INGI 2315 - Group INFO 4

Project report :
DHCP Relay on PIC 18F97J60

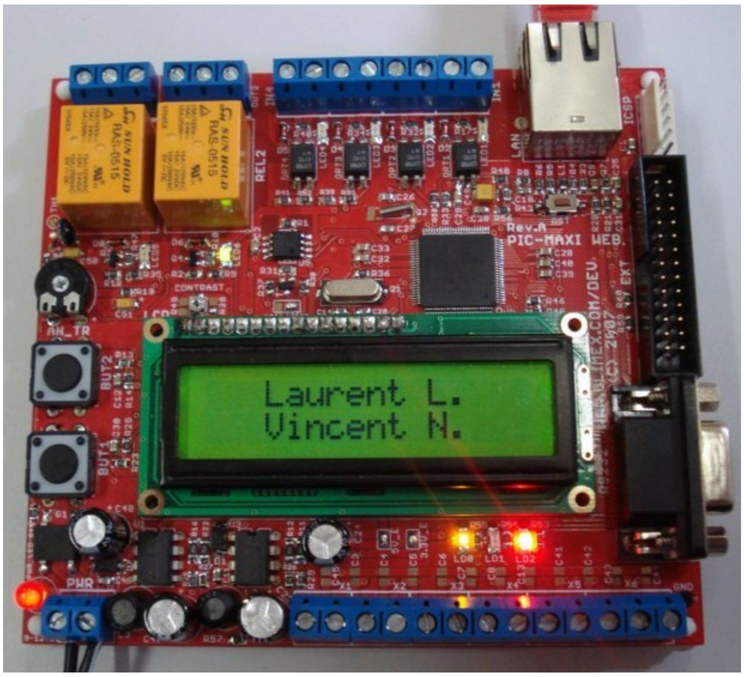
Laurent Lamouline - 3597-05-00
Vincent Nuttin - 5772-05-00

May 21, 2010

Contents

Introduction

For this project, we were asked to implement a DHCP Relay on a PIC 18F97J60 [1].



This report presents the result of our work about that. First, we will explain a little bit how DHCP works, the protocol in general and its behaviour when a relay is present in a network. After that, we will present our model done in ASG and explain why it has been useful during the implementation phase. In addition to that, we will continue with our implementation in C on the PIC and some explanations about the transition between ASG and the C code. Finally, there will be a small description of our working environment and some illustrations of the application running on our PIC.

The appendix of the report contains the source code of the most important file : `DCHPr.c`. Some other pieces of code are also presented throughout this report.

We wish you a pleasant reading.

[1] `http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en026439`

1 DHCP : How does it work ?

1.1 The protocol

DHCP is a protocol used by hosts (DHCP clients) to retrieve IP address assignments. It uses a client-server architecture. DHCP uses the same two ports assigned by IANA for BOOTP: 67/udp for sending data to the server, and 68/udp for data to the client. You can see a typical DHCP packet (with each field) in the figure just below (Figure 1).

UDP Src=0.0.0.0 sPort=68
Dest=255.255.255.255 dPort=67

OP	HTYPE	HLEN	HOPS
0x01	0x01	0x06	0x00
XID			
0x3903F326			
SECS		FLAGS	
0x0000		0x0000	
CIADDR			
0x00000000			
YIADDR			
0x00000000			
SIADDR			
0x00000000			
GIADDR			
0x00000000			
CHADDR			
0x00053C04			
0x8D590000			
0x00000000			
0x00000000			
192 octets of 0's. BOOTP legacy			
Magic Cookie			
0x63825363			
DHCP Options			
DHCP option 53: DHCP Discover			
DHCP option 50: 192.168.1.100 requested			
DHCP option 55: Parameter Request List: Request Subnet Mask (1), Router (3), Domain Name (15), Domain Name Server (6)			

Figure 1: DHCP Discovery

DHCP process is divided in four operations :

- IP discovery : The client broadcasts messages to discover available DHCP servers. A DHCP client can also request its last-known IP address.
- IP lease offer : When a DHCP server receives an IP discovery from a client, it reserves an IP address for the client and send a `DHCP OFFER` message to this client. This message contains the client's MAC address, the IP address that the server is offering, the subnet mask, the lease duration, and the IP address of the DHCP server making the offer. The proposed IP address is specified in the YIADDR (Your IP Address) field.
- IP request : A client can receive DHCP offers from multiple servers, but it will accept only one DHCP offer and broadcast a DHCP request message. Based on the Transaction ID field in the request, servers are informed whose offer the client has accepted. When other DHCP servers receive this message, they withdraw any offers that they might have made to the client and return the offered address to the pool of available addresses. DHCP request message is broadcast because the DHCP client has still not received any IP and hence it cannot unicast the request.
- IP lease acknowledgement : When the DHCP server receives the `DHCP REQUEST` message from the client, the configuration process enters its final

phase. The acknowledgement phase involves sending a `DHCP ACK` packet to the client. This packet includes the lease duration and any other configuration information that the client might have requested. At this point, the IP configuration process is completed [2].

The DHCP protocol also provide some options. Each option has a specific length. In this project, we used some of them but especially "Message Type (53)" and "Requested IP (50)".

