

## FEATURE ARTICLE

# The Ethernet Development Board

Fred Eady

## Part 2: The Software and firmware Exposed

Fred picks up where he left off and takes us through embedded Ethernet, as he explores part two of his online series. Looking at the Ethernet development board from a firmware angle, he covers a lot of information, but still reminds us, just as he does in all his print columns, that "It's not complicated, it's embedded."



Hopefully by now, with the information I gave you last month, you've glued down a CS8900A-CQ Ethernet engine and your Ethernet development board is complete. This time around, I'll take you through what it takes to drive the Ethernet development board from a firmware standpoint. I'll also show you how to use Visual Basic to effect a test panel for your new toy. Again, there's a lot to cover, so let's tear apart the Ethernet development board source code line by line.

### THE BASICS

The first thing to do is establish an IP address for the Ethernet development board. Any valid IP address will work, and your choice of IP addresses depends on where the Ethernet development board will be used. I used 192.168.1.150, as it is one of the addresses reserved for private networks.

The IP address of the Ethernet development board is specified in the IP address definition area of the code. Hexadecimal notation could have been used in the definition, but it's not as easy to read and remember as dotted decimal notation. Thus, the "D" before each IP octet in Listing 1 tells the compiler that the numbers are in decimal format, rather than the compiler's default hexadecimal format.

The next order of business is to assign a hardware or MAC (Media Access Control) address to the Ethernet development board. The hardware address is normally a purchased item that is regulated by the IEEE. If you plan to use the Ethernet development board in a commercial environment, you'll need to purchase a unique hardware identifier. The idea is to not have identical hardware addresses on any NIC.

The common commercial MAC address is most often a combined IEEE-supplied ID merged with the Ethernet device's manufacturing serial number. You can normally find this Organizationally Unique Identifier (OUI) printed on commercial Ethernet NICs. Issuing *ARP-a* in a command window of WIN98 will also show you the OUI if the NIC has registered itself in the ARP cache.

As I progress and view some network dumps, you'll notice that the Ethernet NIC I'm using has a common set of digits that identify it as an SMC NIC. I'm not allowing my Ethernet development board to interface to the Internet, so a MAC of 00EDTP is assigned in the driver code I'm offering. If you plan to use the Ethernet development board commercially or interface it to the Internet, follow the rules for obtaining and using the OUI. For more details, check out the OUI article I wrote for my embedded column in *Circuit Cellar* magazine.

The MAC address is stored in the CS8900A-CQ in the Individual Address (IA) register. The layout of the IA register makes it necessary to place the most significant octet of the MAC address into the least significant octet of the IA register and so on. I defined three MYMACXX variables that put the MAC octets in the right order before they are loaded into the CS8900A-CQ's IA register. To keep you from having to dig out the CS8900A-CQ datasheet, the Ethernet development board MAC and the CS8900A-CQ IA register layout is given in Listing 2.

## REGISTERS

There are numerous CS8900A-CQ registers. The best way to keep up with them is to predefine them and use labels that are readable. Beginning with the PacketPage I/O port definitions, all of the internal CS8900A-CQ registers are listed and defined whether or not they are used.

Labels beginning with "pageport" represent the 16 base PacketPage I/O ports listed in Listing 3. PacketPage internal registers that are accessed using the PacketPage I/O ports are prefixed by "ppage." This area is where the initialization registers are located. Also, you'll find the counters and status registers in the grouping you see in Listing 4.

There are five PacketPage event registers and they are defined by their datasheet names. Some of the CS8900A-CQ registers have names that are simply too long to attempt to code with. I tried to make those register names and their internal bit settings as readable as possible. These long-named registers include:

- PacketPage self control
- PacketPage self status
- PacketPage bus control
- PacketPage bus status
- PacketPage line control
- PacketPage test control
- PacketPage receiver configuration
- PacketPage receiver event
- PacketPage receiver control
- PacketPage transmit configuration
- PacketPage transmit event
- PacketPage transmit command
- PacketPage buffer configuration

The aforementioned registers and register sets can be simplified into six major categories, bus interface registers, status and control registers, initiate transmit registers, address filter registers, receive frame location, and transmit frame location.

The six categories, including all of their subsets, can be found within the 4K area called PacketPage.

## BUS INTERFACE REGISTERS

The bus interface registers are primarily used for interfacing the CS8900A-CQ to a real ISA bus. Be-

cause there is no place for an ISA connector on the PIC or the Ethernet development board, not much of the bus interface register's functionality is used. The ppageBaseIO, which is part of the bus interface register group, defaults to 0x300, which is the Ethernet development board's selected I/O base address. So, you don't have to set it with code (good thing). You could use the ppagePID register to determine which level of CS8900A-CQ is glued to your printed circuit board, but that is just a waste of code, memory space, and CPU cycles for this application.

**Listing 1**—As you'll see later, this reads as C0 A8 01 96 in the Sniffer and PIC IP header umps.

```

;*****
;*      IP ADDRESS DEFINITION
;*      YOU MAY CHANGE THIS TO ANY VALID IP ADDRESS
;*
;*****
#define MYIPU  D'192' ;most significant octet of IP address
#define MYIPH  D'168'
#define MYIPM  D'1'
#define MYIPL  D'150' ;least significant octet of IP address

```

**Listing 2**—The MAC address is also known as the physical address. The MAC and IP addresses are used to make NICs unique and repeatedly reachable on a network.

```

;*****
;*      HARDWARE (MAC) ADDRESS DEFINITION
;*      YOU MAY CHANGE THIS TO ANY VALID MAC ADDRESS
;*
;*****
#define MYMAC0  0 ;most significant octet of MAC address
#define MYMAC1  0
#define MYMAC2  'E'
#define MYMAC3  'D'
#define MYMAC4  'T'
#define MYMAC5  'P' ;least significant octet of MAC address

;*****
;*      INDIVIDUAL ADDRESS LAYOUT IN CS8900
;* MYMAC      5      4      3      2      1      0
;* CS REG |0X15D|0X15C|0X15B|0X15A|0X159|0X158|
;*      | P | T | D | E | O | O |
;*****

#define MYMAC10 (MYMAC1 << 8 | MYMAC0)
#define MYMAC32 (MYMAC3 << 8 | MYMAC2)
#define MYMAC54 (MYMAC5 << 8 | MYMAC4)

