BY FRED EADY

PETER BEST CIRCLES THE DRAIN

NO. IT'S NOT A MISPRINT. Peter has been rotated out of the Design Cycle and into the toilet ... literally! It may comfort you to know that Peter hasn't committed any felonious acts or taken any missteps in the outhouse. Mr. Best has decided to leave his Design Cycle duties and take a position with the Xerolet Corporation. Peter is in the toilet because Xerolet makes potties. However, unlike the dumb (but proud) porcelain monument found in most of our bathrooms, the potties produced by Xerolet are high-tech units designed to be deployed in places devoid of communal waste treatment facilities. You can take a look at what Peter has "fallen into" at www.xerolet.com. And, yes. There are LOTS of potty jokes floating around the EDTP lab these days. I am honored to be able to step in for Peter and we all wish him continued success at Xerolet.

Peter and I worked together on the EDTP Ethernet MINI C-to-PBP (PICBASIC PRO) conversion project that you have been reading about in this series of Design Cycle episodes. So, I'm up to speed and ready to go with this month's MINI driver conversion subject matter. We've already gotten the basic driver stuff up and the latest EDTP Ethernet MINI PBP driver code for ARP, UDP, and ICMP (PING) was put up for you to grab via ftp last month. The PICBASIC PRO MINI code you have access to right now is enough to get the EDTP Ethernet MINI online. However, adding a touch of DHCP to the mix will add some flavor to our MINI stew.

JUST IN CASE

If you're DHCP challenged, Peter presented the details of the innerworkings of DHCP and the embedded coding behind it all in a past Design Cycle column. If the concepts that I am presenting in this discussion are unfamiliar to you, I suggest refreshing your DHCP knowledge by checking out past Design Cycle columns.

DHCP is short for Dynamic Host Configuration Protocol. As Peter pointed out in his DHCP discussions, DHCP is message based and through its services a network client host such as the EDTP Ethernet MINI can automatically obtain IP, gateway, and subnet mask information from a DHCP server participating in the client host's network. DHCP is not difficult to understand, and likewise, is not difficult to code into an embedded application. This month, our goal is to code a PICBASIC PRO-based DHCP engine into the EDTP Ethernet MINI. So, let's get started.

DHCP WITH PICBASIC PRO

We'll need to add some DHCP-specific PBP code modules to our existing PICBASIC PRO-based EDTP Ethernet MINI driver to get DHCP up and running on our EDTP Ethernet MINI hardware. Specifically, at a minimum, we'll need to add a DHCP initialization routine, a DHCP engine routine, a DHCP transmit message routine, and a DHCP receive message routine to our existing code. While we're at it, we'll also code in a routine to allow us to display our received DHCP addressing information on a personal computer running a terminal emulator application. And, if that's not enough excitement for you, we'll throw in some PICBASIC PRO code to broadcast our newly acquired DHCP parameters using UDP and retrieve the UDP datagram using a Microchip PC application called MCHPDetect.

PLANTING THE FIRST DHCP SEED

Before we do anything about building up the aforementioned DHCP modular routines, we'll need to tell our main PICBASIC PRO operating loop about the DHCP stuff we're about to code up. Please consider the following PBP source code snippet:

```
forever con 1
;*********************************************************
;*     MAIN PROGRAM
;*********************************************************
       on interrupt goto int_handler
       OSCTUNE = $40
;*********************************************************
;*     CONFIGURE AND START TIMER2
;*     SET TO OVERFLOW EVERY 1mS
;*********************************************************
       hours = 12
       mins = $00
       secs = $00
       milliseconds = 0
       T2CON = $79
       PR2 = $A3
       TMR2ON = 1
       TMR2IE = 1
       gosub init_EUSART
       gosub init_IO
       gosub init_67J60
       DHCPSTATE = DHCP_ENTRY
       gosub init_DHCP

       while forever
        if EPKTCNT != 0  then
              gosub get_frame
        endif
        gosub dhcp_state_engine
       wend
end
//*********************************************************
```