The aim of the section is not to describe DHCP deeply so for further information, we recommend you to read the following websites :

- `http://www.networksorcery.com/enp/protocol/dhcp.htm`
- `http://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol`

1.2 With a relay

A DHCP relay is an host configured with a static IP address and knows the DHCP server address. It listens to the port 67 for clients requests and to the port 68 for server answers. Its role is to forward packets coming from the client (resp. server) to the server (resp. client). The main difference is that when a client makes a DHCP discovery in broadcast, it is intercepted by the relay. The relay knows the IP address of the server but it still needs to make an ARP request to know its MAC address to be able to forward the packet in unicast to this latter. When it forwards the packet, it adds its own IP address in the `GIADDR` field. The server can then send a `DHCP OFFER` with a proposal for the client, in unicast, to the relay. When the client has received the offer, it then sends in broadcast a DHCP request with the IP it chose. This packet is intercepted by the DHCP relay anew so that it can send it to the server in unicast. Finally, the server sends the DHCP acknowledgement in unicast to the relay. A client always communicates in broadcast with the relay and the relay sends its messages in unicast to the server. In the next section, you will see how all these components are connected to each other.

[2] `http://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol`

2 Description of our model (ASG)

In order to have a working base, we build an ASG (Asynchronous State Graph) model. This model represents the state machine we will use in our relay. To design this model, we use the concepts of parallelism, transitions with and without condition, and rendez-vous.

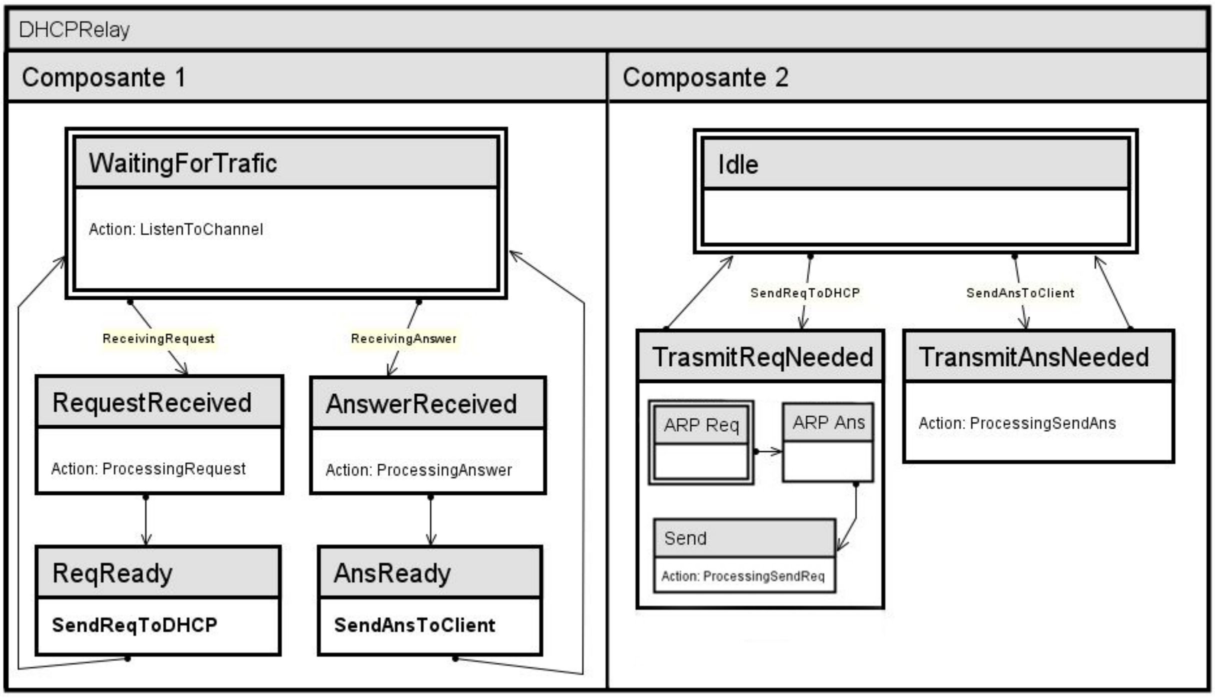


Figure 2: ASG Model

As you can see, there are two main parts. The first component (on the left) is the part that processes the packets received on the Ethernet buffer. The first state waits for incoming traffic. When a packet arrives, we look at the fields (`OP`) (shown in Figure 1) to determine whether the packet is coming from a (new) client or from the server. If the packet is identified as coming from a client, we forward it to the server in unicast mode. On the other hand, if the packet is identified as coming from the server, we forward it to the related client thank to its MAC address in the `CHADDR` field. These two conditions bring us to new states which lead to some "rendez-vous" points with the other part of the model. They are called "SendReqToDHCP" and "SendAnsToClient". As you can see, theses "rendez-vous" are on both components (on the left: inside a state; on the right: as condition on transition). When the left side's process is inside one of these states (bottom most), the transition on the right part is triggered and the execution can continue.

The second part (on the right) is the one that sends the packets to some destination (in broadcast mode when we want to contact a client; in unicast mode when we contact the server). When we have to contact the server, we don't directly have its MAC address so we need to perform an `ARP` request.

