

CIRCUIT CELLAR

THE MAGAZINE FOR COMPUTER APPLICATIONS

WIZnet iEthernet Applications Special Archive Edition

Internet-Based Weather Data Acquisition

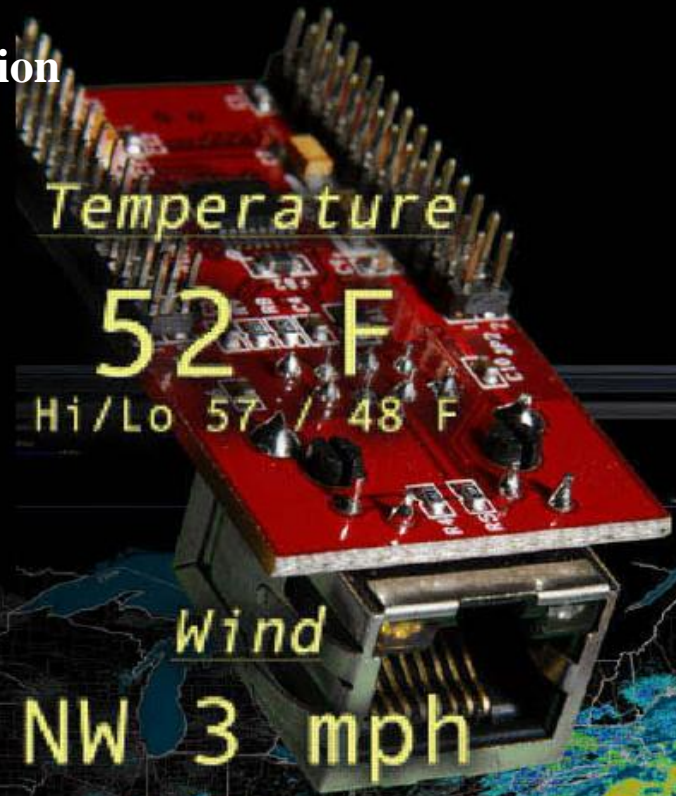
A Compact Webcam Design

Automated Data Mining

Time Server Design

Content Collection and Display

DMX Portal Design

A barcode is located in the bottom left corner. Below the barcode, the ISSN number "0 74470 75349 0 7>" is printed. Below the ISSN, the price information "\$5.95 U.S. (\$6.95 Canada)" is displayed.

0 74470 75349 0 7>
\$5.95 U.S. (\$6.95 Canada)

Welcome to Circuit Cellar, the magazine for computer applications. Over the last 21 years, Circuit Cellar has been dedicated to publishing hands-on articles about embedded design projects, tools, and techniques. These articles are submitted from designers around the world who want to highlight what they've been able to accomplish in their labs.

Circuit Cellar has the good fortune of sharing design contest co-sponsorship roles with a number of high-profile chip companies in the industry. WIZnet joined this impressive list of sponsors recently by co-sponsoring the Circuit Cellar WIZnet iEthernet Design Contest.

The WIZnet iEthernet Design Contest excited Circuit Cellar's readership and encouraged many of the most notable entrants to submit articles that further detailed their accomplishments with the WIZnet devices.

Today, I am pleased to bring you this special archive edition of Circuit Cellar articles so that you may see a sampling of some interesting applications made possible by WIZnet's impressive chip technology. Enjoy!

Sincerely,

Sean Donnelly, Publisher

Inside this issue:

- iEthernet Bootcamp
- Winners Announcement
- The DMX Portal
- Content Collection & Display
- Automated Data Mining
- Time Server Design
- Networked Timing
- Wireless Mobile Robotics
- Web Camera Design
- Internet Weather Display



CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

FOUNDER/EDITORIAL DIRECTOR

Steve Ciarcia

MANAGING EDITOR

C. J. Abate

WEST COAST EDITOR

Tom Cantrell

CONTRIBUTING EDITORS

Jeff Bachiochi
Ingo Cyliax
Robert Lacoste
George Martin
Ed Nisley

NEW PRODUCTS EDITOR

John Gorsky

PROJECT EDITORS

Gary Bodley
Ken Davidson
David Tweed

CHIEF FINANCIAL OFFICER

Jeannette Ciarcia

MEDIA CONSULTANT

Dan Rodrigues

CUSTOMER SERVICE

Debbie Lavoie

CONTROLLER

Jeff Yanco

ART DIRECTOR

KC Prescott

GRAPHIC DESIGNERS

Grace Chen
Carey Penney

STAFF ENGINEER

John Gorsky

ADVERTISING

860.875.2199 • Fax: 860.871.0411 • www.circuitcellar.com/advertise

PUBLISHER

Sean Donnelly
Direct: 860.872.3064, Cell: 860.930.4326, E-mail: sean@circuitcellar.com