Nothing in the original MAIN PROGRAM code has changed, but I did take the liberty of adding some DHCP-relative statements. The DHCPSTATE = DHCP_ENTRY PBP statement allows you to enable or disable the DHCP code at compile time. For instance, equating the DHCPSTATE to DHCP_DISABLED will turn the code execution around at the beginning of the DHCP initialization routine init_DHCP that is called following the declaration of the initial DHCP state you desire. You can easily see how this "turn around" mechanism works by examining the simple PBP statements that make up the DHCP initialization code module:

```
;*********************************************************
;*     INITIALIZE DHCP STATE MACHINE
;*********************************************************
init_DHCP:

       if DHCPSTATE == DHCP_DISABLED then
             return
       endif
       dhcpflags = 0
       DHCPSTATE = 0
       DHCPSTATE = DHCP_ENTRY
return
//*********************************************************
```

Once all of the hardware modules have been initialized and the DHCP state has been determined, the EDTP Ethernet MINI driver will loop forever checking for incoming Ethernet frames and servicing the DHCP engine.

If you were able to get your hands on the previous EDTP Ethernet MINI PICBASIC PRO conversion Design Cycle installments, you'll recall that many a C-to-PBP pitfall lies in our path. One of those spiked booby traps involved the way PBP handles SELECT CASE structures versus the way ANSI C treats its native SELECT structures. PICBASIC PRO handles SELECT CASE perfectly in 99.999% of the situations the SELECT CASE statement is involved in. However, although the PBP SELECT CASE statement in its native state doesn't fit well into our DHCP engine code, it seems that the PICBASIC PRO designers were really just trying to help us out. The ANSI C language delimits each case structure within a SELECT structure with a break statement. A C programmer with too many hours behind the keyboard and too much JOLT soda in his or her system may accidentally omit a break statement.

The C program will compile just fine. However, the case structure immediately following the omitted break statement will execute without performing another select operation. That's normally not a good thing. Guess what? It was done intentionally in the EDTP Ethernet MINI C driver code. Here's a code snippet of the original C version:

```
//*********************************************************
//*     DHCP STATE MACHINE
//*********************************************************
void dhcp_state_engine(void)
{
        unsigned int i;
        switch(DHCPSTATE)
        {
              case DHCP_ENTRY:
                    printf("\r\nDHCP RESET..");
                    for(i=0;i<4;++i)
                          tempipaddrc[i] = 0x00;
                    DHCPSTATE = DHCP_INIT;
              case DHCP_INIT:
                    printf("\r\nDHCP INIT..");
                    for(i=0;i<6;++i)
                          svrmacaddrc[i] = 0xFF;
                    for(i=0;i<4;++i)
                          svridc[i] = 0xFF;
                    msecs_timer = 0;
                    DHCPSTATE = DHCP_WAIT;
              case DHCP_WAIT:
                    if(msecs_timer >= 2000)
                          DHCPSTATE = DHCP_BROADCAST;
              break;
//*********************************************************
```

Note the absence of the break statement between the DHCP_ENTRY, DHCP_INIT, and DHCP_WAIT case structures in the preceding C code snippet. Intentionally omitting the break statements causes the trio of case structures to execute one after the other until a break statement is encountered. That may be clever in C, but it's impossible to do in PICBASIC PRO. Accidental elimination of break statements within SELECT CASE structures is just the thing I believe the PICBASIC PRO designers are protecting us from. That's one we should add to the Universal Rules of Computing: "You can't screw it up if you can't code it."

Since we can't override the absence of SELECT CASE break statements in PBP, we'll have to emulate the original DHCP engine code as best we can with the tools that PICBASIC PRO provides. As you can see in the PICBASIC PRO code snippet that follows, the ported PBP DHCP engine code converts the original EDTP Ethernet MINI C

