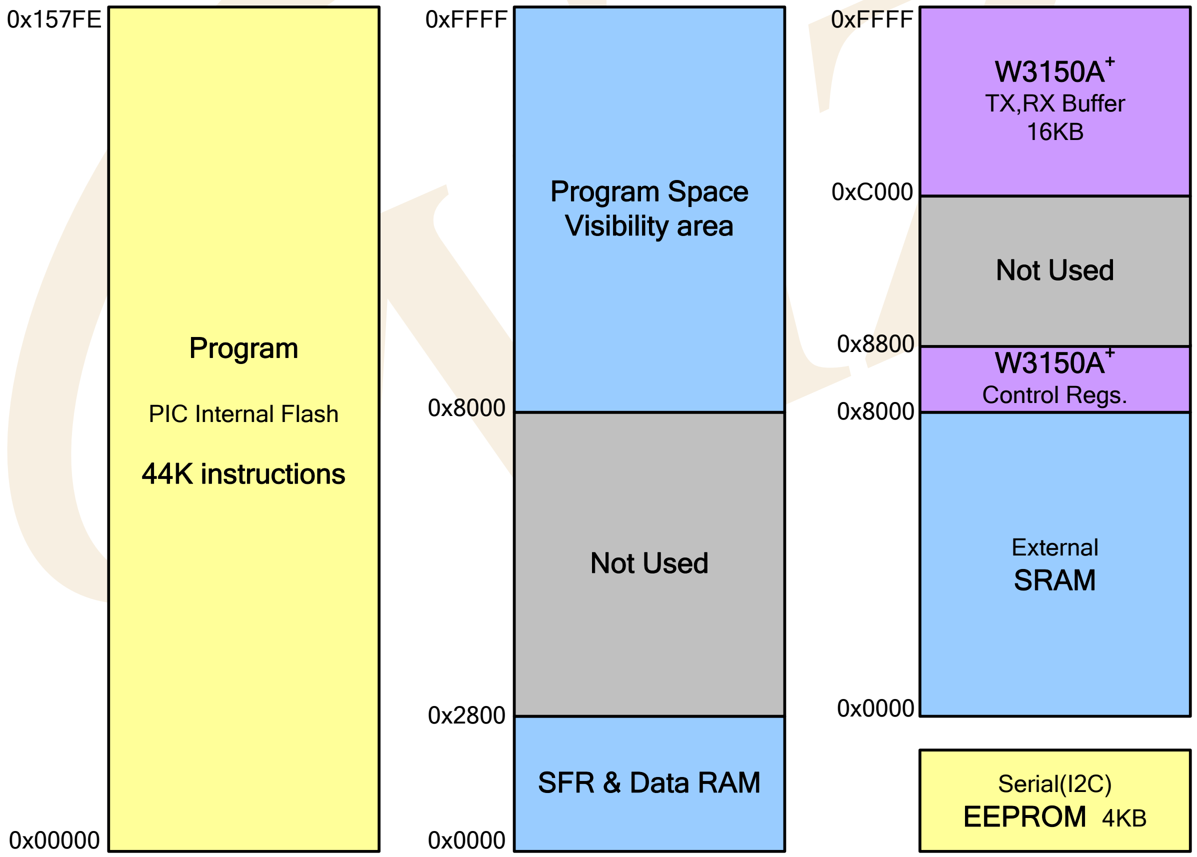
3. Programmer's Guide

3.1. Memory Map

3.1.1. Code & Data Memory Map

Memory Map of EVB-PIC24 has Program Memory 128 Kbytes, SRAM 8Kbytes. External Space(Parallel Master Port) divided into SRAM Area and W3150A+ Area. Other than these, there is 4Kbytes Serial EEPROM. Various kinds 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>

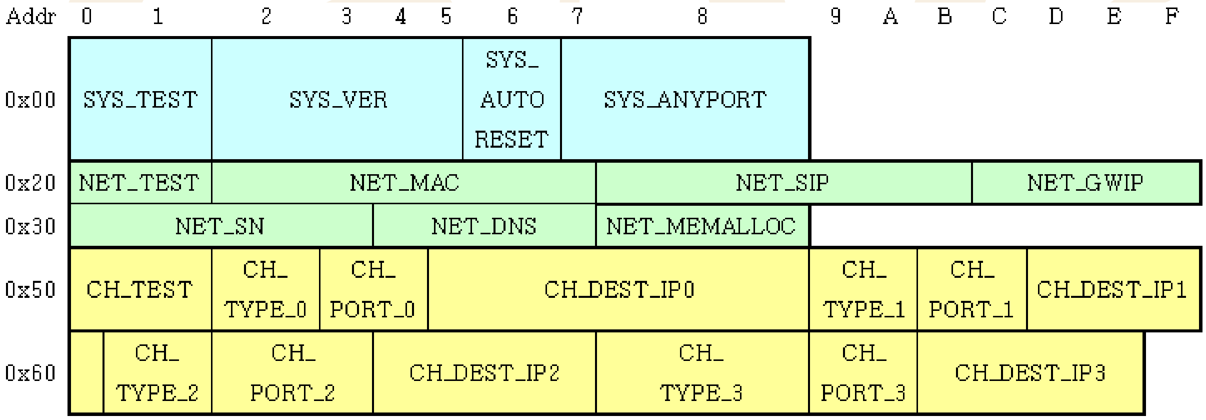
<Table 3-1: Device MAP Definition>

Device	Map Define	Source Code
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) #define __DEF_IINCHIP_MAP_RXBUF__ (COMMON_BASE + 0x6000)	iinChip/iinchip_conf.h

3.1.2. SERIAL EEPROM MAP

<Fig 3.2>, <Table 3.2> are representations of Serial EEPROM의 Map.

Refer to “evb/config.h” and “evb/config.c.”



<Fig 3.2: Serial EEPROM Map>

<Table 3-2: SERIAL 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	System Information's Valid Check	0xA5A5 – Valid Others – Invalid
SYS_VER	F/W Version	0xAABBCCDD (AA.BB.CC.DD)
SYS_AUTORESET	If you set any environmental variable up, Auto Reset Check	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

typedef struct _SYSINFO
{
    u_int   test;
    u_long   ver;
    u_char   auto_reset;
    u_int   any_port;
}**SYSINFO**;
	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	Network Information Valid Check	0xA5A5 – Valid Others – Invalid
NET_SIP	Source IP Address	0xC0 A80002 (192.168.0.2)
NET_GWIP	Gateway IP Address	0xC0 A80001 (192.168.0.1)
NET_SN	Subnet Mask	0xFFFFFF00 (255.255.255.0)
NET_DNS	DNS Server IP Address	0x00000000 (0.0.0.0)
NET_MEMALLOC	iinChip™ Memory Allocation	0x55

Network Information is accessed as NETCONF Data Type.

<Table 3-7: NETCONF Data Type Definition>

Type Definition	Global Instance

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**;
	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 about the 4 Channel Application of iinChip™.

<Table 3-9: Channel Information>

Name	Description	Default Value
CH_TEST	Channel Information의 Valid Check	0xA5A5 – Valid Others – Invalid
CH_TYPE_X	No. "X" Channel's Application Type	Default - LB_TCPS 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
CH_PORT_X	No. "X" Channel's Source/Destination Port Number	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	No. "X" Channel's Destination IP Address	0xC0 A80003 (192.168.0.3)

Channel Information is used for recording 4 Channel Application Types of iinChip™.

Channel Application Type includes Loopback TCP Server, Loopback TCP Client, Loopback UDP,DHCP Client, Web Server, and 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

typedef struct _CHCONF
{
u_int test;
struct _CH_CONF
{
u_char type;
u_int port;
u_long destip;
}ch[4];
}**CHCONF**;
	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. First, Manage Program that sets up various environments for running EVB B/D. And second, Loopback Programs that tests iinChip™ performance and there are Internet Application using Internet Protocols such as DHCP, HTTP, DNS, and ICMP.
Let's look at the sources 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	PIC UART control
	timer.h, timer.c	PIC Timer enable & disable
	types.h	PIC Data Type 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(D1,D2) control
	manage.h, manage.c	Manage App
inet	dhcp.h dchp.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
	sockutil.h, sockutil.c	Utilities relating Socket
	util.h, util.c	Utilities
iinChip	iinchip_conf.h	System Dependant Defintion of iinChip™
	w3150a.h, w3150a.c	I/O Functions of iinchip™
	socket.h, socket.c	Socket APIs for iinChip™

3.2.2. How to Compile

Sources of Chapter 3.2.1 compile in bundle after arranging SRC items.

Using MPLAB IDE, open the "EVBs" project. And, select "Make" or "Build All".

After compiling, the name that is assigned at Makefile TARGET and "Evbs.hex," the file that is assigned FORMAT are created. This can be loaded in PIC24.

< Table 3-14 : iinChip™'s DEFINE Option >

#define LITTLE_ENDIAN	
#define __DEF_IINCHIP_DIRECT_MODE__	1
#define __DEF_IINCHIP_INDIRECT_MODE__	2
#define __DEF_IINCHIP_BUS__	**__DEF_IINCHIP_DIRECT_MODE__**
//#define __DEF_IINCHIP_BUS__	__DEF_IINCHIP_INDIRECT_MODE__

Since EVB B/D is Little-Endian system, LITTLE_ENDIAN is defined and used. If the target system is Big-Endian, the defined items should be commented.

If iinchip™ is intended to be used as different mode other than Direct Bus Mode, use desired Bus Mode Define as __DEF_IINCHIP_BUS__ instead of __DEF_IINCHIP_DIRECT_MODE__. If DEFINE OPTION of iinchip™ is changed the sources must Re-Build. To Re-Build project do "make clean" then "make".

3.2.3. How to download

For downloading the hex file, we use MPLAB IDE and MPLAB ICD 2.

1) Connect ICD Cable to J1 at the PM-PIC24 or J2 at the expension board.
2) Supply the power with EVB B/D.
3) Run MPLAB IDE.
4) Choose the "MPLAB ICD 2" in "Select Programmer" menu items.
5) Clicked "Program target device" button.

Please refer to userguide of MPLAB IDE 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 iinChip™ is executed and initialized using previously set network information.

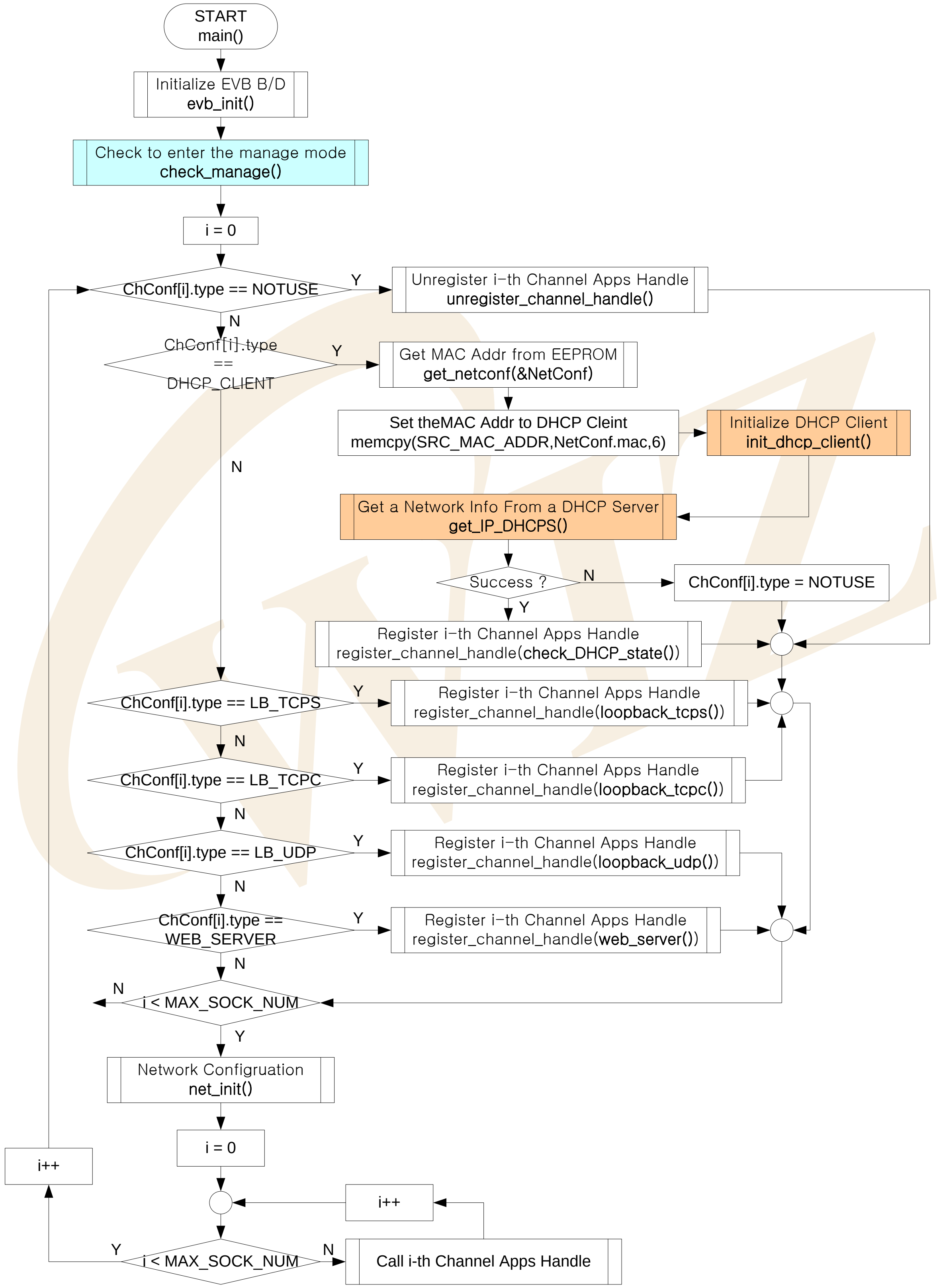
<Fig 3.3> process procedure of EVB B/D main(). Refer to "main/main.c"

If DHCP client exists in the application, the DHCP client obtains the network information from DHCP server by calling 'get_IP_DHCPS()' 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)	PIC, 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_DHCPS(void)	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_tcpc(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

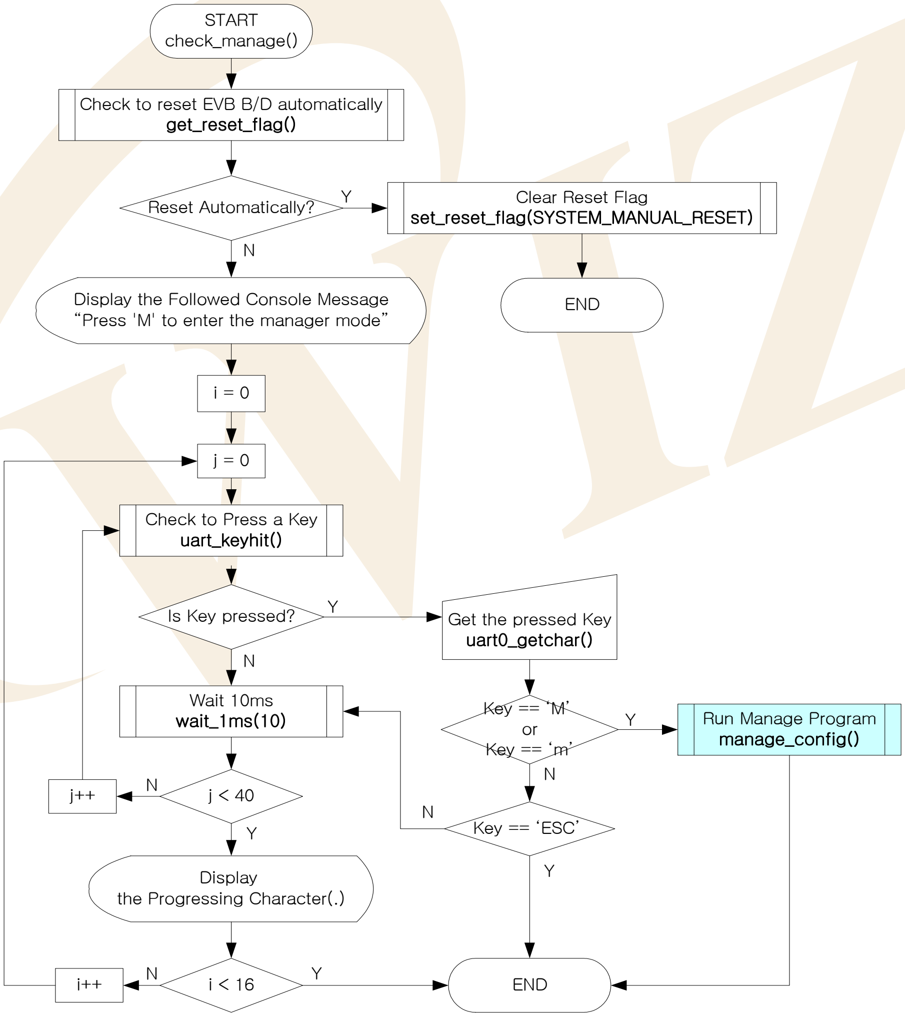

START
main()
Initialize EVB B/D
evb_init()
Check to enter the manage mode
check_manage()
i = 0
ChConf[i].type == NOTUSE
Y
Unregister i-th Channel Apps Handle
unregister_channel_handle()
N
ChConf[i].type
==
DHCP_CLIENT
Y
Get MAC Addr from EEPROM
get_netconf(&NetConf)
Set theMAC Addr to DHCP Cleint
memcpy(SRC_MAC_ADDR,NetConf.mac,6)
Initialize DHCP Client
init_dhcp_client()
N
Get a Network Info From a DHCP Server
get_IP_DHCPS()
Success ?
N
ChConf[i].type = NOTUSE
Y
Register i-th Channel Apps Handle
register_channel_handle(check_DHCP_state())
ChConf[i].type == LB_TCPS
Y
Register i-th Channel Apps Handle
register_channel_handle(loopback_tcps())
N
ChConf[i].type == LB_TCPC
Y
Register i-th Channel Apps Handle
register_channel_handle(loopback_tcpc())
N
ChConf[i].type == LB_UDP
Y
Register i-th Channel Apps Handle
register_channel_handle(loopback_udp())
N
ChConf[i].type ==
WEB_SERVER
Y
Register i-th Channel Apps Handle
register_channel_handle(web_server())
N
N
i < MAX_SOCK_NUM
Y
Network Configruation
net_init()
i = 0
i++
i++
Y
i < MAX_SOCK_NUM
N
Call i-th Channel Apps Handle

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

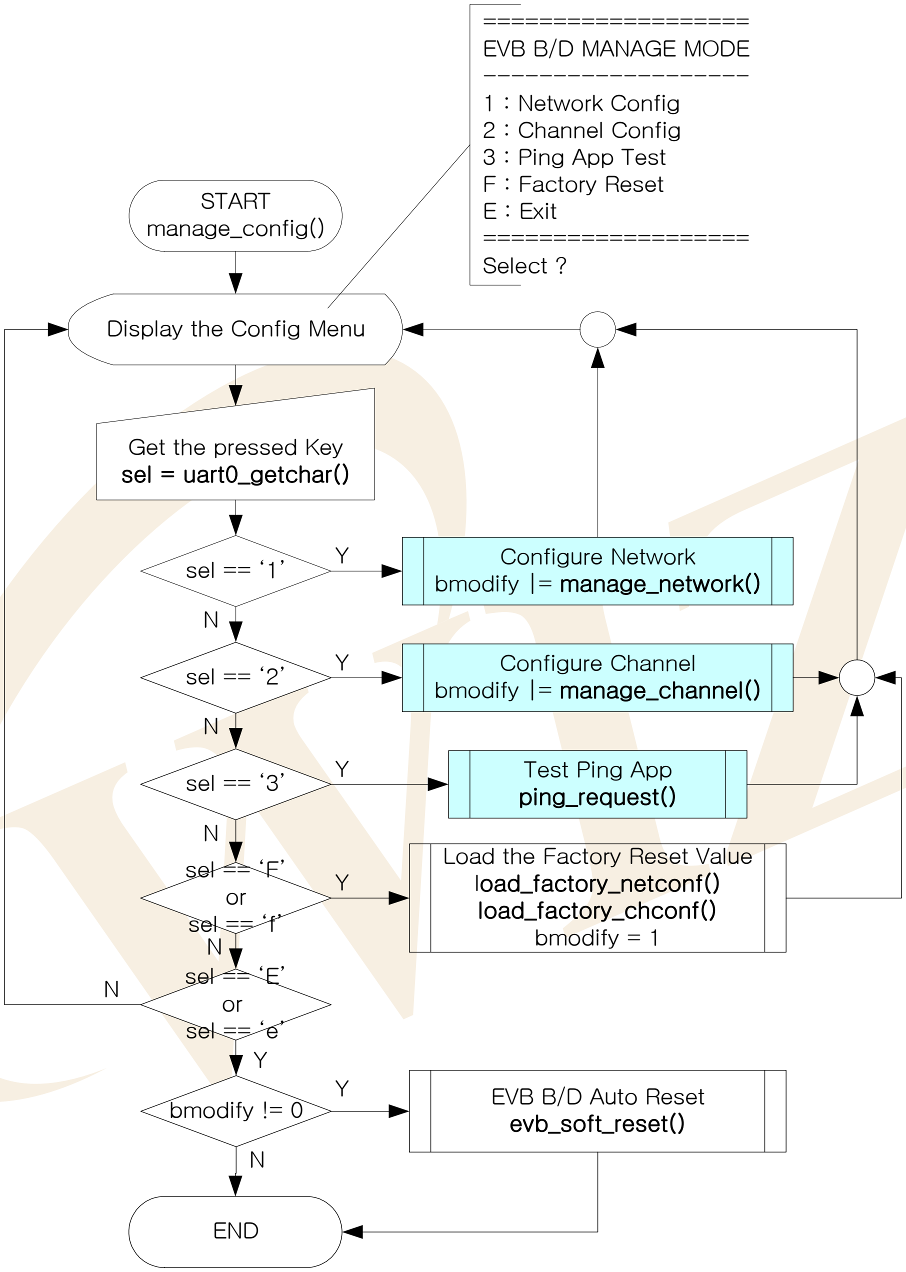
3.2.5. Manage Program

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

Running the manage program can be started by call check_manage() from main() function. check_manage() checks if there is any entering command to Manage Program from RS232 terminal, that if character 'M' or 'm' is input or not. And if there is the command, it will enter to Manage Program through manage_config(). If the user change the configuration, the EVB B/D automatically reboot and check_manage() is skipped.