```

**Listing 3**—Note that each PacketPage I/O port is 16-bits wide. To read data from pageport\_RxTxData0 requires two reads, one from location 0x00 (low-order byte) and another from location 0x01 (high-order byte). All of the offsets in this listing are the low-order byte address.

```

;*****
;*      PacketPage I/O Port Definitions
;*
;*
;*
;*****
#define pageport_RxTxData0 00 ;Receive/Transmit data Port 0
#define pageport_RxTxData1 02 ;Receive/Transmit data Port 1
#define pageport_TxCmd     04 ;Transmit Command
#define pageport_TxLen     06 ;Transmit Length
#define pageport_ISQ       08 ;Interrupt Status Queue
#define pageport_Ptr       0A ;PacketPage Pointer
#define pageport_Data0     0C ;PacketPage Data Port 0
#define pageport_Data1     0E ;PacketPage Data Port 1

```

CS8900A-CQ 8-bit mode does not allow the use of CS8900A-CQ interrupts, EEPROM, or DMA. So, you must poll the CS8900A-CQ INTRQX line to sense transmit, receive, or error activity. For the polling to be realized, you must set up one of the INTRQX lines to transmit activity status to the PIC16F877's input pin. The application will use the INTRQ0 line, which is defined in the ppageINT register using the mask for INTRQ0 (0x00), as shown in Listing 5. After the interrupt is defined and enabled, you can turn on the CS8900A-CQ transmitter and receiver.

## STATUS AND CONTROL REGISTERS

These registers are subdivided into two groups—configuration and control registers and status and event registers. Configuration and control registers determine how frames are transmitted and received. In addition, configuration and control registers determine which frames will be sent and received and which events will cause an interrupt to be sent to the PIC16F877.

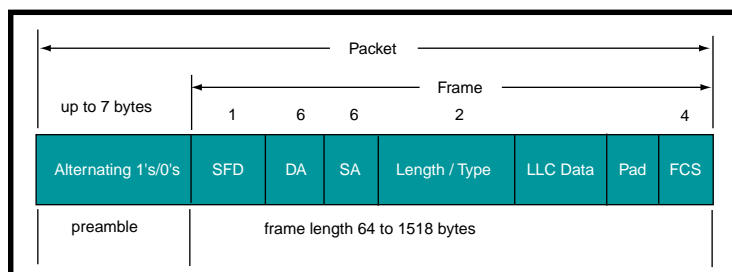
The Ethernet development board uses many of the configuration and control registers early in the code to set up various areas that deal with receive and transmit functions. For instance, in Listing 6 the InitChip

routine uses definitions from the *ppageRxCFG* and *ppageTxCFG* to generate an interrupt when:

- a frame is received with no errors (*RXCFCG\_RX\_OK\_IE*)
- a packet is completely transmitted (*TXCFG\_TX\_OK\_IE*)
- a late (out-of-window) collision occurs (*TXCFG\_OUT\_WIN\_IE*)
- a transmission takes longer than 26 ms (*TXCFG\_JABBER\_IE*)
- a collision or number of collisions occur (*TXCFG\_ALL\_IE* bit 0x0B)
- sixteen collisions occur (*TXCFG\_16\_COLL\_IE*)

If you take apart the control register (*ppageRxCTL*), you find that:

- The CS8900A-CQ accepts frames with correct CRC and length only (*RXCTL\_RX\_OK\_A*).
- The Destination Address in the packet header must match the IA address found in the *ppageIA* (*RXCTL\_IND\_A*) register.
- Broadcast frames with a destination



**Figure 1**—Everything inside the frame area is supplied by you with the exception of the padding and CRC.

address of FFFFFFFF hexadecimal are accepted (*RXCTL\_BCAST\_A*).

## INITCHIP

Continuing our observation of InitChip, it is obvious that the ether will be 10BaseT (*LINECTL\_10BASET*), and the CS8900A-CQ will pump encoded bits onto the ether in full duplex mode (*TESTCTL\_FDX*).

References to the status and control registers can be found throughout the Ethernet development board code because their job is to report the status of transmitted and received frames. Status and control register activity can also exist in situations where you need to know how the CS8900A-CQ feels. Listing 7 is the CS8900A-CQ reset sequence.

The reset bit within *ppageSelfCTL* is set and a 10-ms delay is implemented to allow the CS8900A-CQ to calibrate its on-chip analog circuitry. The *ppageSelfCTL* *RESET* bit (*SELFCTL\_RESET*) is a bit that acts once. That is, it is set and cleared automatically by the action it initiates and, thus, can only kick off an operation one time.

While I'm on the subject of the self-control register, the *SELFCTL\_RESET* mask also determines how the link LED will operate. Obviously, you turn on the link LED function with the *SELFCTL\_RESET* mask. After 10 ms, a bit check is made on the INITD bit in the self status register (*SELFSTAT\_INIT\_DONE\_BIT*). When this bit clears, the global CS8900A-CQ reset is complete.

The last status and control area I will examine here is the bus status register, *ppageBusStatus*. To gain transmit buffer area, the host must bid for transmit space on the CS8900A-CQ.

*BUSSTA\_RDY4TXNOW\_BIT* signals the host that the CS8900A-CQ is ready and willing to accept a frame from the host for transmission. The complex coding needed to affect this is shown in Listing 8.

## INITIATE TRANSMIT REGISTERS

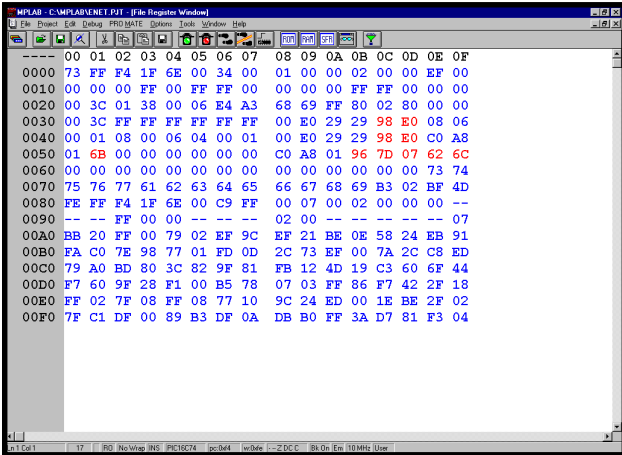


Photo 1—My PICMASTER may be old, but it can still twiddle bits.

These registers tell the CS8900A-CQ how to transmit the next packet. The first four definitions in Listing 9 are values that can be used to instruct the CS8900A-CQ to start transmitting after a certain number of bytes have been transferred to the CS8900A-CQ transmit buffer area. For instance, *TXCMD\_AFTER\_5* says start transmitting after five bytes are in the CS8900A-CQ buffer. In the code snippet that follows the defines, you see that I used *TXCMD\_AFTER\_ALL*, which translates to wait for every byte sent to be loaded into the CS8900A-CQ transmit buffer before transmitting.

