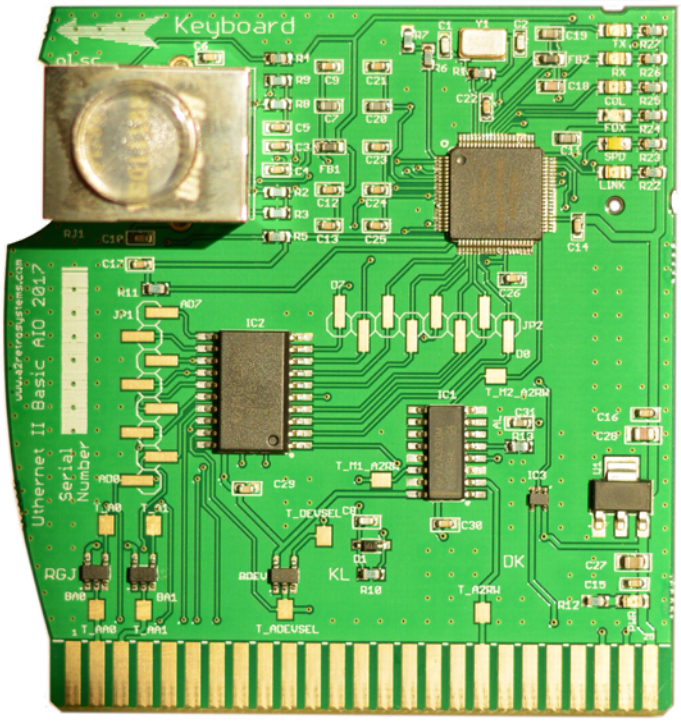
a2RetroSystems presents

UTHERNET II

Ethernet card for the Apple II series



User's and Programmer's Manual

UTHERNET II

©2018 a2RetroSystems
Uthernet II is a trademark of a2RetroSystems.

Disclaimer of all Liability

This card was designed and built for "hobby" computing purposes only. It is strictly forbidden to use it to launch spy satellites, track ICBM missiles or air traffic control paper airplanes. Having said that, you can use it for whatever you want so long as you don't bug me about it. It's your problem not mine.

Warranty and Return Information

You may return the Uthernet II card for any reason within 90 days of receiving it. This should allow you enough time to evaluate its compatibility with your system. I guarantee your Uthernet II card to be free of defects under normal use for a period of one year from the date you receive the product. This means that if the card fails, and you have treated it properly, I will repair, replace, or refund your money at my discretion, to be determined by me on a case by case basis.

If you want to return the product under warranty, please contact me via e-mail to discuss return arrangements. Include your name and the serial number on the front of the card. It is your responsibility to get the product you are returning back to my door. I will not be responsible for lost shipments. Please choose shipping methods and insurance as you deem necessary.

Web site: http://www.a2retrosystems.com

Email: info@a2retrosystems.com

Manual and Uthernet II logo by D. Finnigan.
17 Nov 18

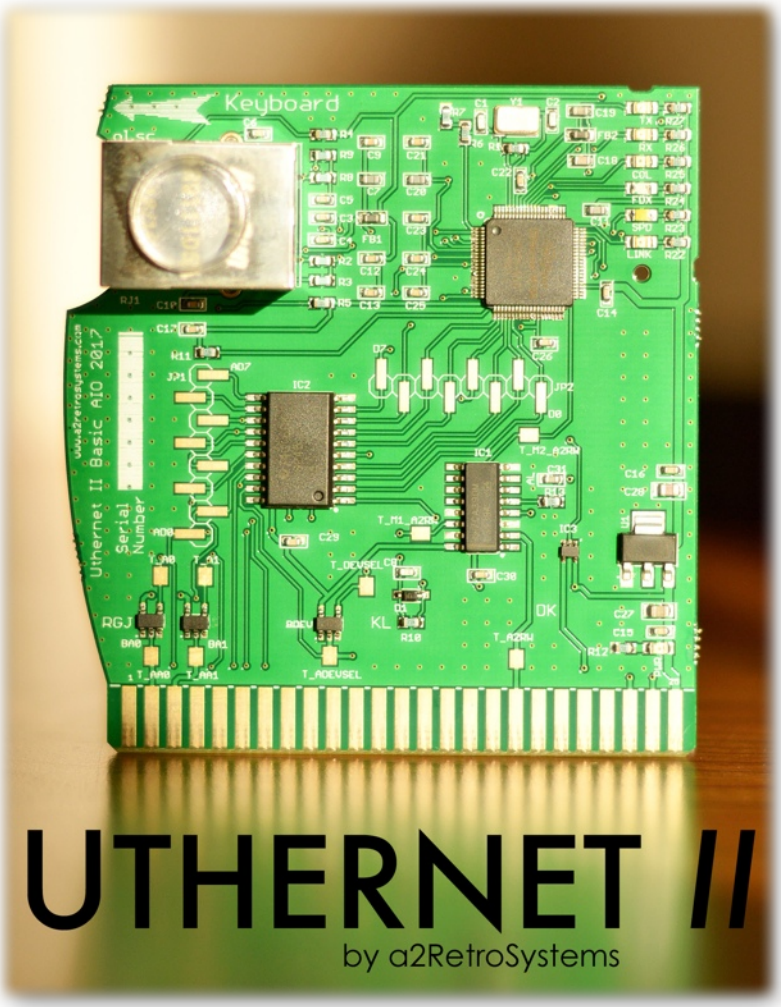
Contents

Presenting the Uthernet II

Congratulations on purchasing the best Ethernet card for the Apple II series! The Uthernet II by a2RetroSystems was the result of over a year of research, testing and development to bring you the best possible network interface card for the Apple. With Uthernet II, your Apple IIGS or enhanced Apple IIe can connect to a home network, or even the Internet to surf the Web, send and receive email, transfer files with FTP, chat with IRC, and so much more.

If you are upgrading from the original Uthernet card, you will be pleased to find that the Uthernet II offers compatibility with nearly all of your existing software, while providing exciting new features that will enable faster communication and more powerful programs. In most cases, only an upgraded driver for the Uthernet II is needed.

Programmers, be sure to study Chapter 4, as you will be thrilled by what the Uthernet II offers over the old Uthernet in terms of programming capabilities.



Installation and Setup

The Uthernet II can be installed in any model of Apple II with slots, though enhanced Apple IIe and Apple IIGS are recommended. Uthernet II may not be fully compatible with older models, the integer Apple II and Apple II Plus. The Uthernet II does not have any on-board firmware so it may be installed in any standard slot.

Installing the Uthernet II

To install the Uthernet II card, begin by following the standard safety precautions: power down your Apple and discharge any static electricity build-up.

Remove the cover of your Apple, and gently insert the card so that the large gray Ethernet port is facing the front (or keyboard) of the Apple. This detail is essential: the small chips and LEDs must be facing away from the Apple power supply. When pressing the Uthernet II into the slot, press only on the circuit board, not the gray Ethernet port.

Next, carefully remove one of the back panel covers (if necessary) and thread the Ethernet cable through it. Be careful removing these plastic covers as they are usually brittle from age. If you needed to remove a cover, it is convenient to tape it to the power supply or somewhere else inside the case so that you do not lose it. The Uthernet II is a 10/100BaseTX device, meaning that a standard Cat-5 or Cat-6 cable is sufficient. Once the cable is inside the Apple's case, loop it around back and insert it into the Ethernet port on the Uthernet II. If you did not need to loop the cable back, then you probably installed the Uthernet II facing the wrong way. Make sure that the Ethernet port is facing *the front* of the Apple.

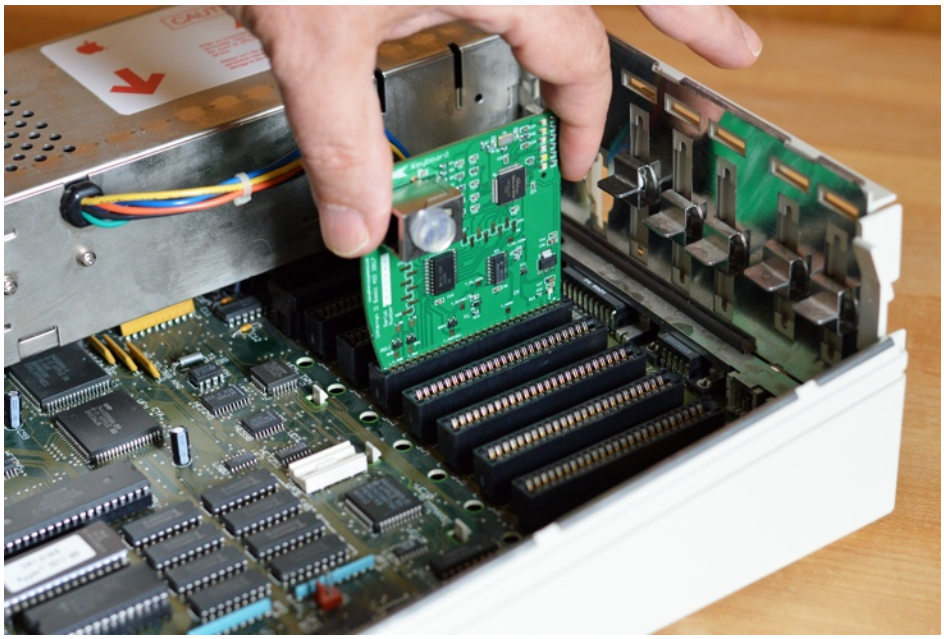


Figure 2.1 - Correct installation of the Uthernet II

The other end of the Ethernet cable should be attached to a switch, hub, router, or another computer.

If you are installing the Uthernet II in an Apple IIGS in a slot other than 3 or 4, be sure to open the Control Panel and set the Uthernet II's slot number to Your Card.

Installing the Optional Panel Mount Cable

Due to the inconvenience of having to open the Apple's case to adjust the Ethernet cable, it may be a good idea to install a short Ethernet extension cable that runs from the Uthernet II to the back panel of the Apple. Adafruit sells a Panel Mount Ethernet Extension Cable (Product ID 909) that fits the small cutouts on the back panel of the Apple IIe for $4.95. This is an optional accessory that you may order separately at any time from Adafruit. If you install this, you will not have to open the Apple's case in order to connect or disconnect the Ethernet cable. The drawback to installing the Panel Mount Cable is that it will block an Apple II slot, either 2 or 4.

Figure 2.2 - The Panel Mount Cable

To install the panel mount cable you will need a small Phillips screwdriver. Begin by turning your Apple IIe around so that you are looking at the back panel with its numbered cutouts. Choose which small cutout you wish to use: either 5, 6, 8, or 9. Carefully remove the plastic cover, if necessary, and tape it to the power supply or put it in some safe place so you do not lose it. On the end of the Panel Mount Cable, use the screwdriver to loosen one of the screws, and completely remove the other screw.

Insert the male end of the cable into the gray Ethernet port on the Uthernet II. Butt the opposite end of the cable against the cutout that you chose, making sure that the loosened screw head is showing on the outside of the back panel. Install the second screw, and tighten both so that they are snug. Do not over tighten the screws! Your completed installation should look like that shown in Figures 2.3 and 2.4.