<Fig 3.4: check_manage()>



```
==================
EVB B/D MANAGE MODE
------------------
1 : Network Config
2 : Channel Config
3 : Ping App Test
F : Factory Reset
E : Exit
==================
Select ?
```

<Fig 3.5: manage_config()>

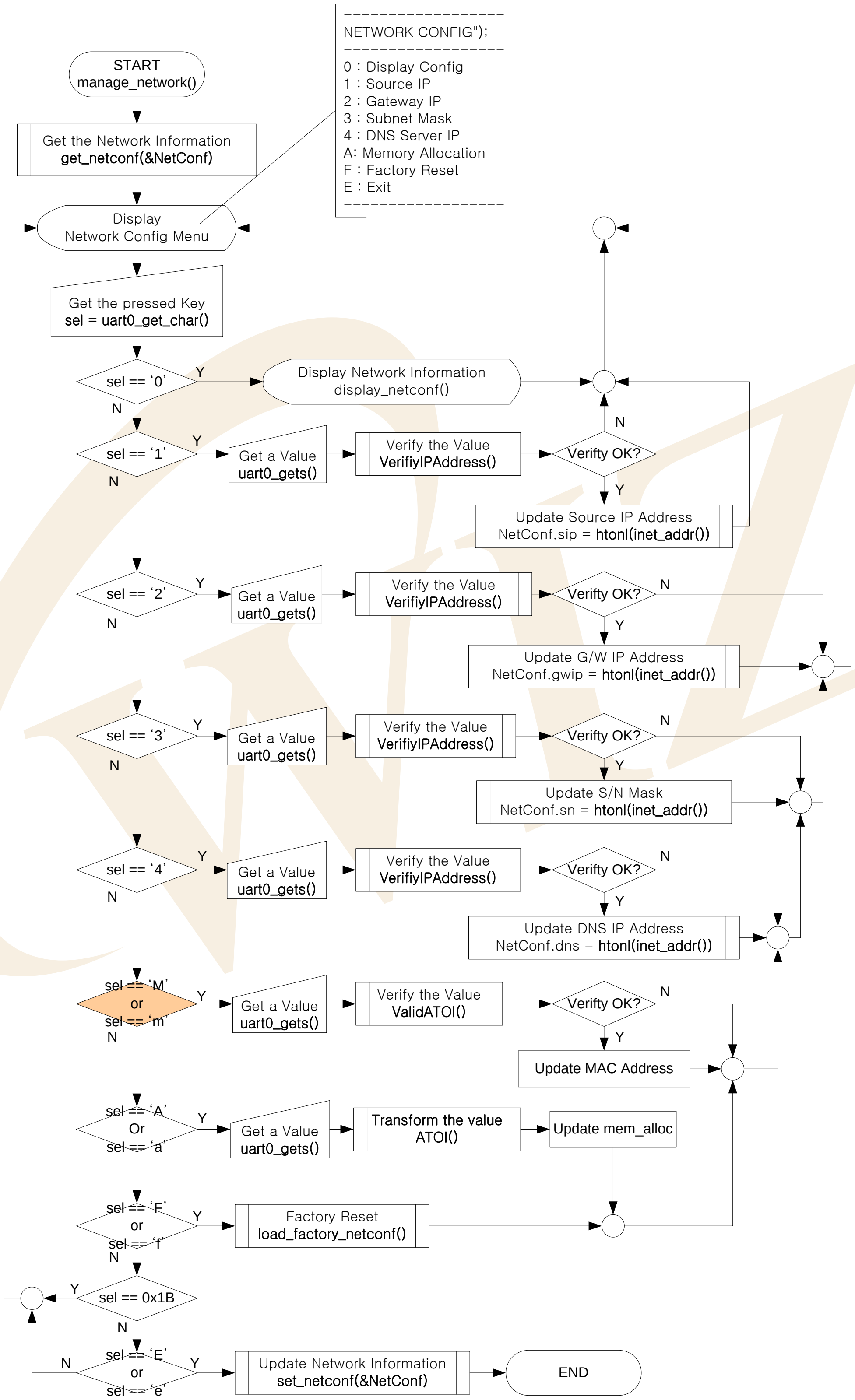
If the EVB B/D is updated, the EVB B/D automatically reboot to apply the updated configuration.

<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 uart0_putchar(char c)	Write one character to 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 change at the time of Factory Reset. MAC Address is updated using 'M' or 'm'.



NETWORK CONFIG");
----
0 : Display Config
1 : Source IP
2 : Gateway IP
3 : Subnet Mask
4 : DNS Server IP
A: Memory Allocation
F : Factory Reset
E : Exit

START
manage_network()
Get the Network Information
get_netconf(&NetConf)
Display
Network Config Menu
Get the pressed Key
sel = uart0_get_char()
sel == '0'
Y
N
Display Network Information
display_netconf()
sel == '1'
Get a Value
uart0_gets()
Verify the Value
VerifiyIPAddress()
Verify OK?
Update Source IP Address
NetConf.sip = htonl(inet_addr())
sel == '2'
Update G/W IP Address
NetConf.gwip = htonl(inet_addr())
sel == '3'
Update S/N Mask
NetConf.sn = htonl(inet_addr())
sel == '4'
Update DNS IP Address
NetConf.dns = htonl(inet_addr())
sel == 'M'
or
sel == 'm'
Verify the Value
ValidATOI()
Update MAC Address
sel == 'A'
Or
sel == 'a'
Transform the value
ATOI()
Update mem_alloc
sel == 'F'
or
sel == 'f'
Factory Reset
load_factory_netconf()
sel == 0x1B
sel == 'E'
or
sel == 'e'
Update Network Information
set_netconf(&NetConf)
END

<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
void uart0_putchar(char c)	Write one character to 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
void uart_puts(u_char uart, char * str)	Write text lines to 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

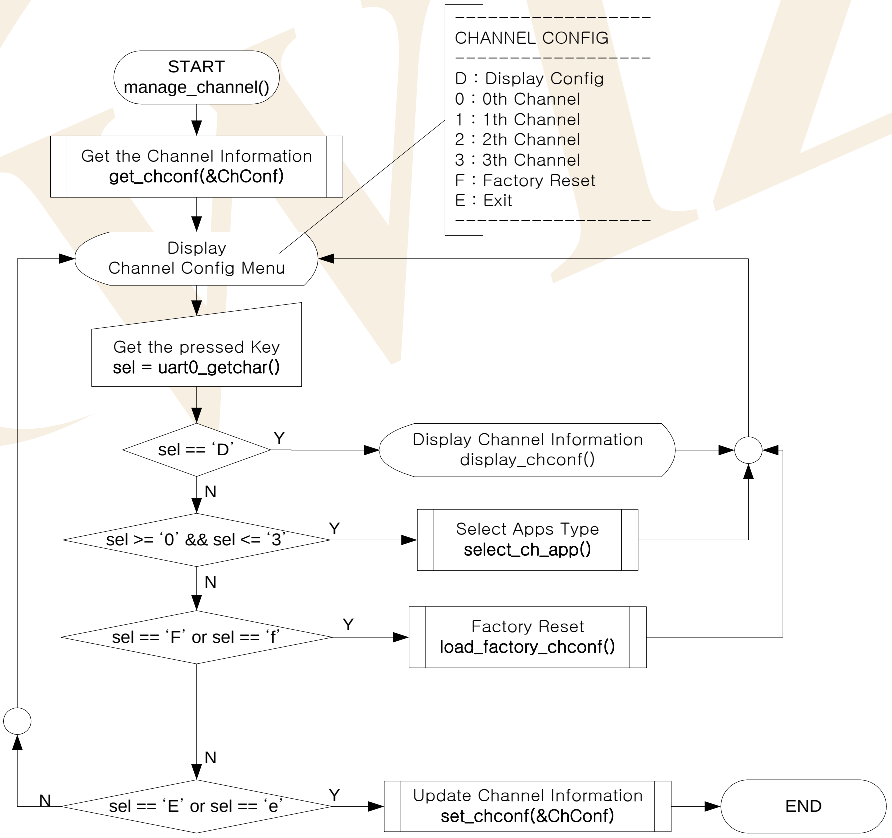
3.2.5.2. Channel Configuration

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

The application type that can be set up, there 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. But the DHCP Client can only be supported by the first channel and the setting cannot be repeated on other channel. TCP Server Program (LB_TCPS,WEB_SERVER) can be set repeatedly by channel and in such case the same port can be used. Here, the number of clients is as many as the same port number. Other application can be set repeatedly by channel but the same port number cannot use.

<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 certain destination. It uses ICMP protocol message on IP protocol and made with ping_request().

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

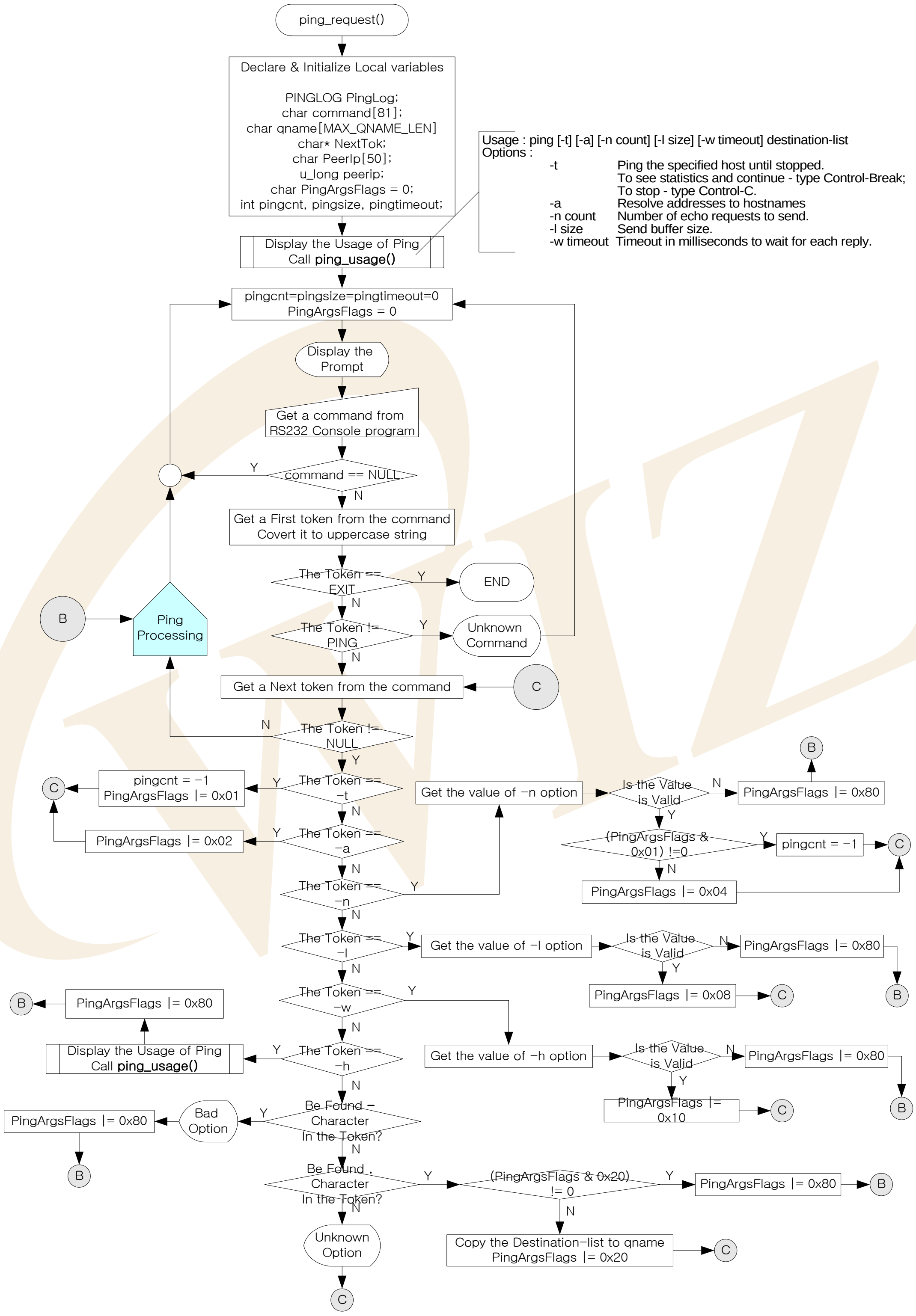
Both domain name and IP address can be used as destination address for Ping request. In case of using domain name, domain name is changed into IP address using gethostbyname() or DNS. With the changed IP address the Ping request is sent.

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.

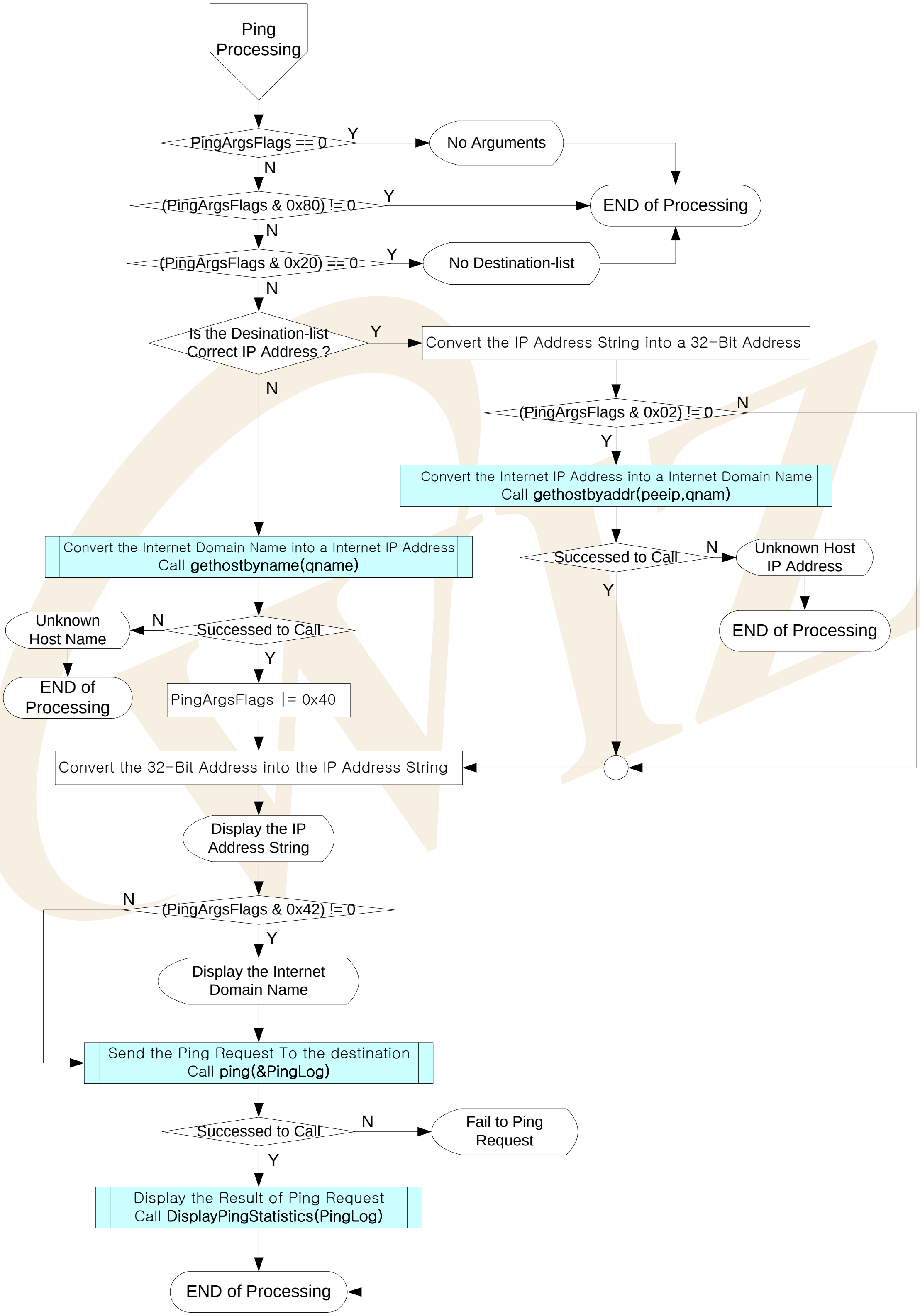
gethostbyname(), gethostbyaddr() is DNS-related functions. For further information, refer to Chapter 3.2.6.6 DNS Client. <Fig 3.8> and <Fig 3.9> are processing procedures of ping_request().

<Fig 3.8> creates tokens of inputs of Command, Option, and Option Value and decides the related Bit of Argument Flag(PingArgsFlags).

<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 certain destination and process the ICMP message which is received from any destination.


ping_request()
Declare & Initialize Local variables
PINGLOG PingLog;
char command[81];
char qname[MAX_QNAME_LEN]
char* NextTok;
char PeerIp[50];
u_long peerip;
char PingArgsFlags = 0;
int pingcnt, pingsize, pingtimeout;
Usage : ping [-t] [-a] [-n count] [-l size] [-w timeout] destination-list
Options :
-t Ping the specified host until stopped.
To see statistics and continue - type Control-Break;
To stop - type Control-C.
-a Resolve addresses to hostnames
-n count Number of echo requests to send.
-l size Send buffer size.
-w timeout Timeout in milliseconds to wait for each reply.
Display the Usage of Ping
Call ping_usage()
pingcnt=pingsize=pingtimeout=0
PingArgsFlags = 0
Display the Prompt
Get a command from RS232 Console program
command == NULL
Y
N
Get a First token from the command
Covert it to uppercase string
The Token == EXIT
Y
END
N
The Token != PING
Y
Unknown Command
N
B
Ping Processing
Get a Next token from the command
C
The Token != NULL
N
Y
The Token == -t
Y
pingcnt = −1
PingArgsFlags |= 0x01
C
N
The Token == -a
Y
PingArgsFlags |= 0x02
N
The Token == -n
Y
Get the value of −n option
Is the Value is Valid
N
PingArgsFlags |= 0x80
B
Y
(PingArgsFlags & 0x01) !=0
Y
pingcnt = −1
C
N
PingArgsFlags |= 0x04
N
The Token == -l
Y
Get the value of −l option
Is the Value is Valid
N
PingArgsFlags |= 0x80
B
Y
PingArgsFlags |= 0x08
C
N
The Token == -w
Y
B
PingArgsFlags |= 0x80
N
The Token == -h
Y
Display the Usage of Ping
Call ping_usage()
Get the value of −h option
Is the Value is Valid
N
PingArgsFlags |= 0x80
B
Y
PingArgsFlags |= 0x10
C
N
Be Found − Character In the Token?
Y
Bad Option
PingArgsFlags |= 0x80
B
N
Be Found . Character In the Token?
Y
(PingArgsFlags & 0x20) != 0
Y
PingArgsFlags |= 0x80
B
N
Copy the Destination−list to qname
PingArgsFlags |= 0x20
C
N
Unknown Option
C

<Fig 3.8: ping_request()>


Ping Processing
PingArgsFlags == 0
Y
N
No Arguments
END of Processing
(PingArgsFlags & 0x80) != 0
Y
N
(PingArgsFlags & 0x20) == 0
Y
N
No Destination-list
Is the Desination-list Correct IP Address ?
Y
N
Convert the IP Address String into a 32-Bit Address
(PingArgsFlags & 0x02) != 0
N
Y
Convert the Internet IP Address into a Internet Domain Name
Call gethostbyaddr(peeip,qnam)
Successed to Call
N
Y
Unknown Host IP Address
END of Processing
Convert the Internet Domain Name into a Internet IP Address
Call gethostbyname(qname)
Successed to Call
N
Y
Unknown Host Name
END of Processing
PingArgsFlags |= 0x40
Convert the 32-Bit Address into the IP Address String
Display the IP Address String
(PingArgsFlags & 0x42) != 0
N
Y
Display the Internet Domain Name
Send the Ping Request To the destination
Call ping(&PingLog)
Successed to Call
N
Y
Fail to Ping Request
Display the Result of Ping Request
Call DisplayPingStatistics(PingLog)
END of Processing

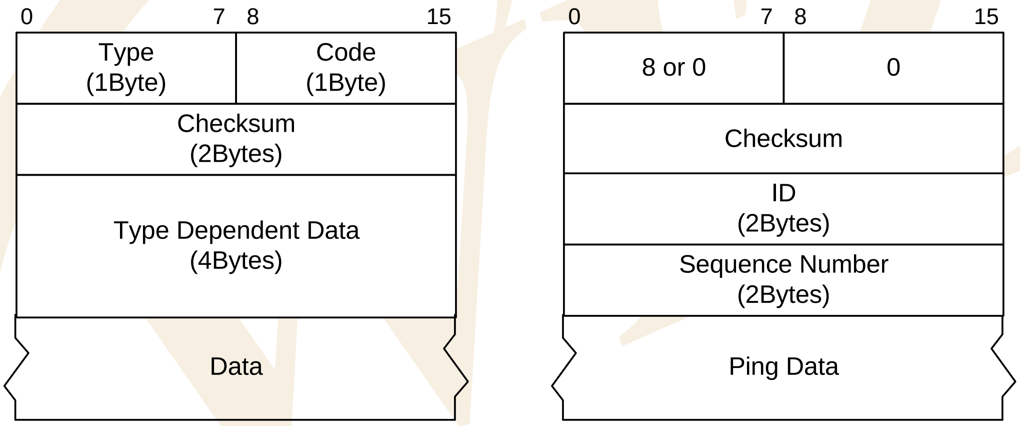
<Fig 3.9: ping_request() – Continue>

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 and Code Field of ICMP Message has 0. Also, Type Dependant Data Field(4Bytes) of ICMP Message can be re-defined as ID Field(2Bytes), Sequence Number Field(2Bytes) respectably. 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.