DHCP engine code's SELECT structures to standard PBP if-then structures:

```
;*****************************************************
;*     DHCP STATE MACHINE
;*****************************************************
dhcp_state_engine:

            if DHCPSTATE == DHCP_DISABLED then
            return
            endif

            if DHCPSTATE == DHCP_ENTRY then
                  hserout[13,10,"DHCP RESET.."]
                  for i8 = 0 to 3
                        tempipaddrc[i8] = $00
                  next i8
                  DHCPSTATE = DHCP_INIT
            endif
            if DHCPSTATE == DHCP_INIT then
                  hserout[13,10,"DHCP INIT.."]
                  for i8 = 0 to 5
                        svrmacaddrc[i8] = $FF
                  next i8
                  for i8 = 0 to 3
                        svridc[i8] = $FF
                  next i8
                  msecs_timer = 0
                  DHCPSTATE = DHCP_WAIT
            endif
            if DHCPSTATE == DHCP_WAIT then
                  if(msecs_timer >= 2000) then
                        DHCPSTATE = DHCP_BROADCAST
            endif
      goto breaker
////////////////////////////////////////////////////////
breaker:
return
//*****************************************************
```

Using consecutive if-then PBP structures allows us to emulate the "breakless" C SELECT constructs. When a break of execution is required, the much C-programmer-maligned BASIC GOTO statement is brought into action. You can use GOTO in many C compiler implementations but odds are you'll never see a GOTO used in a serious C application. I must admit that I tend to avoid using the GOTO statement at all and I was really pulling for straws when I coded it into the PICBASIC PRO EDTP Ethernet MINI driver. The line of forward slashes (///) in the preceding PBP code snippet represents the rest of the dhcp_state_engine subroutine statements. The breaker label lies at the end of the dhcp_state_engine subroutine.

Other than the SELECT emulation, the rest of the dhcp_state_engine subroutine port was logical and straightforward. As you can see in the preceding code snippets, the C printf statements map easily to the PBP hserout statements. C-to-PBP if-then and for-next structures were also easy ports.

Another C form that is not native to PBP is the concept of non-assembler-based macros and function calls. Here's what a bbound macro and send_dhcp function call within a SELECT case structure looks like in the EDTP Ethernet MINI C driver source:

```
char dhcpflags;
#define bound           0x01
#define offerrec        0x02
#define done            0x04
#define trashit         0x08
#define newdhcppkt      0x10
#define bbound  (dhcpflags & bound)
////////////////////////////////////////////////////////
case DHCP_BROADCAST:
      leasetime = 60;
      if(bbound)
      {
            DHCPSTATE = DHCP_REQUEST;
      }
      else
      {
            send_dhcp(DHCP_DISCOVER_MESSAGE);
            msecs_timer = 0;
            DHCPSTATE = DHCP_DISCOVER;
            printf("\r\nSENT DISCOVER MESSAGE..");
      }
break;
```

Since PBP doesn't support function calls with parameters, we must preload our emulated PBP function and convert the C function code to a PBP subroutine. This is illustrated by the PBP code following the else statement in the PBP if-then SELECT emulation code that follows:

```
if DHCPSTATE == DHCP_BROADCAST then
      leasetime = 60
      if bbound then
            DHCPSTATE = DHCP_REQUEST
      else
            dhcpmsgtype = DHCP_DISCOVER_MESSAGE
            gosub send_dhcp
            msecs_timer = 0
            DHCPSTATE = DHCP_DISCOVER
            hserout[13,10,"SENT DISCOVER MESSAGE.."]
      endif
goto breaker
```

Pain is not evident with every modular C-to-PBP porting operation. To handle flags such as bbound, I used an elaborate C macro scheme to clear and set software flags in the C version of the EDTP Ethernet MINI driver. Here's an example:

```
char dhcpflags;
#define bound           0x01
#define offerrec        0x02
#define done            0x04
#define trashit         0x08
#define newdhcppkt      0x10
#define bbound          (dhcpflags & bound)
#define bofferrec       (dhcpflags & offerrec)
#define bdone           (dhcpflags & done)
#define btrashit        (dhcpflags & trashit)
#define bnewdhcppkt     (dhcpflags & newdhcppkt)

#define clr_bound       dhcpflags &= ~bound
#define set_bound       dhcpflags |= bound
#define clr_offerrec    dhcpflags &= ~offerrec
#define set_offerrec    dhcpflags |= offerrec
#define clr_done        dhcpflags &= ~done
#define set_done        dhcpflags |= done
#define clr_trashit     dhcpflags &= ~trashit
#define set_trashit     dhcpflags |= trashit
#define clr_newdhcppkt  dhcpflags &= ~newdhcppkt
#define set_newdhcppkt  dhcpflags |= newdhcppkt
```