Let's talk about the high-order byte of the *TXCMD* register. If the *TXCMD\_NO\_CRC* bit is included in the mask, the CS8900A-CQ generated CRC value is not appended to the transmission. Because you are appending a CRC and the *TXCMD\_NO\_PAD* bit is clear, any packet less than 60 bytes will be padded to 60 bytes and the CRC will be appended to the transmission.

The reason for control is to allow illegal packets to be transmitted. This illegal activity could be used in the development or test phase of a design to test other Ethernet devices or Ethernet data capture software like the Sniffer. The *TXCMD\_FORCE* bit flushes any frames waiting to be transmitted and aborts the transmission of any frame in the process of being transmitted.

## ADDRESS FILTER REGISTERS

The CS8900A-CQ comes equipped with a destination address register

(DA). This address filter determines which frames will pass the CS8900A-CQ receive portal and be placed in the CS8900A-CQ receive buffer. If you recall from recent articles I've written in *Circuit Cellar* on the destination address, the first bit of the physical address should be zero. The reason for this is that if the first bit is not a zero, the address is not a physical address. The IA on the incoming frame must match the physical address in the CS8900A-CQ IA register (see Listing 2).

If the first bit of the incoming DA is a 1, then the frame is a multicast frame, and the address is logical not

physical. The CS8900A-CQ uses a hash technique to determine if it should accept the incoming multicast frame. Take a look at Listing 6 under PacketPage receiver control register bit definitions, and you will find that the multicast (*RXCTL\_MCAST\_A*) bit is not set. Thus, your implementation of the Ethernet development board will ignore multicast addresses.

Another look at Listing 6 indicates that, in addition to physical addresses, the Ethernet development board CS8900A-CQ also accepts broadcast addresses. Unless you know every physical and IP address of every host you want to communicate with and all of those remote hosts know your IP and MAC addresses (highly unlikely), you had better be able to resolve a broadcast message.

## RECEIVE AND TRANSMIT FRAME LOCATIONS

These areas are used to transfer Ethernet frames between the CS8900A-CQ and PIC16F877. Only

Listing 4

```

;*****
;*      PacketPage Internal Register Definitions
;*
; **
;*****
#define INTRQ0          0
#define ppageEISA      0000    ;EISA Registration number of CS8900
#define ppagePID       0002    ;Product ID Number
#define ppageBaseIO    0020    ;I/O Base Address
#define ppageINT       0022    ;Interrupt number (0,1,2, or 3)
#define ppageBaseMemory 002C ;20-bit Memory Base address register
#define ppageRxCfg     0102    ;Receiver Configuration
#define ppageRxCTL      0104    ;Receiver Control
#define ppageTxCFG     0106    ;Transmit Configuration
#define ppageBufCFG    010A    ;Buffer Configuration
#define ppageLineCTL   0112    ;Line Control
#define ppageSelfCTL   0114    ;Self Control
#define ppageBusCTL    0116    ;Bus Control
#define ppageTestCTL   0118    ;Test Control
#define ppageISQ       0120    ;Interrupt status queu
#define ppageRxEvent   0124    ;Receiver Event
#define ppageTxEvent   0128    ;Transmitter Event
#define ppageBufEvent  012C    ;Buffer Event
#define ppageRxMiss    0130    ;Receiver Miss Counter
#define ppageTxColl    0132    ;Transmit Collision Counter
#define ppageLineStatus 0134    ;Line Status
#define ppageSelfStatus 0136    ;Self Status
#define ppageBusStatus 0138    ;Bus Status
#define ppageTxCmd     0144    ;Transmit Command Request
#define ppageTxLength  0146    ;Transmit Length
#define ppageIA        0158    ;Individual Address
#define ppageRxStatus  0400    ;Receive Status

```

one receive and transmit frame is available to the PIC16F877 at any time, and the space for each frame is dynamically allocated by the CS8900A-CQ. Of course, all of the data is transferred between the PIC's RAM and the CS8900A-CQ's 4K buffer area via the PacketPage I/O ports.

That about covers what will concern you with the CS8900A-CQ register set. As you can see, there are a multitude of variables and configurations that can be had by setting and clearing the right set of bits. If I've left anything out, it will surface as I walk you through receiving and transmitting frames with the CS8900A-CQ.

### CS8900A-CQ TRANSMIT AND RECEIVE OPERATIONS

Take a look at Listing 10 before you read on. Listing 10 is a list of the PIC port and pin definitions. Port B of the PIC16F877 is used for both the programming of the part and CS8900A-CQ I/O control. The data bus is synthesized on Port C, and a pseudo-4-bit address bus takes up residence in the lower half of Port D. The first 48 bytes of PIC RAM are the usual counters, flags, pointers, and scratch registers that make up most PIC applications. The actual frame data buffer area begins at the PIC's register address 0x32.

Listing 11 consists of a tight loop that is polling the CS8900A-CQ INTRQ line followed by code to determine if an ARP request has been received. ServiceISQ dispatches rou-

tines based on the protocol contents of their frames.

To receive a frame, the CS8900A-CQ must accept it using the IA register and destination address filter. After the packet is accepted, the preamble and start-of-frame delimiter are ignored and the bits following the SFD are loaded into the CS8900A-CQ receive buffer area. To refresh your understanding of packets versus frames, I've included Figure 1 for your viewing pleasure. *RXC\_CFG\_RX\_OK\_IE* (0x0100) is loaded into the RxCFG register, and the BufferCRC bit (0x0800) is clear. This means you do not include the CRC in the receive buffer contents or the length calculations.

Beginning at the top of Listing 12, after the PIC16F877's Port C is put into input mode, the receive status is read high-order byte first. Notice that the receive status is not stored for later use. Even so, it still must be read. If you're interested in the receive status, it can be determined by reading the RxEvent register. This is a required read no matter what you do with the data.

### THE NEXT READ

The next required read brings in the length of the frame that will be trans-

ferred from the CS8900A-CQ to the PIC16F877. This is the total length of bytes starting with the DA and ending with the last byte before the 4-byte CRC value. The frame length value is stored in the first two locations of your PIC frame buffer area (0x30 and 0x31). The CS8900A-CQ receive buffer is then read low byte to high byte into the PIC16F877 RAM for the length of the buffered frame.

The PIC's frame buffer area extends from 0x32 to 0x7F, which works out to be 78 bytes. There is plenty of leftover RAM in the PIC16F877 in other banks that can be used to make a larger buffer, but for this application, 78 bytes are plenty. This means you must only accept frames of 78 bytes or less or only fill your PIC buffer area with 78 bytes maximum and ignore the rest.