ADVERTISING REPRESENTATIVE

Shannon Barraclough
Direct: 860.872.3064, E-mail: shannon@circuitcellar.com

ADVERTISING COORDINATOR

Valerie Luster
E-mail: val.luster@circuitcellar.com

Cover photography by Chris Rakoczy—Rakoczy Photography
www.rakoczyphoto.com

PRINTED IN THE UNITED STATES

CONTACTS

SUBSCRIPTIONS

Information: www.circuitcellar.com/subscribe, E-mail: subscribe@circuitcellar.com
Subscribe: 800.269.6301, www.circuitcellar.com/subscribe, Circuit Cellar Subscriptions, P.O. Box 5650, Hanover, NH 03755-5650
Address Changes/Problems: E-mail: subscribe@circuitcellar.com

GENERAL INFORMATION

860.875.2199, Fax: 860.871.0411, E-mail: info@circuitcellar.com
Editorial Office: Editor, Circuit Cellar, 4 Park St, Vernon, CT 06066, E-mail: editor@circuitcellar.com
New Products: New Products, Circuit Cellar, 4 Park St, Vernon, CT 06066, E-mail: newproducts@circuitcellar.com

AUTHORIZED REPRINTS INFORMATION

860.875.2199, E-mail: reprints@circuitcellar.com

AUTHORS

Authors' e-mail addresses (when available) are included at the end of each article.

CIRCUIT CELLAR®, THE MAGAZINE FOR COMPUTER APPLICATIONS (ISSN 1528-0608) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Vernon, CT 06066. Periodical rates paid at Vernon, CT and additional offices. One-year (12 issues) subscription rate USA and possessions \$23.95, Canada/Mexico \$34.95, all other countries \$49.95. Two-year (24 issues) subscription rate USA and possessions \$43.95, Canada/Mexico \$59.95, all other countries \$85. All subscription orders payable in U.S. funds only via Visa, MasterCard, international postal money order, or check drawn on U.S. bank. Direct subscription orders and subscription-related questions to Circuit Cellar Subscriptions, P.O. Box 5650, Hanover, NH 03755-5650 or call 800.269.6301.

Postmaster: Send address changes to Circuit Cellar, Circulation Dept., P.O. Box 5650, Hanover, NH 03755-5650.

Circuit Cellar® makes no warranties and assumes no responsibility or liability of any kind for errors in these programs or schematics or for the consequences of any such errors. Furthermore, because of possible variation in the quality and condition of materials and workmanship of reader-assembled projects, Circuit Cellar® disclaims any responsibility for the safe and proper function of reader-assembled projects based upon or from plans, descriptions, or information published by Circuit Cellar®.

The information provided by Circuit Cellar® is for educational purposes. Circuit Cellar® makes no claims or warrants that readers have a right to build things based upon these ideas under patent or other relevant intellectual property law in their jurisdiction, or that readers have a right to construct or operate any of the devices described herein under the relevant patent or other intellectual property law of the reader's jurisdiction. The reader assumes any risk of infringement liability for constructing or operating such devices.

Entire contents copyright © 2009 by Circuit Cellar, Incorporated. All rights reserved. Circuit Cellar is a registered trademark of Circuit Cellar, Inc. Reproduction of this publication in whole or in part without written consent from Circuit Cellar, Inc. is prohibited.



DESIGN CONTEST

iEthernet Bootcamp

Get Started with the W5100

Are you ready to join the Ethernet revolution? If so, it's time to start working with WIZnet's W5100 hardwired TCP/IP embedded Ethernet controller. In this article, Fred helps you get started on your first W5100-based design.

I recently received an e-mail from a reader asking why there were no in-depth TCP/IP stack "how-to" articles. Honestly, I had never given that much thought because I normally forego the formal TCP/IP stack in favor of small, easy-to-follow, home-brewed Ethernet driver packages. As a magazine writer, my first guess on the lack of TCP/IP stack magazine literature is the cost versus interest factor. I have reviewed many commercial TCP/IP stack products and I can say from experience that you get what you pay for. My readers simply can't afford or financially justify a full-blown commercial TCP/IP stack for their applications and projects. Thus, why should I ask them to read about a TCP/IP product that they can't afford to use? My second stab at why TCP/IP stacks aren't in magazine vogue is complexity. Many of you have written articles for magazines and you know that you are limited to so many words per article. It would take a series of articles to explain everything you would need to know about TCP/IP stacks.

I must admit that in the past I have offered up some pretty pricey stuff in my articles. These days, I tend to shy away from super-expensive and complex subjects for the reasons I just outlined. However, when I see something that may be what typical technical magazine

readers like you and I are looking for, I'm all over it. For instance, I have found that Ethernet ICs supported by free TCP/IP stacks are very popular with *Circuit Cellar* readers. I've also discovered that many readers who implement single-IC Ethernet devices don't even use a TCP/IP stack. Instead, like me, they employ simple protocol drivers specifically written for the single-IC Ethernet device that they are deploying in their project. I practice what I preach, and what I'm about to introduce to you is the best of both the garage Ethernet driver and TCP/IP stack worlds. How would you like to solder down a single-IC Ethernet solution that provides the power of a full-blown commercial TCP/IP stack as if it were a set of simple Ethernet drivers? Read on, my friend.

