Micriμm

μC/DHCPc

Dynamic Host Configuration Protocol (Client)

User's Manual

www.Micrium.com

Table of Contents

1.00 Introduction

DHCP is a protocol designed to enable clients to get IP configuration from a centralized database. DHCP protocol haves slightly evolved over the years from the BOOTP protocol, initially designed to enable diskless clients to boot from the network. The µC/DHCPc module implements a part of RFC 2131 and RFC 2132 (`ftp://ftp.rfc-editor.org/in-notes/rfc213{1,2}.txt`).

This document describes how to configure and use the µC/DHCPc module in a µC/TCP-IP and µC/OS-II environment.

We used the Cogent Computer CSB335 embedded computer and IAR's Embedded Workbench to demonstrate the examples but other embedded platforms and tool chains can be used.

1.01 Directories and Files

The code and documentation of the µC/DHCPc module are placed in a directory structure according to "AN-2002, µC/OS-II Directory Structure". Specifically, the files are placed in the following directories:

`\Micrium\Software\uC-DHCPc\Doc`

This directory contains the µC/DHCPc documentation files, including this one.

`\Micrium\Software\uC-DHCPc\Source`

This directory contains the µC/DHCPc source files (`dhcp-c.c` and `dhcp-c.h`).

`\Micrium\Software\EvalBoards\Cogent\CSB335\IAR\BSP`

This directory contains the Board Support Package (BSP) for the CSB335. This directory contains:

```
bsp.c
bsp.h
CSB33x_lnk_ram.xcl
net_bsp.c
net_bsp.h
net_isr.c
```

- `bsp.*` contains the board-dependant code for µC/OS-II and other board specific functions;
- `CSB33x_lnk_ram.xcl` is the IAR Embedded Workbench linker configuration file (memory map).
- `net_bsp.*` contains the board-dependant code for µC/TCP-IP NIC module;
- `net_isr.c` is the board-dependant code for µC/TCP-IP NIC module ISR manager;

\Micrium\Software\EvalBoards\Cogent\CSB335\IAR\uC-DHCPc\Ex1

This directory is the directory that contains the source code for Example #1 running on a CSB335 card with IAR tools. This directory contains:

```
app.c
app_cfg.h
Ex1.*
includes.h
net_cfg.h
os_cfg.h
```

- `app.c` contains the test code;
- `app_cfg.h` is the example/application configuration file;
- `Ex1.*` are the IAR Embedded Workbench project files;
- `includes.h` contains a master include file used by the application;
- `net_cfg.h` is the µC/TCP-IP configuration file;
- `os_cfg.h` is the µC/OS-II configuration file;

2.00 Test Code

As mentioned in the previous section, the test code for this board is found in the following directory and will be briefly described:

\Micrium\Software\EvalBoards\Cogent\CSB335\IAR\uC-DHCPc\Ex1

2.01 Test Code, app.c

This file contains the application code for example #1. `app.c` is written to demonstrate the capabilities of the µC/DHCPc module. The code doesn't really do anything useful except initializing µC/OS-II, µC/TCP-IP and µC/DHCPc, and create a few tasks and other kernel objects that will inform you about the state of the system.

Note that some lines of the listings have been removed to help you focus on the µC/DHCPc module usage.