On both sides, you can observe unconditional transitions from the bottom to the top. These transitions are there to ensure the state machine to run as long as there are packets to process.

3 Implementation on the PIC

In this part, we will explain how we have translated our ASG model into C code, how we have implemented some stuff like state machines, rendez-vous, etc.

- The main method is located in `MainDemo.c`. First, this method initialises the board. Then, it contains an infinite loop :

Listing 1: MainDemo.c

```
while(1){
  ...
  // This tasks invokes each of the core stack application tasks
  StackApplications(); // LV : Triggering DHCPRelayTask()
  ...
} //end of while(1)
```

- Then, in `StackApplications()`, we use the constants predefined for a DHCP server on the PIC to activate our relay. Thank to this, we are sure that we have all the stuff (sockets, data-structures, ...) to perform this properly.

Listing 2: StackTsk.c

```
#if defined(STACK_USE_DHCP_SERVER)
//DHCPServerTask(); // LV : Using relay instead of server !
DHCPRelayTask(); // Our task ! :-)
#endif
```

- Finally, we implemented `DHCPRelayTask()` in a new file called `DHCPr.c` (see code in Appendix B).

Cooperative multitasking loop scheduling

As we saw in `INGI 2315`, the cooperative multitasking loop scheduling can be implemented with an infinite loop including a series of calls to functions corresponding to the different tasks. It is exactly what is done in the previous files we talked about (Listings 1 and 2).

State machines

Concerning the state machines, we implemented them with some `switch/case` operations. Here is a small piece of code to show you how it works :

Listing 3: DHCPr.c

```
switch(SMState){
  case SM_IDLE:
    break;

  case SM_ARP_SEND_QUERY:
    ...
    SMState = SM_ARP_GET_RESPONSE;
    break;

  case SM_ARP_GET_RESPONSE:
    ...
    SMState = SM_MESS_SEND;
    // No break;

  case SM_MESS_SEND:
    ...
    SMState = SM_IDLE;
    break;

  default:
    return;
}
```

Rendez-vous

The "rendez-vous" is supposed to be implemented by another infinite loop in which a condition is checked to know if the "rendez-vous" point is reached or not. In our case, we had to integrate our concept into an already existing code, so instead of launching another separate task, we choose to trigger the send by a simple method call (`UDPFlush()`). This solution seems to be easier and as effective as the "rendez-vous". We could have done this with an infinite loop but this loop would have directly called `UDPFlush()`. By this little explanation, we mean we felt it was more efficient to call this method without spending resources to new (little and not mandatory) task.

4 In practice

4.1 Our working environment

In this part of the report, we want to briefly introduce our working environment and how you can use our relay. You must have the four following things :

- a PIC 18F97J60,
- a router (type TrendNet TW100-S4WW1CA),
- a DHCP server,
- a client (any computer).

On the PIC, you have to install our relay software. On the router, you have to disable the intern DHCP server (enable by default). After that, you have to configure a virtual server on the LAN. More specifically, you have to say to the router that every `UDP` packet received on its WAN interface going to the port 67 will be redirected to `192.168.8.2`, the static IP address of the PIC. Finally, you have to configure the router's IP addresses as on the figure below (Figure 3).

On the DHCP server, you only have to configure its IP address (Figure 3).

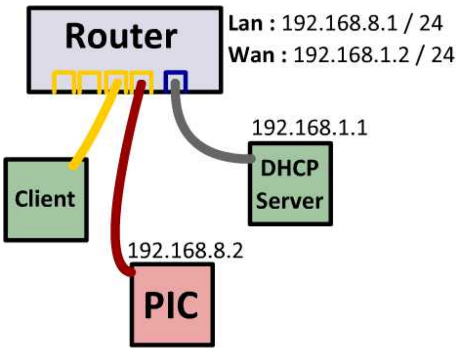
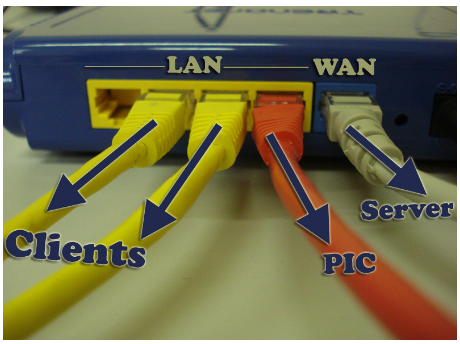


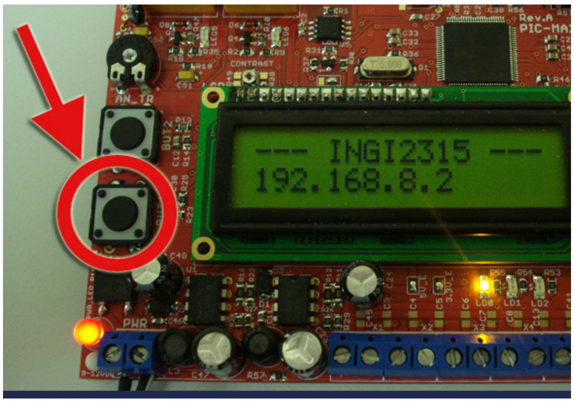
Figure 3: IP addresses

Thank to this configuration, the connexions on the router must be something like that :