WIZnet W5100

The WIZnet W5100 is a single-IC Ethernet solution with a built-in TCP/IP stack. The W5100 folks like to call their on-chip stack a "hardwired stack" because all of the W5100's Internet-enabling goodies are contained within a compact 80-pin LQFP. In addition

to the W5100's hardwired TCP/IP stack, other W5100 Ethernet goodies include an integrated IEEE 802.3 10Base-T and 802.3u 100Base-TX-compliant MAC and PHY. As you would expect, the W5100's TCP/IP stack supports all of the things you need to put an embedded Ethernet gadget on a network. The W5100's TCP/IP stack supports TCP, UDP, ICMP, and ARP, which normally provide enough protocol power for a major portion of embedded Ethernet LAN and Internet projects that are launched by folks like you and me. PPPoE is also supported by the W5100. The inclusion of PPPoE enables you to use the W5100 in ADSL applications.

If you've ever toyed with embedded Ethernet, you know that the lack of a transmit or receive buffer memory can

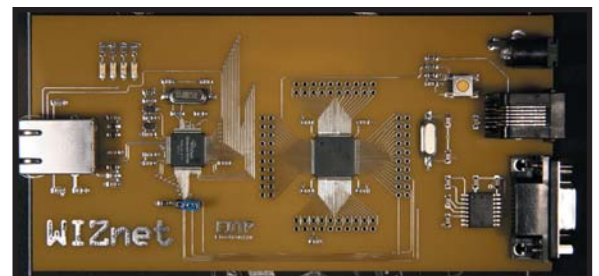


Photo 1—My WIZnet W5100 development board is based on the Microchip Technology PIC18LF8722. The PIC18LF8722 is hefty enough to enable the selective use of Direct Memory mode, Indirect Memory mode, and SPI mode access to the W5100's registers and buffer memory. Using the Microchip PIC18LF8722 also puts the powerful set of Microchip development tools at our disposal.

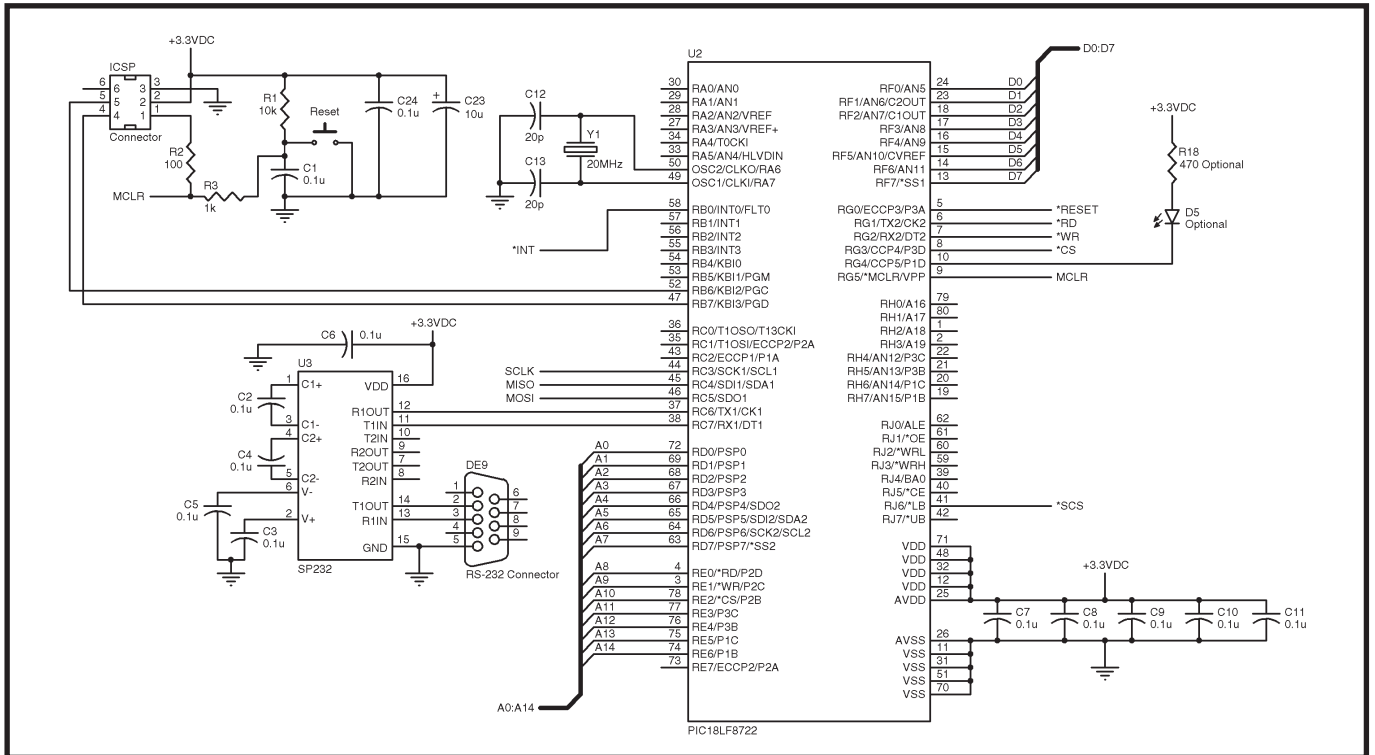


Figure 1—Nothing much I need to say about what you see here. However, the PIC18LF8722 does remind me of my favorite Military Channel quote: “It’s just a good, solid tank.”