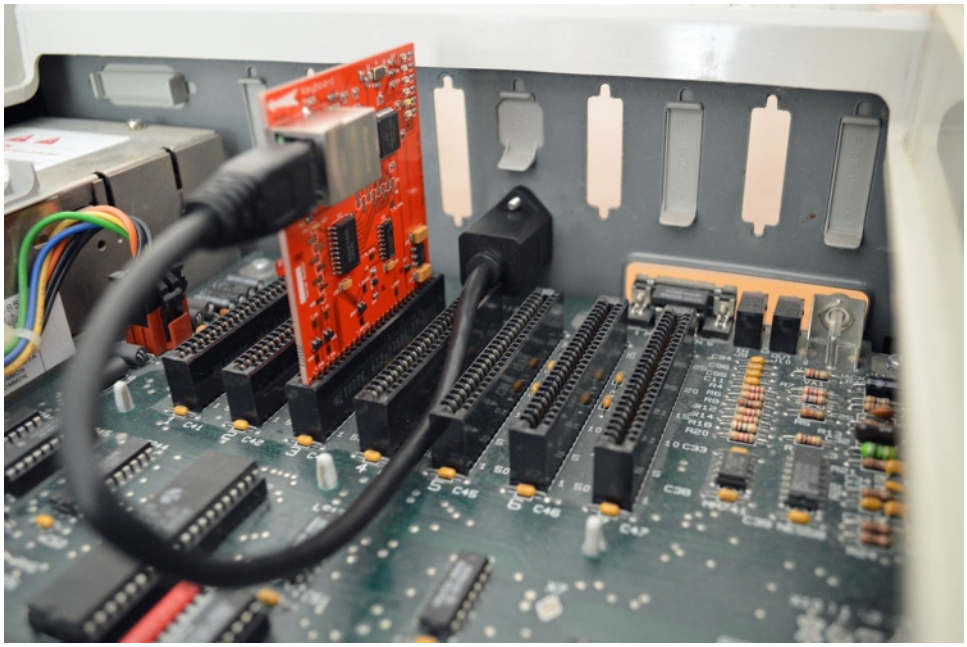


Figure 2.3 - Interior view of Panel Mount Cable

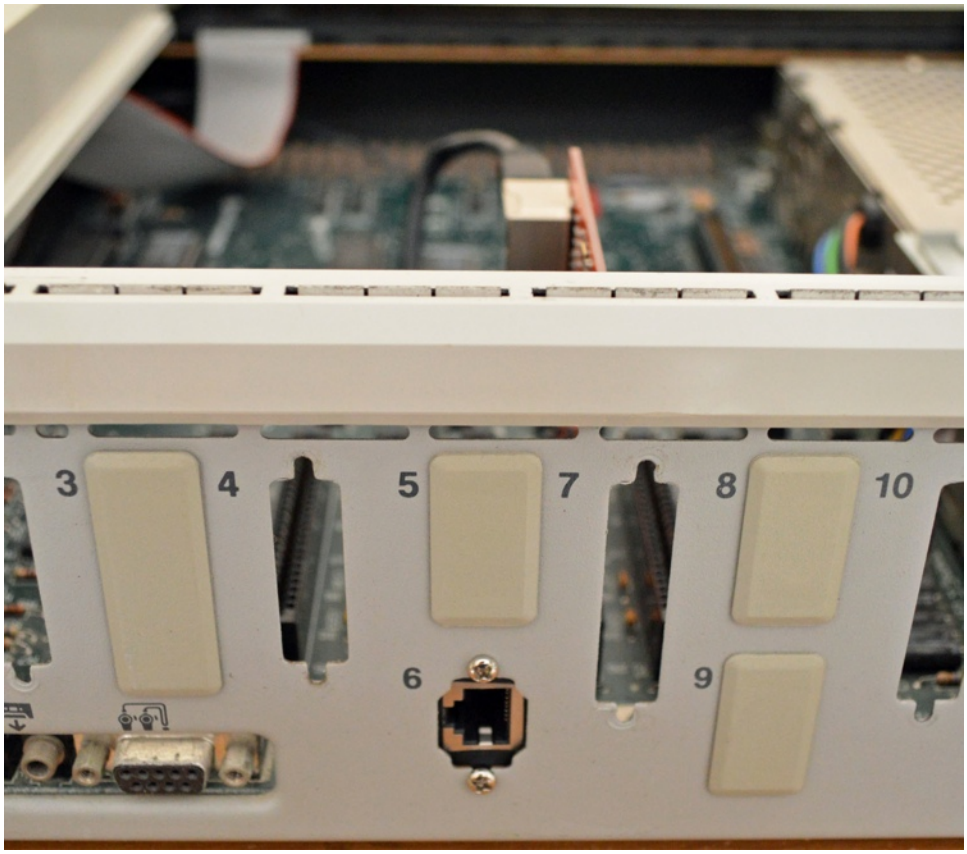


Figure 2.4 - Back panel view of Panel Mount Cable

Testing the Uthernet II

Your Uthernet II card was thoroughly tested before it was shipped to you, but it is a good idea to test it now to confirm that it is working correctly in your system. Power on your Apple II and enter the Monitor by pressing Control-Reset and typing CALL-151 at the prompt.

In the following instructions, what you will type will vary slightly depending on which slot the Uthernet II is installed in. Replace the X with the appropriate number or letter from the following table:

If the Uthernet II is installed in slot...	... replace the X with the following:
0	8
1	9
2	A
3	B
4	C
5	D
6	E
7	F

At the Monitor prompt (*), enter the following three lines, being sure to replace the X with the correct letter or number from the table above. Do not type the asterisk, but do press Return after each line:

```
*C0X4: 80

*C0X4: 03

*C0X4
```

After you press Return, the Apple should print the following line:

```
C0X4- 03
```

The X should be the same letter or number that you were using. If you do not see the value 03 printed on the screen, then verify that you are using the correct letter or number that corresponds to the slot number in which you have installed the Uthernet II. Otherwise, there may be a problem with your Uthernet II. Refer to the Troubleshooting chapter of this manual.

Testing the Link

Once the card is installed and connected to another device, leave the cover off your Apple and power it on. At least two LEDs on the Uthernet II should be lit: a green LED indicating a good link, and a blue LED above it indicating a full-duplex connection. A white LED on indicates a 100 Mb/s link, off means 10 Mb/s. In most cases, all three LEDs will be lit. If the green link LED is not lit, you need to determine whether the Ethernet cable is working, whether it is plugged in on both ends, and whether the device on the other end is powered on and active. Follow the steps in Chapter 5 for troubleshooting the connection.

Occasionally you may see a yellow LED blink periodically. This the receive LED, and it indicates that the Uthernet II is receiving information from the network.

Supported Software

Thanks to standardized programming interfaces used for Uthernet-enabled software on the Apple IIGS and 8-bit Apple, most software that worked with the original Uthernet will also work with the Uthernet II. Only a new driver for the Uthernet II is needed.

Be sure to check the a2RetroSystems web site for the latest version of all software and drivers for the Uthernet II.

TCP/IP Stacks and Operating Systems

Because the original Uthernet had no built-in TCP/IP stack, several implementations were developed for it. A brief overview and a list of features of each is available in this article: TCP/IP stacks for the Apple II. Though the new Uthernet II does have a built-in TCP/IP stack, the existing implementations were updated with device drivers or link layers for it. These are listed in the following sections.

A2osX

A2osX by Rémy Gibert is a new multitasking operating system for the 8-bit Apple II that includes a TCP/IP networking component. It is written in 65C02 assembly language for the enhanced Apple IIe, Apple IIc, or Apple IIGS. As of this writing, it is being continually updated and improved with new features and functionality.

ADTPro

ADTPro by David Schmidt is not a TCP/IP stack, but instead is one of the most popular applications for the Uthernet. ADTPro is the Apple Disk Transfer utility, allowing you to copy disk images to and from the Apple. Also included is a virtual Ethernet drive that allows the Apple II running ProDOS to easily access files across a network. The latest version of ADTPro is v2.0.1 (as of this writing) and it includes a driver for the Uthernet II. Click here to download ADTPro. The source code for ADTPro is available from Sourceforge.

Contiki

Contiki by Adam Dunkels, ported to the 8-bit Apple II family by Oliver Schmidt, is currently at version 3.0 (as of this writing). Contiki is a suite of Internet-enabled applications written in C that includes a web browser, HTTP server, Telnet server, wget, and an IRC client. Click here to download Contiki and its source code.

IP65

IP65 is a TCP/IP stack for the 8-bit Apple II series written in 6502 assembly language and C by Per Olofsson, Jonno Downes, and Oliver Schmidt. It is continually being updated, with source code available here. There is a suite of ready-to-use applications including a telnet client, web server, and HTTP downloader (wget). ADTPro, mentioned earlier, also uses IP65.

Marina

Marina by D. Finnigan is another TCP/IP stack for the 8-bit Apple II series that is also written in 6502 assembly language. The source code for Marina is available on the Marina web site.

Marinetti

Marinetti, originally written by Richard Bennett, is a TCP/IP stack available only for the Apple IIGS. If you do not have Marinetti installed, you will need to download and configure it. Visit the a2RetroSystems Marinetti page to download the latest version, 3.0b9, as of this writing. Installation instructions for Marinetti, as well as the needed Uthernet II link layer, are provided with the download package.

Ewen Wannop developed the Uthernet II link layer needed for Marinetti, which is available from his web site. If you already have Marinetti installed on your IIGS this is the only file that you need.

After you download the Uthernet II link layer file and copy it to your IIGS disk, use ShrinkIt GS to decode and extract it. The resulting file, named UthernetII, should be copied to the :System:TCPIP: folder of your startup disk. Be sure to restart your IIGS after copying. If this TCPIP folder does not exist, then you probably need to install Marinetti.

Marinetti has the widest selection of applications available for it, including web browsers, HTTP servers, FTP clients, an email client, ping, whois, finger, an IRC client, and NTP clients. A partial list of applications is available on the a2RetroSystems Marinetti page. Source code for Marinetti is available from Sourceforge.

PLASMA

PLASMA, by David Schmenk, is a combination of virtual machine and assembler/compiler matched closely to the 6502 architecture. It is a new programming language for the entire Apple series that allows a program to run without any changes on the Apple I, Apple II, and Apple III, similar to the Java VM system. PLASMA includes support for the Uthernet II and a demonstration web server. Download the source code and demos from GitHub,

Programming the Uthernet II

The Uthernet II is built around the Wiznet W5100 Ethernet controller. The W5100 supports 10/100Mb/s Ethernet and includes a built-in TCP/IP stack and 16KB internal buffer for data transmission. The W5100 has the following features:

- Support for TCP, UDP, ICMP, IPv4, ARP, IGMP, and PPPoE
- 10BaseT/100BaseTX Ethernet, PHY embedded
- Auto Negotiation (Full-duplex and half duplex)
- Auto MDI/MDIX
- 4 independent sockets
- Internal 16KB Memory for TX/RX buffers
- Interrupts (IRQ)

While the W5100 includes built-in support for TCP/IP protocols, it can also be operated in MAC Raw mode, which allows the programmer to use it as a traditional Ethernet controller and send any kind of information using any protocol over the network. The MTU of the W5100 has a fixed upper limit of 1500 bytes, not including the 14-byte Ethernet header.

A document that you will need to frequently refer to is the W5100 Datasheet which includes additional programming information. See also the Uthernet II Developer's Wiki.

W5100 Memory Map

The W5100 has its own internal address space that is completely independent from the Apple's memory. Communication between these two spaces is performed through a set of Apple slot I/O addresses that are described in the next section. The W5100 has a 16-bit address space that contains 32,768 locations. The space is divided into areas for common settings, socket settings, transmit buffers, and receive buffers. Table 4.1 shows the general layout and purpose of each range of memory.

Address Range	Purpose
$0-2F	Common registers
$30-3FF	*Reserved*
$400-7FF	Socket registers
$800-3FFF	*Reserved*
$4000-5FFF	Transmit buffers (TX)
$6000-7FFF	Receive buffers (RX)

Table 4.1 - W5100 Memory Map Overview

Apple Slot I/O

All I/O to and from the W5100 is performed through a set of *4* addresses in the $C0x0 range. The x is of course determined by what slot number the Uthernet II is installed in. All examples in this manual will assume slot *4*. On the W5100, this method of access is called Indirect Bus I/F mode. These addresses are all R/W enabled.

These *4* addresses are as follows:

- $C0x*4* - Mode Register
- $C0x*5* - Address High
- $C0x*6* - Address Low
- $C0x*7* - Data Port

Following is a brief overview of each of these addresses:

The Mode Register is generally used only at program initialization time, as it soft resets the W5100 and controls some low-level operational settings. Reading from this location returns the current mode byte.

There are two address registers that set an internal address pointer within the W5100's address space. The W5100 has a 16-bit address space, so there are two bytes for an address. Reading these registers will return the current value of the address pointer. Using the Mode Register, it is possible to configure the W5100 so that each successive read or write to the Data Port (described next) will automatically increment the internal address pointer. After a hardware reset the address pointer is initialized to 0.

The Data Port is a single location that is used to pass data between the Apple and the W5100. As stated above, it is possible to configure the W5100 so that each read or write to this location automatically increments the address pointer afterward. Doing so will lead to more efficient use of the W5100, as all transmit and receive operations must include a loop over every byte to be sent or received.

Mode Register

The Mode Register is used to perform a software reset, as well as set Ping Block mode, PPPoE mode, Address Auto-Increment, and Indirect Bus mode. Indirect Bus mode is the only setting that absolutely must be enabled to use the W5100. However, Address Auto-Increment is nearly always enabled too, as it dramatically improves performance and simplifies programming.

When a program first initializes the W5100, bit *7* should be set in order to perform an internal software reset in the W5100. Note that a software reset *does not* reset the address pointer to 0; it remains unchanged. Following reset, the program should then leave bit *7* clear and set the desired configuration bits, generally setting bits 1 and 0 to enable Address Auto-Increment and Indirect Bus mode. Table 4.2 shows the Mode Register and function of its bits.

Bit	Symbol	Description
7	RST	**Software Reset** Set this bit to initialize the W5100. Automatically cleared after reset.
6	*N/A*	*Reserved*
5	*N/A*	*Reserved*
4	PB	**Ping Block mode** Set this bit to ignore all ping requests when using built-in TCP/IP.
3	PPPoE	**PPPoE Mode** Set this bit if using ADSL without a router.
2	*N/A*	*Not Used*
1	AI	**Address Auto-Increment** Set this bit to automatically increment address pointer after data read/write. It is recommended to use this setting.
0	IND	**Indirect Bus mode** This bit must always be set when using the Uthernet II.

Table 4.2 - Mode Register bits

The following sample code will initialize the W5100 and set Address Auto-Increment and Indirect Bus mode. The Uthernet II should be in slot 4, or else change the $C0C4 to the appropriate value ($C0B4 for slot 3, for example).

```
A9 80       LDA #$80     ; reset the W5100
8D C4 C0    STA $C0C4    ; store in mode register
A9 03       LDA #$03     ; set address auto-inc (bit 1) and indirect mode
8D C4 C0    STA $C0C4    ; store in mode register
```

Accessing W5100 Memory

There are two steps to access the memory space of the W5100. First one must enter the desired address using the Address High and Address Low slot I/O locations. Write the high byte first, then low byte. Second, read or write the desired byte using the Data Port location. If you will be doing a sequential read or write on the W5100, then you only need to set the initial address. Be aware that the W5100 uses big endian byte order, that is, the high byte is in the lower address, and the low byte is in the higher address. This is the opposite of what the Ap-

ple II usually uses, where the high-byte comes second. There is one exception to this rule: if you wish to change only the low-byte of the address pointer, there is no need to set the high-byte first.