■ PHOTO 1. I love my MPLAB ICD2 hockey pucks. However, the REAL ICE is much faster when it comes to loading and debugging the PIC18F67J60 mounted on the EDTP Ethernet MINI. The REAL ICE also seems to be faster and easier to recover when communications between it and the laptop are disrupted.

Note that I simply declared a flag byte (dhcpflags) and manipulated each bit within the byte as a software flag. If you're an ATMEL AVR C programmer, you've seen this method used many times in many places. Although I lose the convenience of just saying set this or clear that, the PBP port is very clean. I utilized the native functionality of PICBASIC PRO to produce this flag manipulation port:

```
dhcpflags       var byte
bbound          var dhcpflags.0
bofferrec       var dhcpflags.1
bdone           var dhcpflags.2
btrashit        var dhcpflags.3
bnewdhcppkt     var dhcpflags.4
```

To clear the bound flag in PBP, I simply code bbound = 0. Conversely, setting the bbound bit is as easy as bbound = 1. The "b" preceding bound identifies the bbound variable is a bit variable. Just as I did in the C version of the EDTP Ethernet MINI driver, testing the flag bits is normally done with an if-then structure. For example, to test and branch on the state of the bbound flag, we code:

```
If bbound then
        do stuff here
else
        do something else here instead
endif
```

The porting techniques I've described are used throughout the new PICBASIC PRO-based EDTP Ethernet MINI driver. There's more ported EDTP Ethernet MINI code than I can show you and talk about here. So, I've provided a full listing of both the C and PICBASIC PRO versions of the EDTP Ethernet MINI driver on the *Nuts & Volts* website (**www.nutsvolts.com**). You can also get the code sets from the EDTP Electronics site. I've tried to keep the C and PICBASIC PRO source code versions in sync so you can follow the port from C to PICBASIC PRO line-by-line.

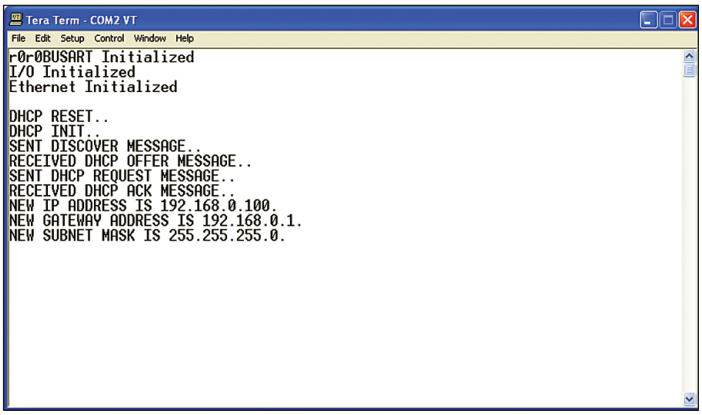
VERIFYING THE PORT

Just sitting behind my ThinkPad plinking out PBP code is one thing. Sitting behind three ThinkPads verifying my plinking is another. Here's the 3-D setup with "our" ThinkPad residing in your virtual lap. The EDTP Ethernet MINI hardware is connected to our ThinkPad via Microchip's REAL ICE, which is attached via USB. Our ThinkPad is running the MPLAB IDE, which is supporting the PICBASIC PRO compiler and the REAL ICE debugger/programmer. All of the porting, loading, and debugging of the EDTP Ethernet MINI hardware is done from our ThinkPad. For those of you that are not familiar with the Microchip REAL ICE, I've put one under the camera in Photo 1.