be painful and hamper the performance of your embedded Ethernet device. The embedded Ethernet IC manufacturers are aware of this. Most of the single-IC Ethernet solutions offered these days include a fair amount of dedicated transmit and receive buffer memory. The W5100 is no exception, and it is equipped with 16 KB of internal transmit/receive buffer memory.

To avoid the exclusion of smaller microcontrollers, the W5100 can communicate with a host of microcontrollers using an SPI, direct memory access, or indirect memory access. To further accommodate the majority of today’s newer microcontrollers, the W5100 is powered with a 3.3-VDC power source. This enables the W5100 to be directly interfaced to low-power microcontrollers that also run on a 3.3-VDC power rail. The W5100 can also be integrated into legacy 5-VDC systems because its I/O subsystem is 5-V tolerant.

The W5100 supports up to four simultaneously active sockets. Thus, all you need to know is basic socket programming because you will be shielded from the W5100’s internal Ethernet engine operations. The W5100 is designed to provide the bulk of everything needed to produce a working

embedded Ethernet device while being easy to use. The only things the W5100 won’t do for you are write its own code and handle IP fragmentation.

I just happen to have a couple of W5100 ICs. Let’s assemble a W5100-based device from scratch. Once we’ve got the W5100 hardware realized, we’ll put together some Microchip Technology PIC18LF8722 driver code for our W5100 development board.

BUILD A DEVELOPMENT BOARD

For your convenience, I am supplying the PCB layout for an EDTP Electronics-designed W5100 device (see Photo 1). The PCB layout file on the *Circuit Cellar* FTP site is in ExpressPCB format. I chose ExpressPCB because it is a relatively inexpensive PCB manufacturing service that is available to everyone. ExpressPCB software is free for download, and the quality of ExpressPCB PCBs is excellent. Another plus associated with using ExpressPCB is that you don’t have to design your W5100 PCB from scratch. You can use my ExpressPCB PCB template and modify it to meet your needs. If you already have a favorite PCB CAD program, you can easily port my design to your CAD format using my original drawing as a guide. As you would

expect, I haven’t done anything to complicate the W5100 project board design.

The EDTP WIZnet W5100 project board is basically a standard PIC18LF8722 configuration that is wired into a basic W5100 configuration. As you can see in Figure 1, the PIC18LF8722 has enough I/O to wire-in the 15-bit W5100 address bus, the 8-bit W5100 data bus, and all of the W5100 control signals (*RD, *WR, *CS, and *INT) with I/O to spare. In addition to wiring in the W5100 in Direct Bus Interface mode (A0:A14 with D0:D7 and control signals), I attached the W5100’s SPI portal and an SPI select pin to the PIC18LF8722’s SPI I/O interface, which enables you to access the W5100’s internals in W5100 SPI mode. Because the W5100’s address lines are all pulled down internally, the Indirect Bus Interface mode of operation, which uses only two of the 15 address lines, all of the eight data lines, and all of the control signals can also be easily implemented with the EDTP WIZnet W5100 design.

All of the PIC18LF8722’s 80 I/O and power lines are pinned out in blocks of 20 pins to standard 0.1” header pads. The PIC18LF8722 is supported by a 20-MHz clock, a Microchip-certified