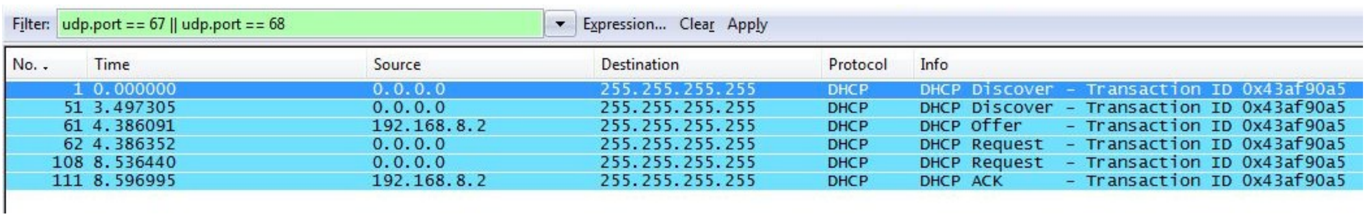
You are now able to use our DHCP relay.

TIP :
In order to be sure that the server and the router are working properly, you can try to "ping" the DHCP server (on `192.168.1.1`) by pushing the `BUTTON0` :



4.2 Running illustrations

We want to show you some illustrations, print-screens and remarks about our solution. Here is a print-screen on the client side :

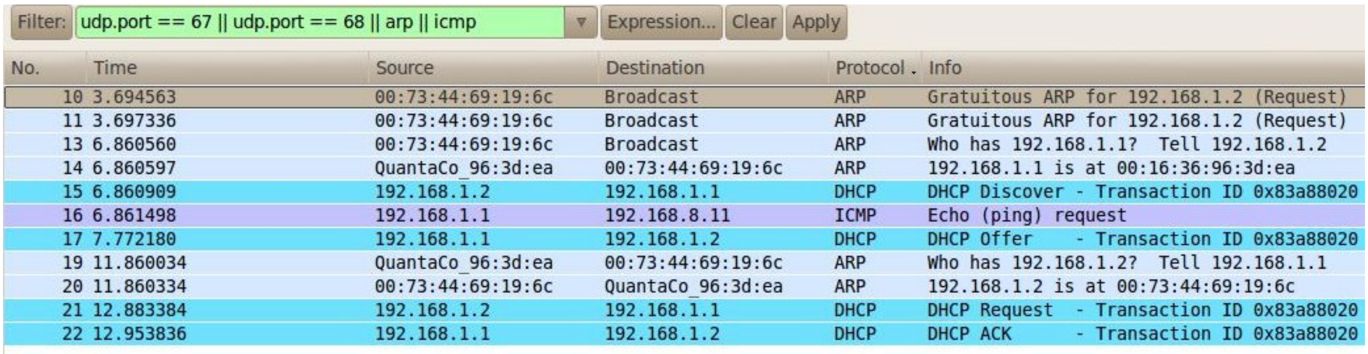

Filter: udp.port == 67 || udp.port == 68   Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0x43af90a5
51	3.497305	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0x43af90a5
61	4.386091	192.168.8.2	255.255.255.255	DHCP	DHCP Offer - Transaction ID 0x43af90a5
62	4.386352	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0x43af90a5
108	8.536440	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0x43af90a5
111	8.596995	192.168.8.2	255.255.255.255	DHCP	DHCP ACK - Transaction ID 0x43af90a5

Figure 4: Capture of the client

Remarks : We can see that the client sends two `DISCOVER` messages. The reason is simple. Our relay does not treat its request fast enough so the client will say it again. The same reason can be invoked for the duplicate `REQUEST` messages.

Now, here is a print screen of the server side :



Filter: udp.port == 67 || udp.port == 68 || arp || icmp — Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
10	3.694563	00:73:44:69:19:6c	Broadcast	ARP	Gratuitous ARP for 192.168.1.2 (Request)
11	3.697336	00:73:44:69:19:6c	Broadcast	ARP	Gratuitous ARP for 192.168.1.2 (Request)
13	6.860560	00:73:44:69:19:6c	Broadcast	ARP	Who has 192.168.1.1? Tell 192.168.1.2
14	6.860597	QuantaCo_96:3d:ea	00:73:44:69:19:6c	ARP	192.168.1.1 is at 00:16:36:96:3d:ea
15	6.860909	192.168.1.2	192.168.1.1	DHCP	DHCP Discover - Transaction ID 0x83a88020
16	6.861498	192.168.1.1	192.168.8.11	ICMP	Echo (ping) request
17	7.772180	192.168.1.1	192.168.1.2	DHCP	DHCP Offer - Transaction ID 0x83a88020
19	11.860034	QuantaCo_96:3d:ea	00:73:44:69:19:6c	ARP	Who has 192.168.1.2? Tell 192.168.1.1
20	11.860334	00:73:44:69:19:6c	QuantaCo_96:3d:ea	ARP	192.168.1.2 is at 00:73:44:69:19:6c
21	12.883384	192.168.1.2	192.168.1.1	DHCP	DHCP Request - Transaction ID 0x83a88020
22	12.953836	192.168.1.1	192.168.1.2	DHCP	DHCP ACK - Transaction ID 0x83a88020

Figure 5: Capture of the server

Remarks : As you can see, even if the client has sent multiple discovery messages, only one message per client arrives at the server. You can also see the ARP request done by the PIC in order to contact the server for the first time. Another thing to mention is the fact that, when the server received the `DHCP DISCOVERY` message and wants to make an offer with address `P`, it performs a ping request to this address `P` in order to check if someone already owns it.