ICMP Message

Ping Message

<Fig 3.10: ICMP Message VS Ping Message>

Checking the Ping Reply about Ping Request can be done by checking if the values of ID, sequence number and ping data field are same. 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.

Ping Request message transmission and checking the Ping Reply message were done by ping(). The elements of ping() are Destination IP Address, Ping Reply Wait Time, number of Ping Requests, and 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 ‘PINGBUF_LEN’ Byte. PINGBUF_LEN is defined ‘32’. But Data field Max size maybe ‘1472’. This is because the sending MTU of iinChip™ 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   ARPErr;
            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 check sum of Ping Reply from peer is received wrongly.

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

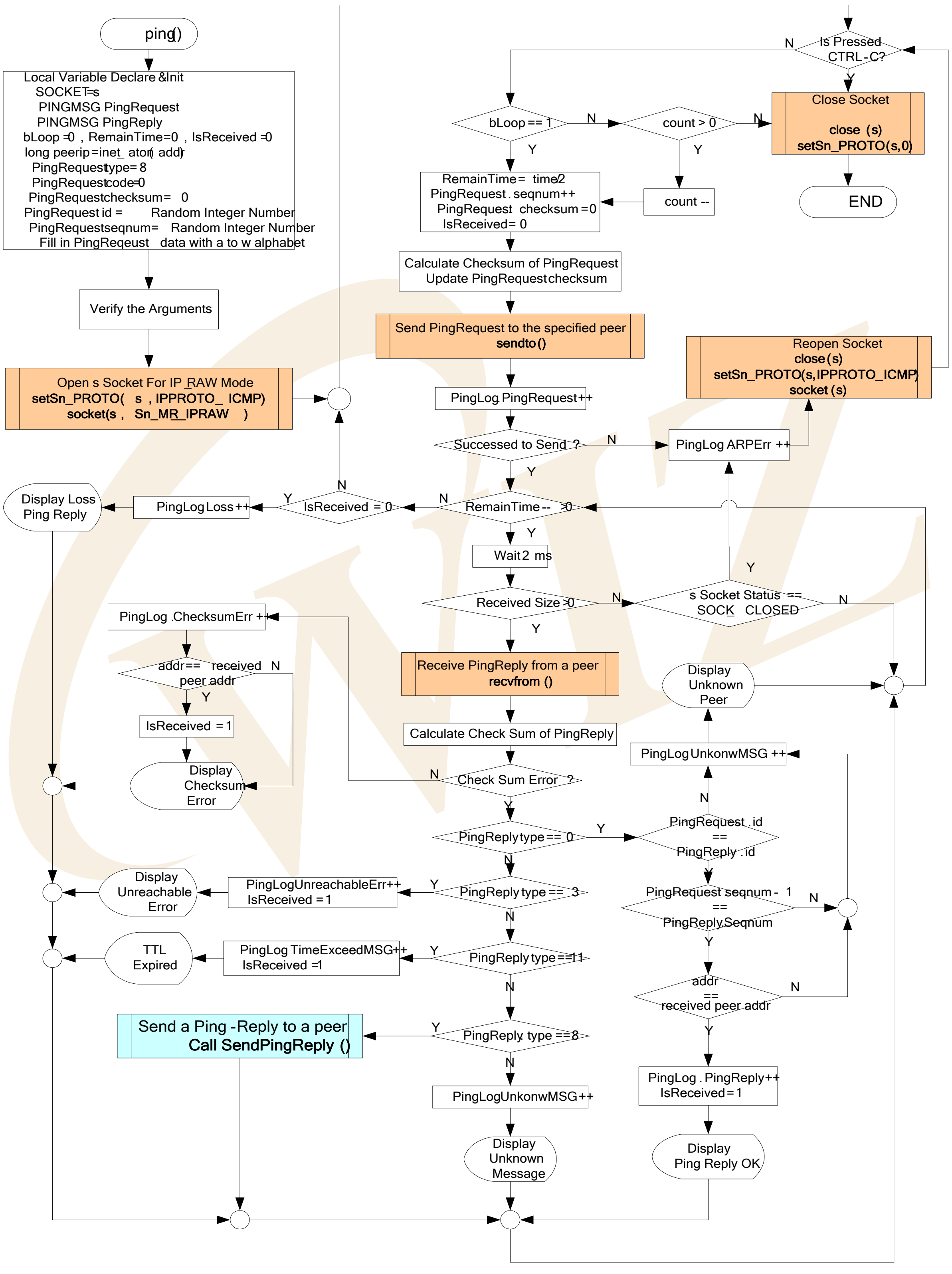
UnknownMSG field is incremented by 1 when the unknown message is received.

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.

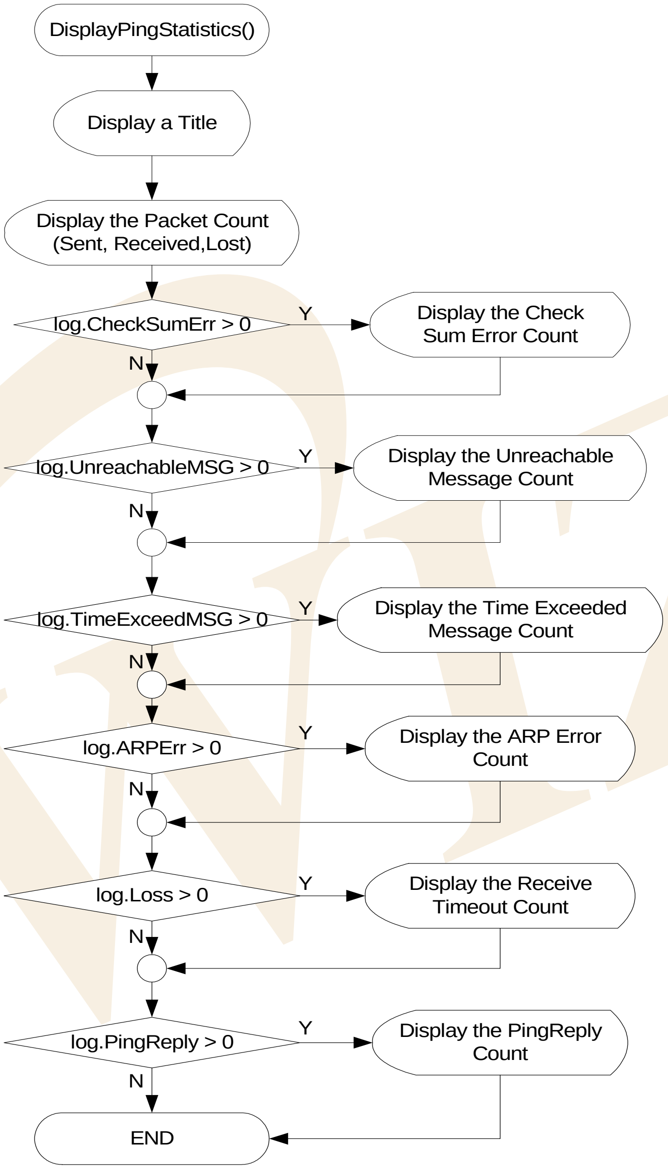
PingRequest field is incremented by 1 whenever ping() sends Ping Request.

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 gets replied to the peer in certain period of time after sending Ping Request.

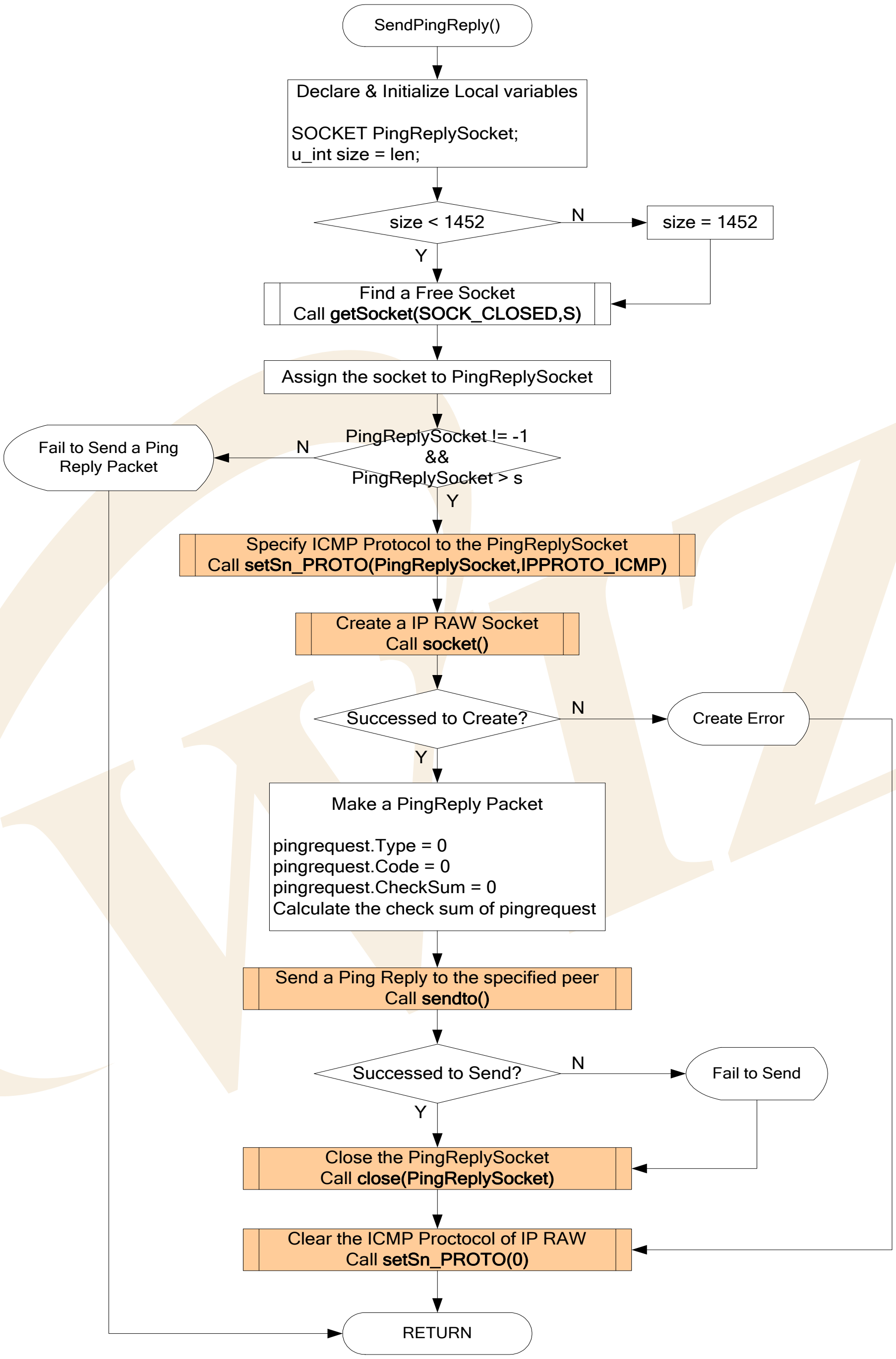

ping()
Local Variable Declare &Init
SOCKET s
PINGMSG PingRequest
PINGMSG PingReply
bLoop =0 , RemainTime=0 , IsReceived =0
long peerip =inet_ aton(addr)
PingRequest.type= 8
PingRequest.code=0
PingRequest.checksum= 0
PingRequest.id = Random Integer Number
PingRequest.seqnum= Random Integer Number
Fill in PingReqeust data with a to w alphabet
Verify the Arguments
Open s Socket For IP_RAW Mode
setSn_PROTO(s , IPPROTO_ ICMP)
socket(s , Sn_MR_IPRAW)
Is Pressed CTRL-C?
N
Y
Close Socket
close (s)
setSn_PROTO(s,0)
END
bLoop == 1
N
count > 0
N
Y
Y
RemainTime= time/2
PingRequest . seqnum++
PingRequest checksum =0
IsReceived= 0
count --
Calculate Checksum of PingRequest
Update PingRequest.checksum
Send PingRequest to the specified peer
sendto ()
Reopen Socket
close (s)
setSn_PROTO(s,IPPROTO_ICMP)
socket (s)
PingLog.PingRequest++
Successed to Send ?
N
PingLog.ARPErr ++
Y
Display Loss Ping Reply
PingLog.Loss++
Y
IsReceived = 0
N
N
RemainTime-- >0
Y
Wait 2 ms
Received Size >0
N
s Socket Status == SOCK_CLOSED
Y
N
Y
PingLog .ChecksumErr ++
addr== received peer addr
N
Y
Receive PingReply from a peer
recvfrom ()
Display Unknown Peer
IsReceived = 1
Calculate Check Sum of PingReply
PingLog.UnkonwMSG ++
Display Checksum Error
N
Check Sum Error ?
Y
N
PingReply.type == 0
Y
PingRequest .id == PingReply .id
N
Y
Display Unreachable Error
PingLog.UnreachableErr++
IsReceived = 1
Y
PingReply.type == 3
PingRequest.seqnum - 1 == PingReply.Seqnum
N
N
Y
TTL Expired
PingLog.TimeExceedMSG++
IsReceived =1
Y
PingReply.type ==11
addr == received peer addr
N
N
Y
Send a Ping -Reply to a peer
Call SendPingReply ()
Y
PingReply. type ==8
N
PingLog.PingReply++
IsReceived = 1
PingLog.UnkonwMSG++
Display Unknown Message
Display Ping Reply OK

<Fig 3.11: ping()>


DisplayPingStatistics()
Display a Title
Display the Packet Count (Sent, Received,Lost)
log.CheckSumErr > 0
Y
Display the Check Sum Error Count
N
log.UnreachableMSG > 0
Y
Display the Unreachable Message Count
N
log.TimeExceedMSG > 0
Y
Display the Time Exceeded Message Count
N
log.ARPErr > 0
Y
Display the ARP Error Count
N
log.Loss > 0
Y
Display the Receive Timeout Count
N
log.PingReply > 0
Y
Display the PingReply Count
N
END

<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 iinChip™, as shown in <Fig 3.11> and <Fig 3.13>, which IP protocol to use must be decided. The socket must be created after calling setSn_PROTO(s, IPPROT_ICMP). IP RAW Channel must be created by calling socket() function when creating the socket. In case of closing ICMP Socket, setSn_PROTO(s, 0x00) should be called after close(s) and clear the ICMP Flag which was set previously.


SendPingReply()
Declare & Initialize Local variables
SOCKET PingReplySocket;
u_int size = len;
size < 1452
N
size = 1452
Y
Find a Free Socket
Call getSocket(SOCK_CLOSED,S)
Assign the socket to PingReplySocket
PingReplySocket != -1
&&
PingReplySocket > s
N
Fail to Send a Ping Reply Packet
Y
Specify ICMP Protocol to the PingReplySocket
Call setSn_PROTO(PingReplySocket,IPPROTO_ICMP)
Create a IP RAW Socket
Call socket()
Successed to Create?
N
Create Error
Y
Make a PingReply Packet
pingrequest.Type = 0
pingrequest.Code = 0
pingrequest.CheckSum = 0
Calculate the check sum of pingrequest
Send a Ping Reply to the specified peer
Call sendto()
Successed to Send?
N
Fail to Send
Y
Close the PingReplySocket
Call close(PingReplySocket)
Clear the ICMP Proctocol of IP RAW
Call setSn_PROTO(0)
RETURN

<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 setSn_PROTO (SOCKET s, uint8 proto)	Assigns IP protocol of the related socket	iinChip/w3150a.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);	Close 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

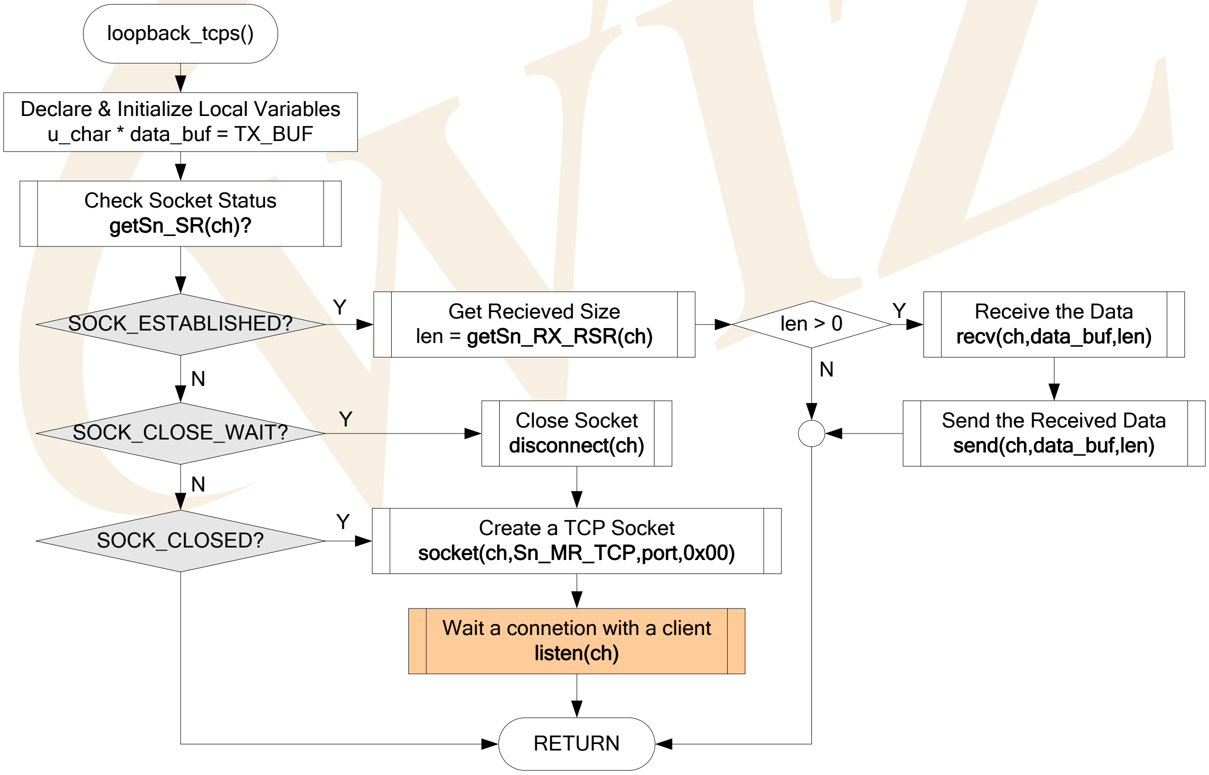
3.2.6. Applications

It's a Network Application using iinChip™ and 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, EVB B/D works as server mode and AX1, the PC testing program 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
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 puts 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 Sn_MR_TCP, Listen Port Number, and Option Flag to create TCP server socket.

The socket() function change the socket status as 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 listen status and maintains 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 "listen" to "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 possible 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

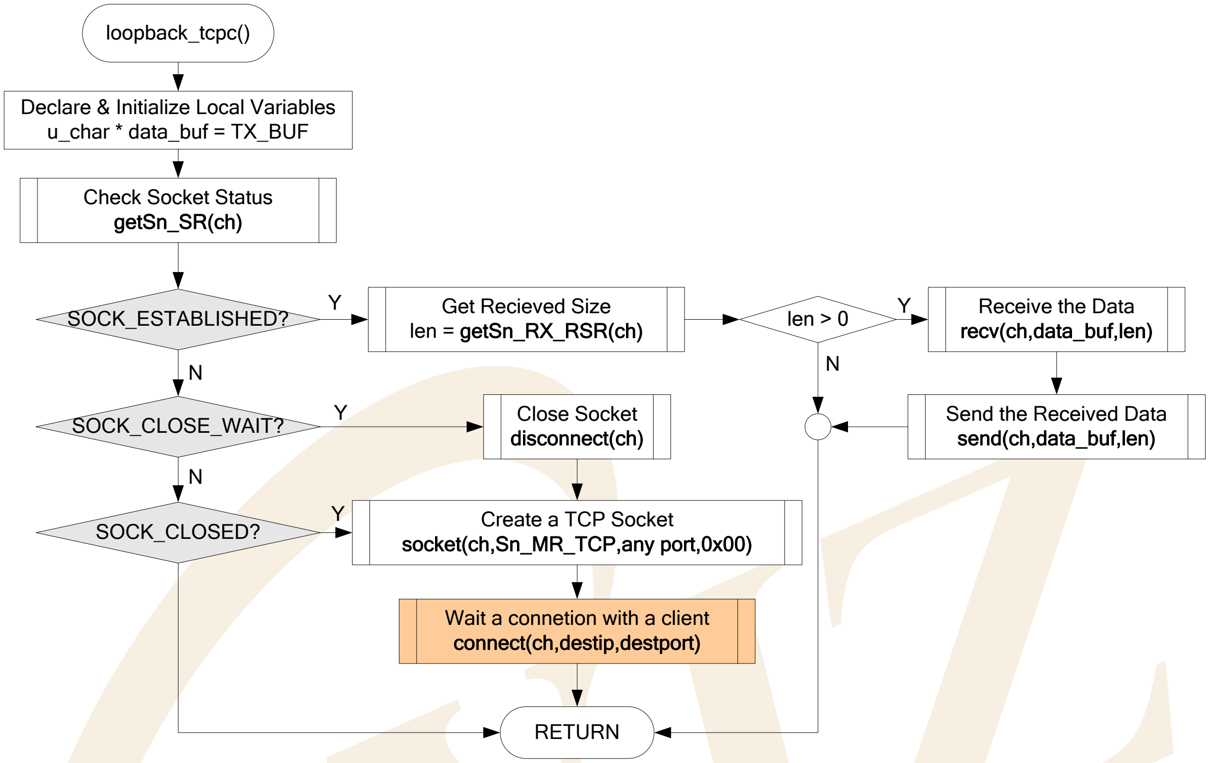
At Loopback TCP Client program, EVB B/D works in client mode and AX1, PC test program, works in server mode. EVB B/D tries to connect to AX1 which is waiting as the server, if the connection is successful EVB B/D receives data stream through TCP channel and then EVB B/D sends back the received data stream to AX1 without processing.