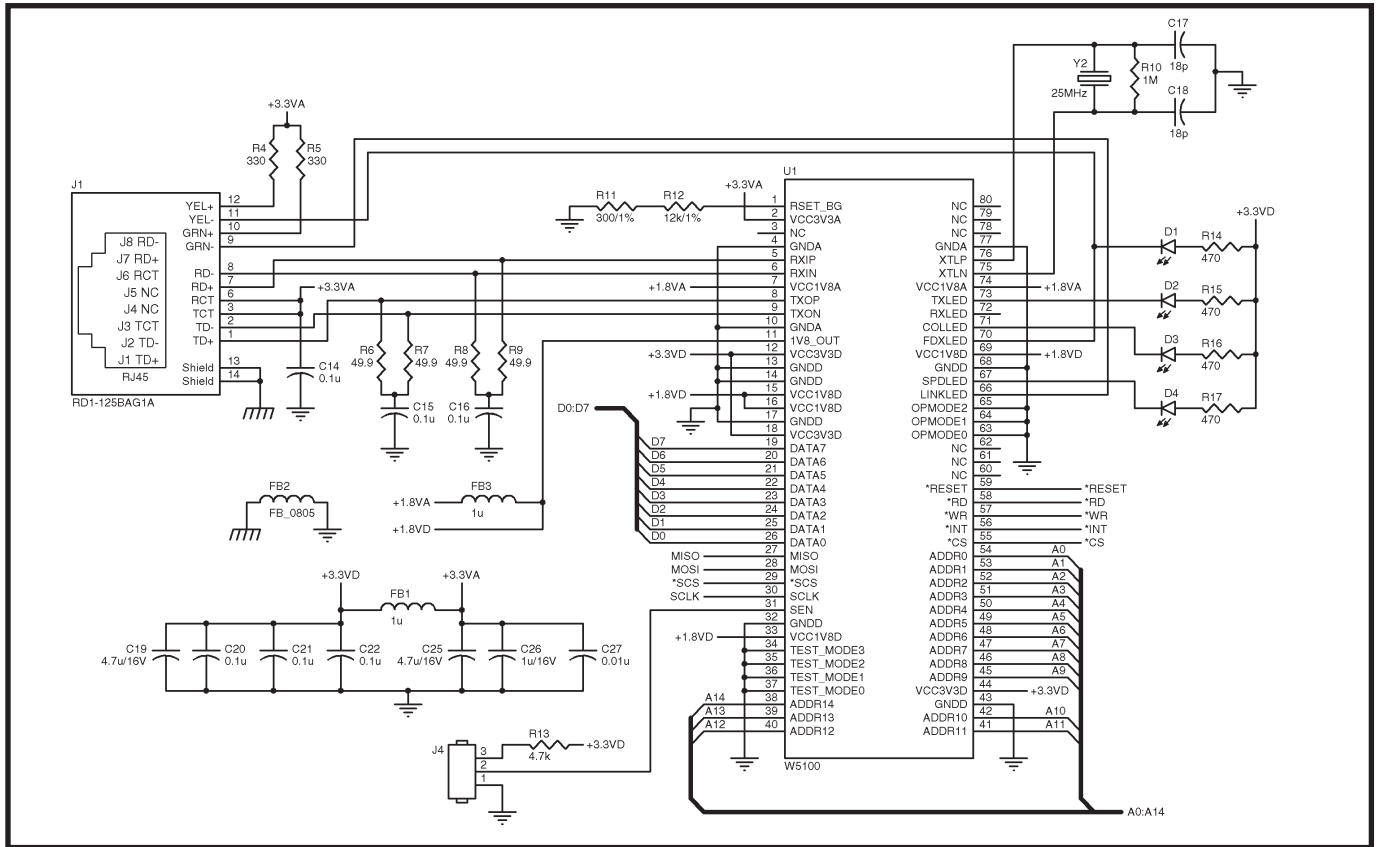


Figure 2—You can get your hands on most everything here from Digi-Key or Mouser Electronics. My friends at Saelig supply the W5100 IC. Saelig doesn't stock the RD1-125BAG1A on its site, but you can probably get the pulse transformer from many of the vendors listed on WIZnet's web site.

fied ICSP programming/debugging portal, and a regulation RS-232 port. I did not include any power supply circuitry because a Digi-Key-supplied 3.3-VDC wall wart does a great job powering the W5100 project board and the external programming/debugging hardware.

On the W5100 side of the EDTP W5100 development board, the W5100 is supported by the required 25-MHz crystal and an all-in-one can of magnetics (see Figure 2). I chose to incorporate the U.D. Electronic RDI-125BAG1A pulse transformer for a couple of reasons. First, the RDI-125BAG1A footprint fits exactly into the old packet whacker pulse transformer footprint, for which I already have a time-proven ExpressPCB pad layout. Second, like the old packet whacker mag jack package, the RDI-125BAG1A has a pair of built-in indicator LEDs in addition to a pair of transmit and receive pulse transformers and the required internal terminating resistors. If you've ever worked with the EDTP ASIX-based and Microchip-based Ethernet development boards, you'll notice that the W5100 PHY connections

are very similar to the EDTP Electronics ASIX and Microchip ENC29J60 designs.

You may wonder why there are no bypass components on the W5100's internally generated 1.8-VDC supply. That question was posted on the

W5100 online technical support question-and-answer board. The W5100 engineering answer was to follow the path that was set forth by the W5100 reference schematic, which is void of 1.8-VDC supply bypass components.

Because the EDTP WIZnet W5100 project board is designed to help you get your W5100 design up and running quickly, I attached all of the W5100 LED indicator lines to LEDs. The pair of RDI-125BAG1A LEDs is connected to the W5100's LINKLED and RXLED status indicator I/O pins. I pulled the TXLED, COLLED, FDXLED, and SPDLED indicators out to discrete LEDs, which you can see in Photo 1 hanging above the city of WIZnet W5100 0805 supporting SMT components. I have also provided a jumper to select W5100 SPI mode if you choose to run your W5100 in that manner. The only oddity I need to point out is the 12.3-kΩ reset resistor pair you see in Figure 2, which is attached to the W5100's RSET_BG pin. A bird's-eye view of the W5100 portion of the EDTP WIZnet W5100 development board is in Photo 2.