Well, there are no bits to limit the incoming frame length, but you can throw away any bytes beyond 78. In Listing 12 under *GetFrameData*, you will find a flag bit called *bitbucket*. If incrementing the FSR (pointing to the next available PIC RAM register) puts the FSR over 0x7F, the rest of the remaining bytes are read from the CS8900A-CQ receive buffer and dumped into the bit bucket.

Instead of throwing away bytes beyond address 0x7F in the CS8900A-CQ receive buffer, you could read the RxEvent register after transferring byte 0x7F and trigger the CS8900A-CQ to move on to the next frame in its buffer. But because you're reading the entire frame, there's no need to do that. My method also ensures that the

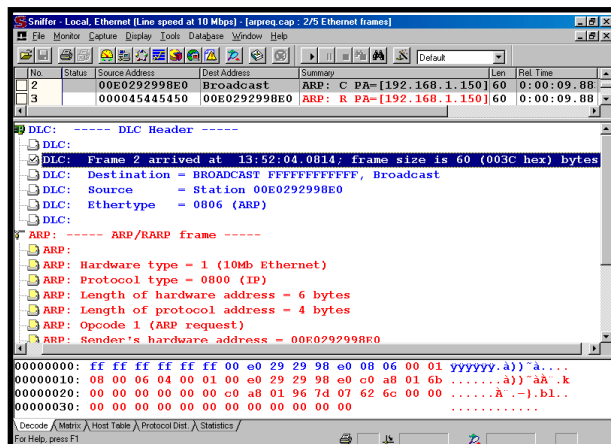


Photo 2—Not only is frame content broken out, it's broken out in coordinating colors.

Listing 5—The first line defines the INTRQ line as the selected output interrupt line, and the rest of the code enables the interrupt and fires up the transmitter and receiver sections of the CS8900A-CQ.

```

;*****
;*      Enable INTRQ
;*
;*
;*
;*****

WPP  ppageINT,INTRQ
RPP  ppageBusCTL
bsf  data_H,BUSCTL_INT_ENBL_BIT
call  PPWrite

RPP  ppageLineCTL
bsf  data_L,LINECTL_RX_ON_BIT
bsf  data_L,LINECTL_TX_ON_BIT
call  PPWrite

```

entire frame is transferred or read whether or not all of the data is used.

Let's assume a host sent an ARP request to the Ethernet development board and all of the code in Listing 12 was executed successfully. Photo 1 shows how it looks inside the PIC16F877. Photo 2 shows what the Sniffer sees in the same situation and same frame.

The frame length is displayed at locations 0x30 and 0x31 in the PIC dump. The Sniffer shot simply tells you the frame size of the area I highlighted. Starting at location 0x32 in the PIC dump photo, you can follow along byte for byte in the Sniffer hex dump area at the bottom of Photo 2. Take another look at Figure 1 as I walk you through this frame. You can correlate the PIC and Sniffer data with the Figure 1 frame sub-areas. Because the Sniffer gives details not obvious in the PIC dump, use the Sniffer shots as your pointer to the relative (and identical) data in the PIC frame dump.

## BROADCAST

In the DLC area of the Sniffer screen shot (see Photo 2), a frame size of exactly 60 tells you that there's probably been some padding of the base data to meet the minimum frame requirements. The highlighted areas (hex dump and DLC area) of Photo 3 show the DA filled with "f"s indicating a broadcast frame was received. This is obvious to the Sniffer because it blatantly says "BROADCAST" twice on the same line.

The source address highlighted in Photo 4 belongs to a PC running an SMC NIC under Windows 98.

Photo 5 highlights an element of the Ethernet header that the Ethernet development board firmware will key on to process the rest of the frame. These two bytes (0806) tell the Ethernet development board code that the frame transferred from the CS8900A-CQ to the PIC is an ARP frame. The rest of the ARP code in Listing 13 checks every field you see in the Sniffer shot (see Photo 6). If it all matches (which it does), then an ARP request has been tendered and the Ethernet development board must

**Listing 6—Definitions that end with `_BIT` are used to directly set, clear or test that bit position in the register. `TXCFG_ALL_IE` is the sum of the masks for all of the bits in the Transmit Configuration Register, `ppageTxCFG`.**

```

;*****
;*      PacketPage Line Control Bit Definitions
;*
;*****
#define      LINECTL_RX_ON_BIT          6
#define      LINECTL_RX_ON             0040
#define      LINECTL_TX_ON_BIT          7
#define      LINECTL_TX_ON             0080
#define      LINECTL_AUI_ONLY           0100
#define      LINECTL_10BASET           0000

;*****
;*      PacketPage Test Control Bit Definitions
;*
;*****
#define      TESTCTL_DIS_LT             0080
#define      TESTCTL_ENDEC_LP           0200
#define      TESTCTL_AUI_LOOP           0400
#define      TESTCTL_DIS_BKOFF          0800
#define      TESTCTL_FDX                4000

;*****
;*      PacketPage Receiver Configuration Bit Definitions
;*
;*****
#define      RXCFG_SKIP_BIT             6
#define      RXCFG_SKIP                 0040
#define      RXCFG_RX_OK_IE             0100
#define      RXCFG_CRC_ERR_IE           1000
#define      RXCFG_RUNT_IE              2000
#define      RXCFG_X_DATA_IE            4000

;*****
;*      PacketPage Receiver Control Register Bit Definitions
;*
;*****
#define      RXCTL_SETUP
(RXCTL_RX_OK_A|RXCTL_IND_A|RXCTL_BCAST_A)
#define      RXCTL_RX_OK_A              0100
#define      RXCTL_MCAST_A              0200
#define      RXCTL_IND_A                0400
#define      RXCTL_BCAST_A              0800
#define      RXCTL_CRC_ERR_A            1000
#define      RXCTL_RUNT_A                2000
#define      RXCTL_X_DATA_A             4000

;*****
;*PacketPage Transmit Configuration Register Bit Definitions
;*
;*****
#define      TXCFG_LOSS_CR_S_IE          0040
#define      TXCFG_SQE_ERR_IE           0080
#define      TXCFG_TX_OK_IE             0100
#define      TXCFG_OUT_WIN_IE           0200
#define      TXCFG_JABBER_IE            0400
#define      TXCFG_16_COLL_IE           8000
#define      TXCFG_ALL_IE                8FC0

;*****
;*Load the CS8900 Basic Parameters
;*10BaseT/Full Duplex/RxOK interrupt/accept broadcast and
;*individual addresses/Tx interrupts enabled
;*****
InitChip
WPP      ppageLineCTL,LINECTL_10BASET
WPP      ppageTestCTL,TESTCTL_FDX
WPP      ppageRxCFG,RXCFG_RX_OK_IE
WPP      ppageRxCTL,RXCTL_SETUP
WPP      ppageTxCFG,TXCFG_ALL_IE

```

assemble and transmit an ARP reply. I threw in Photo 7 to delineate the padded bytes.