Listing 2-1, AppStartTask()

```
void  AppInit_TCPIP (void)
{
    NET_ERR      err;
    CPU_INT08U  *buf;

    APP_DEBUG_TRACE("Initialize TCP/IP stack...\n");
    buf = (CPU_INT08U *)mon_getenv("ETHERADD");                          (1)
    NetASCII_Str_to_MAC(buf, NetIF_MAC_Addr, &err);

    err = Net_Init();                                                    (2)
    if (err != NET_ERR_NONE) {
        APP_DEBUG_TRACE("Net_Init() failed: error #%d, line #%d.\n", err, __LINE__);
        while (DEF_YES) {
            ;
        }
    }

    APP_DEBUG_TRACE("    IP  = 0.0.0.0\n");
    APP_DEBUG_TRACE("    MAC = %s\n", buf);

    NetIP_CfgAddrThisHost(NET_IP_ADDR_THIS_HOST, NET_IP_ADDR_NONE);      (3)
    NetIP_CfgAddrDfltGateway(NET_IP_ADDR_NONE);
}

void  AppInit_DHCPc (void)
{
    NET_ERR      err;
    CPU_INT08U  *opt;
    CPU_INT08U   buf[16];
    CPU_INT08U   param_req_list[] = {DHCP_OPT_HOST_NAME,                 (4)
                                     DHCP_OPT_DOMAIN_NAME,
                                     DHCP_OPT_DOMAIN_NAME_SERVER,
                                     DHCP_OPT_TIME_SERVER,
                                     DHCP_OPT_TIME_OFFSET};

    DHCPc_SetMacAddr(NetIF_MAC_Addr);                                    (5)
    DHCPc_SetClientID(0x01, NetIF_MAC_Addr, NET_IF_ADDR_SIZE);           (6)
    DHCPc_SetVendorClassID("YourCompany", 11);                           (7)
    DHCPc_SetParamRequestList(param_req_list,                            (8)
                sizeof(param_req_list) / sizeof(param_req_list[0]));
```

```
    APP_DEBUG_TRACE("Get DHCP lease... ");
    err = DHCPc_Start();                                                (9)
    if (err != DHCPc_ERR_NONE) {                                        (10)
        APP_DEBUG_TRACE("fail!\n");
        while (DEF_YES)              /* Can't obtain a DHCP lease: nothing to do!       */
    }


    APP_DEBUG_TRACE("success!\n");

    DHCPc_CfgStack();                                                   (11)

    DHCPc_Print(DHCPc_GetHdr());                                        (12)

                                     /* Must Mem_Copy because header may be misaligned.  */
    Mem_Copy (&AppIPAddr,
              &(DHCPc_GetHdr()->yiaddr),
               sizeof(DHCPc_GetHdr()->yiaddr));

    opt = DHCPc_GetOpt(DHCP_OPT_SUBNET_MASK);                           (13)
    if (opt != (void *)0) {
        Mem_Copy ((void *)&AppIPMsk,    (void *)(opt + 2), *(CPU_INT08U *)(opt + 1));
    }

    opt = DHCPc_GetOpt(DHCP_OPT_ROUTER);
    if (opt != (void *)0) {
        Mem_Copy ((void *)&AppIPGw,     (void *)(opt + 2), *(CPU_INT08U *)(opt + 1));
    }

    opt = DHCPc_GetOpt(DHCP_OPT_DOMAIN_NAME_SERVER);
    if (opt != (void *)0) {
        Mem_Copy ((void *)&AppIPDnsSrv, (void *)(opt + 2), *(CPU_INT08U *)(opt + 1));
    }

    NetASCII_IP_to_Str(NET_UTIL_NET_TO_HOST_32(AppIPAddr),   buf, DEF_NO, &err);
    APP_DEBUG_TRACE("DHCP IP address : %s\n", buf);

    NetASCII_IP_to_Str(NET_UTIL_NET_TO_HOST_32(AppIPMsk),    buf, DEF_NO, &err);
    APP_DEBUG_TRACE("DHCP IP mask    : %s\n", buf);

    NetASCII_IP_to_Str(NET_UTIL_NET_TO_HOST_32(AppIPGw),     buf, DEF_NO, &err);
    APP_DEBUG_TRACE("DHCP IP gateway : %s\n", buf);

    NetASCII_IP_to_Str(NET_UTIL_NET_TO_HOST_32(AppIPDnsSrv), buf, DEF_NO, &err);
    APP_DEBUG_TRACE("DHCP IP dns     : %s\n", buf);
}
```

L2-1(1) Obtain the MAC address. Ask the monitor to obtain the MAC address. You have to set the #define LAN91C111_CFG_MAC_ADDR_SEL to LAN91C111_MAC_ADDR_SEL_CFG into app_cfg.h.

L2-1(2) Initialization of µC/TCP-IP stack.

L2-1(3) Configuration of the stack with generic values. DHCP protocol REQUIRES that you use IP / Mask address 0.0.0.0 during lease negotiation.

L2-1(4) This is an example of a parameter request list. If you use µC/DNSc or µC/SNTPc, you may use this example to obtain DNS or SNTP configuration dynamically via DHCP.

L2-1(5) Provision of µC/DHCPc with the MAC address.

L2-1(6) Provision of µC/DHCPc with the Client ID. As an example, we use the MAC address as the Client ID. The Client ID can be any type of information.

L2-1(7) Provision of µC/DHCPc with the Vendor Class ID. As an example, we have set the value “Micriμm”, you can use your own organization name if needed.