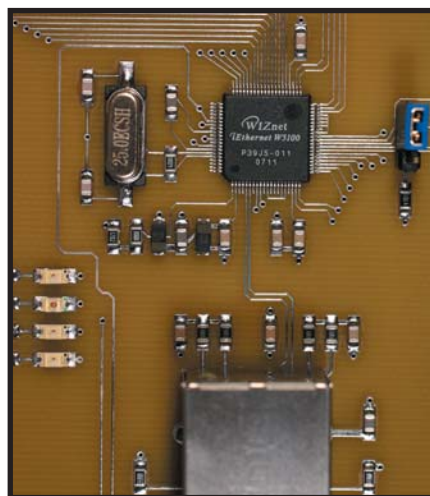


Photo 2—There's nothing here you can't handle. With the exception of the 12.3-kΩ resistor pair, the line of components closest to the W5100 is all filter and bypass components. The PHY components are in the line closest to the pulse transformer. Note the status LEDs and the SPI select jumper at the port and starboard extremes of this photo.

As you can see, the W5100 hardware is a no-brainer. Before we move on to do some W5100 coding, what you don't see in Photo 1 is the heart-beat LED I attached to RG4 on the PIC18LF8722. It's just there as a warm fuzzy to let me know that things are moving on the firmware side. I flash the RG4 LED at a rate of 1 Hz via the PIC18LF8722's Timer3 interrupt-driven real-time clock code.

WIZNET W5100 GARAGE CODE

From a WIZnet W5100 programmer's point of view, the W5100 consists of Common registers, Socket registers, TX memory, and RX memory. The W5100's Common registers consist mostly of W5100 local IP and MAC addressing fields. Also included within the confines of the Common registers are RX and TX memory sizes and PPP/PPPoE parameters. It looks like we will be populating most of the Common registers. So, let's kill two birds with one stone and use the Common registers to test the PIC18LF8722 driver hardware by writing some basic PIC18LF8722 routines to read and write the W5100's registers. I'll use the HI-TECH PICC-18 C compiler in conjunction with MPLAB and a Microchip Technology REAL ICE as my W5100 firmware brewing tools.

About 18 hours later, I returned to write this sentence. I could not get my W5100 to communicate correctly with the PIC18LF8722 to save my life. A cursory look at the W5100 project board didn't indicate any problems. So, I turned to my C code to see if I could find the bug. As it turns out, my C was fine, but my eyes deceived me. A great number of the PIC18LF8722 W5100 address and data I/O pins simply did not get soldered to the W5100 PCB. I use an industrial hot air reflow machine to mount fine-pitched ICs like the W5100 on a regular basis. I've done so many of them that I take the process for granted. Well, this time the reflow machine bit me.

In the meantime, I managed to put some W5100 I/O code together. The official factory W5100 driver code I have is written for AVR devices. So, rather than build my own PIC W5100 include file, I de-Atmeled the factory-supplied W5100 include file. Right now, all I really want from the W5100 include file is

the definition code that lays out all of the W5100's internal register addresses. The W5100 factory include file also contains definitions of all of the W5100 register contents, which I'm sure will come in handy later. I lost a bunch of time chasing my soldering snafu, but I gained some of that time back by Microchip-izing the original AVR include file.

My first official W5100 firmware act was to punch the W5100 into a hardware reset (see Listing 1). Since I went to all of the trouble to fix those W5100 address and data solder joints, I'm going to run in W5100 Direct Bus Interface mode. Running in Direct Bus Interface mode means that I don't have to touch the

W5100 Mode register, which happens to be the very first W5100 Common register. So, we can run our initial W5100/PIC18LF8722 I/O test on the set of Gateway Address registers at address range 0x0001:0x0004. The Gateway Address register addresses GAR0:GAR3 are defined in the include file I converted, which I renamed w5100_pic.h. As you can see in Listing 1, I put together some basic PIC18LF8722 I/O routines to read and write the W5100 registers. Then, I wrote the contents of the gwayipaddr array to the W5100's Gateway Address Common register set. To make sure I performed the Common register write, I turned around and read the contents of GAR0:GAR3 into an array