## ARP

If you look at the top of any of the Sniffer screen shots, you'll notice that the second event is an ARP broadcast, and the third event is the ARP reply from physical address 00EDTP. Photo 8 looks similar to the ARP request screen shot with the exception of an additional physical address supplied by the Ethernet development board (000045445450) and an Opcode 2 in the ARP frame defining the frame as an ARP reply. The ARP process is somewhat like an algebra problem. You use the knowns (IP addresses) to solve for the unknowns (physical addresses).

If you start at the DA and count the total number of bytes in the ARP reply, you'll come up with 42. That's 60 bytes total frame length minus 18 bytes of padding. In Listing 14, Port C is commanded to output 0x2A and 42 bytes of ARP response buffer area are requested from the CS8900A-CQ. This is called a bid in the CS8900A-CQ datasheet. After the CS8900A-CQ allocates the space and sets the *RDY4TXNOW\_BIT*, the bits of the ARP reply flow out of the PIC16F877 RAM in the order of the subareas you see in Figure 1 and Photo 8, starting with the DA. After all of the bytes are collected in the CS8900A-CQ transmit buffer, the CS8900A-CQ generates a preamble that is immediately followed by an SFD. And assuming no collision occurs, the ARP reply hits the ether followed by a CS8900A-CQ-generated CRC.

## NO CHEATING

If you're reading ahead, and I expect you are, you already know that the fourth event is unmasked in Photo 8. You also

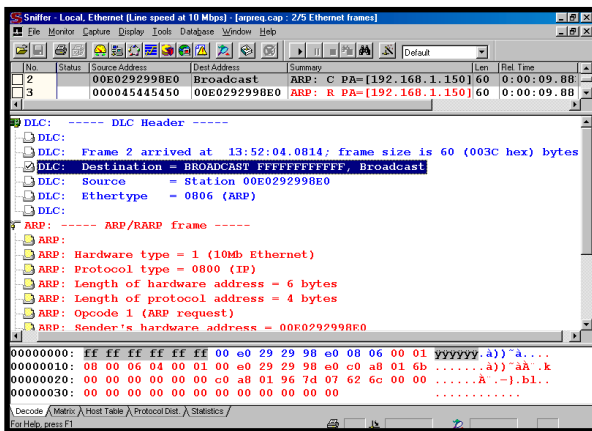


Photo 3—Notice the highlighted areas of the text area correspond with highlighted areas in the hex dump area.

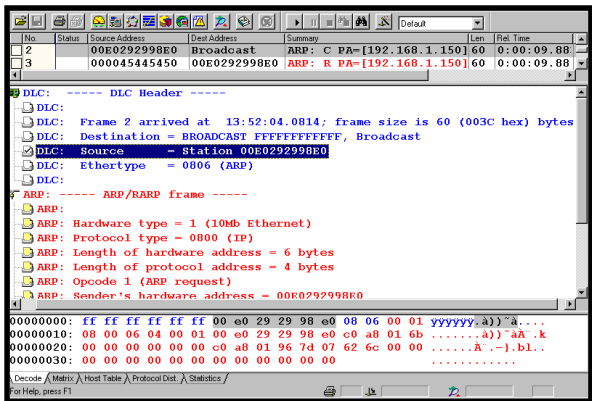


Photo 4—

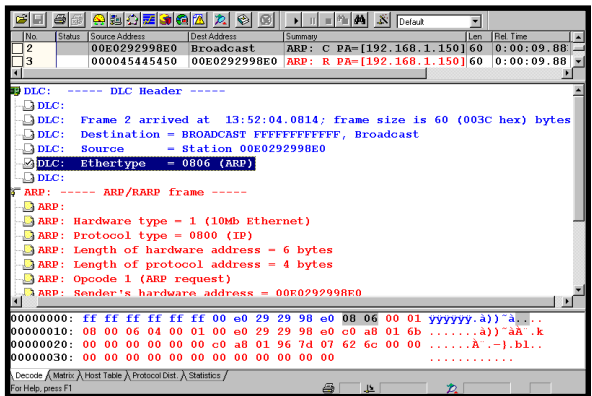


Photo 5—

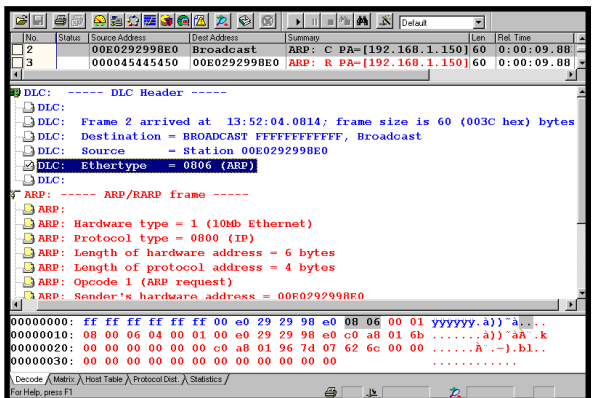


Photo 6—This shot is like being lost at sea and thirsty. Everything around you spells "water" but one essential ingredient is missing. In this case it's the target hardware address.

know it is a UDP transaction between the PC called SPEADY and the Ethernet development board. I've already presented lots of words on IP protocol in other *Circuit Cellar* articles. (Is that a low-down way to get you to read my stuff, or what?) So, I'm not going to go into detail about what the IP fields mean and do here. I do want to point out that the Sniffer fields match the PIC IP field layout in the Ethernet development board code you see in Listing 15.

Photo 9 highlights the DLC-type field in the hex dump area containing 0800, which signifies an IP frame. Following the logic in Listing 16, you see that 0806 caused an ARP replay frame to be built and transmitted, and 0800 sends the code down the path to process an IP frame. After you're headed in the IP frame direction, there are still decisions to be made as to where the code will branch to next. The protocol field within the IP header determines that. In Photo 9, the protocol value is 0x11, or 17 decimal. That means that UDP header information and data is contained within the IP data area.

The Ethernet development board firmware examines the UDP destination port to determine how to handle the UDP data. In this case, the destination is Port 7. When UDP Port 7 is addressed in the Ethernet development board firmware, an echo operation is performed. That is, the data, in this case an "a" (0x61), is echoed back to the source host's port 5002. You can see from the fifth event at the top of Photo 9 that this entails swapping the source and destination ports, IP addresses, and physical addresses. This swapped frame is returned to the sender's UDP port with the UDP data untouched. Compare Photo 9 and Photo 10 to realize the concept.