To conclude this part, you can see the different messages passing through our relay via the little LCD screen on the PIC. Here is an example of messages :

(a) DHCP Discover

(b) DHCP Offer

(c) DHCP Request

(d) DHCP Ack

Figure 6: Messages on the LCD

Conclusion

It is important to begin the work with some technical and theoretical terms in order to build a model. Thank to the ASG tool, the design of our small DHCP system has been made easier. This helps us to begin with a more general approach that has been deepen as long as we refine the model. The implementation is then easier, because it is based on a concrete model and the way to convert ASG concepts into a programming language is known.

We do not claim that our solution is the best one because we are sure that there are many better ways to deal with incoming packets in the Ethernet buffer of the PIC 18, better ways to forward packets but, we think that the solution we propose is acceptable regarding to the objectives of the course. We learned a lot on modelling systems and on programming on real-time machines.

A Annex : Modified files

We mainly used the package "IPonPIC" from Microchip to develop our relay. Here is a list of the files that have been modified :

- MainDemo.c $\Longrightarrow$ In method InitAppConfig
- StackTsk.c $\Longrightarrow$ In method StackApplications
- DHCPr.c $\Longrightarrow$ New file
- TCPIPConfig.h $\Longrightarrow$ Add some new entries
- PingDemo.c $\Longrightarrow$ In method PingDemo