The following program shows how to set the address pointer and write to the W5100 Data Port by assigning the MAC address 00:08:DC:01:02:03 to the W5100:

```
A9 00        LDA   #$00   ; high-byte of MAC address location
A2 09        LDX   #$09   ; low-byte
8D C5 C0     STA   $C0C5  ; set high-byte of address pointer
8E C6 C0     STX   $C0C6  ; data register now points at MAC address
8D C7 C0     STA   $C0C7  ; store first byte of MAC address, then auto-inc
A9 08        LDA   #$08   ; 2nd byte of MAC address
8D C7 C0     STA   $C0C7  ; store in data port
A9 DC        LDA   #$DC
8D C7 C0     STA   $C0C7
A9 01        LDA   #$01
8D C7 C0     STA   $C0C7
A9 02        LDA   #$02
8D C7 C0     STA   $C0C7
A9 03        LDA   #$03
8D C7 C0     STA   $C0C7  ; last byte of MAC address
```

Notice that the Address Auto-Increment feature is being used here: the initial location of the MAC address, $09, is set once, then it is automatically incremented by the W5100 after each STA instruction.

The next example will read back the third and fourth bytes of the MAC address and print them on the screen:

```
A9 00        LDA   #$00   ; high-byte of 3rd byte of MAC address
A2 0B        LDX   #$0B   ; low-byte
8D C5 C0     STA   $C0C5  ; set high-byte of address pointer
8E C6 C0     STX   $C0C6  ; set low-byte of address pointer
AD C7 C0     LDA   $C0C7  ; read 3rd byte from data port
20 DA FD     JSR   $FDDA  ; print on screen
AD C7 C0     LDA   $C0C7  ; read 4th byte
20 DA FD     JSR   $FDDA  ; print
```

After running this program you should find DC01 printed on the screen. If not, be sure that your reset and initialized the W5100 using the instructions on the previous page.

When using Address Auto-Increment, the W5100 address pointer will wrap in two places. First, when the address reaches and surpasses $5FFF, the end of all Transmit buffers, the address wraps back to $4000, the beginning of those buffers. The second place is at $7FFF, the last byte of the Receive buffers. After this point, the address will wrap back to $6000, the beginning of the Receive buffers.

If the address pointer is manually set to $8000, it will continue to advance as usual to $8001, $8002 and so on, but reads and writes to the Data Port will begin from $0. For example, $801A corresponds to $001A, the the RX Memory Size Register. When the address pointer surpasses $FFFF it will wrap back to $E000. It is not recommended to write any program that manually sets the address pointer above $7FFF.

Some Common W5100 Locations

The programmer must refer to pages 19-36 of the W5100 Datasheet for a list and description of all W5100 memory locations, including the sockets. Table 4.3 contains some of the most common locations and their function.

Function	Address	Length (in bytes)
Mode Register (MR)	$0	1
Gateway Address	$1	4
Subnet Mask	$5	4
MAC Address	$9	6
Source IP Address	$F	4
Interrupt (IR)	$15	1
Interrupt Mask (IMR)	$16	1
Retry Count (RCR)	$17	2
RX Memory Size (RMSR)	$1A	1
TX Memory Size (TMSR)	$1B	1

Table 4.3 - Common W5100 Memory Locations

Configuring Host Address, Gateway, and Subnet Mask

The built-in TCP/IP stack does not manage host configuration, that is, allocation or assignment of host addresses. There is no DHCP or BOOTP client or Link Local Addressing implementation built-in to the W5100.

If the programmer wishes to use any of the socket modes beyond MAC Raw, the W5100 should be configured with a source IP address, Gateway (router) address, and Subnet mask. Note that it is still possible to use UDP sockets with no addresses configured, as in the case of a DHCP client attempting to obtain an IP address. In all cases, however, a MAC (hardware) address must be configured too, as shown in an earlier section in this chapter.

Table 4.3 shows the locations of these necessary host configuration parameters. By default they are all zero. The method of setting them is the same as shown for the MAC address.

Configuring W5100 Sockets

No matter which W5100 mode the programmer wishes to use: raw, IP, UDP, or TCP, a socket must be configured and established on the W5100. This section will introduce the basics of sockets that apply to all the above types while the following sections will cover the specifics for each type of socket.

As mentioned earlier, the W5100 allows up to 4 sockets to be active at once. Each socket has 3 main areas reserved for it in the W5100's address space: socket registers, transmit buffer, and receive buffer.

The socket registers contain basic information about the socket including its current mode, command, status, source port, foreign port, foreign IP address, and pointers and lengths for the transmit and receive buffers. Pages 15-18 of the W5100 Datasheet list all of the socket registers and their corresponding addresses for each socket. Notice that the memory layout for each socket's registers is the same; only the starting address changes: $400 for socket 0, $500 for socket 1, $600 for socket 2, and $700 for socket 3.

The transmit and receive buffers are located in another area of W5100 memory. The W5100 has a total of 16 KB of memory for these buffers. However, the maximum buffer size for transmit and receive is 8 KB each, meaning that the combined transmit buffer size of all sockets cannot be larger than 8 KB, nor can the combined receive buffer size of all sockets be larger than 8 KB. It is the responsibility of the programmer to determine how to divide the 8 KB of transmit and receive space between the sockets. If only one socket will ever be used, then the selection is simple: just assign 8 KB for transmit and 8 KB for receive. If using two sockets, then one could assign each socket 4 KB of transmit and receive buffer space. It is not required to assign the same amount of buffer space to each socket, nor must the transmit and receive buffer sizes be the same for a socket.

The buffer sizes are configured by storing a byte in the RX Memory Size and TX Memory Size registers. As Table 4.3 shows, these registers are located at $1A and $1B, respectively, in the W5100's address space. The default value is a 2 KB transmit buffer and a 2 KB receive buffer assigned to each of the 4 sockets. The following example shows how to assign 8 KB transmit and receive buffer to socket 0, meaning that only this socket can be used:

```
A9 00       LDA #$00     ; high-byte of RMSR location
A2 1A       LDX #$1A     ; low-byte
8D C5 C0    STA $C0C5
8E C6 C0    STX $C0C6    ; data register now points at RMSR
A9 03       LDA #$03     ; assign 8 KB to socket 0 receive buffer
8D C7 C0    STA $C0C7    ; set RX buffer size
8D C7 C0    STA $C0C7    ; here we assume auto-inc to set TX buffer size
```

Notice that the program assumes Auto-Increment mode is active. The second STA to $C0C7 sets the TX Memory Size register located at $1B. The following program assigns 4 KB of transmit and receive buffer space to sockets 0 and 1:

```
A9 00       LDA #$00     ; high-byte of RMSR location
A2 1A       LDX #$1A     ; low-byte
8D C5 C0    STA $C0C5
8E C6 C0    STX $C0C6    ; data register now points at RMSR
A9 0A       LDA #$0A     ; assign 4 KB to socket 0 and 1 receive buffer
8D C7 C0    STA $C0C7    ; set RX buffer size
8D C7 C0    STA $C0C7    ; here we assume auto-inc to set TX buffer size
```

For details on how to assign other amounts of buffer space to the sockets, refer to page 23 of the W5100 Datasheet.

Managing Socket Buffers

Though the W5100's built-in TCP/IP stack simplifies a great many details for the programmer, there are a few minor complexities involved with managing socket buffers. The first concern is to compute the correct base address within the W5100's memory space for the transmit and receive buffers. Recall from Table 4.1 that the transmit buffers start at $4000 and the receive buffers at $6000. In the simplest example, a single socket that uses 8 KB of buffer space for transmit and receive, the base addresses for those buffers are $4000 and $6000, respectively. The formula to compute the receive buffer base address for a given socket is simple: start with $6000 and add the receive buffer sizes of all preceding sockets. For example, let us say that socket 0 has a receive buffer size of 4 KB, socket 1 has 2 KB, and sockets 2 and 3 have 1 KB each. The base address for the receive buffer of each socket is $6000 for socket 0, $7000 for socket 1, $7800 for socket 2, and $7C00 for socket 3.

The method of computing addresses of the transmit buffers is identical. Just ensure that you start at $4000 for socket 0, and keep in mind that the transmit buffer size for a given socket does not necessarily have to be equal to its receive buffer size.

The second piece of information needed to manage socket buffers is the buffer mask. The buffer mask ensures that a program does not try to read or write outside of the valid buffer space for a socket. Fortunately, computing the mask is simple: take the buffer size and subtract 1. For example, if a socket's transmit buffer size is 4 KB ($1000) then its buffer mask is $0FFF. An 8 KB buffer has a mask of $1FFF. A buffer mask must be computed for both the socket's transmit and receive buffers, but if they are the same size, then the buffer mask for them is also the same. Otherwise the socket's transmit buffer and receive buffer will each have a different mask.

The table below shows example buffer sizes, base addresses, and masks for 4 sockets, two of which use a receive buffer that is a different size than the transmit buffer. Notice that for all sockets except 1 and 2, the TX and RX sizes are the same, therefore the TX and RX masks are the same. Socket 1 uses a 1 KB transmit buffer and a 2 KB receive buffer, while socket 2 uses a 2 KB transmit buffer but only a 1 KB receive buffer, so the masks differ. Though this setup is admittedly quite contrived, it shows the flexibility in configuring buffers.

Socket	TX size	TX base	TX mask	RX size	RX base	RX mask
0	4 KB	$4000	$0FFF	4 KB	$6000	$0FFF
1	1 KB	$5000	$03FF	2 KB	$7000	$07FF
2	2 KB	$5400	$07FF	1 KB	$7800	$03FF
3	1 KB	$5C00	$03FF	1 KB	$7C00	$03FF

Configuring and Opening a Socket

Once the buffer sizes have been configured, the programmer may configure and open a socket on the W5100. The first step is to determine which socket mode should be used. Next, write the appropriate socket mode byte to the Socket Mode Register. Each socket has an independent Socket Mode Register that determines the socket type, whether to enable Multicasting, and delayed ACK (if using a TCP socket). Only socket 0 can be configured with a MAC filter, and then only when in MAC Raw mode. Table 4.4 shows the Socket Mode Register and the function of each of its bits. The low 4 bits of the mode byte determine the socket type. Table 4.5 shows how to set the bits for each socket type.

Socket 0 is special in that it is the only socket that can be used for MAC Raw and PPPoE modes. Otherwise, it can be used like any other socket.

The following example shows how to open socket 0 in MAC Raw mode with the MAC filter enabled. Page 15 of the W5100 Datasheet shows that the address for the socket 0 Mode Register is $400.

```
A9 04       LDA #$04     ; high-byte of s0 MR
A2 00       LDX #$00     ; low-byte
8D C5 C0    STA $C0C5
8E C6 C0    STX $C0C6    ; data register now points at s0 MR
A9 44       LDA #$44     ; MAC Raw mode with MAC filter on
8D C7 C0    STA $C0C7    ; set socket mode
```

Bit	Symbol	Description
7	MULTI	**Multicasting** Set this bit to enable multicasting. Only has effect with UDP sockets.
6	MF	**MAC Filter** (for Socket 0 only) Set this bit to filter out all packets not addressed as broadcast or to the W5100's MAC address. When clear, all packets are received. This option, sometimes referred to as promiscuous mode, only applies to socket 0 when used in MAC Raw mode.
5	ND/MC	**Delayed ACK/Multicast version** When a TCP socket is used, setting this bit will disable the delayed ACK algorithm. When a socket is being used with Multicast, setting this bit will use IGMP version 1; clearing the bit will use IGMP version 2.
4	*N/A*	*Reserved*
3	P3	**Socket Type** Together these 4 bits determine the socket type (protocol). Table 4.5 shows how to set the bits for each socket type. Only socket 0 can be configured in MAC Raw or PPPoE modes.
2	P2	
1	P1	
0	P0	

Table 4.4 - Socket Mode Register bits

P3	P2	P1	P0	Socket Type
0	0	0	0	Closed
0	0	0	1	TCP
0	0	1	0	UDP
0	0	1	1	IP Raw
0	1	0	0	MAC Raw (Socket 0 only)
0	1	0	1	PPPoE (Socket 0 only)

Table 4.5 - Socket Type bits

This example shows how to configure socket 2 in UDP mode with Multicasting enabled and IGMP version 1:

```
A9 06        LDA #$06     ; high-byte of s2 MR
A2 00        LDX #$00     ; low-byte
8D C5 C0     STA $C0C5
8E C6 C0     STX $C0C6    ; data register now points at s2 MR
A9 A2        LDA #$A2     ; UDP, Multicast on, IGMP v1
8D C7 C0     STA $C0C7    ; set socket mode
```

Once the socket type and settings have been written to the W5100, the socket must be opened by using the OPEN command on the socket's Command Register. Pages 26-27 in the W5100 Datasheet list all the valid command codes for sockets. The code for OPEN is $01.

The following example shows how to open socket 0, assuming that the W5100's address pointer had been automatically incremented after setting the Socket Mode Register, as shown in the previous examples, and now points at $0401, the socket 0 Command Register.

```
A9 01        LDA #$01     ; socket open command
8D C7 C0     STA $C0C7    ; send command, assuming W5100 address is $0401
```

Whether or not Auto-Increment Mode is being used, it may be a good idea to explicitly set the W5100's address pointer before issuing a command to the socket, especially if a socket needs additional parameters to be set (such as a destination address or port) before it can be opened. The following example uses this approach to open socket 2.

```
A9 06        LDA #$06     ; high-byte of s2 CR
A2 01        LDX #$01     ; low-byte
8D C5 C0     STA $C0C5
8E C6 C0     STX $C0C6    ; data register now points at s2 CR
A9 01        LDA #$01     ; socket open command
8D C7 C0     STA $C0C7    ; send command
```

It is important to note that the value of the Command Register returns to 0 after the W5100 has finished executing the command.

Checking Socket Status

Each socket has a Status Register that contains the socket's type and state. If the socket is in TCP mode, the programmer can determine from the Status Register what TCP state the socket is currently in, such as ESTABLISHED or SYN SENT. The Status Register is located at $403 for socket 0, $503 for socket 1, and so on. Pages 28-31 of the W5100 Datasheet list all possible status values, and the location of the Status Register for each socket.