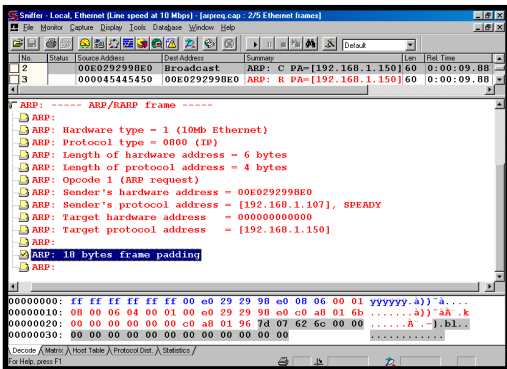


Photo7—

Port 7 and a basic IP-address-selectable and UDP-Port-selectable on/off switch mechanism for Port A of the Ethernet development board's PIC16F877. For those who wish to play, I'll include the VB project files . You can get an idea of the functionality of the VB-based test panel by looking at Photo 14.

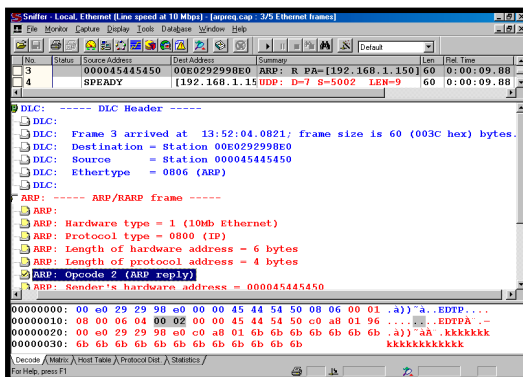


Photo 8—This shot is a steak dinner perfectly cooked.

## TIGER WOODS' PUTTER

ARP and UDP are the basic set of protocols the Ethernet development board uses to move data over the ether. If you know absolutely nothing about Internet protocols, you've probably heard someone talking about "pinging" someone else's computer. Ping is actually an application of sorts that is based on the ICMP protocol. It's a quick and nasty way to establish that a remote host is online. I won't go into detail because Photos 11 and 12 pretty much tell the story. But, while I'm already on the subject, I can tell you ping is incorporated in the Ethernet development board firmware. Photo 13 shows the ping command being given from a Windows98 DOS screen to the Ethernet development board running under PICMASTER control.

## WHAT'S THE POINT?

OK, I've gone to all this trouble to show you how you can build the equipment and operate on the Ethernet with a PIC and a handful of various other components. What good is it if you can't do something with the data that is encapsulated in the Ethernet packets?

I'm sure you'll find a need for Ethernet connectivity in some of your projects, and to help you get started, I'm including a simple Visual Basic (VB) program that sends and receives data using the UDP protocol. You can select the IP address and UDP port and transfer data between the PC and other uniquely addressed Ethernet development boards on a network.

This little program also has a built-in echo function dedicated to UDP

## AND IT PROGRAMS, TOO!

I haven't said much about the PIC16F877 control and programming code . The program feeds on hex and

code files from the standard Microchip MPLAB environment. Just write your code, assemble it, and make sure the hex output and code output files are in the same directory. Then, load the hex

Listing 7—There's also a busy bit for the EEPROM in the Self Status Register. We aren't concerned about it because the CS8900A-CQ doesn't support the use of the EEPROM in 8-bit mode.

```

;*****
;
;*   Reset the CS8900
;*
;*
;*****
                WPP      ppageSelfCTL, SELFCTL_RESET
CS_Reset
                movlw   6
                movwf   ARG0
holdit3
                movlw   19
                movwf   ARG1
holdit2
                movlw   6E
                movwf   ARG2
holdit1
                decfsz  ARG2, F
                goto   holdit1
                decfsz  ARG1, F
                goto   holdit2
                decfsz  ARG0, F
                goto   holdit3

RPP      ppageSelfStatus
btfss   data_L, SELFSTAT_INIT_DONE_BIT
goto    CS_Reset

```

Listing 8

```

UDPrdy4TX
                RPP      ppageBusStatus
                btfss   data_H, BUSSTA_RDY4TXNOW_BIT
                goto    UDPrdy4TX

```

Caption: This register and bit has to be polled as the Rdy4TxNOW bit does not generate an interrupt.

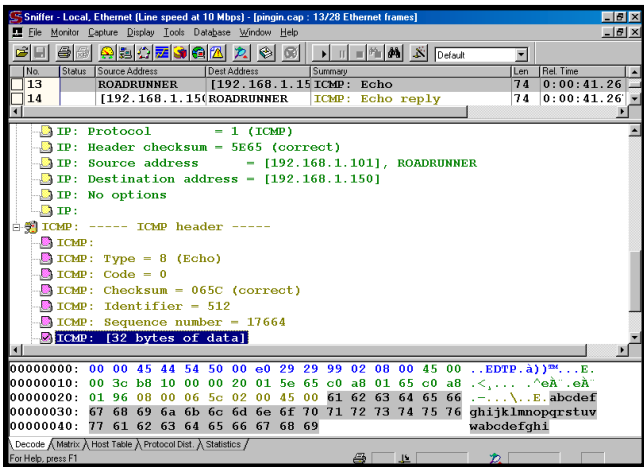


Photo 11

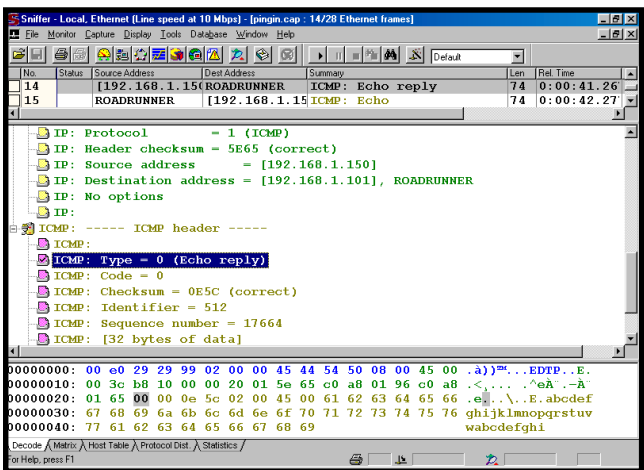


Photo 12

**Listing 9**—TXCMD\_ONE\_COLL terminates transmission after a single collision. If this bit is clear, the normal 16 collisions will be tolerated before terminating the transmission.

```

*****
;
; *   PacketPage Transmit Command Register Bit Definitions
; *
; *
; *
; *****
#define      TXCMD_AFTER_5           0000
#define      TXCMD_AFTER_381        0080
#define      TXCMD_AFTER_1021       0040
#define      TXCMD_AFTER_ALL        00C0
#define      TXCMD_FORCE             0100
#define      TXCMD_ONE_COLL         0200
#define      TXCMD_NO_CRC           1000
#define      TXCMD_NO_PAD            2000

movlw TXCMD_AFTER_ALL
WpppL pageport_TxCmd
movlw 0
WpppH pageport_TxCmd

movfw packetstart+enetpacketLenL
WpppL pageport_TxLen
movfw packetstart+enetpacketLenH
WpppH pageport_TxLen