B Annex : Code of DHCPr.c

```
/*********************************************************************
 *
 *  Dynamic Host Configuration Protocol (DHCP) Relay
 *  Module for Microchip TCP/IP Stack
 *
 *********************************************************************
 * FileName:        DHCPr.c
 * Processor:       PIC18, PIC24F, PIC24H, dsPIC30F, dsPIC33F, PIC32
 * Compiler:        sdcc
 * Company:         UCLouvain.be - EPL 2010
 *
 * Author                Date       Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Lamouline Laurent
 * Nuttin Vincent      05/15/10     Original
 ********************************************************************/
#define __DHCPS_C
#define __18F97J60
#define __SDCC__
#include <pic18f97j60.h> //ML

#include "../Include/TCPIPConfig.h"

#if defined(STACK_USE_DHCP_SERVER)

#include "../Include/TCPIP_Stack/TCPIP.h"

static union
{
  union
  {
//ML  ROM BYTE *szROM;
    BYTE *szRAM;
  } RemoteHost;
  NODE_INFO DHCPRemote;
} StaticVars;

static enum
  {
    SM_IDLE = 0,
    SM_ARP_SEND_QUERY,
    SM_ARP_GET_RESPONSE,
    SM_MESS_SEND
  } SMState = SM_ARP_SEND_QUERY;

int counter = -1;  // LV debug
```

```
IP_ADDR ReqIP;
int reqIPnonNull = 0;
static UDP_SOCKET     MySocket; // Socket used by DHCP Server
static UDP_SOCKET     MySocket2;  // Socket used by DHCP Client
static IP_ADDR        DHCPNextLease;  // IP Address to provide for next lease
BOOL          bDHCPRelayEnabled = TRUE; // Whether or not the DHCP server is enabled

static void ForwardToServer(BOOTP_HEADER *Header, int type);
static void ForwardToClient(BOOTP_HEADER *Header, int type);

/*****************************************************************************
  Function:
  void DHCPRelayTask(void)

  Summary:
  Performs periodic DHCP relay tasks.

  Description:
  This function performs any periodic tasks requied by the DHCP relay
  module, such as forwarding DHCP messages.

  Precondition:
        None

  Parameters:
  None

  Returns:
    None
  ***************************************************************************/
void DHCPRelayTask(void)
{
  BYTE        i;
  BYTE        Option, Len;
  BOOTP_HEADER    BOOTPHeader;
  DWORD       dw;
  BOOL        bAccept;
  static enum
  {
    DHCP_OPEN_SOCKET,
    DHCP_LISTEN
  } smDHCPServer = DHCP_OPEN_SOCKET;

#if defined(STACK_USE_DHCP_CLIENT)
  // Make sure we don't clobber anyone else's DHCP server
  if(DHCPIsServerDetected(0))
    return;
#endif

  if(!bDHCPRelayEnabled)
    return;
  /* DHCP State Machine */
  switch(smDHCPServer)
  {
    case DHCP_OPEN_SOCKET:
      // Obtain a UDP socket to listen/transmit on
      MySocket = UDPOpen(DHCP_SERVER_PORT, NULL, DHCP_CLIENT_PORT);
      MySocket2 = UDPOpen(DHCP_CLIENT_PORT, NULL, DHCP_SERVER_PORT);
      if(MySocket == INVALID_UDP_SOCKET || MySocket2 == INVALID_UDP_SOCKET){
        DisplayString (0,"Invalid socket");
        break;
      }

      // Decide which address to lease out
      // Note that this needs to be changed if we are to
      // support more than one lease
      DHCPNextLease.Val = (AppConfig.MyIPAddr.Val & AppConfig.MyMask.Val) + 0x02000000;
      if(DHCPNextLease.v[3] == 255u)
```

```
        DHCPNextLease.v[3] += 0x03;
      if(DHCPNextLease.v[3] == 0u)
        DHCPNextLease.v[3] += 0x02;

      smDHCPServer++;

    case DHCP_LISTEN:
      // Check to see if a valid DHCP packet has arrived
      if(UDPIsGetReady(MySocket) < 241u)
        break;
      counter++;
      // DisplayWORD(counter, counter);
      // Retrieve the BOOTP header
      UDPGetArray((BYTE*)&BOOTPHeader, sizeof(BOOTPHeader));

      bAccept = (BOOTPHeader.ClientIP.Val == DHCPNextLease.Val) || (BOOTPHeader.
          ClientIP.Val == 0x00000000u);

      // Validate first three fields
      /* LV : We remove this because we are a relay. Type 1 and 2 are allowed !
      if(BOOTPHeader.MessageType != 1u)
        break;
      */
      if(BOOTPHeader.HardwareType != 1u)
        break;
      if(BOOTPHeader.HardwareLen != 6u)
        break;

      // Throw away 10 unused bytes of hardware address,
      // server host name, and boot file name -- unsupported/not needed.
      for(i = 0; i < 64+128+(16-sizeof(MAC_ADDR)); i++)
        UDPGet(&Option);

      // Obtain Magic Cookie and verify
      UDPGetArray((BYTE*)&dw, sizeof(DWORD));
      if(dw != 0x63538263ul)
        break;

      // Obtain options
      while(1)
      {
        // Get option type
        if(!UDPGet(&Option)){
          break;
        }
        if(Option == DHCP_END_OPTION)
          break;