The ThinkPad to the left is attached via CAT5 cable to a LinkSys EtherFast Cable/DSL router, which is acting as the DHCP server for our little EDTP Ethernet MINI/ThinkPad network. In addition to being the second host on the EDTP Ethernet MINI network, the left-hand ThinkPad is also running Tera Term Pro, an RS-232 terminal emulator application, at 57600 bps. Since these new-fangled laptops dropped their native RS-232 ports in favor of USB interfaces, I'm forced to connect the left-hand ThinkPad's emulated COM2 to the EDTP Ethernet MINI's serial port via a USB-to-serial dongle. The third ThinkPad to the right is dedicated to running Network General's Sniffer Portable LAN Suite, which we will use to capture all of the DHCP activity flying around. Photo 2 shows them all chugging along.

If you've got your copy of the PBP source code port handy, you'll see that I converted and coded a DHCP send routine, a DHCP receive routine, and a DHCP send_bound_message routine. Each routine that is important to us has a "talker" routine within it that sends a human readable serial message to the left-hand ThinkPad running

■ PHOTO 2. The array of ThinkPad laptops used for development and debugging. On the left, the DHCP server and comm front end; in the center, the development system for the MINI, connected through REAL ICE; and on the right, the network sniffer.



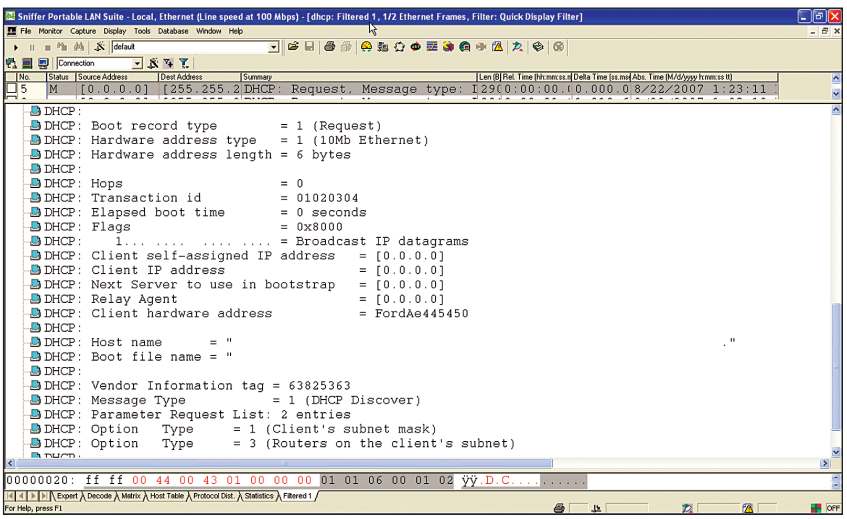
■ SCREENSHOT 1. This all goes back to my love of using an embedded device's RS-232 port as a debugging tool. During the debugging phase, if I didn't get the message I wanted to see here, or the messages were looping, I would then turn my attention to reading dumps and registers produced by the Microchip REAL ICE to find a fix.

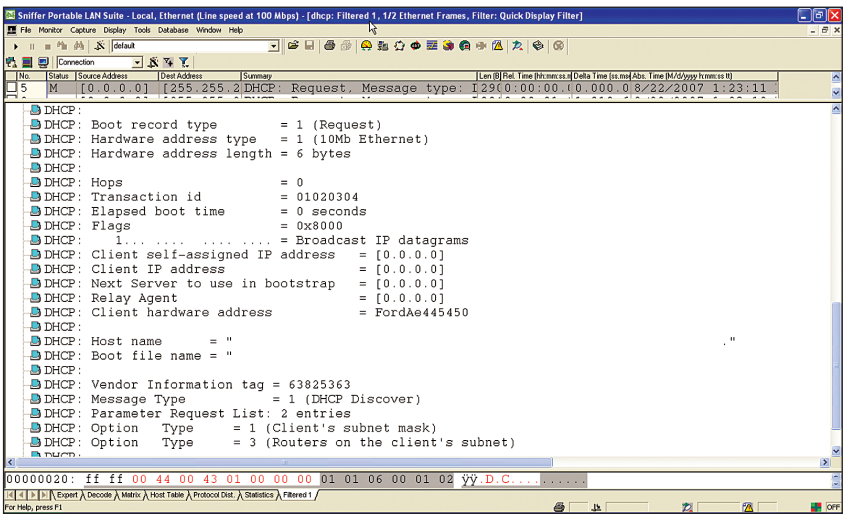
Tera Term Pro. If everything works as designed, that same ThinkPad that is receiving the serial messages from the EDTP Ethernet MINI's RS-232 port is also tasked with running the Microchip MCHPDetect program. The MCHPDetect program is looking for UDP broadcast messages on the network aimed at UDP port 30303. The send_bound_message routine in our PBP EDTP Ethernet MINI driver heaves the DHCP-parameters-laden UDP datagram at UDP port 30303. The left-hand ThinkPad, being an active participant in the network, is listening for UDP traffic addressed to UDP port 30303 by way of the MCHPDetect application it is running.