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

If the client socket is in SOCK_CLOSED status, loopback_tcpc() calls socket() with the elements of Sn_MR_TCP, 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 for loopback_tcps().



<Fig 3.15: loopback_tcpc()>

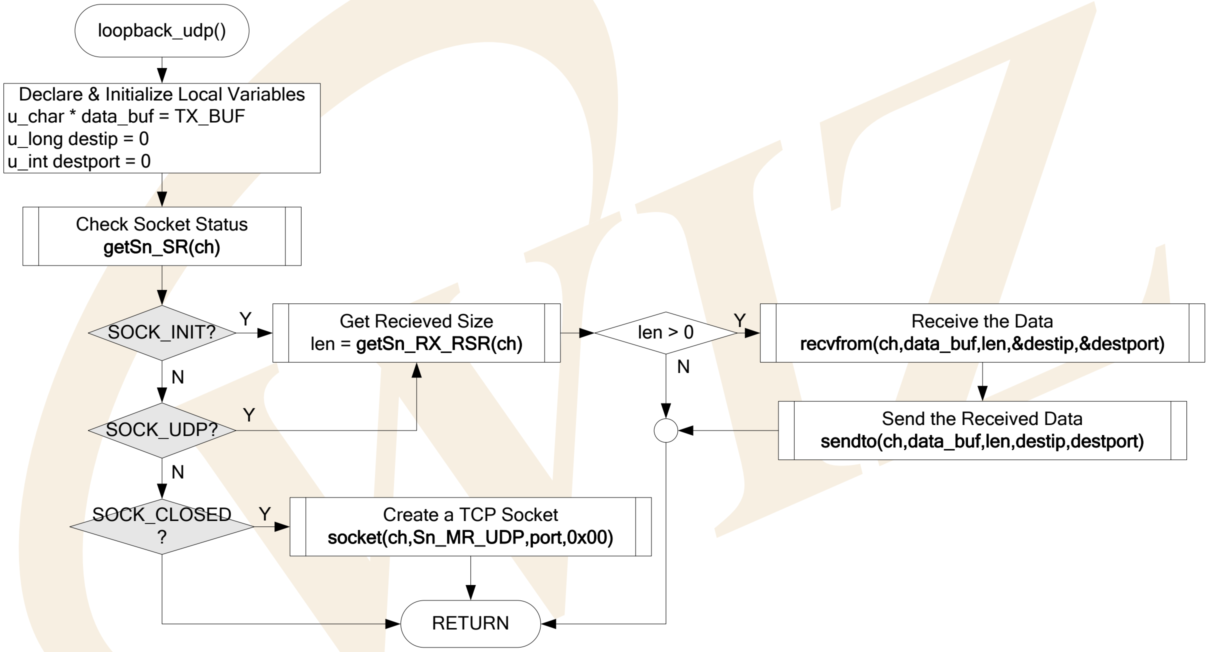
<Table 3-24: Reference Functions in loopback_tcpc()>

Function Name	Description	Location
void loopback_tcpc(u_char ch)	Loopback TCP Client Program	app/loopback.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
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 Sn_MR_UDP, 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 called always and 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

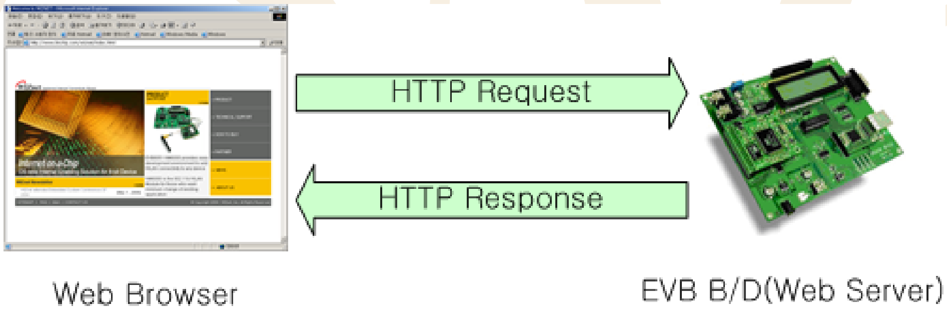
--> Client(Web Browser) requests document wanted from URL

--> 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, and in case the related URI simply requests for web page, then the page will be sent. If it requests an action such as CGI(Common Gateway Interface) then 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 and <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 depending on web browser. <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

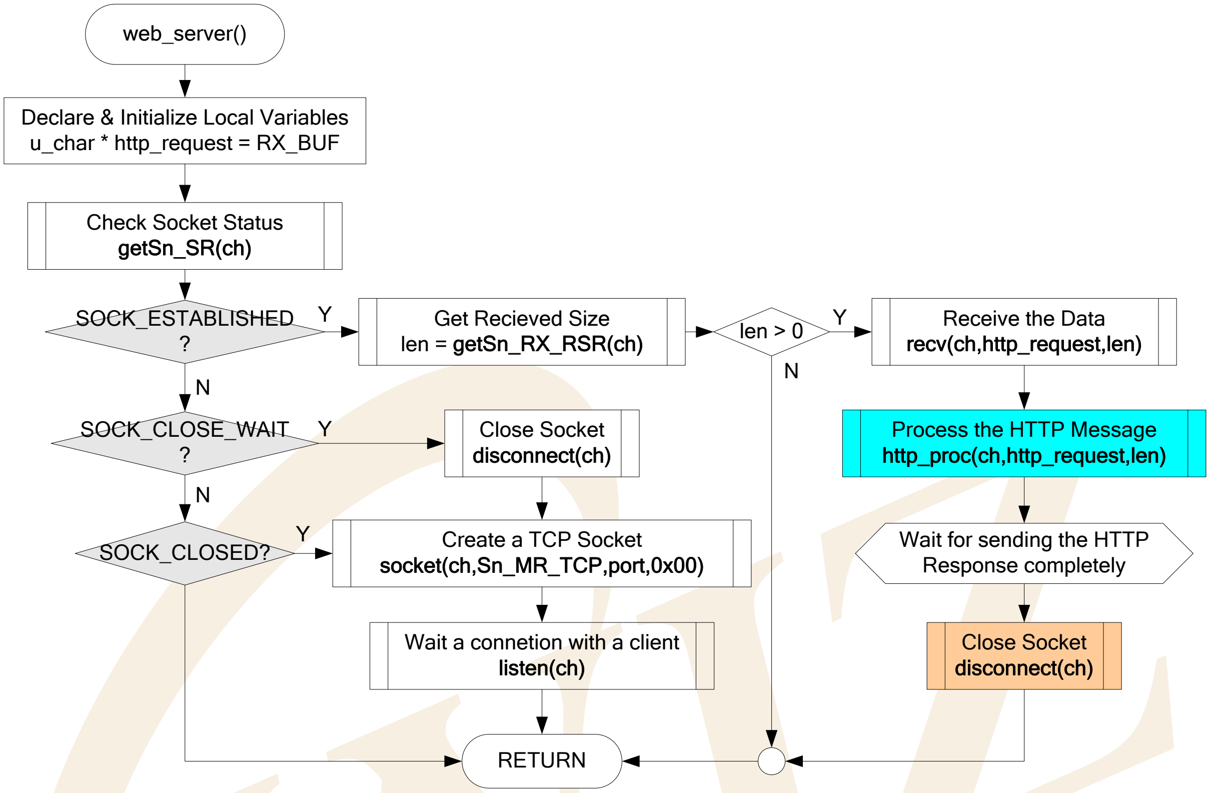
```
Ex1>    GET wiz_log.gif HTTP/1.1CRCF
        Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.-ms-
           powerpoint, application/vnd.-ms-excel, application/ms-word, */*CRCF
        Accept Language: koCRCF
        Accept Encoding: gzip, deflateCRCF
        User-Agent: 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: 1451CRCFCRCF
        [Html Document]
Ex2>    HTTP/1.1 200 OKCRCF
        Content-Type: gif/imageCRCF
        Content-Length: 613CRCFCRCF
        [GIF IMAGE]
```

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

<Fig 3.18> is processing procedure.

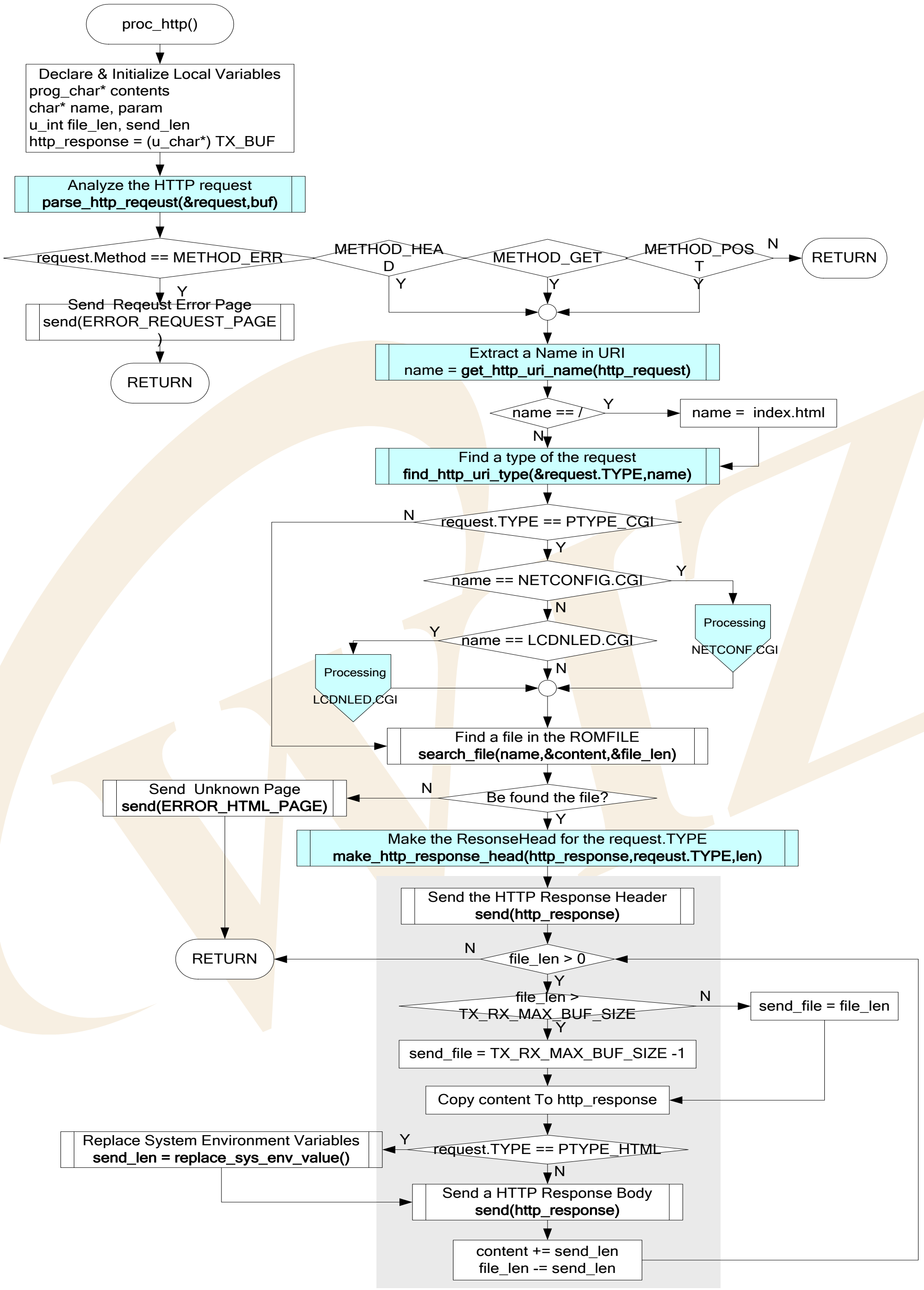


<Fig 3.18: web_server()>

Since web_server() is TCP server program 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 about 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 support to connect other client's connection.


proc_http()
Declare & Initialize Local Variables
prog_char* contents
char* name, param
u_int file_len, send_len
http_response = (u_char*) TX_BUF
Analyze the HTTP request
parse_http_reqeust(&request,buf)
request.Method == METHOD_ERR
METHOD_HEAD
METHOD_GET
METHOD_POST
N
RETURN
Y
Y
Y
Y
Send Reqeust Error Page
send(ERROR_REQUEST_PAGE)
RETURN
Extract a Name in URI
name = get_http_uri_name(http_request)
name == /
Y
name = index.html
N
Find a type of the request
find_http_uri_type(&request.TYPE,name)
N
request.TYPE == PTYPE_CGI
Y
name == NETCONFIG.CGI
Y
Processing
NETCONF.CGI
N
Y
name == LCDNLED.CGI
Processing
LCDNLED.CGI
N
Find a file in the ROMFILE
search_file(name,&content,&file_len)
Send Unknown Page
send(ERROR_HTML_PAGE)
N
Be found the file?
Y
Make the ResonseHead for the request.TYPE
make_http_response_head(http_response,reqeust.TYPE,len)
Send the HTTP Response Header
send(http_response)
RETURN
N
file_len > 0
Y
file_len > TX_RX_MAX_BUF_SIZE
N
send_file = file_len
Y
send_file = TX_RX_MAX_BUF_SIZE -1
Copy content To http_response
Replace System Environment Variables
send_len = replace_sys_env_value()
Y
request.TYPE == PTYPE_HTML
N
Send a HTTP Response Body
send(http_response)
content += send_len
file_len -= send_len

<Fig 3.19: proc_http()>

proc_http() calls parse_http_request() to analyses 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 "/" then replaces 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” then it performs the related CGI command process.

After processing CGI commands or in case that HTTP Request Type is not 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, since 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 iinChip™. Hence it has to be divided into maximum size of iinChip™ before the transmission. At this point, if system environment variables that are defined in EVB B/D in HTTP Response Body are existent, it calls replace_sys_env_value() and replaces system environment variables to system environment value that was 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>
```

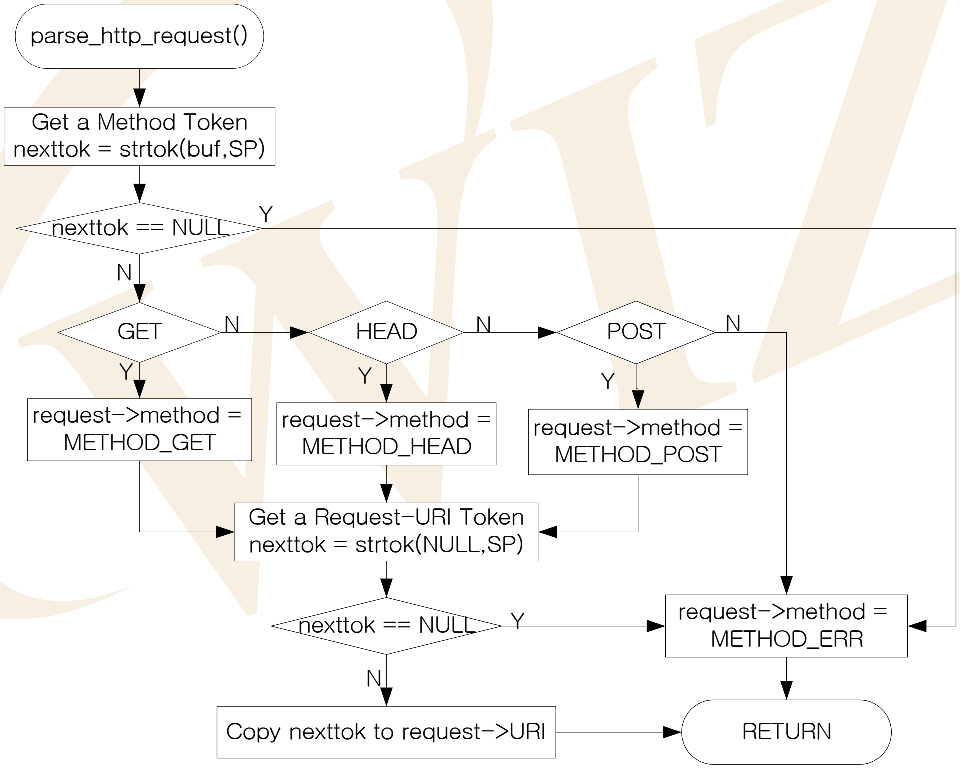
<Table 3-30> is part of “evbctrl.html” in ROM File Image of EVB B/D.

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.

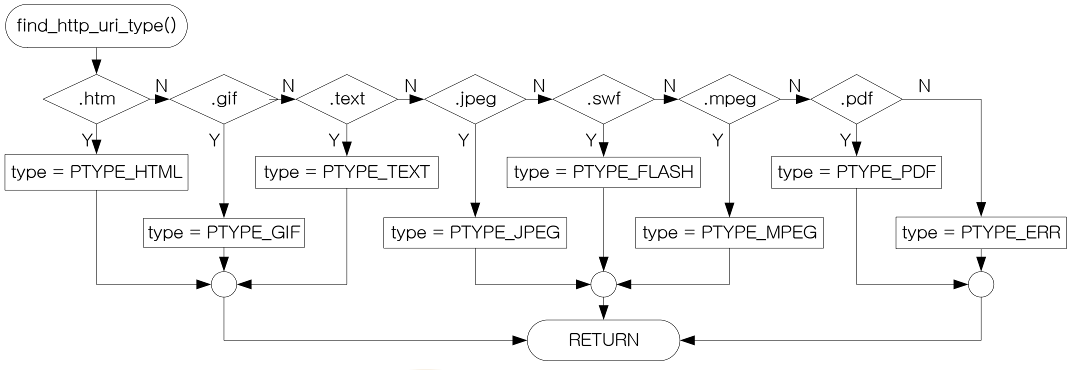
HTTP Request message can be divided into Method and Request-URI by parse_http_request() and stored in 'st_http_request' Date Type which is defined in <Table 3-31>. It gets the requested URI Type with get_http_uri_type().

<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;
```

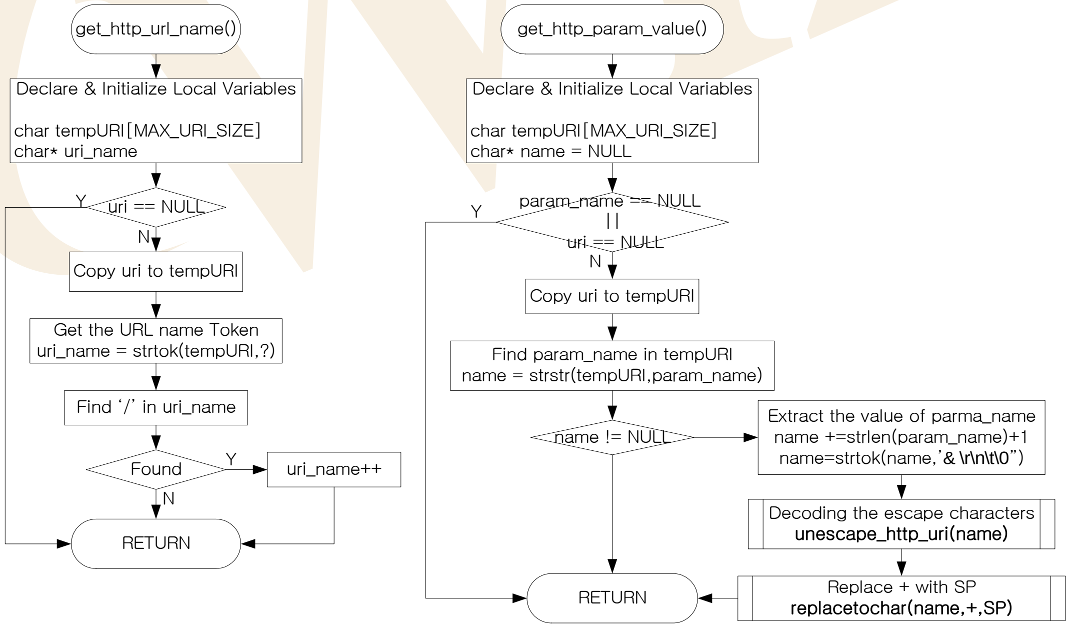


<Fig 3.20: parse_http_request()>



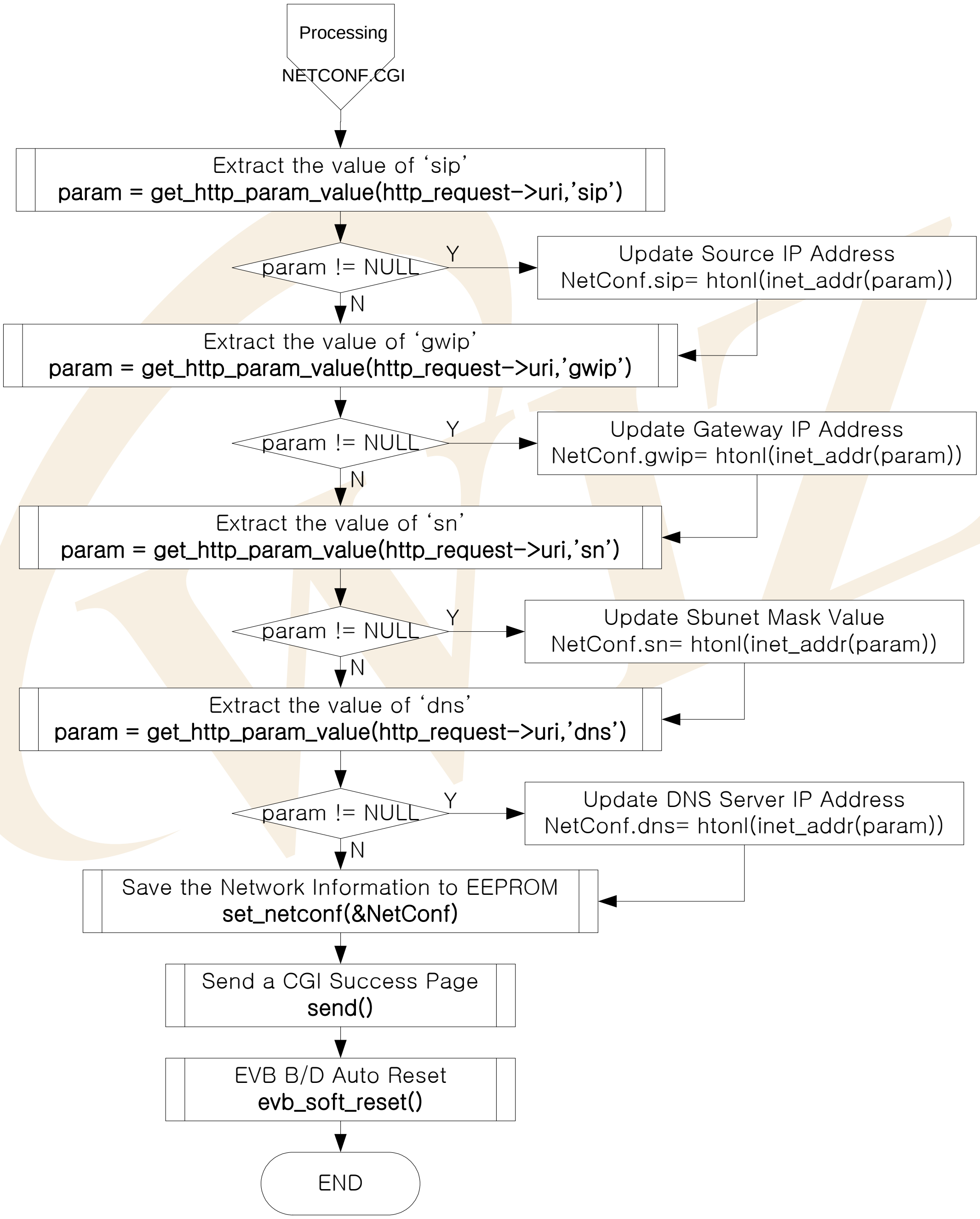
<Fig 3.21: find_http_uri_type()>

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.