```

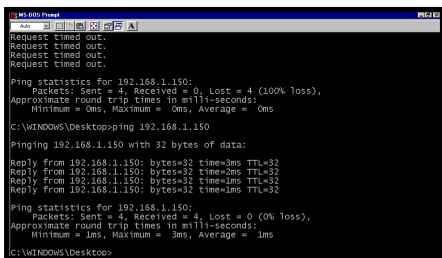
file and hit the Program button. It's a simple program to manipulate.

Over the past few articles I've covered quite a bit of territory. If you run up a tree and can't get down, e-mail me or check out current and back issues of *Circuit Cellar* and *Circuit Cellar Online* for the rest of the articles on embedded Ethernet. I'll leave you with a screen shot of the PIC16F877 programmer panel (see Photo 15).

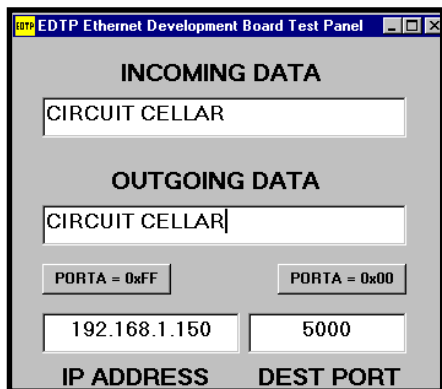
In my column in *Circuit Cellar*, I always remind you that it doesn't have to be complicated to be embedded. Online, it's not complicated and it's embedded.

*Fred Eady has more than 20 years of experience as a systems engineer. He has worked with computers and communication systems large and small, simple and complex. His forte is embedded-systems design and communications. Fred may be reached at fred@edtp.com.*

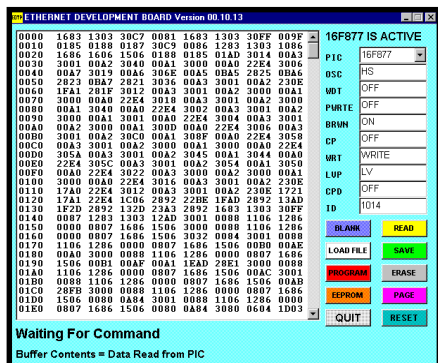
Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, subscribe@circuitcellar.com or www.circuitcellar.com/subscribe.htm.



**Photo 13**—This ping ran against the code running on the PICMASTER at 10 MHz. Notice at the top of the screen, I had not yet started the code when the ping failed.



**Photo 14**—Come now, is there any other magazine compared to this one? The IP address and destination port windows are input-driven. So is the outgoing data window. The two buttons send a 0xFF or 0x00 word to the destination port of the remote



**Photo 15**—Leave out the CS8900A-CQ and supporting components, run this program with the Ethernet development board attached, and you have a PIC16F874/877 programmer for PLCC packages.

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, subscribe@circuitscellar.com or www.circuitscellar.com/subscribe.htm.

**Listing 10**—Nothing fancy here. Just standard PIC stuff.

```

;*****
;*
;* PIC16F87X PortB Bit Definitions
;*
;*****
;
; RB7 used by ICSP/LVP
; RB6 used by ICSP/LVP
#define IOR 5
#define IOW 4
; RB3 used by ICSP/LVP
#define AEN 2
#define RESET 1
#define INTR 0

;*****
;*
;* PIC16F87X PortC Bit Definitions
;*
;*****
;*****
#define SD0 0
#define SD1 1
#define SD2 2
#define SD3 3
#define SD4 4
#define SD5 5
#define SD6 6
#define SD7 7

;*****
;*
;* PIC16F87X PortD Bit Definitions
;*
;*****
;*****
#define SA0 0
#define SA1 1
#define SA2 2
#define SA3 3

;*****
;*
;* PIC16F87X RAM (Register) Definitions
;*
;*****
;*****
#define data_H 20
#define data_L 21
#define poffsetH 22
#define poffsetL 23
#define ARG3 24
#define ARG2 25
#define ARG1 26
#define ARG0 27
#define BARG1 28
#define BARG0 29
#define cntr 2A
#define scratch_H 2B
#define scratch_L 2C
#define frameflags 2D
#define len_H 2E
#define len_L 2F
#define packetstart 30

```

**Listing 11**—The arp reply is built on the fly. All other frame processing is taken care of by swapping and extracting data areas within the respective protocol type routines.

```

;*****
;*
;* MAIN SERVICE LOOP
;*
;*****
;*****
chkint

    btfss PORTB,INTR
    goto chkint

    call ServiceISQ
    btfss framereceived
    goto chkint
    bcf framereceived
    btfss txarp
    goto chkint
    bcf txarp
    call XMIT_ARP
    goto chkint

```

**Listing 12**—Data read under *GetReal* less than or equal to 0x7E bytes out of the CS8900A-CQ buffer is retained. All of the rest is trashed.

```

;*****
;*      Receive a Frame
;*
;*****
RECEIVE_DATA
    banksel TRISC
    movlw OFF
    movwf TRISC
    banksel PORTC

    bcf    bitbucket

    RpppH  pageport_RxTxData0
    RpppL  pageport_RxTxData0

    movlw  packetstart+2
    movwf  FSR

    RpppH  pageport_RxTxData0
    movwf  packetstart+enetpacketLenH
    movwf  len_H
    movwf  data_H
    RpppL  pageport_RxTxData0
    movwf  packetstart+enetpacketLenL
    movwf  len_L
    movwf  data_L

GetFrameData
    btfss  bitbucket
    goto  GetReal
    RpppL  pageport_RxTxData0
    movwf  scratch_L
    RpppH  pageport_RxTxData0
    movwf  scratch_H
    goto  DecDataCnt_0

GetReal
    RpppL  pageport_RxTxData0
    movwf INDF
    incf  FSR,F
    RpppH  pageport_RxTxData0
    movwf INDF
    incf  FSR,F

    movlw 80
    xorwf FSR,W
    bnz   DecDataCnt_0
    bsf   bitbucket
    goto  GetFrameData

DecDataCnt_0
    decfsz len_L,F
    goto  DecDataCnt_1
    tstf  len_H
    bz    FrameIn

DecDataCnt_1
    movlw OFF
    xorwf len_L,W
    bnz   DecDataCnt_2
    decf  len_H,F

DecDataCnt_2
    decfsz len_L,F
    goto  DecDataCnt_3
    tstf  len_H
    bz    FrameIn

DecDataCnt_3
    movlw OFF
    xorwf len_L,W
    bnz   GetFrameData
    decf  len_H,F
    goto  GetFrameData

```

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, [subscribe@circuitcellar.com](mailto:subscribe@circuitcellar.com) or [www.circuitcellar.com/subscribe.htm](http://www.circuitcellar.com/subscribe.htm).