When the bits all line up, here's what should happen. Since we have not specified that the DHCP engine be disabled at compile time, the dhcp_state_engine subroutine will be called. The dhcp_state_engine DHCP_ENTRY and DHCP_INIT if-then constructs will execute and we will see the talker statements DHCP RESET and DHCP INIT appear in the Tera Term Pro terminal emulator window. I've captured the Tera Term Pro output in Screenshot 1.

Approximately two seconds later, the PBP EDTP Ethernet MINI driver will issue a DHCP Discover message via the send_dhcp subroutine. The talker statement SENT DISCOVER MESSAGE will appear on the ThinkPad running Tera Term Pro when the Discover message is sent. I just happened to have the right-hand ThinkPad and Sniffer Portable LAN Suite online and I captured the EDTP Ethernet MINI's outgoing DHCP Discover message in Screenshot 2.

■ SCREENSHOT 2. Since you can't really see the entire Sniff, this shot is aimed at providing a verification of the EDTP Ethernet MINI sending a correctly assembled DHCP Discover message. Note that I have asked for the network's subnet mask and the IP addresses of any routers on the network. The fabricated non-IEEE-issued MAC address of 00EDTP results in the Sniffer incorrectly identifying us as IEEE-legal FordAe45450.



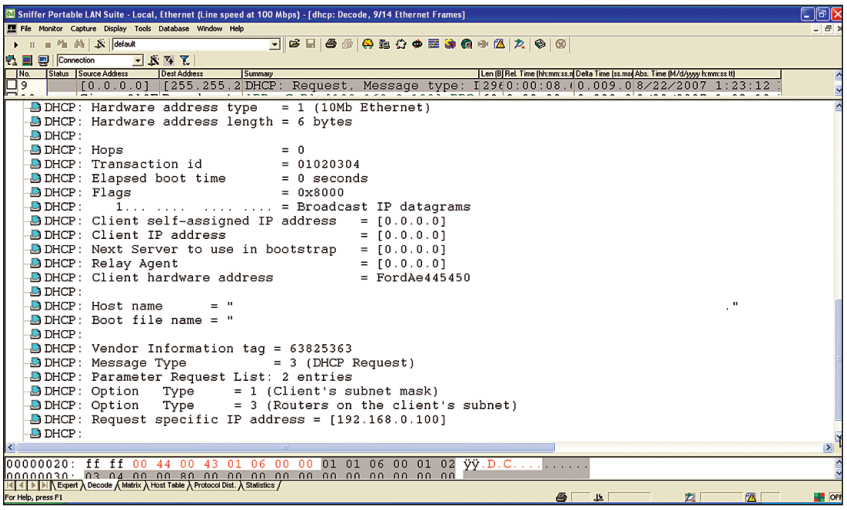


With a certified DHCP Discover on the way to the Linksys router, the PBP EDTP Ethernet MINI driver will then sit back and wait for a response from the Linksys router, which is configured as the EDTP Ethernet MINI LAN's DHCP server. The ThinkPad running Tera Term Pro has a hard-coded IP address below 192.168.0.100, which means our EDTP Ethernet MINI hardware should request and receive the first DHCP pool IP address of 192.168.0.100. The ThinkPad running the Sniffer Portable LAN Suite is not an active participant in the network it is monitoring and will not request an IP address from the Linksys router.