Listing 1—You won't find this level of coding in the W5100 datasheet examples. Nothing will whizz about without these base register I/O routines.

```
char gwayipaddr[4] = {192,168,0,1};
char svrmacaddr[6];

#define make8(var,offset) ((unsigned int)var >> (offset * 8)) & 0x00FF
#define TO_WIZ          TRISF = 0x00
#define FROM_WIZ        TRISF = 0xFF

*****
void wr_wiz_addr(unsigned int addr)
{
    addr_hi = (make8(addr,1));
    addr_lo = addr & 0x00FF;
}
void wr_wiz_reg(char reg_data,unsigned int reg_addr)
{
    TO_WIZ;
    wr_wiz_addr(reg_addr);
    data_out = reg_data;
    clr_WR;
    NOP();
    set_WR;
    FROM_WIZ;
}
char rd_wiz_reg(unsigned int reg_addr)
{
    char data;
    wr_wiz_addr(reg_addr);
    clr_RD;
    NOP();
    data = data_in;
    set_RD;
    return(data);
}
*****
clr_RSET;
msecs_timer2 = 0;
while(msecs_timer2 < 2);
set_RSET;
addr_i = GAR0;
for(i8=0;i8<4;+i8)
    wr_wiz_reg(gwayipaddr[i8],addr_i++);
addr_i = GAR0;
for(i8=0;i8<4;+i8)
    svrmacaddr[i8] = rd_wiz_reg(addr_i++);
```

called `svrmacaddrc`. You can imagine how pleased I was to see the gateway IP address represented in hexadecimal format in the MPLAB WATCH window shot you see in Photo 3. I tested a bit further by using my W5100 register read routines to read the RTR0 Common register pair, which defaults to 0x07D0 and the RCR Common register that follows and defaults to 0x08. All went well. So, I initialized the W5100's gateway, MAC address, subnet mask, and IP address Common registers.

The next step on our way to putting the W5100 project board online involves setting up and defining the socket memory information. We'll use the default of 2-KB-per-socket sizing, which means we don't touch the RMSR (RX memory size) and TMSR (TX memory size) default register values (0x55). As you can see in Listing 2, all we are really doing is establishing the receive and transmit memory boundaries for each of the four sockets that the W5100 supports. With the socket memory allocation task behind us, we can concentrate on what it takes to manipulate a W5100 socket.

The topmost portion of Listing 3 is the code we will execute to open a W5100 UDP socket. The first order of business is to tell the W5100 what type of socket we want to work with. We are working with UDP at the moment. So, I loaded the socket 0 Mode register with a UDP socket value. I already have an application (EDTP Internet test panel)

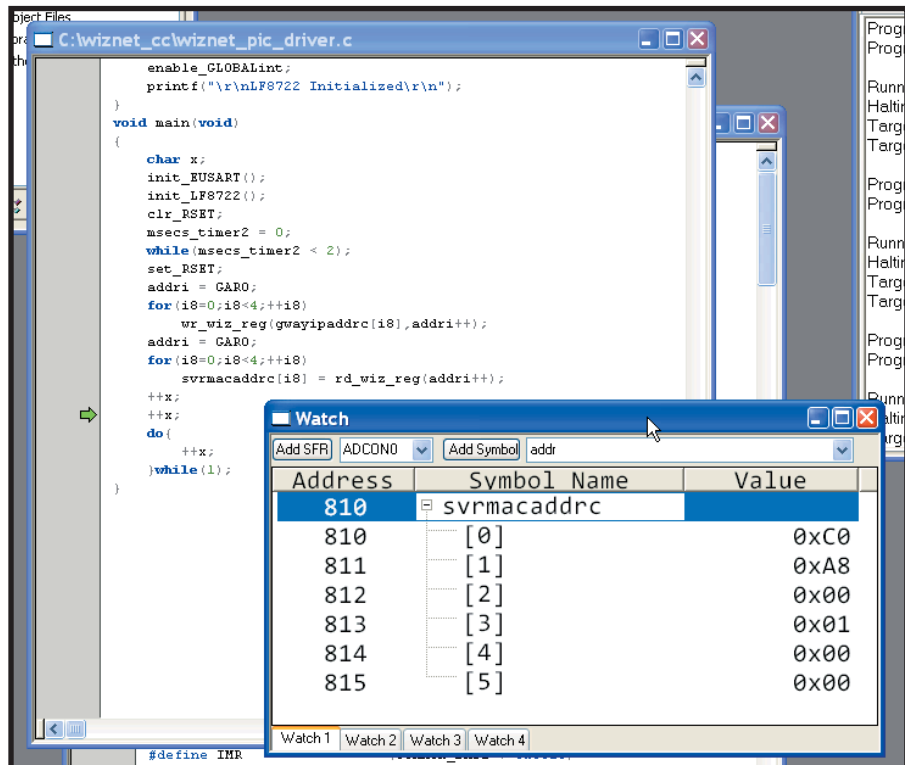


Photo 3—With the success of reading back what I stored in the WIZnet gateway IP address register, we've established a base of operations for reading and writing the W5100's internal registers.

that will send ASCII characters to well-known port 7 and, as you can see in Listing 3, I've loaded the socket 0 Source Port register with 0x0007. We've already loaded our IP and MAC information. Thus, the addition of the UDP source port value enables us to open a UDP socket. From the proliferation of zeros in the Listing 3 socket initialization code, it should be obvious that we will open the W5100's socket 0 in

UDP mode.