L2-1(8) Provision of µC/DHCPc with the Parameter Request List. The µC/DHCPc module will request these options to the server.

L2-1(9) Start DHCP negotiation.

L2-1(10) Check for successful negotiation. On failure, you can restart the process of bail out. You cannot assign yourself an IP address which is in the DHCP server scope because it may create IP address duplications.

L2-1(11) Configure µC/TCP-IP stack with the IP configuration got from server. IP address, mask and gateway are configured. Any other options are to be configured by the application.

L2-1(12) Print DHCP ACK packet header for debugging purposes.

L2-1(13) Obtain the value of the subnet mask option. See section 3.01 for explanations about how to fetch options using the `DHCPc_GetOpt()` function.

3.00 µC/DHCPc module

DHCP stands for Dynamic Host Configuration Protocol. The 'c' means 'client'. Other modules ends by 's', which means 'server'. The files are located at:

```
\Micrium\Software\uC-DHCPc\Source
```

These files are:

```
dhcp-c.h
dhcp-c.c
```

3.01 µC/DHCPc module use

The µC/DHCPc module relies on the µC/TCP-IP stack to work. As you can see in the example file (see section 2.01), the configuration of µC/TCP-IP stack is different with the use of DHCP than by its own.

µC/TCP-IP with µC/DHCPc configuration:

- The stack must be configured initially with generic IP, mask and gateway;
- The µC/DHCPc module must be configured (see section 3.02);
- The DHCP negotiation process is started (call `DHCPc_Start()`);
- On success, the `DHCPc_CfgStack()` is called by the application to have the IP, mask and gateway set with the values obtained from the DHCP server. The usage of `DHCPc_CfgStack()` is optional. The user may choose to configure the stack directly in the application as opposed to call this function because there might be other parameters that you need to set.
- The application may use the `DHCPc_GetHdr()` and the `DHCPc_GetOpt()` functions to obtain more information about the lease obtained. The `DHCPc_GetHdr()` return the DHCP ACK packet header and the `DHCPc_GetOpt()` returns a pointer to the option entry in the DHCP ACK packet. If the option is not found, the `NULL` value is returned.

DHCP header format:
The DHCP header format is defined in `dhcp-c.h`, line 280, from the RFC 2131, p. 9.

DHCP option entry format:
The format of the option entry depends of the option itself.

- The first byte is the value of the option itself.
- The second byte is the length of the option data. For your convenience, in the 'DHCP options' section of the file `dhcp-c.c`, we have defined all DHCP options defined in RFC 2132, and in the comments you will find the valid length of each option.
- Starting from third byte, the format depends of the option. Many options contain one or more IP addresses. Their length byte will be a multiple of 4 and their data will be the IP addresses bytes, in network order.

DHCP lease negotiation:
Here is a description of what happens when you call the `DHCPc_Start()` function.

- An UDP socket is opened and bound to IP address 0.0.0.0, port 68. This enable to receive packet from any host on port 68 (DHCP client).

- A DHCP DISCOVER packet is built using your MAC address, Client ID and Vendor Class ID. There is also a field called “Transaction ID” which is a counter incremented for each lease negotiation and helps discard packets from old negotiations or negotiations from other clients.
- The DHCP DISCOVER packet is broadcasted on your LAN, port 67 (DHCP server). Generally, only one server will reply, but more servers can respond on complex configurations. The packet can also be relayed outside of your LAN by DHCP/BOOTP relay agents.
- The first server to answer is selected by µC/DHCPc module, assuming that the first is the nearer. This is the DHCP OFFER packet.
- In the case of no or bad server reply, a new DHCP DISCOVER packet will be send after 4, 8, 16, 32 and 64 seconds. After the fifth retry, the `DHCP_Start()` function will return with an error.
- A DHCP REQUEST packet is built using your MAC address, Client ID, Vendor Class ID, Transaction ID, and an option to specify that we request a lease to the server we have selected previously.
- In the DHCP REQUEST packet, there is the parameter list, which includes all the options we want from the DHCP server. By default, µC/DHCPc requests subnet mask and gateway. The application can request more options by using the `DHCPc_SetParamRequestList()` function.
- The DHCP REQUEST packet is broadcasted on your LAN, port 67 (DHCP server). This time, only the selected server will reply. This is the DHCP ACK packet.
- In the case of no or bad server reply, a new DHCP DISCOVER packet will be send after 4, 8, 16, 32 and 64 seconds. After the fifth retry, the `DHCP_Start()` function will return with an error.
- The lease is acquired and the `DHCP_Start()` returns with no error.