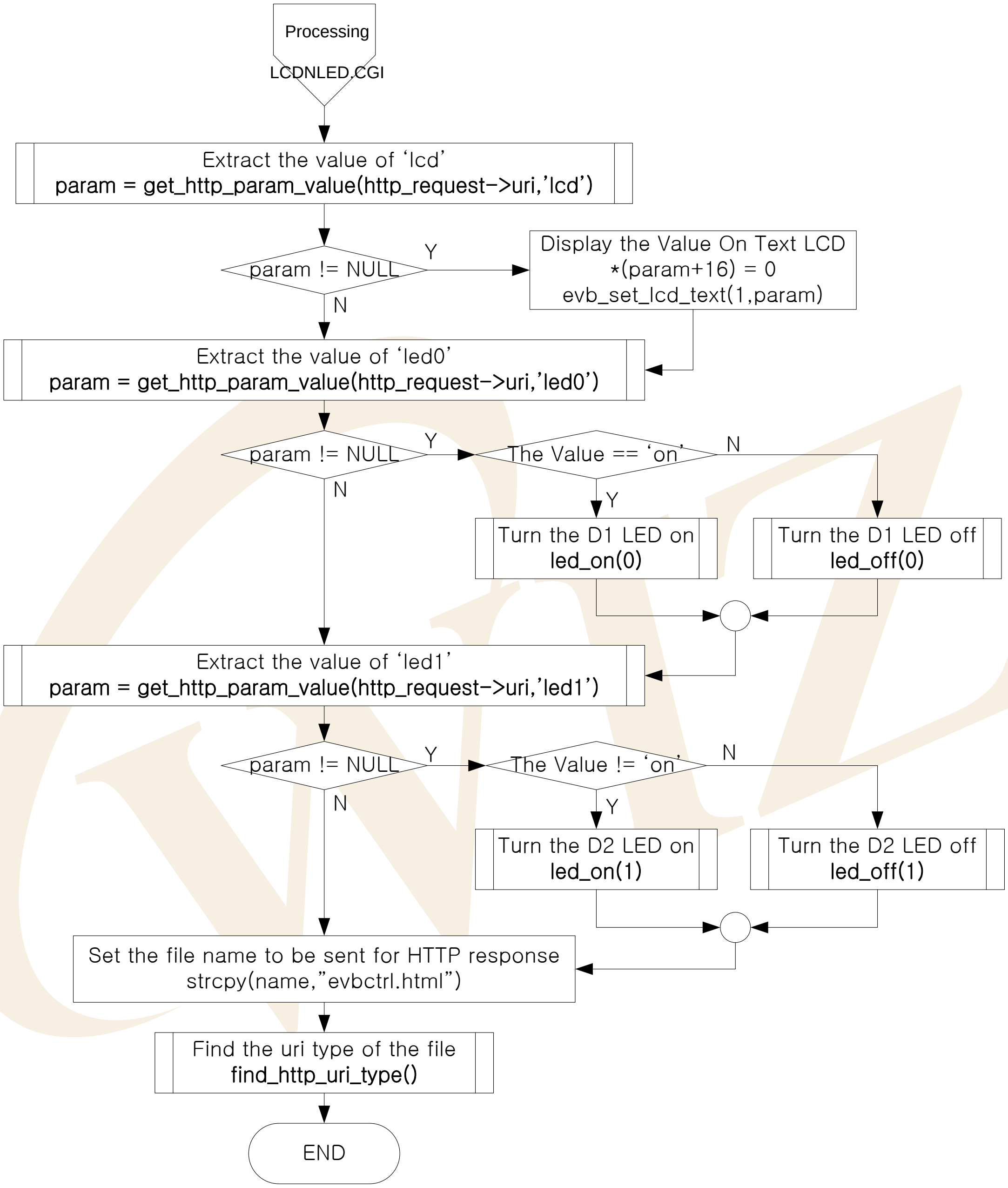

get_http_url_name()
Declare & Initialize Local Variables
char tempURI[MAX_URI_SIZE]
char* uri_name
uri == NULL
Y
N
Copy uri to tempURI
Get the URL name Token
uri_name = strtok(tempURI,?)
Find '/' in uri_name
Found
Y
N
uri_name++
RETURN
get_http_param_value()
Declare & Initialize Local Variables
char tempURI[MAX_URI_SIZE]
char* name = NULL
param_name == NULL
||
uri == NULL
Y
N
Copy uri to tempURI
Find param_name in tempURI
name = strstr(tempURI,param_name)
name != NULL
Extract the value of parma_name
name +=strlen(param_name)+1
name=strtok(name,'& \r\n\t\0")
Decoding the escape characters
unescape_http_uri(name)
Replace + with SP
replacetochar(name,+,SP)
RETURN

<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 case 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.


Processing
NETCONF.CGI
Extract the value of 'sip'
param = get_http_param_value(http_request->uri,'sip')
param != NULL
Y
N
Update Source IP Address
NetConf.sip= htonl(inet_addr(param))
Extract the value of 'gwip'
param = get_http_param_value(http_request->uri,'gwip')
param != NULL
Y
N
Update Gateway IP Address
NetConf.gwip= htonl(inet_addr(param))
Extract the value of 'sn'
param = get_http_param_value(http_request->uri,'sn')
param != NULL
Y
N
Update Sbunet Mask Value
NetConf.sn= htonl(inet_addr(param))
Extract the value of 'dns'
param = get_http_param_value(http_request->uri,'dns')
param != NULL
Y
N
Update DNS Server IP Address
NetConf.dns= htonl(inet_addr(param))
Save the Network Information to EEPROM
set_netconf(&NetConf)
Send a CGI Success Page
send()
EVB B/D Auto Reset
evb_soft_reset()
END

<Fig 3.23: NETCONF.CGI Processing>


Processing
LCDNLED.CGI
Extract the value of 'lcd'
param = get_http_param_value(http_request->uri,'lcd')
param != NULL
Y
N
Display the Value On Text LCD
*(param+16) = 0
evb_set_lcd_text(1,param)
Extract the value of 'led0'
param = get_http_param_value(http_request->uri,'led0')
param != NULL
Y
N
The Value == 'on'
N
Y
Turn the D1 LED on
led_on(0)
Turn the D1 LED off
led_off(0)
Extract the value of 'led1'
param = get_http_param_value(http_request->uri,'led1')
param != NULL
Y
N
The Value != 'on'
N
Y
Turn the D2 LED on
led_on(1)
Turn the D2 LED off
led_off(1)
Set the file name to be sent for HTTP response
strcpy(name,"evbctrl.html")
Find the uri type of the file
find_http_uri_type()
END

<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()>

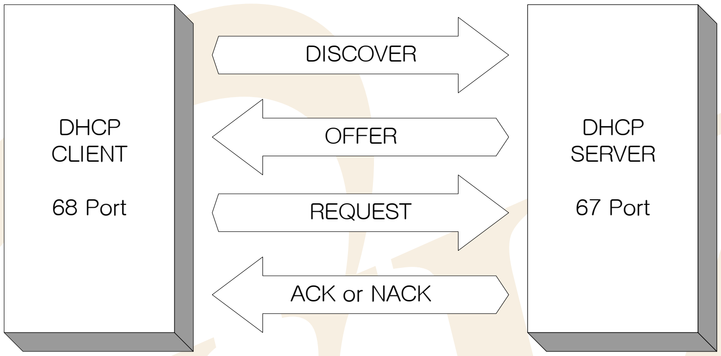
Function Name	Description	Location
void web_server(u_char ch)	Web Server Program	app/webserver.c
void proc_http(SOCKET s, u_char * buf, int length)	Processes HTTP Message using related socket	app/webserver.c
u_int replace_sys_env_value (u_char* base, u_int len)	Change Pre-defined System Environment Variables 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
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, If DHCP Client program must be started prior to other program 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.

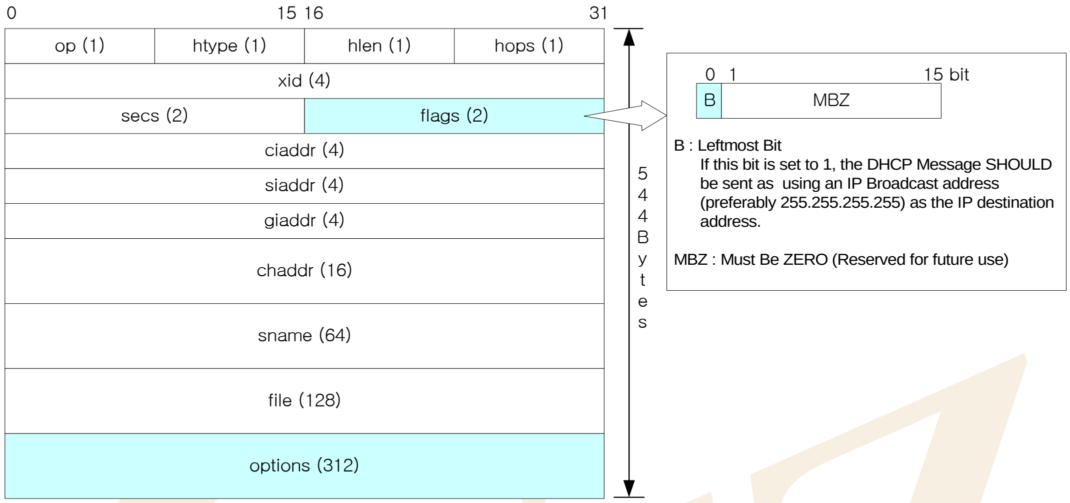
DHCP uses UDP protocol in Transport Layer and communicates with DHCP server using broadcast of UDP. The reason why it uses broadcast is because it has no IP address and DHCP server IP address is unknown. When UDP broadcast at iinChip™, destination IP address needs to be set '255.255.255.255' for broadcast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


0
15 16
31
op (1)
htype (1)
hlen (1)
hops (1)
xid (4)
secs (2)
flags (2)
ciaddr (4)
siaddr (4)
giaddr (4)
chaddr (16)
sname (64)
file (128)
options (312)
544Bytes
0 1
15 bit
B
MBZ
B : Leftmost Bit
If this bit is set to 1, the DHCP Message SHOULD be sent as using an IP Broadcast address (preferably 255.255.255.255) as the IP destination address.
MBZ : Must Be ZERO (Reserved for future use)