After a socket is opened with the OPEN command, the programmer should check the Status Register to make sure that the operation succeeded. The correct status value to check

for is different depending on the socket type; see the following sections in this chapter on each socket type for more information.

Common Socket Parameters

After the Command Status registers are several more registers used for common socket parameters such as local and foreign port, and foreign address, time to live (TTL) and DS/ECN (former TOS byte). Some of these registers are only used for certain socket types. Table 4.6 shows these socket parameters.

Using TCP Sockets

TCP sockets are generally thought of as being either a client or a server. The client is connecting to another TCP host, whereas the server is waiting for another TCP host to connect to it. To open any TCP socket, the W5100 must have an IP address configured.

A client socket must have a foreign address, foreign port, and local port configured before the socket is opened. See Table 4.6 for the locations of these three parameters. The next step is to issue the CONNECT command ($04) to the socket's Command Register. Finally, the program must wait for the socket to become established with the TCP 3-way handshake.

Function	Address	Length (in bytes)
Local Port Only used in UDP and TCP modes	$x04	2
Destination MAC Address Only used with SEND_MAC command in UDP mode	$x06	6
Foreign IP Address	$x0C	4
Foreign Port	$x10	2
Maximum Segment Size (MSS) Only used in TCP mode. Default is 0	$x12	2
Protocol Only used in IP mode. Sets IP protocol. Default is 0	$x14	1
DS/ECN (TOS) Sets this field in IP header. Default is 0	$x15	1
Time to Live (TTL) Sets this field in IP header. Default is 128	$x16	1

Table 4.6 - Common Socket Parameters

When the Status Register reads SOCK_ESTABLISHED ($17), the connection is complete and the program may begin communicating with the remote host. If however the connection should fail, the Status Register will return SOCK_CLOSED ($0), and the programmer must issue the CLOSE command ($10) to the socket.

TCP Client Connection

The following sample shows how to open a TCP connection from socket 0 in client mode to a remote host. The local port is 49152, the destination port is 80 and the destination address is 192.168.2.1.

```
* Configure socket 0 for TCP
A9 04        LDA #$04     ; high-byte of s0 MR
A2 00        LDX #$00     ; low-byte
8D C5 C0     STA $C0C5
8E C6 C0     STX $C0C6    ; data register now points at s0 MR
A9 01        LDA #$01     ; TCP mode
8D C7 C0     STA $C0C7    ; set socket mode
* Set Address Pointer for local port register
A9 04        LDA #$04     ; high byte of s0 local port
A2 04        LDX #$04     ; low byte
8D C5 C0     STA $C0C5
8E C6 C0     STX $C0C6
A9 C0        LDA #$C0     ; high byte of local port
8D C7 C0     STA $C0C7    ; notice that the W5100 is big endian!
A9 00        LDA #$00     ; low byte of local port
8D C7 C0     STA $C0C7    ; the low byte comes second, not first!
* Set Address Pointer for foreign address register
A9 04        LDA #$04     ; high byte of s0 foreign address
A2 0C        LDX #$0C     ; low byte
8D C5 C0     STA $C0C5
8E C6 C0     STX $C0C6
A9 C0        LDA #$C0     ; 1st byte of foreign address
8D C7 C0     STA $C0C7
A9 11        LDA #$A8
8D C7 C0     STA $C0C7
A9 0D        LDA #2
8D C7 C0     STA $C0C7
A9 24        LDA #1
8D C7 C0     STA $C0C7    ; last byte of foreign address stored
* With auto-inc, Address Pointer is now at foreign port
A9 00        LDA #$00     ; high byte of foreign port
8D C7 C0     STA $C0C7    ; remember that the W5100 is big endian!
A9 50        LDA #$50     ; low byte of foreign port
8D C7 C0     STA $C0C7    ; the low byte comes second, not first!
* Now open the socket
A9 04        LDA #$04     ; high-byte of s0 CR
A2 01        LDX #$01     ; low-byte
```

```
8D C5 C0      STA $C0C5
8E C6 C0      STX $C0C6     ; data register now points at s0 CR
A9 01         LDA #$01      ; socket open command
8D C7 C0      STA $C0C7     ; send command
* Check status register to see if the open command succeeded
A9 04         LDA #$04      ; high-byte of s0 SR
A2 03         LDX #$03      ; low-byte
8D C5 C0      STA $C0C5
8E C6 C0      STX $C0C6     ; data register now points at s0 SR
AD C7 C0      LDA $C0C7
C9 13         CMP #$13            ; is it in SOCK_INIT?
F0 01         BEQ :OPENED         ; yes, continue
00            BRK           ; no, there's some problem
* TCP socket is now waiting for its next command
       :OPENED
A9 03         LDA #$04      ; high-byte of s0 CR
A2 01         LDX #$01      ; low-byte
8D C5 C0      STA $C0C5
8E C6 C0      STX $C0C6     ; data register now points at s0 CR
A9 01         LDA #$04      ; socket connect command
8D C7 C0      STA $C0C7     ; send command
* Now wait for the socket to connect and become established
       :CHECKEST
A9 04         LDA #$04      ; high-byte of s0 SR
A2 03         LDX #$03      ; low-byte
8D C5 C0      STA $C0C5
8E C6 C0      STX $C0C6
AD C7 C0      LDA $C0C7     ; get socket status
F0 05         BEQ :ERRDONE  ; 0 = SOCK_CLOSED, error
C9 17         CMP #$17      ; is it SOCK_ESTABLISHED?
D0 ED         BNE :CHECKTEST      ; need more time to establish
* At this point, socket is ready for data transmission
60            RTS
       :ERRDONE
00            BRK
```

If execution reaches the RTS then the TCP socket has been successfully connected and the program may now send or receive data on it. If using tcpdump or Wireshark to monitor network traffic, you will see the W5100 send an ARP request followed by the TCP 3-way handshake.

Because the socket is connected to foreign port 80, typically used for HTTP, the program could then send a GET request to the server and wait for an HTTP response in return.

TCP Server

Opening a TCP socket in server mode is identical except that the foreign address and port are unnecessary, and the LISTEN command ($02) is used instead of CONNECT. After

issuing the LISTEN command, the Status Register should show SOCK_LISTEN ($14). The status will remain so until an incoming connection attempt is received, at which point the socket's status will change to ESTABLISHED ($17).

If the program is not using interrupts, then it could either wait in a loop, repeatedly checking the Status Register for a transition to ESTABLISHED, or it could do something else and only check periodically. In most cases, however, a program will wait in a polling loop. Once it has established a connection, the server can communicate with the client.

Receiving TCP Data

The method for receiving data over an established TCP connection is the same regardless if the connection on the Apple is a client or a server. Unless the program is using interrupts, the programmer must periodically poll the Received Size Register. If there is some data to read, the value returned will be greater than 0, indicating the amount of data available to be read. Otherwise, the value will be 0 to indicate that there is no new data for the program.

So far, everything to do with programming the W5100 has been simple and straightforward. If you were wondering when things were going to become difficult, wonder no longer. The procedures for both sending and receiving data on the W5100 are fairly involved, though if you are an experienced assembly language programmer, you will not find the challenge too difficult.

The source of the challenge is rooted in the fact that the W5100 does not automatically manage buffer rollover, despite the fact that all buffers are circular. Therefore, it is possible, and probably very common, that a program will begin reading data that is near the end of the buffer, and then must reset the W5100's internal address pointer to the beginning of the buffer, and resume reading. The same is true when sending data to the W5100 to be transmitted. If you recall the explanation of computing the buffer base addresses and masks several pages ago, this is where they come into play. The W5100 maintains a read and write pointer (for receive and transmit, respectively) for each socket, but these pointers are only useful after they have been AND'd with the buffer mask, and the result added to the buffer base address.

Let's cover an example scenario before proceeding further. Assume that we are using socket 1 configured as shown in the table on page 15, that is, the receive buffer size is 2 KB, the base address for this buffer is $7000, and the buffer mask is $07FF. The program has determined by reading the Received Size Register that there are 628 bytes of data available. The program has computed the logical AND of buffer read pointer and buffer mask, and the value is $0738. Next, the program adds the buffer base address, $7000. The result is $7738, so the program sets the W5100 address pointer to this value and begins reading. After reading 200 bytes of data, the address pointer is now at $77FF, the end of the socket receive buffer (with auto-increment, it will actually point to $7800 after the last read, which is outside the buffer). Yet there are still 428 bytes left to read. Because all buffers in the W5100 are circular, the remaining 428 bytes are stored starting at the beginning of the buffer, $7000. Therefore, the program must explicitly set the W5100 internal address pointer to $7000 and resume reading

the remaining 428 bytes. After all 628 bytes have been read, the program must increase the buffer read pointer by 628.

A program can read less than the amount of data available. The procedure in this case is the same; just ensure that the buffer read pointer is only increased by the amount of data actually read by the program. In no case should the pointer be increased by *more* than the amount of data available to be read (the value of the Received Size Register).

The final step after reading the data and increasing the read pointer is to issue the RECV command ($40) to the socket Command Register.

The following example checks the Received Size Register for socket 0 and clears the processor Z flag if there is any data to read:

```
A9 04        LDA #$04     ; high-byte of s0 RX_RSR
A2 26        LDX #$26     ; low-byte
8D C5 C0     STA $C0C5
8E C6 C0     STX $C0C6    ; data register now points at s0 RX_RSR
AD C7 C0     LDA $C0C7    ; get high byte of received data size
0D C7 C0     ORA $C0C7    ; OR with low byte to check for 0
```

After the ORA instruction, BNE can be used to branch to further instructions to read data from the W5100. The Z flag is set instead if there is no data to read (Received Size Register was 0).