DHCP broadcast vs unicast messages:
Some implementations of DHCP servers send DHCP OFFER and DHCP ACK messages in unicast packets addressed to the newly assigned IP address. The µC/TCP-IP package doesn’t support this behavior since it can accept packets addressed to the *current* IP address or 0.0.0.0, but not the newly assigned IP address. When µC/DHCPc is used over µC/TCP-IP, the compile-time variable `DHCP_BCAST_BIT` MUST set to `DEF_ENABLED` to make the system work properly.

3.02 µC/DHCPc module configuration

There is some configuration that you need to provide to the µC/DHCPc module before to start the IP lease negotiation.

- MAC address (mandatory)
- Client ID (optional)
- Vendor Class ID (optional)
- Parameter Request List (optional)

MAC address: The DHCP protocol requires the MAC address of the client to index the configuration information in the server database. Note that recent DHCP servers can use the Client ID instead of the MAC address to index database. In this case, the MAC address is still mandatory.

Client ID: In an environment with multiple physical layers supporting the IP network, it may be inconsistent to mix MAC addresses with other address spaces to index server database. For this reason, recent DHCP servers can use the Client ID instead of hardware address. Client ID must be unique in the subnet. It can represent a MAC address (type = 1), other hardware addresses or anything else (type = 0).

Vendor Class ID: The vendor class ID can be used instruct server to send special configuration / options to clients sold by a single vendor. Some vendor units may need configuration which is not defined in the

DHCP standard, using option #43 (vendor-specific information) or options with values greater than 128 (site-specific options).

Parameter Request List: A list of options requested to the DHCP server. By default, µC/DHCPc requests the subnet mask and gateway. You can request any other option you need in your application. When the server replies, you can get the option value(s) using the `DHCPc_GetOpt()` function. Note that the DHCP server may not have a value for all options. In that case, the `DHCPc_GetOpt()` function returns a `NULL` pointer.

3.03 µC/DHCPc module limitations

- This DHCP client implements a part of RFC 2131.
- This DHCP client supports infinite DHCP leases only.

To avoid problems because two or more devices on your subnet may obtain the same IP address, you MUST configure your DHCP server to give infinite leases to clients using the µC/DHCPc client module.

Infinite DHCP leases help you obtain more predictability in your application. After the inherent uncertainty of the lease negotiation at the application startup, there are no more risks about IP configuration for the rest of the application's life. With DHCP finite leases, you have to re-negotiate IP configuration at regular interval of time, creating periodic uncertainty.

References

µC/OS-II, The Real-Time Kernel, 2nd Edition
Jean J. Labrosse
R&D Technical Books, 2002
ISBN 1-57820-103-9

Embedded Systems Building Blocks
Jean J. Labrosse
R&D Technical Books, 2000
ISBN 0-87930-604-1

Contacts

CMP Books, Inc.
1601 W. 23rd St., Suite 200
Lawrence, KS 66046-9950
USA
+1 785 841 1631
+1 785 841 2624 (FAX)
e-mail: rushorders@cmpbooks.com
WEB: http://www.cmpbooks.com

Cogent Computer Systems, Inc.
1130 Ten Rod Road, Suite A-201
North Kingstown, RI 02852 USA
USA
+1 401 295 6505
+1 401 295 6507 (Fax)
WEB: www.CogComp.com

IAR Systems
Century Plaza
1065 E. Hillsdale Blvd
Foster City, CA 94404
USA
+1 650 287 4250
+1 650 287 4253 (FAX)
e-mail: Info@IAR.com
WEB : www.IAR.com

Micriμm
949 Crestview Circle
Weston, FL 33327
USA
+1 954 217 2036
+1 954 217 2037 (FAX)
e-mail: Jean.Labrosse@Micrium.com
WEB: www.Micrium.com

Validated Software
Lafayette Business Park
2590 Trailridge Drive East, Suite 102
Lafayette, CO 80026
USA
+1 303 531 5290
+1 720 890 4700 (FAX)
e-mail: Sales@ValidatedSoftware.com
WEB: www.ValidatedSoftware.com