        // Get option length
        UDPGet(&Len);

        // Process option
        switch(Option)
        {
          case DHCP_MESSAGE_TYPE:
            UDPGet(&i);
            //DisplayString(0,"gotDHCP"); // LV debug
            switch(i)
            {
              case DHCP_DISCOVER_MESSAGE:
                //DisplayWORD(16+counter,i); // LV debug
                //DisplayString (16+counter,"D"); // LV debug
                DisplayString (16,"DHCP DISCOVERY");
                LED5_IO = 1;
                LED6_IO = 0; // A new client is there ! LED 5 on :-)
                StaticVars.DHCPRemote.IPAddr.Val = AppConfig.DHCPServer.Val;
                ForwardToServer(&BOOTPHeader, 1);
                break;

              case DHCP_OFFER_MESSAGE:
```

```
                //DisplayWORD(16,i); // LV debug
                //DisplayString (16+counter,"O"); // LV debug
                DisplayString (16,"DHCP OFFER");
                LED5_IO = 0;
                LED6_IO = 1;
                StaticVars.DHCPRemote.IPAddr.Val = AppConfig.DHCPServer.Val;
                ForwardToClient(&BOOTPHeader, 1);
                break;

              case DHCP_REQUEST_MESSAGE:
                //DisplayWORD(16,i); // LV debug
                //DisplayString (30,"R"); // LV debug
                DisplayString (16,"DHCP REQUEST");
                LED5_IO = 1;
                LED6_IO = 0;
                StaticVars.DHCPRemote.IPAddr.Val = AppConfig.DHCPServer.Val;
                ForwardToServer(&BOOTPHeader, 2);
                break;

              case DHCP_ACK_MESSAGE:
                //DisplayWORD(16,i); // LV debug
                //DisplayString (31,"A"); // LV debug
                DisplayString (16,"DHCP ACK");
                LED5_IO = 0;
                LED6_IO = 1;
                StaticVars.DHCPRemote.IPAddr.Val = AppConfig.DHCPServer.Val;
                ForwardToClient(&BOOTPHeader, 2);
                break;

              // Need to handle these if supporting more than one DHCP lease
              case DHCP_RELEASE_MESSAGE:
              case DHCP_DECLINE_MESSAGE:
                break;
              default:
                break;
            }
            break;

          /*
          case DHCP_PARAM_REQUEST_IP_ADDRESS:
            if(Len == 4u)
            {
              // Get the requested IP address and see if it is the one we have on
                  offer.
              UDPGetArray((BYTE*)&dw, 4);
              Len -= 4;
              bAccept = (dw == DHCPNextLease.Val);
            }
            break;
          */
          case DHCP_END_OPTION:
            UDPDiscard();
            return;

        }

        // Remove any unprocessed bytes that we don't care about
        while(Len--)
        {
          UDPGet(&i);
        }
      }

      UDPDiscard();
      break;
  }
}


/*****************************************************************************
```

```
  Function:
  static void ForwardToServer(BOOTP_HEADER *Header, int type)

  Summary:
  Forwards a message received from a client to the server

  Description:
  This function forwards to a DHCP server message sent by a client

  Precondition:
  None

  Parameters:
  Header - the BootP header to forward
  Type - 1 : Discovery
         2 : Request

  Returns:
    None
  ***************************************************************************/
static void ForwardToServer(BOOTP_HEADER *Header, int type)
{
  BYTE i;
  UDP_SOCKET_INFO *p;

  /* ARP State Machine */
  switch(SMState)
    {
     case SM_IDLE:
      break;

    case SM_ARP_SEND_QUERY:
      LED1_IO = 1;
      SMState = SM_ARP_GET_RESPONSE;
      ARPResolve(&StaticVars.DHCPRemote.IPAddr);
      break;

  case SM_ARP_GET_RESPONSE:
      // See if the ARP reponse was successfully received
      LED2_IO = 1;
      if(!ARPIsResolved(&StaticVars.DHCPRemote.IPAddr,
                       &StaticVars.DHCPRemote.MACAddr)) break;

      SMState = SM_MESS_SEND;
      // No break;

  case SM_MESS_SEND:

    // Set the correct socket to active and ensure that
    // enough space is available to generate the DHCP response
    if(UDPIsPutReady(MySocket2) < 300u)
      return;

    // Search through all remaining options and look for the Requested IP address
        field
    // Obtain options
    while(UDPIsGetReady(MySocket))
    {
      BYTE Option, Len;
      DWORD dw;

      // Get option type
      if(!UDPGet(&Option))
        break;
      if(Option == DHCP_END_OPTION)
        break;

      // Get option length
      UDPGet(&Len);