**Listing 13**—*This had better be an ARP request because there's no code in this application to generate an ARP request from the Ethernet Development Board!*

```
FrameIn
    bsf    framereceived

    movlw  08
    xorwf  packetstart+enetpacketTypeH,W
    bnz   IType
    movlw  06
    xorwf  packetstart+enetpacketTypeL,W
    bz    ProcessARP

ProcessARP

    movlw  1
    xorwf  packetstart+arp_hwtype+1,W
    skpz
    return

    movlw  08
    xorwf  packetstart+arp_prtype,W
    skpz
    return

    tstf   packetstart+arp_prtype+1
    skpz
    return

    movlw  06
    xorwf  packetstart+arp_hwlen,W
    skpz
    return

    movlw  04
    xorwf  packetstart+arp_prlen,W
    skpz
    return

    movlw  1
    xorwf  packetstart+arp_op+1,W
    skpz
    return

    movlw  MYIPU
    xorwf  packetstart+arp_tipaddr,W
    skpz
    return

    movlw  MYIPH
    xorwf  packetstart+(arp_tipaddr+1),W
    skpz
    return

    movlw  MYIPM
    xorwf  packetstart+(arp_tipaddr+2),W
    skpz
    return

    movlw  MYIPL
    xorwf  packetstart+(arp_tipaddr+3),W
    skpz
    return

    bsf    txarp
    return
```

*Listing 14—This is basically plugging in the correct values in the correct order and inserting the unknown physical address of the Ethernet Development Board.*

```

;*****
;*      ARP Layout
;*
;*****
#define  arp_hwtype           10
#define  arp_prtype           12
#define  arp_hwlen            14
#define  arp_prlen            15
#define  arp_op                16
#define  arp_shaddr           18
#define  arp_sipaddr           1E
#define  arp_thaddr           22
#define  arp_tipaddr          28

;*****
;*      SEND ARP RESPONSE
;*
;*****
XMIT_ARP
    banksel    TRISC
    clr        TRISC
    banksel    PORTC
    movlw     TXCMD_AFTER_ALL
    WpppL     pageport_TxCmd
    movlw     0
    WpppH     pageport_TxCmd
    movlw     2A
    WpppL     pageport_TxLen
    movlw     0
    WpppH     pageport_TxLen
arprdy4TX
    RPP       ppageBusStatus
    btfs     data_H,BUSSTA_RDY4TXNOW_BIT
    goto     arprdy4TX

    banksel    TRISC
    clr        TRISC
    banksel    PORTC
xmitdestaddr
    movfw     packetstart+enetpacketSrc0H
    WpppL     pageport_RxTxData0
    movfw     packetstart+enetpacketSrc0L
    WpppH     pageport_RxTxData0
    movfw     packetstart+enetpacketSrc1H
    WpppL     pageport_RxTxData0
    movfw     packetstart+enetpacketSrc1L
    WpppH     pageport_RxTxData0
    movfw     packetstart+enetpacketSrc2H
    WpppL     pageport_RxTxData0
    movfw     packetstart+enetpacketSrc2L
    WpppH     pageport_RxTxData0
xmitsrcaddr
    movlw     MYMAC0
    WpppL     pageport_RxTxData0
    movlw     MYMAC1
    WpppH     pageport_RxTxData0
    movlw     MYMAC2
    WpppL     pageport_RxTxData0
    movlw     MYMAC3
    WpppH     pageport_RxTxData0
    movlw     MYMAC4
    WpppL     pageport_RxTxData0
    movlw     MYMAC5
    WpppH     pageport_RxTxData0
xmityplen
    movlw     08

```

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, [subscribe@circuitcellar.com](mailto:subscribe@circuitcellar.com) or [www.circuitcellar.com/subscribe.htm](http://www.circuitcellar.com/subscribe.htm).

**Listing 15**—The UDP bytes all reside inside the IP data area beginning at ip\_data.

```

;*****
;*      IP Header Layout
;*
;*
;*****
#define      IPHEADERLEN      14
#define      ip_vers_len      10      ;IP version and header
length
#define      ip_tos           11      ;IP type of service
#define      ip_pktlen        12      ;packet length
#define      ip_id            14      ;datagram id
#define      ip_frag_offset   16      ;fragment off-
set
#define      ip_ttl           18      ;time to live
#define      ip_proto         19      ;proto-
col (ICMP=1, CP=6,UDP=11)
#define      ip_hdr_cksum     1A      ;header checksum
#define      ip_srcaddr       1C      ;IP address of source
#define      ip_destaddr      20      ;IP address of destina-
tion
#define      ip_data          24      ;guess

```

**Listing 16**—This is the pivotal code that switches the PIC firmware to the correct protocol processing routines.

```

FrameIn
    bsf      framereceived

    movlw   08
    xorwf   packetstart+enetpacketTypeH,W
    bnz     Iptype
    movlw   06
    xorwf   packetstart+enetpacketTypeL,W
    bz      ProcessARP

Iptype
    movlw   08
    xorwf   packetstart+enetpacketTypeH,W
    skpz
    return

    tstf   packetstart+enetpacketTypeL
    bz     ProcessIP
    return

trashtheframe
    RPP     ppageRxCFG
    bsf     data_L,RXCFG_SKIP_BIT
    call    PPWrite
    return

;*****
;*      Process An IP Packet
;*
;*****
ProcessIP
    movlw   PROT_ICMP
    xorwf   packetstart+ip_proto,W
    bz     ProcessICMP
    movlw   PROT_UDP
    xorwf   packetstart+ip_proto,W
    skpz
    return

ProcessUDP
    tstf   packetstart+UDP_destport
    skpz
    goto   noecho
    movlw   7
    xorwf   packetstart+(UDP_destport+1),W
    skpz

```

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, subscribe@circuitscellar.com or www.circuitcellar.com/subscribe.htm.