The Linksys router sees our DHCP Discover message and returns a DHCP Offer message, which kicks off the talker message RECEIVED DHCP OFFER MESSAGE on our ThinkPad that is running the Tera Term Pro terminal emulator application. The Sniffer Portable LAN Suite application is still monitoring the network and captures the Linksys-generated DHCP Offer

■ SCREENSHOT 3. I squeezed in all of the important stuff into this shot. Not only did we get all of the optional parameters we asked for, we also received an available IP address and lease time value that we may choose to use or reject.

■ SCREENSHOT 4. We could opt to reject the offer. However, there are no other routers on the network and the EDTP Ethernet MINI would just sit there asking for and rejecting offers from the Linksys router forever. So, in this shot we are asking to lease the IP address 192.168.0.100.



Sniff you see in Screenshot 3.

Taking a look at Screenshot 3, we find everything we need to put the EDTP Ethernet MINI on the network. Note the Request IP address lease time value. Yet another pothole for PICBASIC PRO; 172800 decimal seconds equates to $2A300 PICBASIC PRO hexadecimal seconds. The lease time value is beyond the 16-bit arithmetic limit of PICBASIC PRO. Although there are 32-bit tricks PICBASIC PRO can perform, we'll fill this pothole with caveman code. The lease time variable is 32 bits in length. To handle it with PBP, we need to arrange the 32 bits into PBP-chewable bytes:

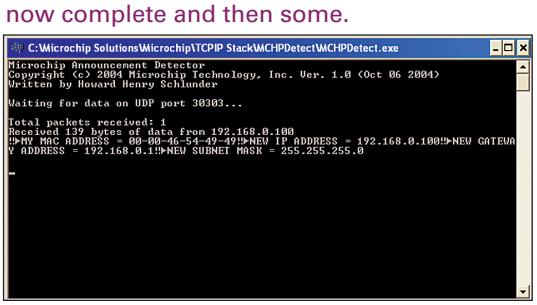
```
templeasetimec[0]       ; MSB
templeasetimec[1]
templeasetimec[2]
templeasetimec[3]       ; LSB
```

The lease time byte arrangement allows us to use caveman code to perform the 32-bit arithmetic as shown in the code that follows:

```
if templeasetimec[2] >= $07  then
for i16 = 0 to 1799
       templeasetimec[3]=templeasetimec[3]-1
       if templeasetimec[3] == $FF then
              templeasetimec[2] =
                templeasetimec[2] - 1
       endif
       if templeasetimec[2] == $FF then
              templeasetimec[1] = templeasetimec[1] - 1
       endif
       if templeasetimec[1] == $FF then
              templeasetimec[0] = templeasetimec[0] - 1
       endif
next i16
```

The lease time code I have presented is used to subtract 1800 seconds from the lease time value that is offered to us by the Linksys router within the DHCP Offer message. We can use the same caveman algorithm to determine when the lease expires:

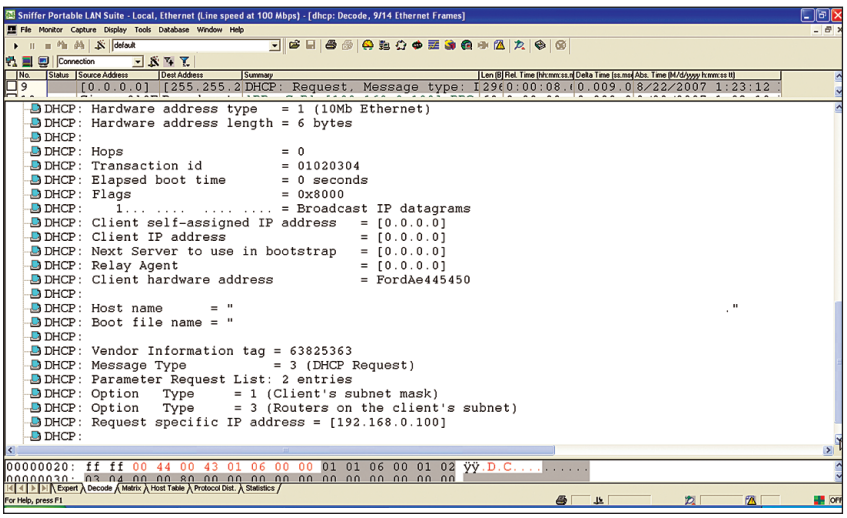
■ SCREENSHOT 6. This is simply gravy over the meat and potatoes. The verification process is now complete and then some.




```
if DHCPSTATE == DHCP_BOUND then
    if bleaseflag then
        bleaseflag = 0
        templeasetimec[3] = templeasetimec[3] - 1
        if templeasetimec[3] == $FF then
            templeasetimec[2] = templeasetimec[2] - 1
        endif
        if templeasetimec[2] == $FF then
            templeasetimec[1] = templeasetimec[1] - 1
        endif
        if templeasetimec[1] == $FF then
            templeasetimec[0] = templeasetimec[0] - 1
        endif
        if templeasetimec[0] == 0 && _
        templeasetimec[1] == 0 && _
        templeasetimec[2] == 0 && _
        templeasetimec[3] == 0 then
            hserout[13,10,"DHCP LEASE TIME EXPIRED.."]
            DHCPSTATE = DHCP_INIT
        endif
    endif
endif
```

I added a bleaseflag bit variable to the existing dhcpflags byte variable. If you look at the timer interrupt routine in the PBP driver source code, you'll see that I set the bleaseflag

■ SCREENSHOT 5. This is the DHCP message we've been working to receive. Everything the EDTP Ethernet MINI needs to get onto the network is here.



every second. When the dhcp_state_engine subroutine is called, the bleaseflag value is checked and if the bleaseflag is set and the EDTP Ethernet MINI is bound to the Linksys router, the 32-bit lease time value is decremented. If the lease time value rolls into zero, the dhcp_state_engine code kicks off another DHCP lease request to the Linksys router.

Before we can use any of that nifty 32-bit caveman arithmetic, we must accept the gracious offer that the Linksys router has given to us. We do that by blasting out a DHCP Request message, which is seen in Sniffer format in Screenshot 4.

Up to this point, our ported PICBASIC PRO EDTP Ethernet MINI driver code is working perfectly. There's only one more step to gaining access to the network: receiving the acknowledgment message to our lease request. The positive acknowledgment talker message in Screenshot 1 backs up the DHCP Ack Sniffer capture you see in Screenshot 5. All that's left to do is parse the incoming DHCP Ack message and place the IP addresses, subnet masks, and lease time values into their proper memory slots.

Once the EDTP Ethernet MINI received its new IP and gateway addresses, the send_bound_datagram subroutine pushed a UDP datagram out onto the network. The MCHP Detect application captured the data you see in Screenshot 6.

ANOTHER BRICK IN THE WALL

You have joined an elite club. You are now part of the initial PICBASIC PRO EDTP Ethernet MINI driver rollout. To my knowledge, at this time there are no other free PICBASIC PRO ports of PIC18F67J60 network drivers available to the public.

The only brick that is yet to be placed in our PICBASIC PRO wall of networking fame is TCP. I think you all have enough knowledge to get that done without talking about it in an additional installment of Design Cycle. You have the C template in the PIC BASIC PRO port download package. And, you know all of my tricks. I'm sure you also have some tricks of your own. So, go forth and network, you Basic lovers. Make Peter and his potties proud! See you next time. **NV**

SOURCES

- *microEngineering Labs* (**www.melabs.com**): PICBASIC PRO
- *EDTP Electronics, Inc.* (**www.edtp.com**): EDTP Ethernet MINI
- *Microchip* (**www.microchip.com**): PIC18F67J60; MPLAB; REAL ICE
- *Network General* (**www.networkgeneral.com**): Sniffer Portable LAN Suite
- *HI-TECH Software* (**www.htsoft.com**): The original EDTP Ethernet MINI driver C application was coded with HI-TECH PICC-18.

CONTACT THE AUTHOR

■ *Fred Eady can be reached via email at fred@edtp.com.*