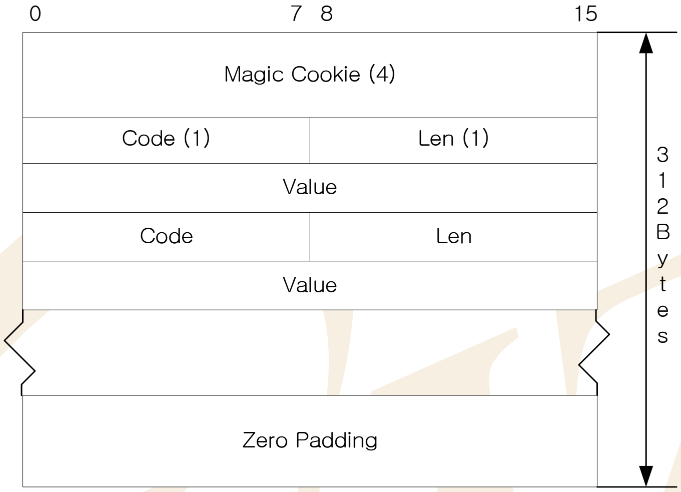
<Fig 3.26: DHCP Message Format>

<Table 3-32: DHCP Message Data Type>

```
typedef struct _RIP_MSG
{
        u_char   op;              // DHCP_BOOTREQEUST 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, Options 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 Options 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.

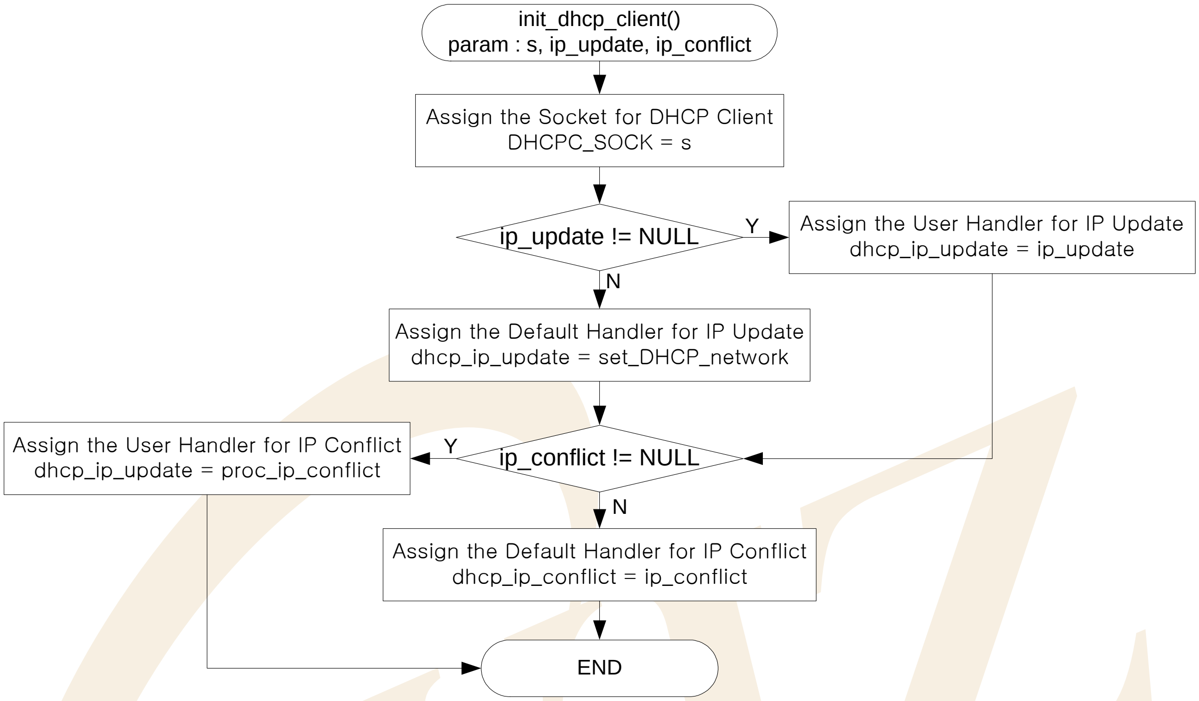
<Table 3-34> is defined as Enumeration Data Type in "inet/dhcp.h" and shows most common Option Codes that are used in DHCP Client Program.

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: EVB B/D's main()>

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 function to be called in case of collision of the IP received from DHCP Server and in case of renewal the IP from DHCP Server.

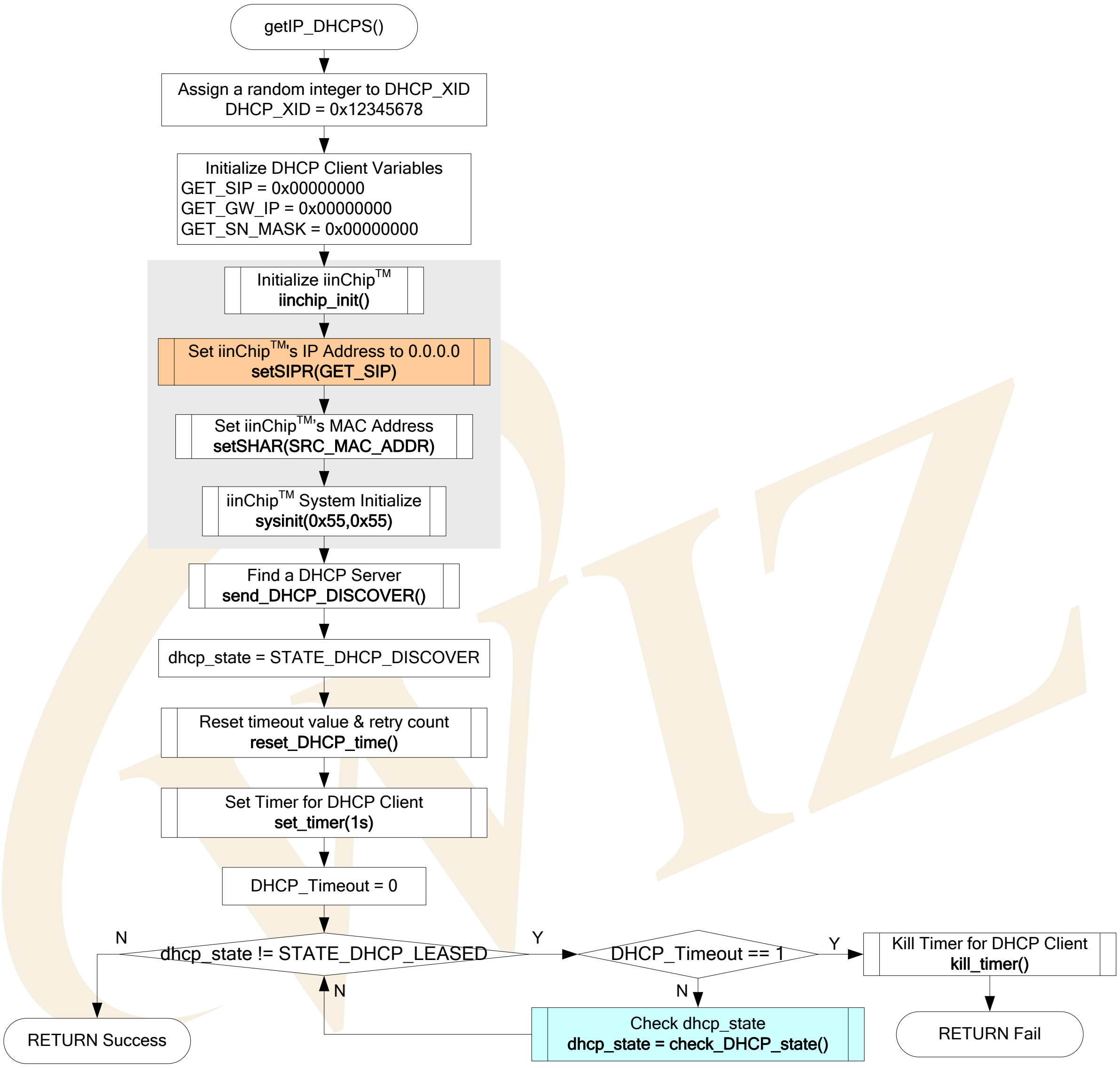
When calling init_dhcp_client(), if each function is not specified, set_DHCP_network() and proc_ip_conflict() of DHCP Client Program are registered respectively.


init_dhcp_client()
param : s, ip_update, ip_conflict
Assign the Socket for DHCP Client
DHCPC_SOCK = s
ip_update != NULL
Y
Assign the User Handler for IP Update
dhcp_ip_update = ip_update
N
Assign the Default Handler for IP Update
dhcp_ip_update = set_DHCP_network
Assign the User Handler for IP Conflict
dhcp_ip_update = proc_ip_conflict
Y
ip_conflict != NULL
N
Assign the Default Handler for IP Conflict
dhcp_ip_conflict = ip_conflict
END

<Fig 3.28: init_dhcp_client()>

When network information is renewal 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_DHCPS().



<Fig 3.29: getIP_DHCPS()>

getIP_DHCPS(), it initializes iinChip™ using setSIPR(),setSHAR(),and etc. And then 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 it's defines at 'MAX_DHCP_RETRY.' While waiting for 'DHCP_WAIT_TIME x 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 is obtained 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 x MAX_DHCP_RETRY' check_DHCP_state() sets DHCP_Timeout to 1. When DHCP_Timeout is 1, getIP_DHCPS() 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 previous 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 maintained 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 random value.

Be advised to set Source IP Address as '0.0.0.0.' when initializing iinChip™ for communication with DHCP server. You can use any IP address to set Source IP address of iinChip™ 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.

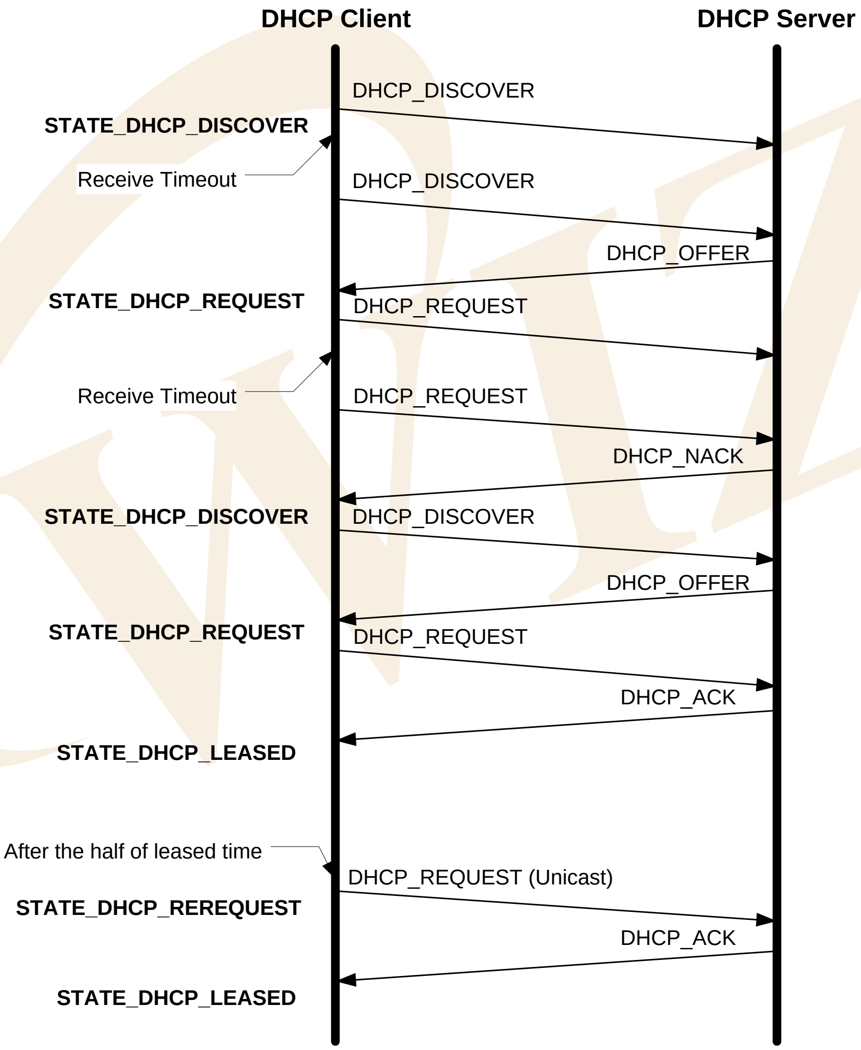
For DHCP server to transmit UDP broadcast packet, note that Flag field MSB of DHCP message must be set 1. Refer to <Fig 3.26: DHCP Message Format>.

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

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

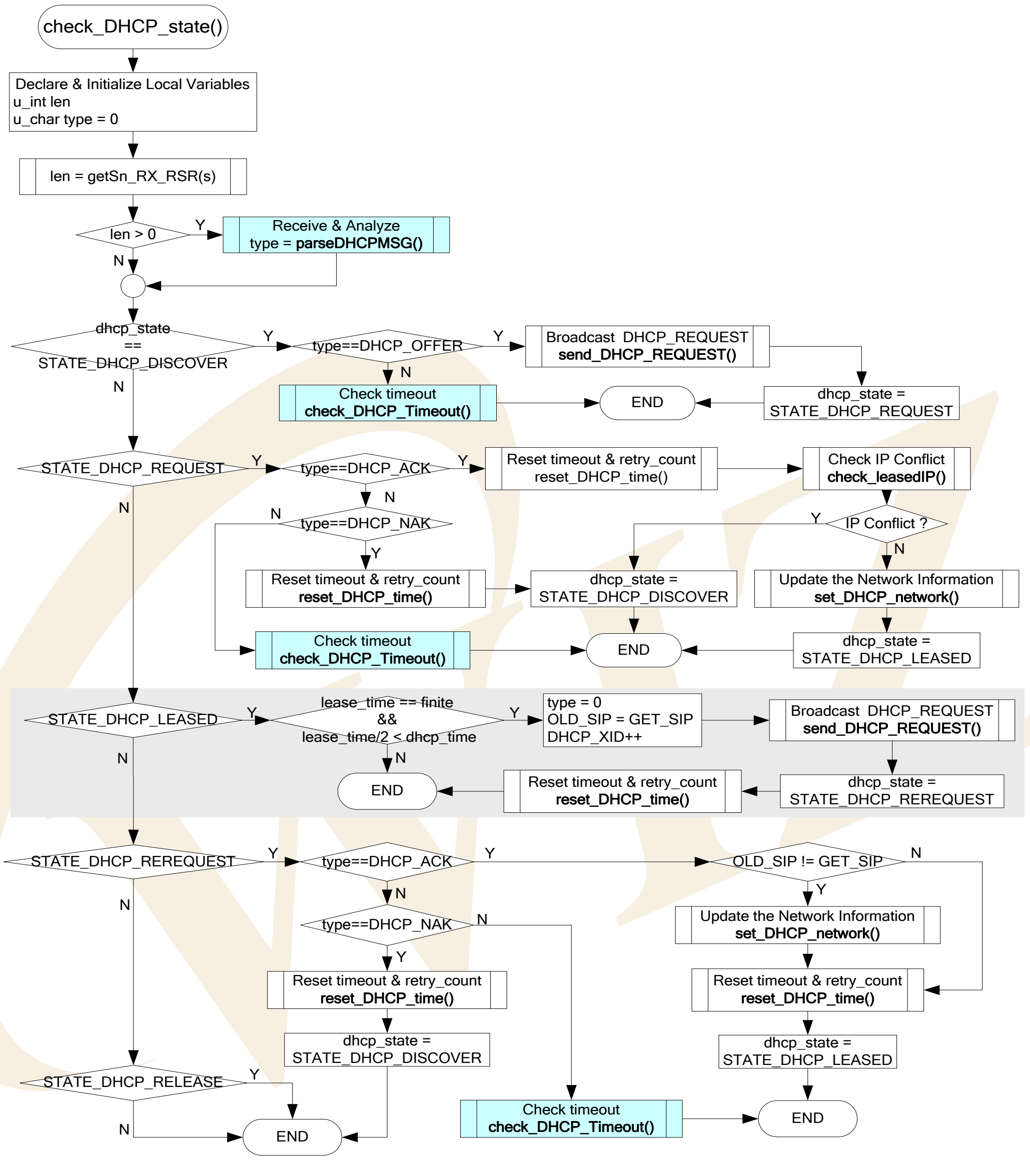
```
#define DHCP_FLAGSBROADCAST            0x8000
pRIPMSG->flags = htons(DHCP_FLAGSBROADCAST);
```

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.



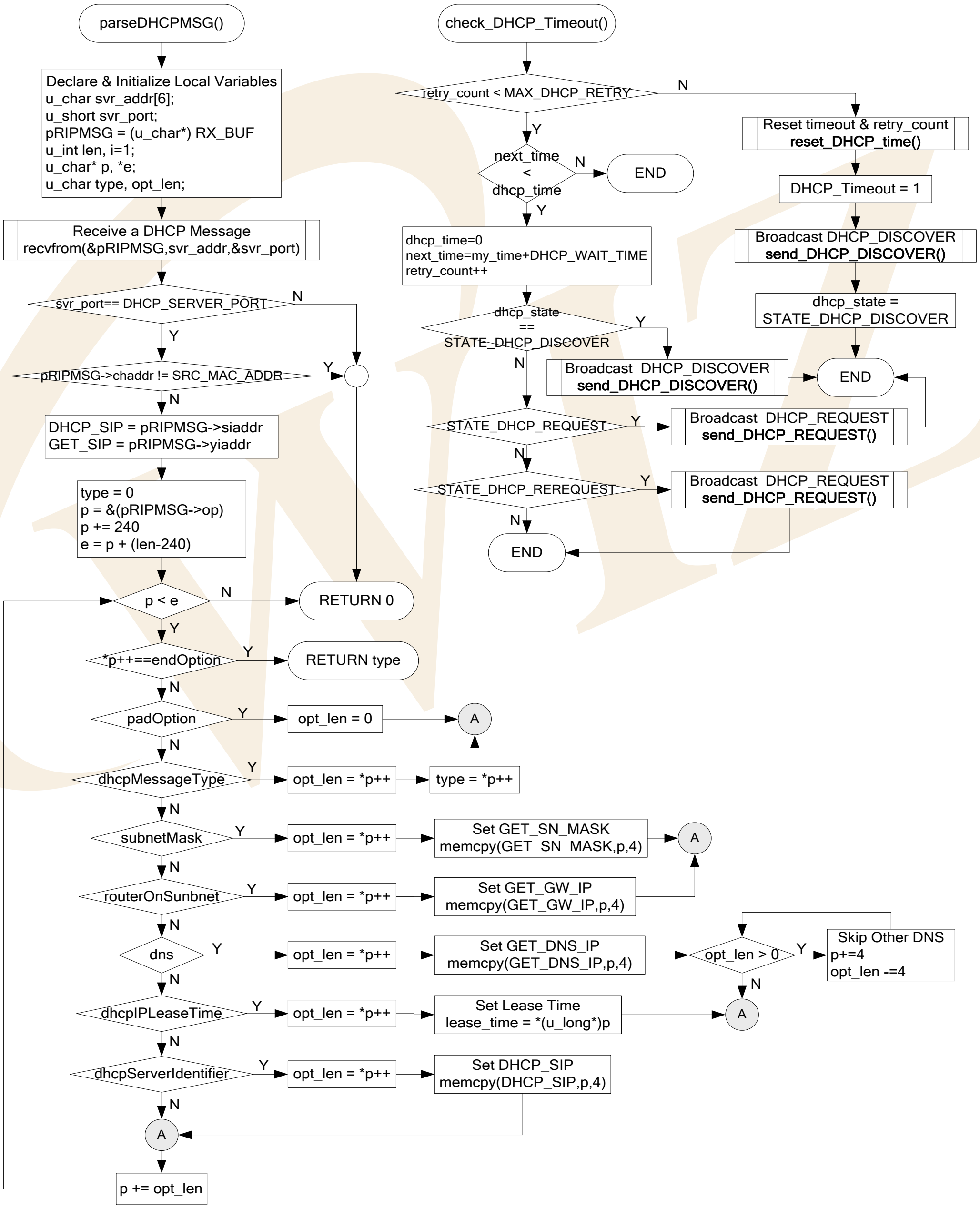
<Fig 3.30: DHCP Message Flow by DHCP Client State>

check_DHCP_state() checks if there is DHCP message from DHCP server, it receives and analyze DHCP message. By 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 DHCP Message Flow of <Fig 3.30> indicates.


check_DHCP_state()
Declare & Initialize Local Variables
u_int len
u_char type = 0
len = getSn_RX_RSR(s)
len > 0
Y
N
Receive & Analyze
type = parseDHCPMSG()
dhcp_state
==
STATE_DHCP_DISCOVER
Y
N
type==DHCP_OFFER
Y
N
Broadcast DHCP_REQUEST
send_DHCP_REQUEST()
Check timeout
check_DHCP_Timeout()
END
dhcp_state =
STATE_DHCP_REQUEST
STATE_DHCP_REQUEST
Y
N
type==DHCP_ACK
Y
N
Reset timeout & retry_count
reset_DHCP_time()
Check IP Conflict
check_leasedIP()
type==DHCP_NAK
N
Y
IP Conflict ?
Y
N
Reset timeout & retry_count
reset_DHCP_time()
dhcp_state =
STATE_DHCP_DISCOVER
Update the Network Information
set_DHCP_network()
Check timeout
check_DHCP_Timeout()
END
dhcp_state =
STATE_DHCP_LEASED
STATE_DHCP_LEASED
Y
N
lease_time == finite
&&
lease_time/2 < dhcp_time
Y
N
type = 0
OLD_SIP = GET_SIP
DHCP_XID++
Broadcast DHCP_REQUEST
send_DHCP_REQUEST()
END
Reset timeout & retry_count
reset_DHCP_time()
dhcp_state =
STATE_DHCP_REREQUEST
STATE_DHCP_REREQUEST
Y
N
type==DHCP_ACK
Y
N
OLD_SIP != GET_SIP
N
Y
type==DHCP_NAK
N
Y
Update the Network Information
set_DHCP_network()
Reset timeout & retry_count
reset_DHCP_time()
Reset timeout & retry_count
reset_DHCP_time()
dhcp_state =
STATE_DHCP_DISCOVER
dhcp_state =
STATE_DHCP_LEASED
STATE_DHCP_RELEASE
Y
N
END
Check timeout
check_DHCP_Timeout()
END

<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_REQEUST 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_DHCPS(void)	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 iinChip™.	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_dhcpc_ch(SOCKET s)	Creates DHCP client socket.	inet/dhcp.c

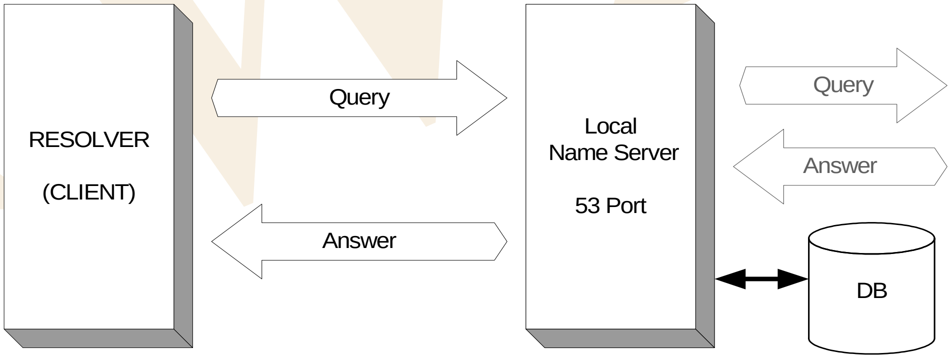
u_char socket(SOCKET s, u_char protocol, u_int port, u_char flag)	Creates sockets as TCP/UDP/IP	iinChip/socket.c
u_int sendto(SOCKET s, const u_char * buf, u_int len, u_char * addr, u_int port)	Transmits data through specific port of specific Destination	iinChip/socket.c
u_int recvfrom(SOCKET s, u_char * 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.

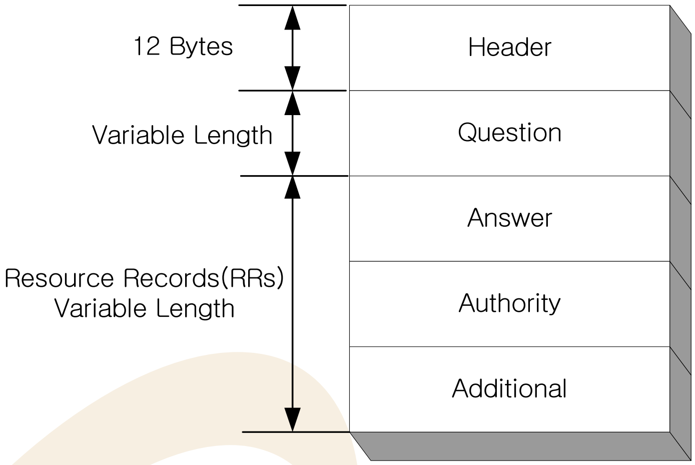
DNS is a system that transforms Internet Domain Name to Internet IP Address or Internet IP Address to Internet Domain Name. DNS is composed of Name Server that contains mapping table between IP Address and Domain Name, and DNS resolver that receives query results by transmitting query to Name Server.

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.



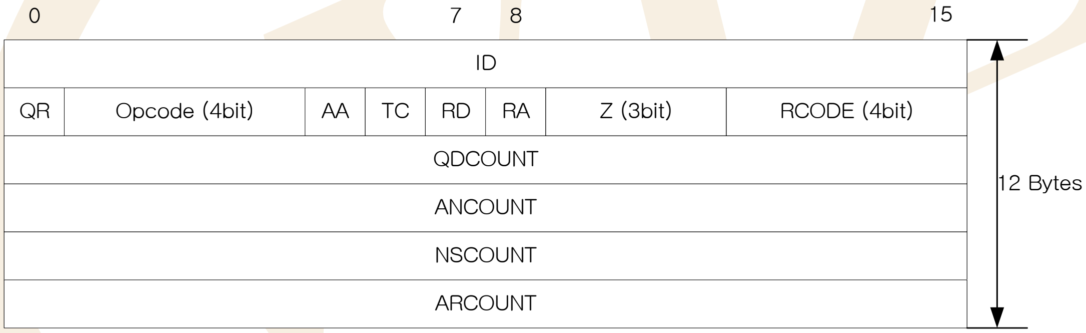
<Fig 3.33: Domain Name System Structure & DNS Message Flow>

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>.



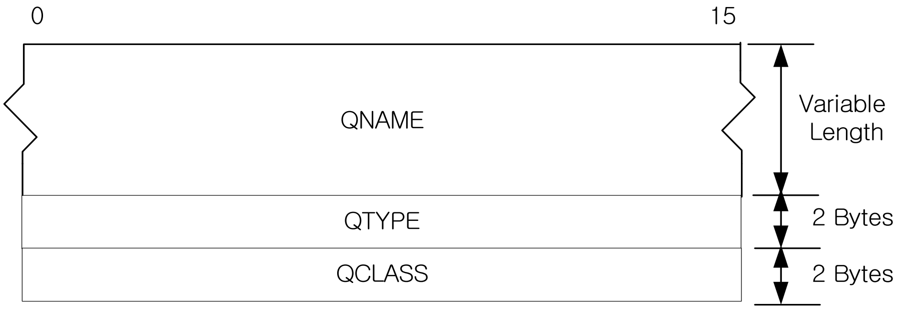
<Fig 3.34: DNS Message Format>

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.

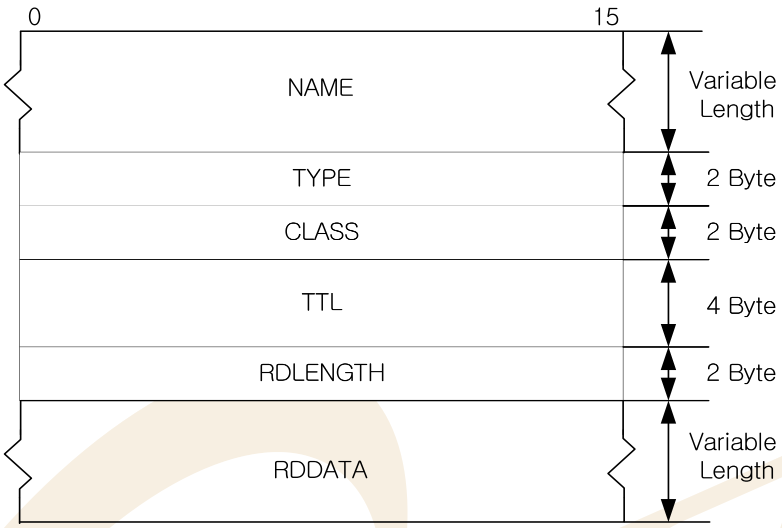


ID							
QR	Opcode (4bit)	AA	TC	RD	RA	Z (3bit)	RCODE (4bit)
QDCOUNT							
ANCOUNT							
NSCOUNT							
ARCOUNT							

<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 Block of <Fig 3.36: Question Section Format> Recode Resources(RRs) which are Format. Answer, Authority, and Additional Sections are composed of Block of <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> and Answer, Authority, and Additional Section are composed of 10 Blocks 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>