```

```
    // Process option
    if((Option == DHCP_PARAM_REQUEST_IP_ADDRESS) && (Len == 4u))
    {
      // Get the requested IP address
      UDPGetArray((BYTE*)&ReqIP, 4);
      reqIPnonNull = 1;
    break;
    }

    // Remove the unprocessed bytes that we don't care about
    while(Len--)
    {
      UDPGet(&i);
    }
  }

  UDPIsPutReady(MySocket2);

  //Copy of the header to forward it!
  UDPPutArray((BYTE*)&(Header->MessageType), sizeof(Header->MessageType));
  UDPPutArray((BYTE*)&(Header->HardwareType), sizeof(Header->HardwareType));
  UDPPutArray((BYTE*)&(Header->HardwareLen), sizeof(Header->HardwareLen));
  UDPPutArray((BYTE*)&(Header->Hops), sizeof(Header->Hops));
  UDPPutArray((BYTE*)&(Header->TransactionID), sizeof(Header->TransactionID));
  UDPPutArray((BYTE*)&(Header->SecondsElapsed), sizeof(Header->SecondsElapsed));
  UDPPutArray((BYTE*)&(Header->BootpFlags), sizeof(Header->BootpFlags));
  UDPPutArray((BYTE*)&(Header->ClientIP), sizeof(Header->ClientIP));
  UDPPutArray((BYTE*)&(Header->YourIP), sizeof(Header->YourIP));
  UDPPutArray((BYTE*)&(Header->NextServerIP), sizeof(Header->NextServerIP));
  UDPPutArray((BYTE*)&(AppConfig.PrimaryDNSServer), sizeof(AppConfig.
      PrimaryDNSServer));  //Fill the giadrr addr with the relay address
  UDPPutArray((BYTE*)&(Header->ClientMAC), sizeof(Header->ClientMAC));

  // Set chaddr[6..15], sname and file as zeros.
    for ( i = 0; i < 202u; i++ ) UDPPut(0);

  // Put magic cookie as per RFC 1533.
    UDPPut(99);
    UDPPut(130);
    UDPPut(83);
      UDPPut(99);

  // Options: change if we have a discover or a request
  UDPPut(DHCP_MESSAGE_TYPE);
  UDPPut(DHCP_MESSAGE_TYPE_LEN);

  if(type == 1){
    UDPPut(DHCP_DISCOVER_MESSAGE);
    //DisplayString (30,"Di"); // LV debug
  }
  else{
    UDPPut(DHCP_REQUEST_MESSAGE);
    //DisplayString (30,"Re"); // LV debug
  }

  // Option: Server identifier
  UDPPut(DHCP_SERVER_IDENTIFIER);
  UDPPut(sizeof(IP_ADDR));
  UDPPutArray((BYTE*)&AppConfig.MyIPAddr, sizeof(IP_ADDR));

  // Option: Router/Gateway address
  UDPPut(DHCP_ROUTER);
  UDPPut(sizeof(IP_ADDR));
  UDPPutArray((BYTE*)&AppConfig.MyIPAddr, sizeof(IP_ADDR));

  /* Requested IP in field 50 ! */
  if (reqIPnonNull == 1){
    //DisplayString(0, "Addr Requested!"); // LV debug
    //DisplayIPValue(ReqIP.Val); // LV debug
    UDPPut(DHCP_PARAM_REQUEST_IP_ADDRESS);
```

```
        UDPPut(DHCP_PARAM_REQUEST_IP_ADDRESS_LEN);
        UDPPutArray((BYTE*)&ReqIP, sizeof(IP_ADDR));
        reqIPnonNull = 0;
      }

      // No more options, mark ending
      UDPPut(DHCP_END_OPTION);

      // Add zero padding to ensure compatibility with old BOOTP relays that discard small packets (<300 UDP octets)
      while(UDPTxCount < 300u)
        UDPPut(0);

      UDPIsPutReady(MySocket2);
      p = &UDPSocketInfo[activeUDPSocket];
      p->remoteNode.IPAddr.Val = StaticVars.DHCPRemote.IPAddr.Val; // Unicast mode : Set up DHCP Server IP
      for(i = 0; i < 6; i++){
        p->remoteNode.MACAddr.v[i] = StaticVars.DHCPRemote.MACAddr.v[i]; // Remote HADDR filled in by the result of ARP
      }
        UDPFlush();
        LED1_IO = 0;
        LED2_IO = 0;
      SMState = SM_ARP_SEND_QUERY; // Inconditionnal transition to the top-state (ASG)

      break;

    default:
        return;
    }
  }

  /*****************************************************************************
    Function:
    static void ForwardToClient(BOOTP_HEADER *Header, int type)

    Summary:
    Forwards a message received from the server to the related client

    Description:
    This function forwards to a client message sent by the DHCP server

    Precondition:
    None

    Parameters:
    Header - the BootP header to forward
    Type - 1 : Offer
           2 : Ack

    Returns:
      None
    ***************************************************************************/
  static void ForwardToClient(BOOTP_HEADER *Header, int type)
  {
    BYTE i;
    UDP_SOCKET_INFO *p;

    /* ARP State Machine : Useless here */
    switch(SMState)
      {
      case SM_IDLE:
        break;

      case SM_ARP_SEND_QUERY:
        SMState = SM_MESS_SEND;

    case SM_MESS_SEND:
```

```c

        // Set the correct socket to active and ensure that
        // enough space is available to generate the DHCP response
        if(UDPIsPutReady(MySocket) < 300u)
          return;
        p = &UDPSocketInfo[activeUDPSocket]; // Activation of the socket on local port 67 to remote port 68
        p->remoteNode.IPAddr.Val = AppConfig.Br.Val; // Broadcast !
        p->remotePort = DHCP_CLIENT_PORT; // Contact the client on port 68

        // Copy of the MAC address of the client (from CHADDR field)
        for ( i = 0; i < 6u; i++ ){
          p->remoteNode.MACAddr.v[i] = Header->ClientMAC.v[i];
        }

        //Print the two last part of the MAC address
        /*
        DisplayString(0, "MAC Addr =");
        DisplayWORD(16, Header->ClientMAC.v[4]);
        DisplayWORD(20, Header->ClientMAC.v[5]);
        */

        //Copy of the header to forward it !
        UDPPutArray((BYTE*)&(Header->MessageType), sizeof(Header->MessageType));
        UDPPutArray((BYTE*)&(Header->HardwareType), sizeof(Header->HardwareType));
        UDPPutArray((BYTE*)&(Header->HardwareLen), sizeof(Header->HardwareLen));
        UDPPutArray((BYTE*)&(Header->Hops), sizeof(Header->Hops));
        UDPPutArray((BYTE*)&(Header->TransactionID), sizeof(Header->TransactionID));
        UDPPutArray((BYTE*)&(Header->SecondsElapsed), sizeof(Header->SecondsElapsed));
        UDPPutArray((BYTE*)&(Header->BootpFlags), sizeof(Header->BootpFlags));
        UDPPutArray((BYTE*)&(Header->ClientIP), sizeof(Header->ClientIP));
        UDPPutArray((BYTE*)&(Header->YourIP), sizeof(Header->YourIP));
        UDPPutArray((BYTE*)&(Header->NextServerIP), sizeof(Header->NextServerIP));
        UDPPutArray((BYTE*)&(AppConfig.PrimaryDNSServer), sizeof(AppConfig.PrimaryDNSServer)); //Fill the giadrr addr with the relay address
        UDPPutArray((BYTE*)&(Header->ClientMAC), sizeof(Header->ClientMAC));


        // Set chaddr[6..15], sname and file as zeros.
          for ( i = 0; i < 202u; i++ ) UDPPut(0);

        // Load magic cookie as per RFC 1533.
          UDPPut(99);
          UDPPut(130);
          UDPPut(83);
            UDPPut(99);

        // Options: change if we have an offer or an ack
        UDPPut(DHCP_MESSAGE_TYPE);
        UDPPut(DHCP_MESSAGE_TYPE_LEN);
        if(type == 1){
          UDPPut(DHCP_OFFER_MESSAGE);
          //DisplayString (30,"Of"); // LV debug
        }
        else{
          UDPPut(DHCP_ACK_MESSAGE);
          //DisplayString (30,"Ac"); // LV debug
        }

        // Add zero padding to ensure compatibility with old BOOTP relays that discard small packets (<300 UDP octets)
        while(UDPTxCount < 300u)
          UDPPut(0);

          UDPFlush();

        SMState = SM_ARP_SEND_QUERY; // Inconditionnal transition to the top-state (ASG)

        break;
```

```
  default:
      return;
  }
}

#endif //#if defined(STACK_USE_DHCP_SERVER)
```