The following is a complete program that connects to a TCP server on port 20,000 and echoes all received data on the screen using socket 0. It is convenient to use nc -l 20000 (netcat, for *nix based systems) to act as the server. An accompanying Applesoft program allows easy configuration of the local and destination IP addresses. Be sure to save the assembly language program to disk as TCPDEMO.

```
:ASM
                1    * TCP SOCKET DEMO FOR W5100/UTHERNET II
                2    * BY D. FINNIGAN
                3    * OCTOBER 2015
                4    *
                5    * UPDATED 09 JAN 2016
                6    *
                7    * UPDATED 13 FEB 2017, C. TORRENCE
                8    *  -REMOVED SEPARATE PATH FOR WRAP, ADD DEBUG PRINT
                9    *
                10   *
                11   * SLOT 3 I/O ADDRESSES FOR THE W5100
                12   *
                13   WMODE    EQU   $C0B4
                14   WADRH    EQU   $C0B5
                15   WADRL    EQU   $C0B6
                16   WDATA    EQU   $C0B7
                17   *
                18   *
```

```
19   * W5100 LOCATIONS
20   *
21   MACADDR  EQU   $0009      ; MAC ADDRESS
22   SRCIP    EQU   $000F      ; SOURCE IP ADDRESS
23   RMSR     EQU   $001A      ; RECEIVE BUFFER SIZE
24   *
25   * SOCKET 0 LOCATIONS
26   *
27   S0MR     EQU   $0400      ; SOCKET 0 MODE REGISTER
28   S0CR     EQU   $0401      ; COMMAND REGISTER
29   S0IR     EQU   $0402      ; INTERRUPT REGISTER
30   S0SR     EQU   $0403      ; STATUS REGISTER
31   S0LOCALPORT EQU $0404     ; LOCAL PORT
32   S0FORADDR EQU  $040C      ; FOREIGN ADDRESS
33   S0FORPORT EQU  $0410      ; FOREIGN PORT
34   S0MSS    EQU   $0412      ; MAX SEGMENT SIZE
35   S0PROTO  EQU   $0414      ; IP PROTOCOL
36   S0TOS    EQU   $0415      ; DS/ECN (FORMER TOS)
37   S0TTL    EQU   $0416      ; IP TIME TO LIVE
38   S0TXFSR  EQU   $0420      ; TX FREE SIZE REGISTER
39   S0TXRR   EQU   $0422      ; TX READ POINTER REGISTER
40   S0TXWR   EQU   $0424      ; TX WRITE POINTER REGISTER
41   S0RXRSR  EQU   $0426      ; RX RECEIVED SIZE REGISTER
42   S0RXRD   EQU   $0428      ; RX READ POINTER REGISTER
43   *
44   * SOCKET 0 PARAMETERS
45   *
46   RXBASE   EQU   $6000      ; SOCKET 0 RX BASE ADDR
47   RXMASK   EQU   $1FFF      ; SOCKET 0 8KB ADDRESS MASK
48   TXBASE   EQU   $4000      ; SOCKET 0 TX BASE ADDR
49   TXMASK   EQU   RXMASK     ; SOCKET 0 TX MASK
50   *
51   *
52   * SOCKET COMMANDS
53   *
54   SCOPEN   EQU   $01        ; OPEN
55   SCLISTEN EQU   $02        ; LISTEN
56   SCCONNECT EQU  $04        ; CONNECT
57   SCDISCON EQU   $08        ; DISCONNECT
58   SCCLOSE  EQU   $10        ; CLOSE
59   SCSEND   EQU   $20        ; SEND
60   SCSENDMAC EQU  $21        ; SEND MAC
61   SCSENDKEEP EQU $22        ; SEND KEEP ALIVE
62   SCRECV   EQU   $40        ; RECV
63   *
64   * SOCKET STATUS
65   *
66   STCLOSED EQU   $00
67   STINIT   EQU   $13
```

```
                    68   STLISTEN EQU   $14
                    69   STESTABLISHED EQU $17
                    70   STCLOSEWAIT EQU $1C
                    71   STUDP    EQU   $22
                    72   STIPRAW  EQU   $32
                    73   STMAXRAW EQU   $42
                    74   STPPOE   EQU   $5F
                    75   *
                    76   * MONITOR SUBROUTINES
                    77   *
                    78   KBD      EQU   $C000
                    79   KBDSTRB  EQU   $C010
                    80   COUT     EQU   $FDED
                    81   PRBYTE   EQU   $FDDA
                    82   PRNTAX   EQU   $F941
                    83   *
                    84   * ZERO-PAGE STORAGE
                    85   *
                    86   PTR      EQU   $06        ; 2 BYTES FOR APPLE BUFFER
                    87   GETSIZE  EQU   $08        ; 2 BYTES FOR RX_RSR
                    88   GETOFFSET EQU  $0A        ; 2 BYTES FOR OFFSET ADDR
                    89   GETSTARTADR EQU $0C       ; 2 BYTES FOR PHYSICAL ADDR
                    90   *
                    91   *
                    92   * RESET AND CONFIGURE W5100
                    93   *
                    94   *
8000: A9 06         95            LDA   #6         ; 5 RETRIES TO GET CONNECTION
8002: 85 06         96            STA   PTR        ; NUMBER OF RETRIES
8004: 10 10         97            BPL   RESET      ; ALWAYS TAKEN
                    98   *
8006: 0A 00 01      99   SRCADDR  HEX   C0A80205   ; 192.168.2.5  W5100 IP
8009: FC
800A: 0A 00 01     100   FADDR    HEX   C0A80201   ; 192.168.2.1   FOREIGN IP
800D: 11
800E: 4E 20        101   FPORT    HEX   4E20       ; 20000 FOREIGN PORT
8010: 00 08 DC     102   MAC      HEX   0008DC010203 ; W5100 MAC ADDRESS
8013: 01 02 03
                   103   *
                   104   RESET
8016: A9 80        105            LDA   #$80       ; RESET
8018: 8D B4 C0     106            STA   WMODE
801B: A9 03        107            LDA   #3         ; CONFIGURE WITH AUTO-INCREMENT
801D: 8D B4 C0     108            STA   WMODE
                   109   *
                   110   * ASSIGN MAC ADDRESS
                   111   *
8020: A9 00        112            LDA   #>MACADDR
8022: 8D B5 C0     113            STA   WADRH
```

```
8025: A9 09     114           LDA   #<MACADDR
8027: 8D B6 C0  115           STA   WADRL
802A: A2 00     116           LDX   #0
802C: BD 10 80  117  :L1      LDA   MAC,X
802F: 8D B7 C0  118           STA   WDATA      ; USING AUTO-INCREMENT
8032: E8        119           INX
8033: E0 06     120           CPX   #6         ; COMPLETED?
8035: D0 F5     121           BNE   :L1
                122  *
                123  * ASSIGN A SOURCE IP ADDRESS
                124  *
8037: A9 0F     125           LDA   #<SRCIP
8039: 8D B6 C0  126           STA   WADRL
803C: A2 00     127           LDX   #0
803E: BD 06 80  128  :L2      LDA   SRCADDR,X
8041: 8D B7 C0  129           STA   WDATA
8044: E8        130           INX
8045: E0 04     131           CPX   #4
8047: D0 F5     132           BNE   :L2
                133  *
                134  * CONFIGURE BUFFER SIZES
                135  *
8049: A9 1A     136           LDA   #<RMSR
804B: 8D B6 C0  137           STA   WADRL
804E: A9 03     138           LDA   #3         ; 8KB TO SOCKET 0
8050: 8D B7 C0  139           STA   WDATA      ; SET RECEIVE BUFFER
8053: 8D B7 C0  140           STA   WDATA      ; SET TRANSMIT BUFFER
                141  *
                142  * CONFIGURE SOCKET 0 FOR TCP
                143  *
8056: A9 04     144           LDA   #>S0MR
8058: 8D B5 C0  145           STA   WADRH
805B: A9 00     146           LDA   #<S0MR
805D: 8D B6 C0  147           STA   WADRL
8060: A9 01     148           LDA   #1         ; TCP MODE
8062: 8D B7 C0  149           STA   WDATA
                150  *
                151  * SET LOCAL PORT NUMBER
                152  *
8065: A9 04     153           LDA   #<S0LOCALPORT
8067: 8D B6 C0  154           STA   WADRL
806A: A9 C0     155           LDA   #$C0       ; HIGH BYTE OF LOCAL PORT
806C: 8D B7 C0  156           STA   WDATA
806F: A9 00     157           LDA   #0         ; LOW BYTE
8071: 8D B7 C0  158           STA   WDATA
                159  *
                160  * SET FOREIGN ADDRESS
                161  *
8074: A9 0C     162           LDA   #<S0FORADDR
```

```
8076: 8D B6 C0  163           STA   WADRL
8079: A2 00     164           LDX   #0
807B: BD 0A 80  165  :L3      LDA   FADDR,X
807E: 8D B7 C0  166           STA   WDATA
8081: E8        167           INX
8082: E0 04     168           CPX   #4
8084: D0 F5     169           BNE   :L3
                170  *
                171  * SET FOREIGN PORT
                172  *
8086: AD 0E 80  173           LDA   FPORT      ; HIGH BYTE OF FOREIGN PORT
8089: 8D B7 C0  174           STA   WDATA      ; ADDR PTR IS AT FOREIGN PORT
808C: AD 0F 80  175           LDA   FPORT+1    ; LOW BYTE OF PORT
808F: 8D B7 C0  176           STA   WDATA
                177  *
                178  * OPEN SOCKET
                179  *
8092: A9 01     180           LDA   #<S0CR
8094: 8D B6 C0  181           STA   WADRL
8097: A9 01     182           LDA   #SCOPEN    ; OPEN COMMAND
8099: 8D B7 C0  183           STA   WDATA
                184  *
                185  * CHECK STATUS REGISTER TO SEE IF SUCCEEDED
                186  *
809C: A9 03     187           LDA   #<S0SR
809E: 8D B6 C0  188           STA   WADRL
80A1: AD B7 C0  189           LDA   WDATA
80A4: C9 13     190           CMP   #STINIT    ; IS IT SOCK_INIT?
80A6: F0 33     191           BEQ   OPENED
80A8: A0 00     192           LDY   #0
80AA: B9 B6 80  193  :L4      LDA   :SOCKERR,Y
80AD: F0 06     194           BEQ   :LDONE
80AF: 20 ED FD  195           JSR   COUT
80B2: C8        196           INY
80B3: D0 F5     197           BNE   :L4
80B5: 00        198  :LDONE   BRK
80B6: D5 D4 C8  199  :SOCKERR ASC   "UTHERNET II: COULD NOT OPEN SOCKET!"
80B9: C5 D2 CE C5 D4 A0 C9 C9
80C1: BA A0 C3 CF D5 CC C4 A0
80C9: CE CF D4 A0 CF D0 C5 CE
80D1: A0 D3 CF C3 CB C5 D4 A1
80D9: 8D 00     200           HEX   8D00       ; CR+NULL
                201  *
                202  * TCP SOCKET WAITING FOR NEXT COMMAND
                203  *
                204  OPENED
80DB: A9 01     205           LDA   #<S0CR
80DD: 8D B6 C0  206           STA   WADRL
80E0: A9 04     207           LDA   #SCCONNECT
```

```
80E2: 8D B7 C0  208           STA   WDATA
                209  *
                210  * WAIT FOR TCP TO CONNECT AND BECOME ESTABLISHED
                211  *
                212  CHECKTEST
80E5: A9 03     213           LDA   #<S0SR
80E7: 8D B6 C0  214           STA   WADRL
80EA: AD B7 C0  215           LDA   WDATA      ; GET SOCKET STATUS
80ED: F0 06     216           BEQ   FAILED     ; 0 = SOCKET CLOSED, ERROR
80EF: C9 17     217           CMP   #STESTABLISHED
80F1: F0 4A     218           BEQ   CHECKRECV  ; SUCCESS
80F3: D0 F0     219           BNE   CHECKTEST
                220  *
                221  FAILED
80F5: C6 06     222           DEC   PTR
80F7: F0 08     223           BEQ   ERRDONE    ; TOO MANY FAILURES
80F9: A9 AE     224           LDA   #"."
80FB: 20 ED FD  225           JSR   COUT
80FE: 4C 16 80  226           JMP   RESET      ; TRY AGAIN
                227  *
                228  ERRDONE
8101: A0 00     229           LDY   #0
8103: B9 0F 81  230  :L       LDA   ERRMSG,Y
8106: F0 06     231           BEQ   :DONE
8108: 20 ED FD  232           JSR   COUT
810B: C8        233           INY
810C: D0 F5     234           BNE   :L
810E: 00        235  :DONE    BRK
                236  *
810F: D3 CF C3  237  ERRMSG   ASC   "SOCKET COULD NOT CONNECT - CHECK REMOTE HOST"
8112: CB C5 D4 A0 C3 CF D5 CC
811A: C4 A0 CE CF D4 A0 C3 CF
8122: CE CE C5 C3 D4 A0 AD A0
812A: C3 C8 C5 C3 CB A0 D2 C5
8132: CD CF D4 C5 A0 C8 CF D3
813A: D4
813B: 8D 00     238           HEX   8D00
                239  *
                240  *
                241  * CHECK FOR ANY RECEIVED DATA
                242  *
                243  CHECKRECV
813D: 2C 00 C0  244           BIT   KBD        ; KEYPRESS?
8140: 10 06     245           BPL   :NEXT
8142: AD 10 C0  246           LDA   KBDSTRB
8145: 4C F8 81  247           JMP   CLOSECONN  ; CLOSE CONNECTION
                248  :NEXT
8148: A9 26     249           LDA   #<S0RXRSR  ; S0 RECEIVED SIZE REGISTER
814A: 8D B6 C0  250           STA   WADRL
```

```
814D: AD B7 C0  251           LDA   WDATA      ; HIGH BYTE OF RECEIVED SIZE
8150: 0D B7 C0  252           ORA   WDATA      ; LOW BYTE
8153: F0 03     253           BEQ   NORECV     ; NO DATA TO READ
8155: 4C 5D 81  254           JMP   RECV       ; THERE IS DATA
                255  *
                256  NORECV
8158: EA        257           NOP              ; LITTLE DELAY...
8159: EA        258           NOP
815A: 4C 3D 81  259           JMP   CHECKRECV  ; CHECK AGAIN
                260  *
                261  * THERE IS DATA TO READ - COMPUTE THE PHYSICAL ADDRESS
                262  *
                263  RECV
815D: A9 26     264           LDA   #<S0RXRSR  ; GET RECEIVED SIZE AGAIN
815F: 8D B6 C0  265           STA   WADRL
8162: AD B7 C0  266           LDA   WDATA
8165: 85 09     267           STA   GETSIZE+1  ; HIGH BYTE
8167: AD B7 C0  268           LDA   WDATA
816A: 85 08     269           STA   GETSIZE    ; LOW BYTE
                270  *
                271  * CALCULATE OFFSET ADDRESS USING READ POINTER AND RX MASK
                272  *
816C: A9 28     273           LDA   #<S0RXRD
816E: 8D B6 C0  274           STA   WADRL
8171: AD B7 C0  275           LDA   WDATA      ; HIGH BYTE
8174: 29 1F     276           AND   #>RXMASK
8176: 85 0B     277           STA   GETOFFSET+1
8178: AD B7 C0  278           LDA   WDATA      ; LOW BYTE
817B: 29 FF     279           AND   #<RXMASK
817D: 85 0A     280           STA   GETOFFSET
                281  *
                282  * CALCULATE PHYSICAL ADDRESS WITHIN W5100 RX BUFFER
                283  *
817F: 18        284           CLC
8180: A5 0A     285           LDA   GETOFFSET
8182: 69 00     286           ADC   #<RXBASE
8184: 85 0C     287           STA   GETSTARTADR
8186: A5 0B     288           LDA   GETOFFSET+1
8188: 69 60     289           ADC   #>RXBASE
818A: 85 0D     290           STA   GETSTARTADR+1
                291  *
                292  * SET BUFFER ADDRESS ON APPLE
                293  *
818C: A9 00     294           LDA   #0         ; LOW BYTE OF BUFFER
818E: 85 06     295           STA   PTR
8190: A9 50     296           LDA   #$50       ; HIGH BYTE
8192: 85 07     297           STA   PTR+1
                298  *
                299  * SET BUFFER ADDRESS ON W5100
```

```
                300  *
                301  * JSR   DEBUG   ; UNCOMMENT FOR W5100 DEBUG INFO
8194: A5 0D     302           LDA   GETSTARTADR+1 ; HIGH BYTE FIRST
8196: 8D B5 C0  303           STA   WADRH
8199: A5 0C     304           LDA   GETSTARTADR
819B: 8D B6 C0  305           STA   WADRL
                306  *
                307  * BEGIN COPY
                308  *
819E: A0 00     309           LDY   #0
81A0: A6 09     310           LDX   GETSIZE+1
81A2: F0 10     311           BEQ   :LAST       ; LESS THAN 256 BYTES
81A4: AD B7 C0  312  :L       LDA   WDATA
81A7: 91 06     313           STA   (PTR),Y
                314  *
81A9: 20 1A 82  315           JSR   CLEANOUT    ; DEBUG PRINT
                316  *
81AC: C8        317           INY
81AD: D0 F5     318           BNE   :L
81AF: E6 07     319           INC   PTR+1       ; Y WRAPPED TO 0, GO TO NEXT PAGE
81B1: CA        320           DEX
81B2: D0 F0     321           BNE   :L
                322  :LAST
81B4: A6 08     323           LDX   GETSIZE
81B6: AD B7 C0  324  :L2      LDA   WDATA
81B9: 91 06     325           STA   (PTR),Y
                326  *
81BB: 20 1A 82  327           JSR   CLEANOUT    ; DEBUG PRINT
                328  *
81BE: C8        329           INY
81BF: CA        330           DEX
81C0: D0 F4     331           BNE   :L2
                332  *
81C2: A9 8D     333           LDA   #$8D        ; <CR>
81C4: 20 ED FD  334           JSR   COUT        ; DEBUG PRINT
                335  *
                336  *
                337  * UPDATE RXRD TO REFLECT DATA WE JUST READ
                338  *
                339  UPDATERXRD
81C7: 18        340           CLC
81C8: A9 04     341           LDA   #>S0RXRD    ; NEED HIGH BYTE HERE
81CA: 8D B5 C0  342           STA   WADRH
81CD: A9 28     343           LDA   #<S0RXRD
81CF: 8D B6 C0  344           STA   WADRL
81D2: AD B7 C0  345           LDA   WDATA       ; HIGH BYTE
81D5: A8        346           TAY               ; SAVE
81D6: AD B7 C0  347           LDA   WDATA       ; LOW BYTE
81D9: 65 08     348           ADC   GETSIZE     ; ADD LOW BYTE OF RECEIVED SIZE
```

```
81DB: AA        349           TAX             ; SAVE
81DC: 98        350           TYA             ; GET HIGH BYTE BACK
81DD: 65 09     351           ADC   GETSIZE+1 ; ADD HIGH BYTE OF RECEIVED SIZE _
81DF: A8        352           TAY             ; SAVE
81E0: A9 28     353           LDA   #<S0RXRD
81E2: 8D B6 C0  354           STA   WADRL
81E5: 8C B7 C0  355           STY   WDATA     ; SEND HIGH BYTE
81E8: 8E B7 C0  356           STX   WDATA     ; SEND LOW BYTE
                357  *
                358  * SEND THE RECV COMMAND
                359  *
81EB: A9 01     360           LDA   #<S0CR
81ED: 8D B6 C0  361           STA   WADRL
81F0: A9 40     362           LDA   #SCRECV
81F2: 8D B7 C0  363           STA   WDATA
                364  *
                365  *
                366  *
81F5: 4C 3D 81  367           JMP   CHECKRECV
                368  *
                369  *
                370  * CLOSE TCP CONNECTION
                371  *
                372  CLOSECONN
81F8: A9 04     373           LDA   #>S0CR    ; HIGH BYTE NEEDED
81FA: 8D B5 C0  374           STA   WADRH
81FD: A9 01     375           LDA   #<S0CR
81FF: 8D B6 C0  376           STA   WADRL
8202: A9 08     377           LDA   #SCDISCON ; DISCONNECT
8204: 8D B7 C0  378           STA   WDATA     ; SEND COMMAND
                379  *
                380  * CHECK FOR CLOSED STATUS
                381  *
                382  CHECKCLOSED
8207: A2 00     383           LDX   #0
8209: A9 03     384  :L       LDA   #<S0SR
820B: 8D B6 C0  385           STA   WADRL
820E: AD B7 C0  386           LDA   WDATA
8211: F0 06     387           BEQ   ISCLOSED
8213: EA        388           NOP
8214: EA        389           NOP
8215: EA        390           NOP
8216: E8        391           INX
8217: D0 F0     392           BNE   :L        ; DON'T WAIT FOREVER
                393  ISCLOSED
8219: 60        394           RTS             ; SOCKET IS CLOSED
                395  *
                396  *
                397  * SUPPORT SUBROUTINE: CLEANOUT
```

```
                398  * "CLEANS UP" OUTPUT FOR THE APPLE BY
                399  * SETTING THE HIGH BIT AND DOING SOME SUBSTITUTIONS
                400  CLEANOUT
821A: 09 80     401           ORA   #%10000000 ; SET HIGH BIT
821C: C9 8A     402           CMP   #$8A       ; NEWLINE?
821E: D0 02     403           BNE   :OUT
8220: A9 8D     404           LDA   #$8D       ; CONVERT TO <CR>
                405  :OUT
8222: 4C ED FD  406           JMP   COUT       ; THIS WILL DO THE RTS
                407  *
                408  * DEBUG - PRINT W5100 STARTADR AND SIZE
                409  *
                410  DEBUG
8225: A9 A0     411           LDA   #" "
8227: 20 ED FD  412           JSR   COUT
822A: A9 A4     413           LDA   #"$"
822C: 20 ED FD  414           JSR   COUT
822F: A5 0D     415           LDA   GETSTARTADR+1
8231: A6 0C     416           LDX   GETSTARTADR
8233: 20 41 F9  417           JSR   PRNTAX
8236: A9 A0     418           LDA   #" "
8238: 20 ED FD  419           JSR   COUT
823B: A9 A4     420           LDA   #"$"
823D: 20 ED FD  421           JSR   COUT
8240: A5 09     422           LDA   GETSIZE+1
8242: A6 08     423           LDX   GETSIZE
8244: 20 41 F9  424           JSR   PRNTAX
8247: A9 8D     425           LDA   #$8D
8249: 4C ED FD  426           JMP   COUT       ; THIS WILL DO THE RTS
                427  *


--End assembly, 588 bytes, Errors: 0
```