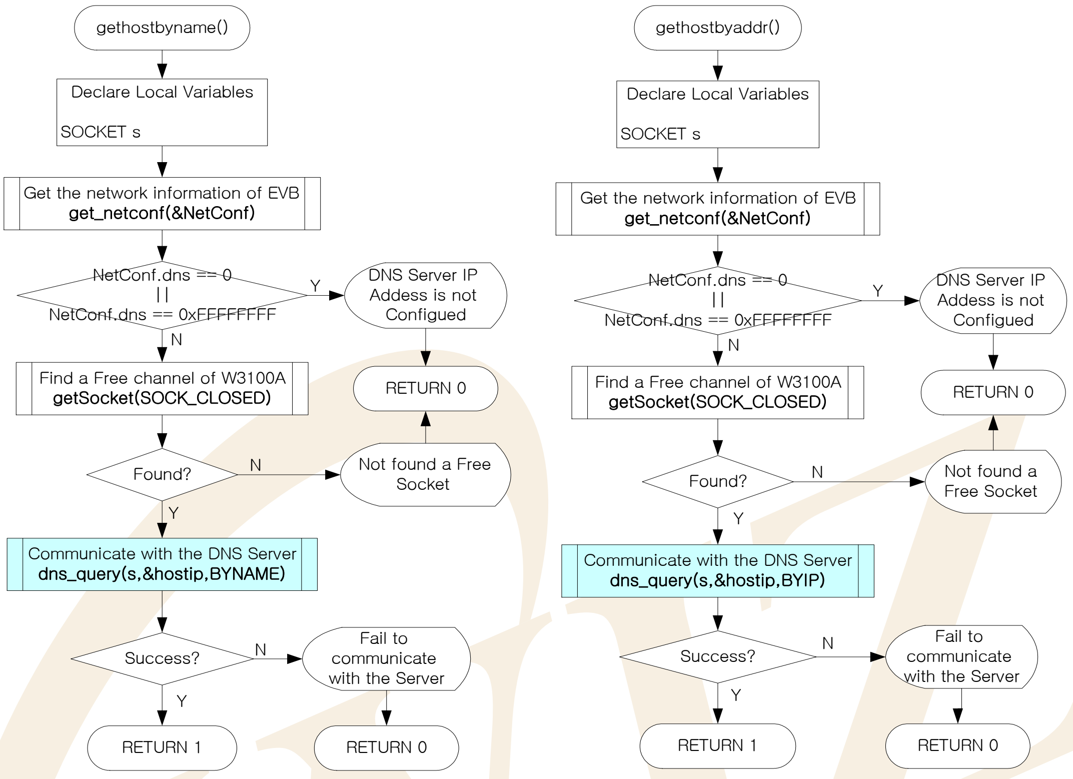
```
/* Header Section */
typedef struct _DHDR
{
        u_int   id;                             /* Identification */
        u_char flag0;
        u_char flag1;
        u_int   qdcount;  /* Question count */
        u_int   ancount;  /* Answer count */
        u_int   nscount;  /* Authority (name server) 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 Channel of iinChip™ needed for connection with DNS Name Server. If Free Channel of iinChip™ 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 */
```


gethostbyname()
Declare Local Variables
SOCKET s
Get the network information of EVB
get_netconf(&NetConf)
NetConf.dns == 0
||
NetConf.dns == 0xFFFFFFFF
Y
DNS Server IP Addess is not Configued
N
Find a Free channel of W3100A
getSocket(SOCK_CLOSED)
RETURN 0
Found?
N
Not found a Free Socket
Y
Communicate with the DNS Server
dns_query(s,&hostip,BYNAME)
Success?
N
Fail to communicate with the Server
Y
RETURN 1
RETURN 0
gethostbyaddr()
Declare Local Variables
SOCKET s
Get the network information of EVB
get_netconf(&NetConf)
NetConf.dns == 0
||
NetConf.dns == 0xFFFFFFFF
Y
DNS Server IP Addess is not Configued
N
Find a Free channel of W3100A
getSocket(SOCK_CLOSED)
RETURN 0
Found?
N
Not found a Free Socket
Y
Communicate with the DNS Server
dns_query(s,&hostip,BYIP)
Success?
N
Fail to communicate with the Server
Y
RETURN 1
RETURN 0

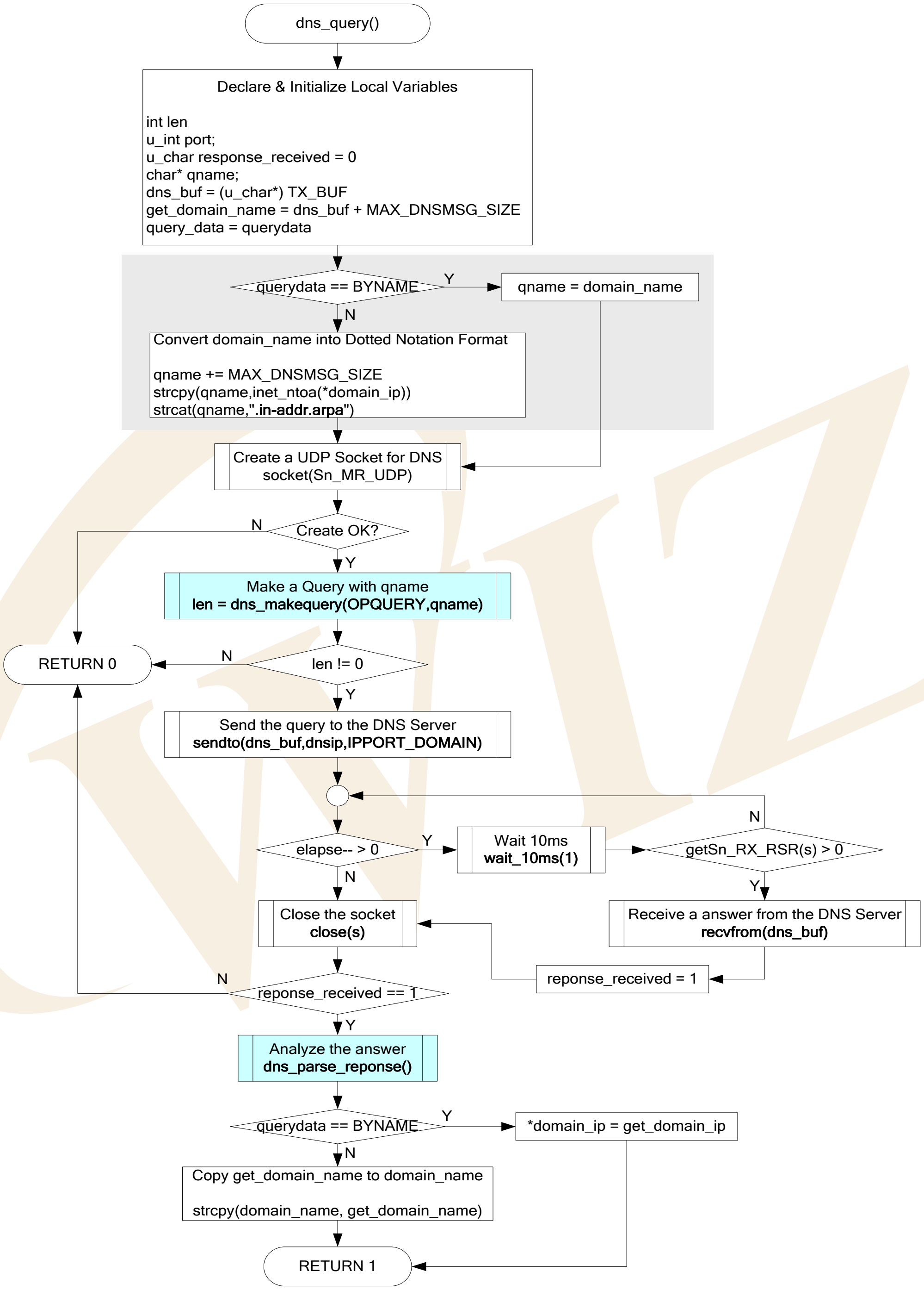
<Fig 3.38: gethostbyaddr() & gethostbyname()>

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.

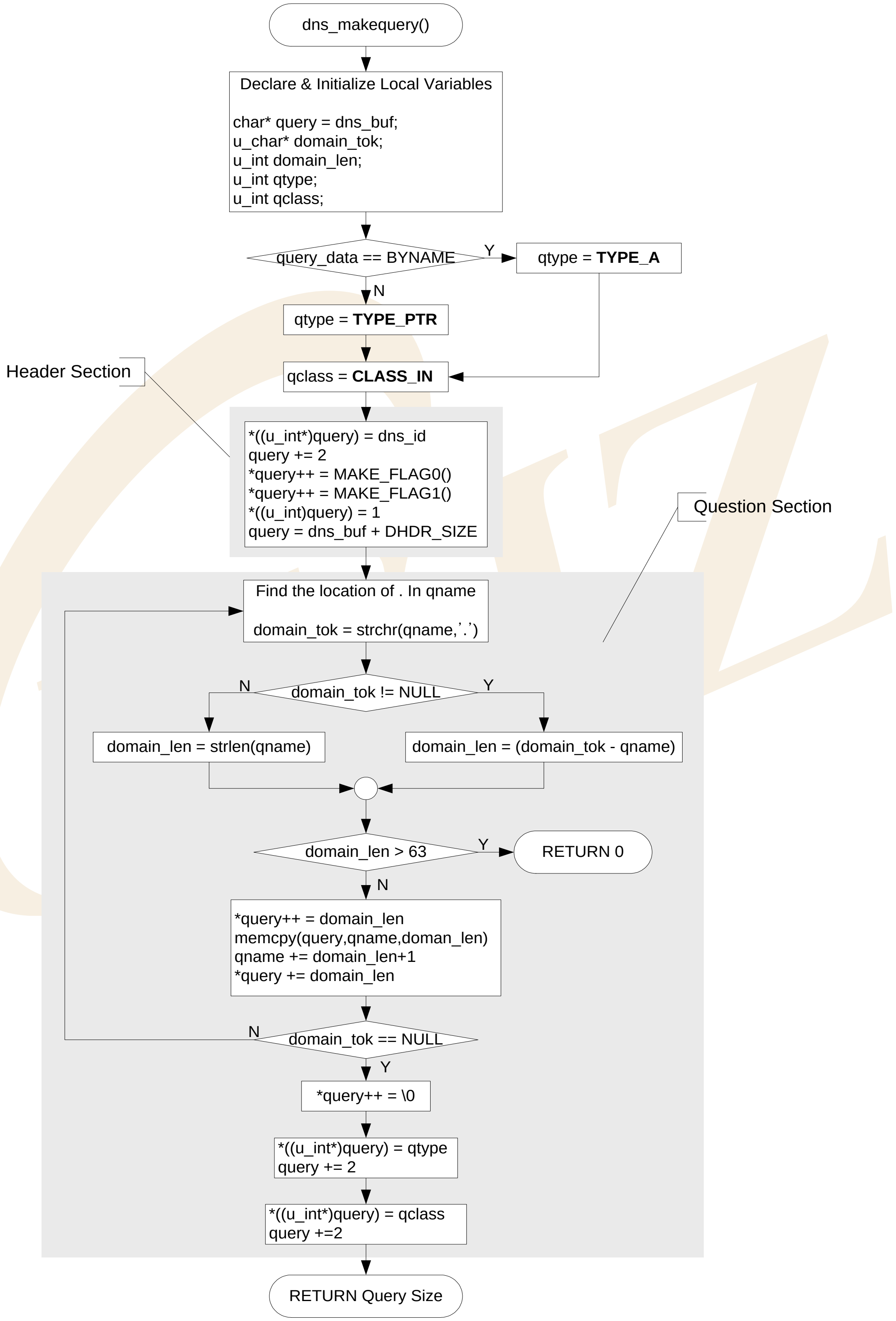
When Query Type is 'BYIP,' that is, when querying the Domain Name with IP Address, change IP Address to IP Address String and QNAME is used after adding "**in-addr.arpa"** to the changed IP Address String. After the creation of QNAME, UDP Socket is created for DNS inter-working and DNS Request Message 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


dns_query()
Declare & Initialize Local Variables
int len
u_int port;
u_char response_received = 0
char* qname;
dns_buf = (u_char*) TX_BUF
get_domain_name = dns_buf + MAX_DNSMSG_SIZE
query_data = querydata
querydata == BYNAME
Y
qname = domain_name
N
Convert domain_name into Dotted Notation Format
qname += MAX_DNSMSG_SIZE
strcpy(qname,inet_ntoa(*domain_ip))
strcat(qname,".in-addr.arpa")
Create a UDP Socket for DNS
socket(Sn_MR_UDP)
N
Create OK?
Y
Make a Query with qname
len = dns_makequery(OPQUERY,qname)
RETURN 0
N
len != 0
Y
Send the query to the DNS Server
sendto(dns_buf,dnsip,IPPORT_DOMAIN)
N
elapse-- > 0
Y
Wait 10ms
wait_10ms(1)
getSn_RX_RSR(s) > 0
N
Y
Close the socket
close(s)
Receive a answer from the DNS Server
recvfrom(dns_buf)
reponse_received = 1
N
reponse_received == 1
Y
Analyze the answer
dns_parse_reponse()
querydata == BYNAME
Y
*domain_ip = get_domain_ip
N
Copy get_domain_name to domain_name
strcpy(domain_name, get_domain_name)
RETURN 1

<Fig 3.39: dns_query()>


dns_makequery()
Declare & Initialize Local Variables
char* query = dns_buf;
u_char* domain_tok;
u_int domain_len;
u_int qtype;
u_int qclass;
query_data == BYNAME
Y
qtype = **TYPE_A**
N
qtype = **TYPE_PTR**
Header Section
qclass = **CLASS_IN**
((u_int)query) = dns_id
query += 2
*query++ = MAKE_FLAG0()
*query++ = MAKE_FLAG1()
*((u_int)query) = 1
query = dns_buf + DHDR_SIZE
Question Section
Find the location of . In qname
domain_tok = strchr(qname,'.')
N
domain_tok != NULL
Y
domain_len = strlen(qname)
domain_len = (domain_tok - qname)
domain_len > 63
Y
RETURN 0
N
*query++ = domain_len
memcpy(query,qname,doman_len)
qname += domain_len+1
*query += domain_len
N
domain_tok == NULL
Y
*query++ = \0
((u_int)query) = qtype
query += 2
((u_int)query) = qclass
query +=2
RETURN Query Size

<Fig 3.40: dns_makequery()>

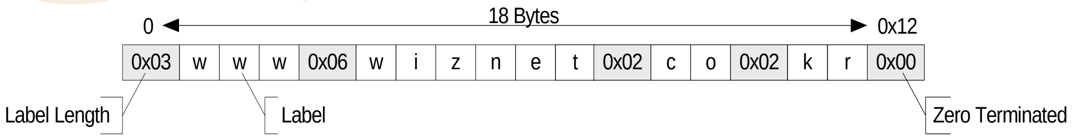
dns_makequery() creates DNS Request message to be sent to DNS Name Server. Since DNS Request Message can query only with Header, Question Section, RRs Sections is not needed to be created. If you examine the header section creation at dns_makequery(), first, it sets ID Field values as any value in DNS 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_IQUREY**	1	/* an inverse query (IQUERY) */
#define OP_STATUS	2	/*a server status request (STATUS)*/
#define **MAKE_FLAG0(qr, op, aa, tc, rd)** (((qr & 0x01) << 7) + ((op & 0x0F) << 3) + ((aa & 0x01) << 2) + ((tc & 0x01) << 1) + (rd & 0x01))		
#define **MAKE_FLAG1(ra, z, rcode)** (((ra & 0x01) << 7) + ((z & 0x07) << 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 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.



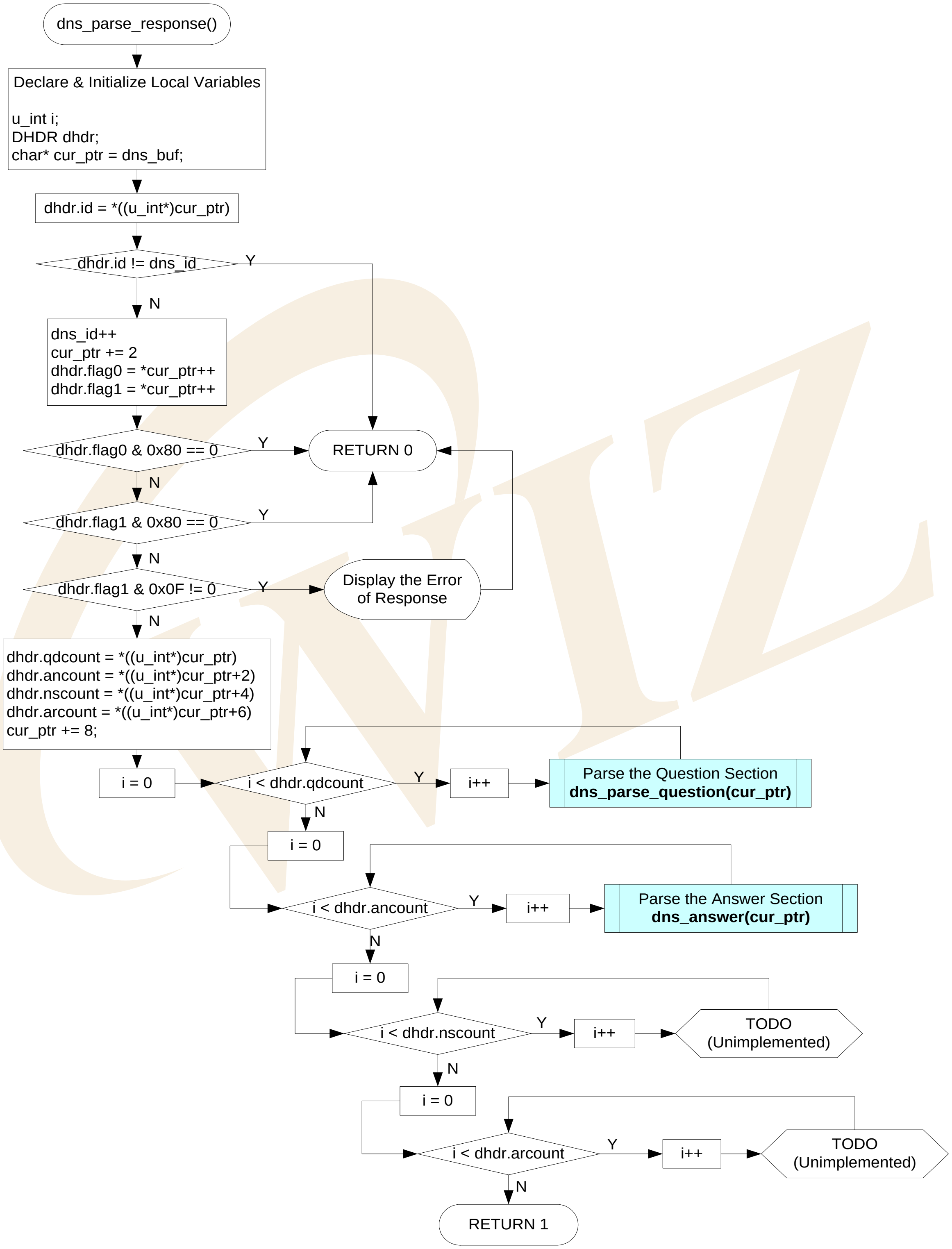
<Fig 3.41: Example of QNAME Field transformation of Question Section >

QTYPE Field of Question Section is set 'TYPE_PTR' when is 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


dns_parse_response()
Declare & Initialize Local Variables
u_int i;
DHDR dhdr;
char* cur_ptr = dns_buf;
dhdr.id = *((u_int*)cur_ptr)
dhdr.id != dns_id
Y
N
dns_id++
cur_ptr += 2
dhdr.flag0 = *cur_ptr++
dhdr.flag1 = *cur_ptr++
dhdr.flag0 & 0x80 == 0
Y
RETURN 0
N
dhdr.flag1 & 0x80 == 0
Y
N
dhdr.flag1 & 0x0F != 0
Y
Display the Error of Response
N
dhdr.qdcount = *((u_int*)cur_ptr)
dhdr.ancount = *((u_int*)cur_ptr+2)
dhdr.nscount = *((u_int*)cur_ptr+4)
dhdr.arcount = *((u_int*)cur_ptr+6)
cur_ptr += 8;
i = 0
i < dhdr.qdcount
Y
i++
Parse the Question Section
dns_parse_question(cur_ptr)
N
i = 0
i < dhdr.ancount
Y
i++
Parse the Answer Section
dns_answer(cur_ptr)
N
i = 0
i < dhdr.nscount
Y
i++
TODO
(Unimplemented)
N
i = 0
i < dhdr.arcount
Y
i++
TODO
(Unimplemented)
N
RETURN 1

<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.

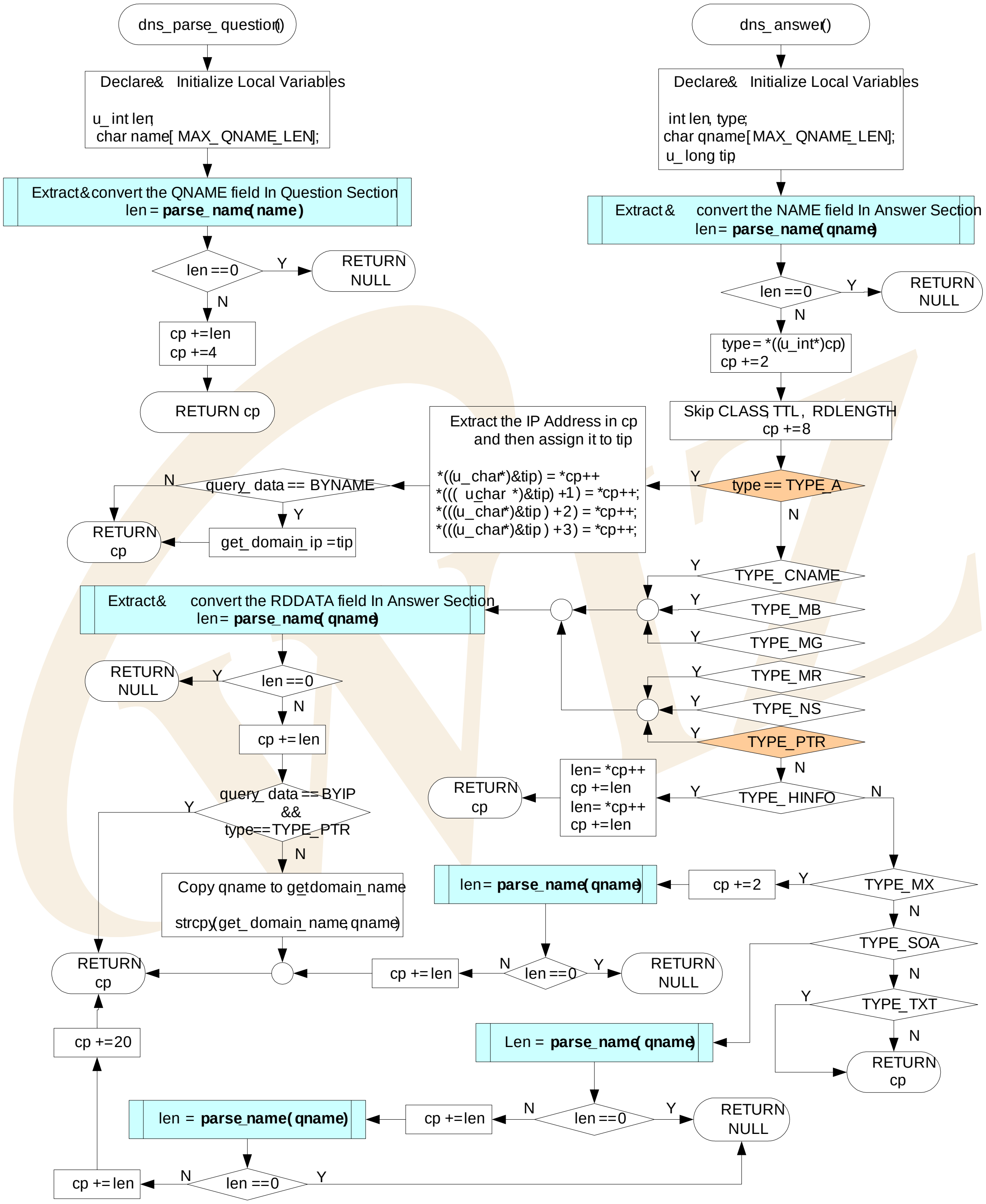
<Table 3-42> is definition of constants that are used in RCODE Field.

<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 then variable length sections such as Question, Answer, Authority, and Additional Section are analyzed. Since the necessary information is set in Answer Section, in this case, it's analyzed and processed up to Answer Section, and other section analysis and process are not performed. If you need information on Authority and Addition 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().



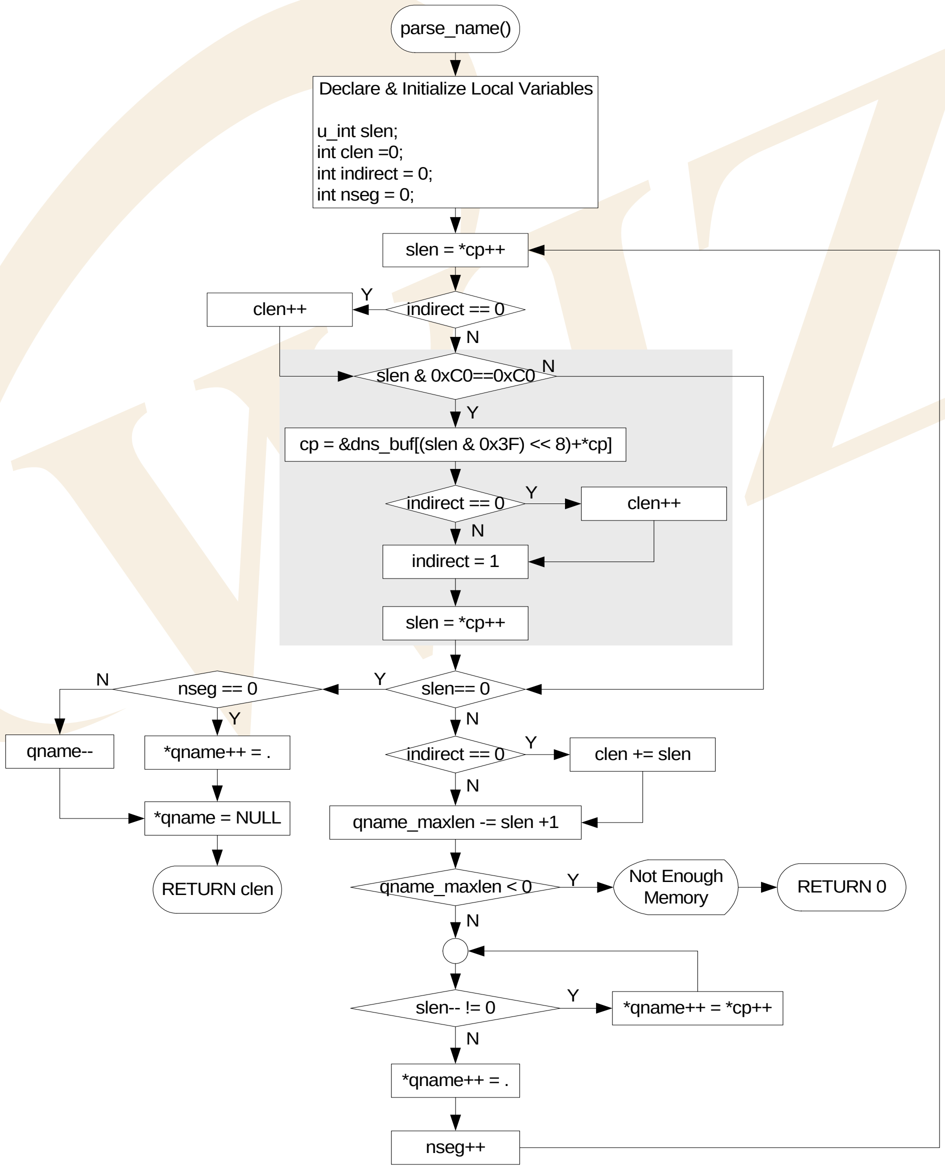