Following is an Applesoft program that patches SRCADDR and FADDR:

```
10 D$ =  CHR$ (4): REM  CTRL+D
20  PRINT D$;"BLOAD TCPDEMO"
30 BASE = 32768:B1 = BASE + 6
35  DIM IP(5)
40  PRINT "UTHERNET II TCPDEMO CONFIGURATION"
50  PRINT "ENTER LOCAL (APPLE II) IP ADDRESS: ";
60  GOSUB 1000
70 B1 = B1 + 4
80  PRINT "ENTER DESTINATION IP ADDRESS: ";
90  GOSUB 1000
100  PRINT "SUCCESSFULLY CONFIGURED TCPDEMO. RUNNING..."
110  CALL BASE
```

```
120  END
900  REM  PARSE THE IP ADDRESS AND POKE INTO PROGRAM
1000  INPUT "";A$
1010 IP(0) = 0:IP(4) =  LEN (A$) + 1
1020 C = 1
1030  FOR I = 2 TO  LEN (A$): IF  MID$ (A$,I,1) = "." THEN IP(C) = I:C = C + 1
1040  NEXT
1050  IF C <  > 4 THEN  PRINT "*** ILLEGAL IP ADDRESS ***": GOTO 50
1060  FOR I = 1 TO 4
1070 NUM =  VAL ( MID$ (A$,IP(I - 1) + 1,IP(I) - IP(I - 1) - 1))
1080  POKE B1 + I - 1,NUM
1090  NEXT
1100  RETURN
```

Sending TCP Data

Sending data follows the same pattern as receiving data, except that the values and registers for the TX (transmit) buffer are used instead. The programmer still needs to check for buffer overflow, and copy the data in two parts if the size of the data to send exceeds the end point of the buffer.

Begin by accessing the TX Free Size Register to determine how much space is left in the buffer. If the free size is less than your send size, then the W5100 may be in process of sending a TCP segment. Simply poll again until there is enough free space.

Next, compute the offset address by performing the logical AND of the TX Write Pointer and the TX mask, then add the resulting value to the TX Base Address.
After the W5100 offset address is computed, check if the size of the data to send will exceed the end of the buffer. If so, the copy from the Apple to the W5100 must be split in two. If not, a single copy loop will suffice. No matter which path was taken, the final steps are to increase the TX Write Pointer by the size of the data sent, then issue to the SEND command ($20) to the Command Register.

The W5100 will always assert the Push (PSH) flag for data segments. The Urgent (URG) flag and pointer cannot be set. There is no way to alter this behavior.

Checking for TCP FIN and Closing the Socket

The socket Status Register will be updated to show if the remote host has closed its end of the connection. If so, the status will be SOCK_CLOSE_WAIT ($1C). In response to this status, the programmer should close the socket using either the DISCON ($08) or CLOSE ($10) commands. The DISCON command will send a FIN segment to the remote host, whereas CLOSE immediately shuts down the connection without communicating anything to the other host. In most cases, the programmer should use DISCON, as shown in this example:

```
* Close TCP connection
A9 04         LDA #$04     ; high-byte of s0 CR
A2 01         LDX #$01     ; low-byte
8D C5 C0      STA $C0C5
8E C6 C0      STX $C0C6    ; data register now points at s0 CR
A9 08         LDA #$08     ; socket discon command
8D C7 C0      STA $C0C7    ; send command
* Check status register to see if the discon command succeeded
       CHECKCLOSED
A9 04         LDA #$04     ; high-byte of s0 SR
A2 03         LDX #$03     ; low-byte
8D C5 C0      STA $C0C5
8E C6 C0      STX $C0C6    ; data register now points at s0 SR
AD C7 C0      LDA $C0C7
D0 F1         BNE CHECKCLOSED ; not closed yet
60            RTS          ; socket is closed
```

Be aware that the socket can also close from timeout. The programmer must check the Status Register before sending or receiving data to ensure that the connection is still established.

Using UDP Sockets

Using a UDP socket is much simpler than TCP. In general, the procedures are the same as for TCP, except that there is no connection process. The W5100 does not necessarily have to be configured with an IP address, such as if DHCP is being used to obtain host configuration.

Opening a UDP Socket

The first step is to configure the socket Mode Register for UDP. Then configure the local port socket parameter before issuing the OPEN command ($01) to the Command Register. To see if the command succeeded, check the Status Register for SOCK_UDP ($22). Close the socket and start again if SOCK_UDP is not returned in the Status Register. The following program demonstrates how to open socket 1 in UDP mode using local port 49152:

```
* Configure socket 1 for UDP
A9 05         LDA #$05     ; high-byte of s1 MR
A2 00         LDX #$00     ; low-byte
8D C5 C0      STA $C0C5
8E C6 C0      STX $C0C6    ; data register now points at s1 MR
A9 02         LDA #$02     ; UDP mode
8D C7 C0      STA $C0C7    ; set socket mode
* Set Address Pointer for local port register
A9 05         LDA #$05     ; high byte of s1 local port
A2 04         LDX #$04     ; low byte
8D C5 C0      STA $C0C5
8E C6 C0      STX $C0C6
A9 C0         LDA #$C0     ; high byte of local port
```

```
8D C7 C0      STA $C0C7     ; notice that the W5100 is big endian!
A9 00         LDA #$00      ; low byte of local port
8D C7 C0      STA $C0C7     ; the low byte comes second, not first!
* Now open the socket
A9 05         LDA #$05      ; high-byte of s1 CR
A2 01         LDX #$01      ; low-byte
8D C5 C0      STA $C0C5
8E C6 C0      STX $C0C6     ; data register now points at s1 CR
A9 01         LDA #$01      ; socket open command
8D C7 C0      STA $C0C7     ; send command
* Check status register to see if the open command succeeded
A9 05         LDA #$05      ; high-byte of s1 SR
A2 03         LDX #$03      ; low-byte
8D C5 C0      STA $C0C5
8E C6 C0      STX $C0C6     ; data register now points at s1 SR
AD C7 C0      LDA $C0C7
C9 22         CMP #$22      ; is it in SOCK_UDP?
F0 01         BEQ :OPENED   ; yes, continue
00            BRK           ; no, there's some problem
* UDP socket is now waiting for its next command
       :OPENED
60            RTS
```

As soon as the UDP socket is opened it is ready to send and receive data.

Receiving UDP Data

The procedure is identical to that for TCP: first check the Received Size Register for a value greater than 0. If there is data to read, proceed by copying the UDP data from the W5100 RX buffer to the Apple. An 8-byte socket header proceeds the actual UDP data payload. This header contains, in order: the foreign IP address (4 bytes), foreign port (2 bytes), and data size (2 bytes). All values are in network byte order, which is high byte first. The data size does include the size of the 8-byte socket header. Note that this 8 byte header is not the same as the UDP header, which is also 8 bytes. For example, if the UDP data payload is 3 bytes, the W5100 will report a total receive size of 11 bytes: 8 bytes socket header and 3 bytes UDP payload.

The foreign IP address and port in the socket header should be saved if the program will send a response back. Unlike in a TCP socket where the program may choose to read less than the total amount of data available, with UDP, the program must read all available UDP data. After reading the received data into the Apple's memory, the program must advance the RXRD pointer and send the RECV command ($40), same as with TCP sockets.

Sending UDP Data

The send process is similar to using a TCP socket, except that the foreign IP address and port must be specified. See table 4.6 for the addresses of these two socket parameters.

Closing a UDP Socket

The socket can be closed at any time by sending the CLOSE command ($10) to the socket Command Register.

Using IP Raw Sockets

The IP Raw socket mode allows the programmer to implement any protocol within IP, such as AppleTalk over IP, ICMP, RDP, or any other protocol.

Opening an IP Raw Socket

The first step is to configure the socket Mode Register for IP Raw ($03). Then configure the Protocol socket parameter before issuing the OPEN command ($01) to the Command Register. To see if the command succeeded, check the Status Register for SOCK_IPRAW ($32). As soon as the socket is opened it is ready to send and receive data.

Receiving IP Raw Data

Receiving data is much the same as with a UDP socket: first check the Received Size Register for a value greater than 0. If there is data to read, proceed by copying the IP Raw data from the W5100 RX buffer to the Apple. A 6-byte socket header proceeds the actual IP data payload. This header contains, in order: destination address (4 bytes), and data size (2 bytes). All values are in network byte order, which is high byte first. The data size does include the size of the 6-byte socket header.

Sending IP Raw Data

The send process is similar to using a UDP socket, except that the remote port does not need to be specified, only the destination address.

Closing an IP Raw Socket

The socket can be closed at any time by sending the CLOSE command ($10) to the socket Command Register.

Using a MAC Raw Socket

The MAC Raw socket provides the programmer with the lowest-possible access to the W5100, allowing one implement any protocol and send any data across the wire. This socket mode is only available on socket 0. At minimum, a MAC address should be configured, but otherwise, there is no need to configure any of the IP settings because it is up to the program to manage the protocol that will be used.

Opening a MAC Socket

The first step is to configure the socket Mode Register for MAC Raw ($04). Then configure the Protocol socket parameter before issuing the OPEN command ($01) to the Command Register. To see if the command succeeded, check the Status Register for SOCK_MACRAW ($42). As soon as the socket is opened it is ready to send and receive data.

Receiving MAC Raw Data

Receiving data is much the same as with a UDP socket: first check the Received Size Register for a value greater than 0. If there is data to read, proceed by copying the MAC Raw data from the W5100 RX buffer to the Apple. The data is prepended with a 2 byte data length header. This length includes all received data plus the length of this header. For example, if the received data were a 14 byte Ethernet header plus 20 bytes of protocol data, the length reported would be 36 bytes.

Sending IP Raw Data

The send process is similar to using a UDP socket, except that the remote address and port do not need to be specified.

Closing a MAC Raw Socket

The socket can be closed at any time by sending the CLOSE command ($10) to the socket Command Register.

Probing for the Uthernet II

Because the Uthernet II works in any slot in the Apple, programs should also be written to work with it in any slot. While it is always possible to ask the user which slot contains the Uthernet II, a clever program can instead scan the slots in attempt to probe each one for the presence of an Uthernet II. There are a few difficulties to this approach, however. The principal obstacle to probing the slots for an Uthernet II is that the card has no firmware, and thus no ID bytes to check for. Furthermore, the W5100 chip used on the Uthernet II has no ID bytes either. Therefore, it must be detected by testing for its expected behavior.

The probing algorithm is thus: for each slot, $80, the reset byte, will be stored at $C0x4, the Mode Register. If the slot contains an Uthernet II, this will cause the W5100 to reset, and the Mode Register to read $00. Next, the Mode Register will be set to $03, and again read back to verify that it contains $03. With this last check satisfied, the probe is complete.

The danger in this algorithm is that it involves blindly writing and reading two slot I/O locations which could cause unpredictable behavior on other peripheral cards. In an attempt to minimize this danger, the slots are scanned in a specific order, based on the presumed probability of having an Uthernet II installed therein. This order is: 3, 4, 2, 1, 5, 6, 7.

This algorithm has been tested to work on a platinum (enhanced) Apple IIe, a standard (unenhanced) Apple IIe, and an Integer Basic Apple II. Each Apple was tested both with and without a Transwarp accelerator, and with three to four other peripheral cards installed.

Following is the Uthernet II probe program:

```
                    1    *
                    2    * UTHERNET II PROBE
                    3    *
                    4    * SCAN THE SLOTS FOR AN UTHERNET II
                    5    *
                    6    * WRITTEN BY D. FINNIGAN - 06 JAN 2016
                    7    *
                    8    *
                    9    MR       EQU   $C084
                    10   DATA     EQU   $C087
                    11   *
                    12   COUT     EQU   $FDED
                    13   PRBYTE   EQU   $FDDA
                    14   *
                    15   *
                    16   *
                    17   START
8000: A0 07         18            LDY   #SLOTLEN-SLOTS ; NUMBER OF SLOTS TO SCAN
8002: B9 4F 80      19   :L       LDA   SLOTS,Y    ; GET SLOT NUMBER
8005: AA            20            TAX
                    21   *
```

```
                22  * SEND THE RESET COMMAND
                23  *
8006: A9 80     24          LDA   #$80
8008: 9D 84 C0  25          STA   MR,X
800B: EA        26          NOP
800C: EA        27          NOP
800D: BD 84 C0  28          LDA   MR,X       ; SHOULD GET ZERO
8010: D0 2C     29          BNE   :NEXTSLOT
                30  *
                31  * CONFIGURE OPERATING MODE WITH AUTO-INCREMENT
                32  *
8012: A9 03     33          LDA   #3         ; OPERATING MODE
8014: 9D 84 C0  34          STA   MR,X
8017: BD 84 C0  35          LDA   MR,X       ; READ BACK MR
801A: C9 03     36          CMP   #3
801C: D0 20     37          BNE   :NEXTSLOT
                38  *
                39  * PROBE SUCCESSFUL
                40  *
801E: 8A        41          TXA
801F: 48        42          PHA
8020: A0 00     43          LDY   #0
8022: B9 56 80  44  :FL     LDA   FOUNDMSG,Y
8025: F0 06     45          BEQ   :FOUND2
8027: 20 ED FD  46          JSR   COUT
802A: C8        47          INY
802B: D0 F5     48          BNE   :FL
                49  :FOUND2
802D: 68        50          PLA
802E: 6A        51          ROR
802F: 6A        52          ROR
8030: 6A        53          ROR
8031: 6A        54          ROR
8032: 09 B0     55          ORA   #$B0
8034: 20 ED FD  56          JSR   COUT
8037: A9 8D     57          LDA   #$8D
8039: 20 ED FD  58          JSR   COUT
803C: D0 10     59          BNE   :DONE      ; ALWAYS TAKEN
                60  *
                61  * TRY NEXT SLOT
                62  *
                63  :NEXTSLOT
803E: 88        64          DEY
803F: 10 C1     65          BPL   :L
                66  *
                67  * UTHERNET II NOT FOUND
                68  *
8041: A0 00     69          LDY   #0
8043: B9 72 80  70  :NFL    LDA   NOTFOUNDMSG,Y
```

```
8046: F0 06     71            BEQ   :DONE
8048: 20 ED FD  72            JSR   COUT
804B: C8        73            INY
804C: D0 F5     74            BNE   :NFL
                75   :DONE
804E: 00        76            BRK
                77   *
                78   *
                79   * ORDER OF SLOTS TO BE SCANNED
                80   * THE GOAL HERE IS TO ARRANGE THE SLOT NUMBERS IN ORDER
                81   * OF PROBABILITY OF FINDING AN UTHERNET II.
                82   * THESE ARE IN REVERSE ORDER, $N0 FORMAT.
                83   *
                84   * YOU CAN REMOVE A SLOT FROM THIS TABLE IF YOU DON'T
                85   * WANT TO SCAN IT, FOR EXAMPLE, SLOTS 5 OR 6.
                86   *
804F: 70 60 50  87   SLOTS    HEX   70605010204030
8052: 10 20 40 30
                88   SLOTLEN
                89   *
                90   *
                91   *
8056: D5 D4 C8  92   FOUNDMSG ASC   "UTHERNET II FOUND IN SLOT: "
8059: C5 D2 CE C5
805D: D4 A0 C9 C9
8061: A0 C6 CF D5
8065: CE C4 A0 C9
8069: CE A0 D3 CC
806D: CF D4 BA A0
8071: 00        93            HEX   00
8072: D5 D4 C8  94   NOTFOUNDMSG ASC "UTHERNET II NOT FOUND!"
8075: C5 D2 CE C5
8079: D4 A0 C9 C9
807D: A0 CE CF D4
8081: A0 C6 CF D5
8085: CE C4 A1
8088: 8D 00     95            HEX   8D00


--End assembly, 138 bytes, Errors: 0
```

An Improved Uthernet II Probe Subroutine

Benoît Gilon wrote an improved subroutine to detect the Uthernet II. He writes "The changes from the one published is that it automatically skips all Disk II interface cards (detected by a checksum routine on the $Cnxx ROM space). and also skips any card with a firmware entry

point as the Uthernet II card doesn't have any at this time. The checked bytes are at offsets: $05 (value $38) $07 (value $18) and $0B (value $01)." Following is his subroutine:

```
* Routine pour deviner ou se cache la carte Uthernet II
AUXPTR          EQU   $06
SLOT            EQU   $FE
SLOT16          EQU   $FF
YMODR           EQU   $85          ; Offset to Wizchip mode register
COUT1           EQU   $FDF0
                ORG $0300
                JSR   CHKSLTS
                BCS   :0
                LDA   #MESSOK
                LDY   #>MESSOK
                JSR   PRINT
                LDA   SLOT
                ORA   #$B0
                JSR   COUT1
                LDA   #$8D
                JMP   COUT1
:0              LDA   #MESSNOK
                LDY   #>MESSNOK
PRINT           STA   AUXPTR
                STY   AUXPTR+1
                LDY   #0
]LOOP           LDA   (AUXPTR),Y
                BEQ   :0
                JSR   COUT1
                INY
                BNE   ]LOOP       ; Always
:0 RTS
MESSOK          HEX   8D
                ASC   "FOUND A UII CARD IN SLOT #",00
MESSNOK         HEX   8D
                ASC   "NO UII CARD DETECTED",8D,00
CHKSLTS         LDA   #0
                STA   AUXPTR
                LDA   #$C7
                STA   AUXPTR+1
                AND   #7
                STA   SLOT
]LOOP1          LDX   #MVAL-MOFFST-1
]LOOP           LDY   MOFFST,X
                LDA   (AUXPTR),Y
                CMP   MVAL,X
                BNE   :1
                DEX
                BPL   ]LOOP
]NEXT           DEC   AUXPTR+1
```

```
               DEC   SLOT
               BPL   ]LOOP1
               SEC
               RTS
* No firmware card found in this slot: good!
* Is it a drive II controller card?
:1             LDA #0
               TAX
               TAY
]LOOP          ADC   (AUXPTR),Y
               BCC   *+3
               INX
               INY
               BNE   ]LOOP
               CMP   #$C2  ;$7BC2 is the checksum for DII interface cards
               BNE   *+6
               CPX   #$7B
               BEQ   ]NEXT
               LDA   AUXPTR+1
               LUP   4
               ASL
               --^
               STA   SLOT16
               ORA   #YMODR
               TAY
               LDA   #$80
               STA   $BFFF,Y
               LDA   #3
               STA   $BFFF,Y
               CMP   $BFFF,Y
               BNE   ]NEXT
               CLC
               RTS
MOFFST         HEX   05070B
MVAL           HEX   381801
```

Indexed Addressing and Slot Independent Code

Due to the 6502's phantom read that occurs in indexed addressing modes, the usual indexed addressing method cannot be used to access the W5100. Instead, the base address must be $BFFF so that the false read occurs on a different page and does not affect the W5100. This is the same method that is used for the Super Serial Card firmware. The reason why this alternate method is necessary is that the W5100 Auto-Increment mode will advance the address pointer on any read or write to the Data Port, and the false read will trigger this increment before the 6502 actually reads or writes any data.

First obtain the slot number in the form of $n0, for example, $30 for slot 3, or $40 for slot 4. Then add $85 to reach the Mode Register, $86 for Address High, and so on. This example shows how to reset and configure the W5100 with a MAC address:

```
       LDA #SLOT     ; Uthernet slot in $n0 format
       ORA #$85      ; Point to Mode Register
       TAX
       LDA #$80      ; reset byte
       STA $BFFF,X   ; reset the W5100
       LDA #3        ; standard config
       STA $BFFF,X
* Assign MAC address
       INX           ; point to Address High
       LDA #0        ; hi byte of MAC addr
       STA $BFFF,X
       INX           ; point to Address Low
       LDA #9        ; lo byte
       STA $BFFF,X
       LDY #0
       INX           ; point to Data Port
:MACL  LDA MACADDR,Y
       STA $BFFF,X
       INY
       CPY #6
       BNE :MACL
       BRK
*
MACADDR HEX 0008DC010203
```

Uthernet II Interrupts

The W5100 is capable of generating IRQs, or interrupts. Interrupts can be triggered on the following events: socket connection, socket disconnection, incoming data, or timeout. The programmer can clear the interrupt status by writing to the Interrupt Register (IRQR) or to the socket's individual Interrupt Register. All interrupts are maskable.

How to Enable Interrupts

To enable interrupts, you need to set the Interrupt Mask Register, which we will hereafter refer to as IRQMR. The IRQMR is a single byte located at $0016 within the W5100's address space. Only *7* bits are actually used; bit *4* is reserved and should remain clear. All bits are clear upon power-up, thus masking out all interrupts. To enable a particular interrupt, set its bit to 1. Here is a table of each bit and what it masks:

Bit	IRQ Masked
7	**IP Conflict** another machine has this same IP address
6	**Destination unreachable**
5	**PPPoE Close Enable**
4	*Reserved* should always be 0
3	**Socket 3 Interrupt Enable**
2	**Socket 2 Interrupt Enable**
1	**Socket 1 Interrupt Enable**
0	**Socket 0 Interrupt Enable**

If you are going to be using interrupts at all, you may as well enable all of them, because the Interrupt Register makes it easy to check for and clear any type of interrupt that may occur. To enable all interrupts, simply store the byte $EF at the IRQMR location, which is $0016. Also, ensure that your program clears the Interrupt disable flag in the 6502 process status register.

How to Check for Interrupts

When an interrupt comes through on the W5100, the appropriate bit will be changed in the Interrupt Register (IRQR) to show the cause. The interrupt flag will be shown with a cleared

bit, and it will remain that way until all bits that have been masked from the IRQMR have been set to 1 by the programmer. The IRQR status byte has the same bit layout as IRQMR, as shown in the table above.

Whenever an interrupt occurs, the corresponding bit in IRQR will be set to 1. It is the responsibility of the programmer to handle the interrupt, then reset the bit to 0. Otherwise, no more interrupts of that type will be flagged. Curiously, the method to clear the bit is to write a 1 to it. A minor exception exists for the four Socket interrupt flags. These flags will be automatically cleared when their corresponding Socket Interrupt Register is cleared to $00.

The Destination unreachable flag has some extra functionality to aid the programmer. When this type of interrupt occurs, the unreachable port address (UIPR) and the unreachable port register (UPORT) will contain the destination port and address.