<Fig 3.43: dns_parse_question() & dns_answer()>

dns_parse_question() analyses and processes Question Section. There are 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 Filed> 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().

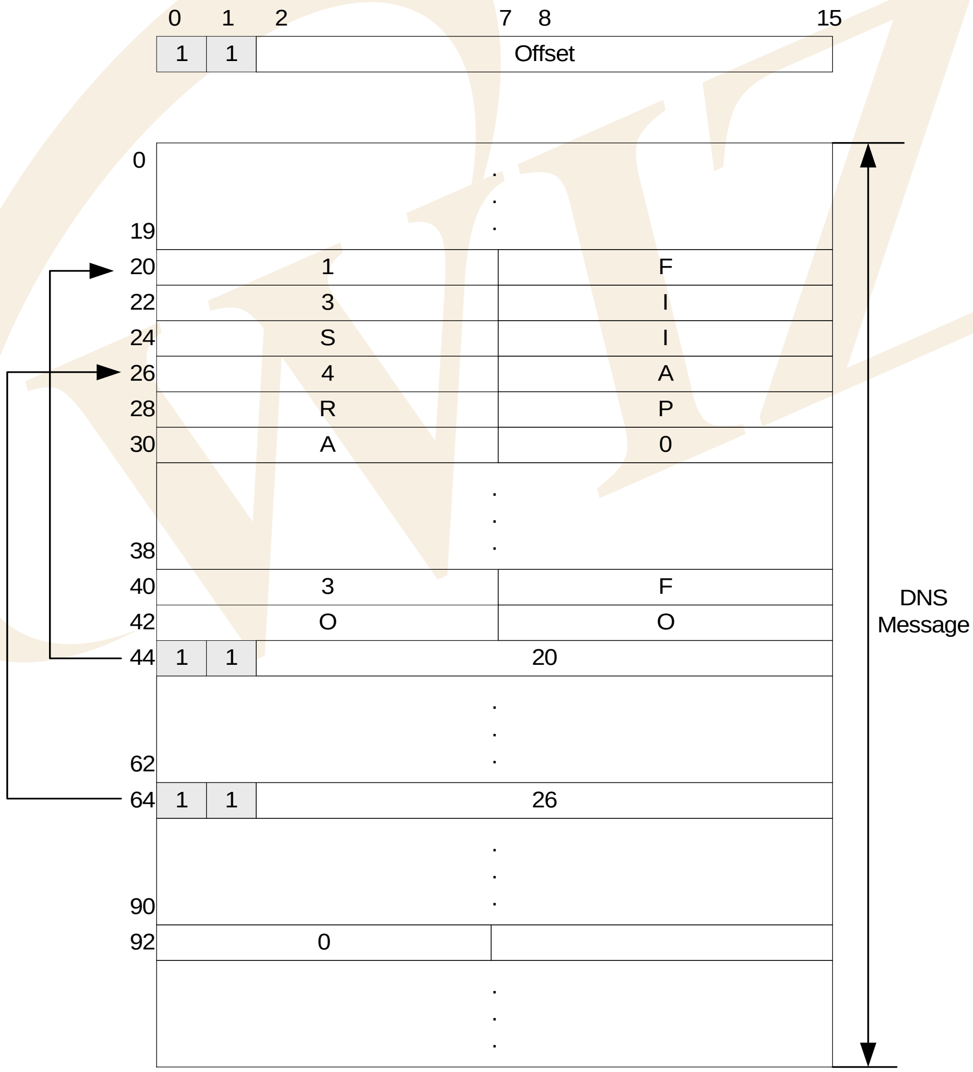


<Fig 3.44: parse_name()>

parse_name() processes QNAME Field of Question Section or NAME, RDDATA Field of RRs Section.

QNAME, NAME, RDDATA 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, that is, the case where 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.

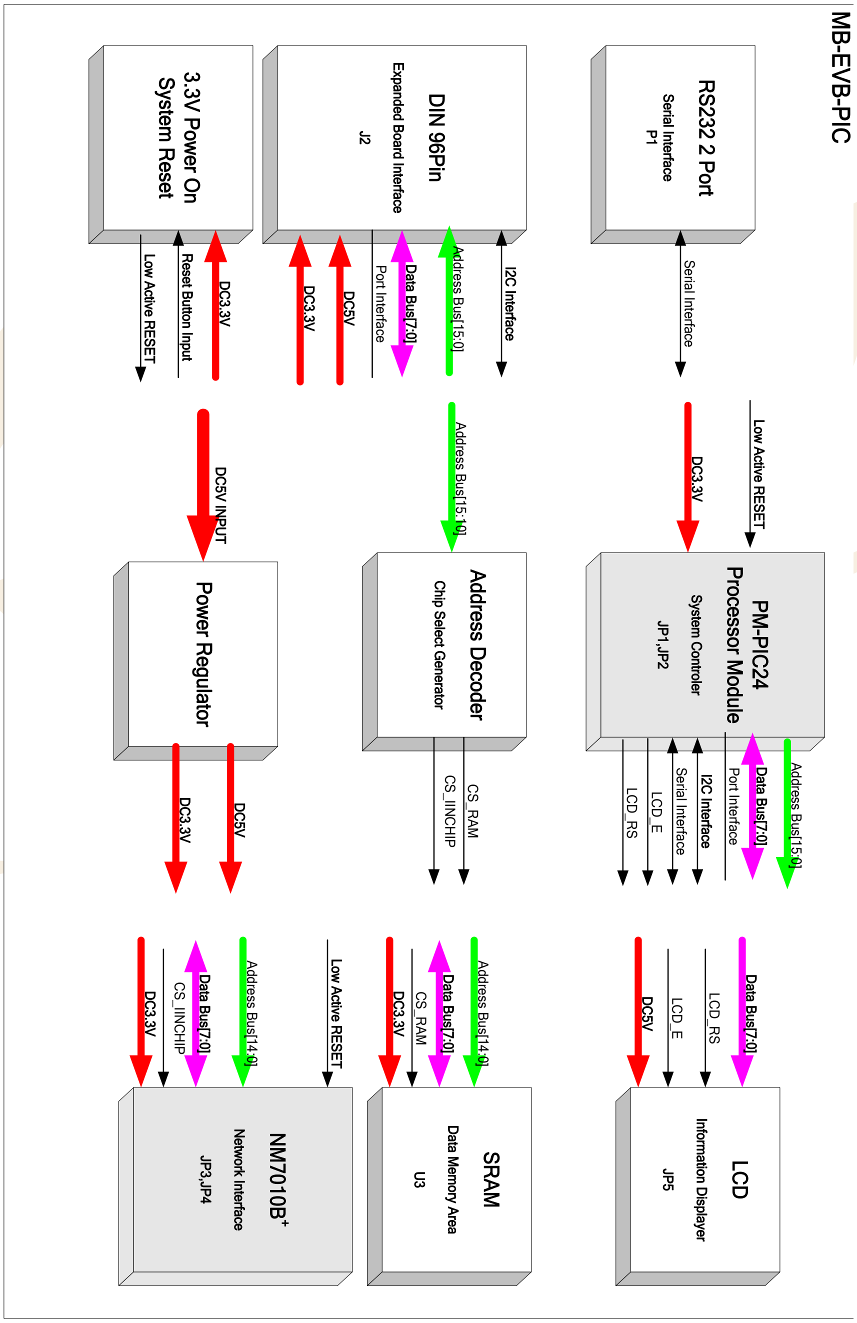
parse_name(), before analysis of Name, checks if upper 2 bits of Label Length Byte are 11, if it's ‘11’ the related Label analyzes the Label at the offset of DNS Message where the Label is located. If it's no ‘11’ then the Label is analyzed and processed like as <Fig 3.41: Example of QNAME Field transformation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

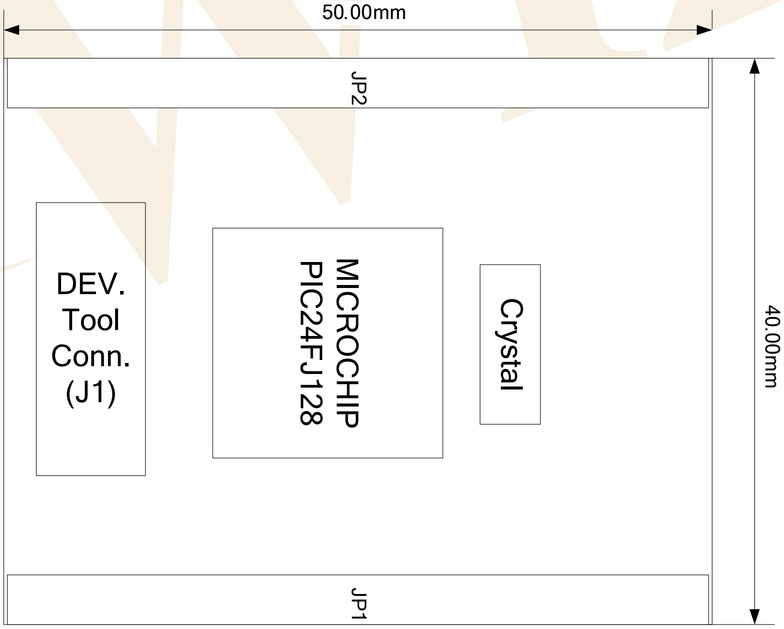
EVB B/D is composed of MB-EVB-PIC(EVB Base Board) and PM-PIC24(PIC MODULE).

Following 9 blocks are components of EVB B/D.

- PM-PIC24
- NM7010B+
- LCD
- SRAM
- RS-232 Port
- Expanded Board Interface
- Power Regulator
- 3.3V Power On System Reset

4.2.1. PM-PIC24

PM-PIC24 is composed of PIC Processor, 8MHz external crystal and header for interfacing to Base board(JP1,JP2), and ICD2(J1) Interface.



<Fig 4.2: PM-PIC24 MODULE Dimension>

For easy development using EVB Board, all the port pin except for OSC(RC12, RC15) are connected to MB-EVB-PIC through module Interface(JP1,JP2). Pin description of Interface is shown in <Table 4-1: PM-PIC24

MODULE Pin Description>.

<Table 4-1: PM-PIC24 MODULE Pin Description>

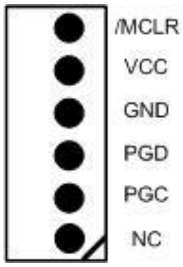
PM-PIC24 MODULE Header #	Pin #	Pin Name	Dir.	Description
JP1	1, 2			3.3V power input
JP1	3	PGC		For development tools
JP1	4,5,6	RB1, RB2, RB3	I/O	Port B
JP1	7	LCD_RS	O	LCD RS signal.
JP1	8	LCD_E	O	LCD E signal.
JP1	9	/IINCHIP_IRQ	I	External interrupt pin
JP1	10, 11, 12	/WR /RD PMBE	O	PMP bus control signal
JP1	13	RB8	I/O	GPIO

JP1	14
	~
	21
		D0(RE0)
		~
		D7(RE7)
			I/O	PMP Data or PortE[0:7]

JP1	25 ~ 31	A9 ~ A15	I/O	PMP Address(alternatively I/O)
JP1	33	/MCLR	I	Reset Signal Input process generated by EVB B/D's Reset Switch
JP1	40, 42	LED0, LED1	O	LED output
JP1	22, 31, 32, 34, 35, 36, 37, 38, 39, 41, 43, 44, 46			Unused
JP1	23, 24, 45, 46, 47, 48, 49, 50	GND		Signal Ground

PM-PIC24 MODULE Header #	Pin #	Pin Name	Dir.	Description
JP2	1, 2			3.3V power input
JP2	6	SCL	O	I2C Clock
JP2	7	SDA	I/O	I2C Data
JP2	8	RF0	I/O	Port F
JP2	9	PGD		For development tools
JP2	10	RF1	I/O	Port F
JP2	11	TX1	O	RS-232 Tx
JP2	13	RX1	I	RS-232 Rx
JP2	14, 17 ~ 24	A0, A1 ~ A8	O	PMP Address(alternatively I/O)
JP2	37 ~ 39, 41 ~ 44	RD0 ~ RD2, RD6 ~ RD9	I/O	Port D
JP2	3 ~ 5, 12, 15, 16, 27 ~ 32, 35, 36, 40, 45 ~ 48			Unused
JP2	33	AVss		AVss
JP2	34	AVdd		AVdd
JP2	25, 26, 49, 50	GND		Signal Ground

PIC ICD2 and ICSP (J1) Pin Mapping



<Table 4-2: ISP Pin Description>

SIGNAL	Pin Number	Description
/MCLR	1	Reset
VCC	2	VCC
GND	3	Ground
PGD	4	Programming Data
PGC	5	Programming Clock
NC	6	Not connected

PIC ICD2 connector(J2) Pin map is same as above. Please refer to Chapter 9.3 in MPLAB ICD2 User's Guide.

4.2.2. NM7010B+ MODULE

NM7010B+ is a network module that is composed of W3150A+(TCP/IP hardwired chip), RTL8201CP(Ethernet PHY), and MAG-JACK(RJ45 with X'FMR).

TCP/IP,MAC protocol layer : W3150A+
Physical layer : Ethernet PHY
Connector : MAG-JACK

For details on NM7010B+ MODULE, refer to "**NM7010B+ Datasheet Vx.x.pdf**"

4.2.3. LCD

LCD is used for debugging and system status display.
Pin Description of LCD Interface (JP5) 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	LCD_RS / RS	I	Data/Instruction register select
5	GND / RW	I	Read/Write
6	LDC_E / E	I	Enable signal,start data read/write
7 ~ 14	RE0 / DB0 ~ RE7 / DB7	I/O	Data Bus Line
15	NC1 / LED A	O	LED Anode, power supply+
16	NC2 / LED K	O	LED Cathode,ground 0V

4.2.4. SRAM

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

4.2.5. RS232 Port

It's a interface for Serial USARTs that is supported by PIC.

EVB B/D uses 9Pin DSUB male Type connector.

(PM-PIC24 use only one RS232 port.)

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 PIC24, power and many reserved pin are connected to Expanded Board Interface.

<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	PMP Address[0:15]
77,45,78,46 79,47,80,48	D0, D1, D2, D3, D4, D5, D6, D7	I/O	PMP Data Bus[0:7]
53 76 86	/RD PMBE /WR	O	PMP Bus Read Strobe PMP BUS Enable PMP Bus Write Strobe
81 49	SDA SCL	I/O O	I2C Bus Data Line I2C Bus Clock Line
Port Interface			
10 11 12	RB1 RB2 RB3	I/O	Port B
18 19 20 22	RD0 RD1 RD2 RD6	I/O	Port D

1 84 85	RD7 RD8 RD9		
90	RF1	I/O	Port F
ETC			
4 5	LED1 LED2	O	LED pin
13 14	LCD_RS LCD_E	O O	LCD RS signal LCD E signal
15 16	PGC PGD		Programming Clock Programming Data
23	/IINCHIP_IRQ	I	External interrupt 0
33	/MCLR	I	Reset signal
91 92	TX1 RX1	O I	RS-232 signals
2, 3, 6, 17, 21, 25, 26, 27, 28, 29, 43, 50, 51, 52, 54, 56, 57, 58, 59, 60, 61, 75, 83, 87, 89, 93, 94			Not Available(Do not use)
Power Interface			
31,32	5V	O	5V Power Supply
63,64	3.3V	O	3.3V Power Supply
7	AVdd		AVdd
8	AVss		AVss
9,24,30,44, 55,62,65,72, 82,88	GND		Ground

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. ICD2 Port

It's a interface for ICSP & ICD that is supported by MICROCHIP.

EVB B/D uses 6Pin RJ11 connector. Please refer to Chapter 9.3 in MPLAB ICD2 User's Guide.

4.2.8. 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 LM1117MPX-3.3(U2).

4.2.9. 3.3V Power On System Reset

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

4.3. Schematic

4.3.1. MB-EVB-PIC

Please refer to “**MB-EVB-PIC_Schematics.PDF**” in the CD.

4.3.2. PM-PIC24

Please refer to “**PM-PIC24_Schematics.PDF**” in the CD.

4.3.3. NM7010B+

Please refer to “**NM7010B+_Schematics.PDF**” in the CD.

4.4. Parts List

4.4.1. MB-EVB-PIC Parts List

Please refer to "**MB-EVB-PIC_Partlist.PDF**" in the CD.

4.4.2. PM-PIC24 Parts List

Please refer to "**PM-PIC24_Partlist.PDF**" in the CD.

4.4.3. NM7010B$^+$

Please refer to "**NM7010B+_Partlist.PDF**" in the CD.

4.5. Physical Specification

4.5.1. 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	-	210	-	mA
3.3V	-	110	-	mA

Total Power consumption is 210mA X 5V = 1.5 Watt.