The Socket Interrupt Register

Each of the four sockets has a byte devoted to its own Interrupt Register. This socket interrupt register is located at $0402 for socket 0, $0502 for socket 1, and so on. Only 5 bits are significant. They will be set to 1 when the condition has occurred. They are:

Bit	Status
7	*Reserved*
6	*Reserved*
5	*Reserved*
4	**Send OK**
3	**Timeout** Connection establishment/termination or data transmission
2	**Receive** New data is received, or more is remaining after CMD_RECV
1	**Disconnect** Connection termination is requested or completed
0	**Connection** Connection is established

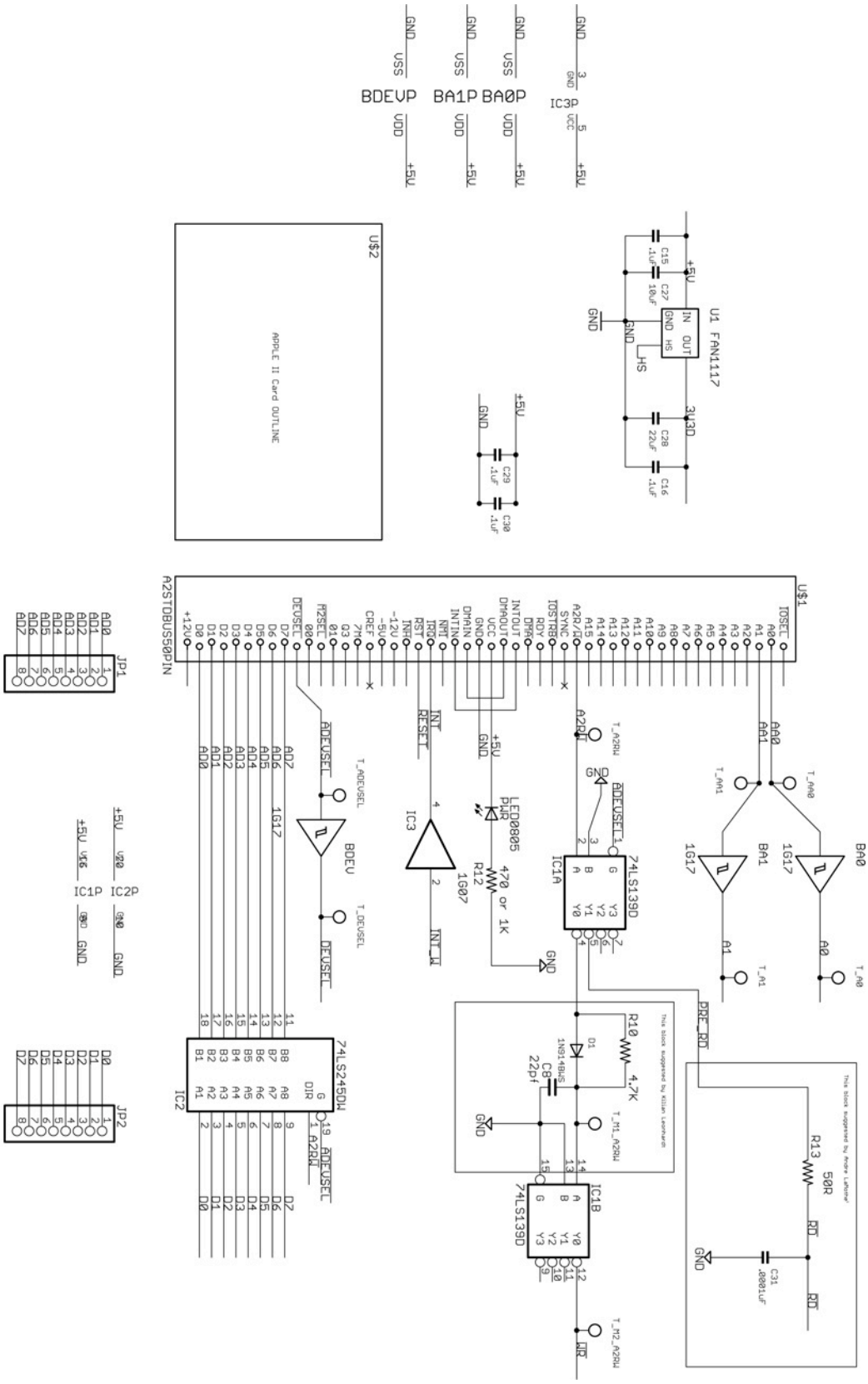
Clear the status by writing a 0 to it. If the programmer writes a $00 byte to this register, then the corresponding socket bit in the IRQR will also be cleared.

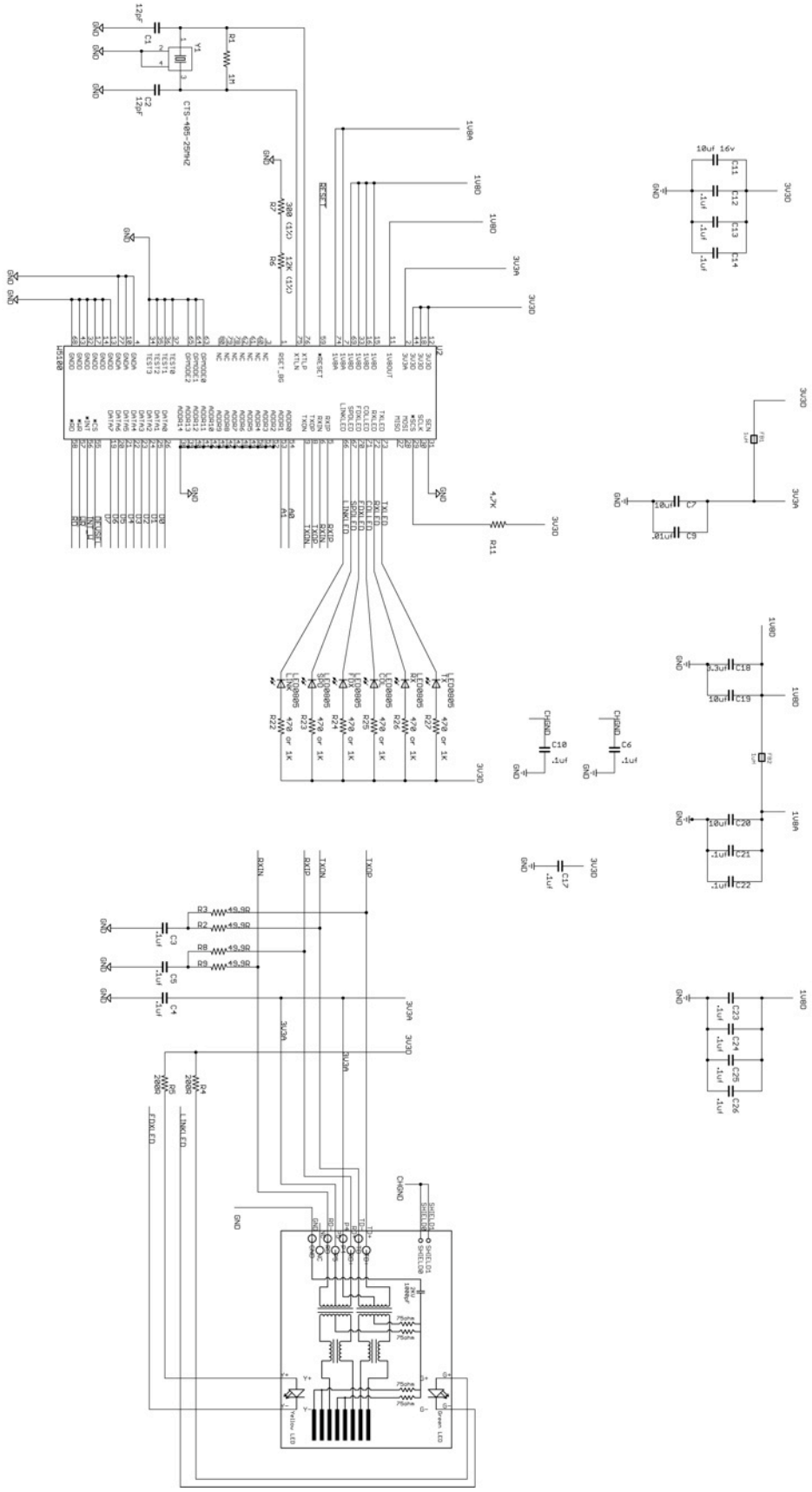
Troubleshooting

Problem	Possible Solutions
The test procedure on page 5 did not return a 03 on the screen.	•Ensure that the Uthernet II is properly seated in its slot. •Ensure that you are using the correct address, ex: C094 for slot 1, or C0C4 for slot 4. •If you are using a IIGS and the Uthernet II is installed in a slot *other than* slot 3 or 4, ensure that the slot setting in the Control Panel is set to Your Card. •Uthernet II is not fully compatible with the original Apple II and Apple II Plus; see problem and solutions below.
The test procedure on page 5 worked, but the green link LED is not lit.	•Ensure that an Ethernet cable is connected from the Uthernet II to another computer, switch, hub, router, or other network device. •Ensure that the other network device is powered on. •The Uthernet II is a 10/100Mb/s device. Ensure that the other device is able to auto-negotiate one of these two link speeds. If the white LED on the Uthernet II is lit, the link speed is 100 Mb/s.
A program which worked with the original Uthernet no longer works with the new Uthernet II.	•If your program is listed in the Supported Software section of this manual, check its web site for an updated version. •If you are using an Apple IIGS with Marinetti you need to download an updated link layer driver for the Uthernet II.
The Uthernet II is not working reliably in an integer Apple II, Apple II Plus, or standard Apple IIe.	•Other peripheral cards may affect the Uthernet II's operation in an older Apple II model. •Try removing other peripheral cards, one at a time, until the Uthernet II operates reliably. •If you are using a Transwarp accelerator, try setting the Uthernet II slot to slow speed using the DIP switch. •If nothing else worked, use Uthernet II in an enhanced Apple IIe or Apple IIGS.

For further assistance, email support@a2retrosystems.com

Schematic


U1 FAN1117
IN
OUT
GND
HS
3V3D
+5V
C15 .1uF
C27 10uF
C28 22uF
C16 .1uF
C29 .1uF
C30 .1uF
IC3P
BDEVP
BA1P
BA0P
VSS
VDD
U$2
APPLE II Card OUTLINE
A2STDBUS50PIN
U$1
IOSEL
A2RST
SYNC
IOSTRB
RDY
DMA
INTOUT
DMAOUT
DMAIN
INTIN
NMI
IRQ
RST
INH
-12V
-5V
CREF
7M
Q3
01
Q0
DEVSEL
D7
D6
D5
D4
D3
D2
D1
D0
+12V
A15
A14
A13
A12
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
T_A2RST
T_AA0
T_AA1
T_A0
T_A1
BA0
BA1
1G17
1G07
ADEVSEL
BDEV
T_ADEVSEL
T_DEVSEL
INT
RESET
IC3
LED0805
PWR
470 or 1K
R12
IC1A
74LS139D
R10
4.7K
D1
1N914BWS
C8
22pf
T_M1_A2RW
T_M2_A2RW
IC1B
R13
50R
RD
C31
.0001uF
PRE_RD
IC2
74LS245DW
DIR
JP1
JP2
IC1P
IC2P
GND


Wiznet W5100

Credits

2018 Update

In addition the original credits I would like to thank André LaMothe for helping me figure out why the card was failing in older Apple II+(RFI) and unenhanced IIe's after what appeared to be some simple changes and layout cleanup.

Thank you to the beta testers who provided feedback: William Andrew, Bob Brown, Patrick Kloepfer, Andrew Madsen, Neil Pobuda, Jeff Ramsey, Lars Wesenick, Matt Wilbur.

I would also like to recognize all the developers that keep the code updated and release new functionality that works with our hardware.

Last but not least, I would like to thank David Finnigan for keeping the manual updated and for designing the new getting started flyer included with all future orders.

Original Credits

I dedicate the Uthernet II to my mom (my assistant tester/shipper). Thanks for all your help!

We all owe Oliver Schmidt a large thank you for his many and continuing open source contributions to Contiki, IP65, and ADTPro. Oliver consulted with me in the early stages of this product's development, on which chip should be chosen for this project, and provided all the primary software support to ensure the correct operation of the hardware. After that he went on to provide drivers and enhancements to Contiki, IP65 and ADTPro.

Ewen Wannop is another force to be reckoned with in the Apple II software world with his own suite of programs for GS/OS, that use both the Uthernet I and II cards. Ewen developed both the original Uthernet I and II link layers that make it possible to use Marinetti and associated applications on GS/OS.

David Schmidt continues to enhance and support ADTPro. David was responsible for an early test version of ADTPro that was compatible with the Uthernet II.

On the hardware front Kilian Leonhardt suggested a solution to resolve compatibility issues with the Apple II Plus and Unenhanced IIe, and Daniel Kruszyna suggested a solution for an issue with the interrupt line.

I would like to thank Tim Haynes for his support and encouragement over the years. He was there when I started working on my first original Ethernet card and has been extremely helpful over the years by loaning me various Apple II systems in order to properly test my creations.

Last but not least is my sincere thanks to my alpha and beta hardware testers whose additional testing help give the confidence to proceed with production of this project:
Jonno Downes, Ed Eastman, Sean Fahey, David Finnigan, Bill Garber, Daniel Kruszyna, Kilian Leonhardt, John Keoni Morris, Andrew Roughan, Oliver Schmidt, David Schmidt, Nigel Sheldon (CL), Antoine Vignau, Ryan Wallmow, Ewen Wannop, Sean Zabriskie.