Once the socket comes online, we have the power to send and receive UDP datagrams. There are a couple of ways to sense an incoming UDP datagram. We can poll the socket's Received Size register or look for the RECV bit in the socket's Interrupt register. As you can see in the UDP datagram receive code that occupies the center section of Listing 3, I have chosen to use the latter.

An incoming UDP datagram sets the RECV bit of the socket's Interrupt register. Our first reaction to this is to clear the RECV bit by writing a "1" to correspond to the RECV bit's position within the Interrupt register. The W5100 takes care of checksums internally and we, as programmers, never see them in our UDP datagram information. The size of the incoming UDP datagram is automatically posted in the socket's Receive Size register. Here, we retrieve the contents of the Receive Size register and place the value into the `get_size` variable. I used the W5100's datasheet variable names where possible to make it a bit easier for you to compare my W5100 driver code with the UDP pseudocode flow example in the W5100 datasheet. The receive buffer's read pointer value is kept in the

Listing 2—The W5100 datasheet talks about this with pseudocode. Here's my translation.

```
#define chip_base_address      0x0000
#define RX_memory_base_address 0x6000
#define gS0_RX_BASE          chip_base_address + RX_memory_base_address
#define gS0_RX_MASK          0x0800 - 1
#define gS1_RX_BASE          gS0_RX_BASE + (gS0_RX_MASK + 1)
#define gS1_RX_MASK          0x0800 - 1
#define gS2_RX_BASE          gS1_RX_BASE + (gS1_RX_MASK + 1)
#define gS2_RX_MASK          0x0800 - 1
#define gS3_RX_BASE          gS2_RX_BASE + (gS2_RX_MASK + 1)
#define gS3_RX_MASK          0x0800 - 1
#define TX_memory_base_address 0x4000
#define gS0_TX_BASE          chip_base_address + TX_memory_base_address
#define gS0_TX_MASK          0x0800 - 1
#define gS1_TX_BASE          gS0_TX_BASE + (gS0_TX_MASK + 1)
#define gS1_TX_MASK          0x0800 - 1
#define gS2_TX_BASE          gS1_TX_BASE + (gS1_TX_MASK + 1)
#define gS2_TX_MASK          0x0800 - 1
#define gS3_TX_BASE          gS2_TX_BASE + (gS2_TX_MASK + 1)
#define gS3_TX_MASK          0x0800 - 1
```

socket's Read Pointer register. We will use the read pointer value to form the basis for the variable `get_offset`, whose value we will combine with the socket's receive buffer base address to calculate the beginning address of the UDP datagram's header. The UDP datagram header offered by the W5100 is made up of 4 bytes of destination IP address, 2 bytes of destination port address, and 2 bytes of data size information. Thus, the `header_size` variable value is eight. Once all of the addressing calculations have been made, we can use our W5100 read register routine to store the data away in the PIC18LF8722's SRAM for later.

Logically, what is not header information must be data information because we are protected from checksums by the W5100 architecture. With that, we can deduce that the `udp_data_size` variable will contain the number of data bytes we need to retrieve and store. Again, using our home-brewed W5100 I/O code, we read the data from the W5100 receive buffer memory and store it in the appropriate PIC18LF8722 SRAM locations. Our absorption of the UDP datagram and its header is complete. We end our receive session by issuing the `RECV` command in the socket's Command register, which updates the receive buffer pointers. For those of you who are following along with the pseudocode flow in the W5100 datasheet, note that the `udp_data_size` variable is not a W5100 datasheet variable. It's a Fred variable.

Sending a UDP datagram is very similar to receiving one. We'll reuse the information we received earlier and bounce a UDP message back at the sender. Recall that our received UDP datagram header contained a destination IP address and a destination UDP port value. We thought ahead and stored both of the header values. Now all we have to do is retrieve them from the PIC18LF8722's SRAM and load them into the proper W5100 Socket registers. I begin my UDP datagram transmission in that manner within the bottom portion of the code in Listing 3. Using the socket's transmit buffer write pointer, I calculate where in the W5100 transmit buffer to begin stuffing the data I wish

Listing 3—Think about it. All you ever do with any communications device is receive and transmit. I pulled the logic behind this code from the pseudocode flow in the W5100's datasheet.

```
//SOCKET INTI*****
do{
wr_wiz_reg(Sn_MR_UDP,Sn_MR(0)); //protocol = UDP
wr_wiz_reg(0x00,Sn_PORT0(0)); //well-known ECHO port
wr_wiz_reg(0x07,Sn_PORT1(0));
wr_wiz_reg(Sn_CR_OPEN,Sn_CR(0)); //give the open command
if(rd_wiz_reg(Sn_SR(0)) != SOCK_UDP) //wait for the socket to come online
wr_wiz_reg(Sn_CR_CLOSE,Sn_CR(0));
}while(rd_wiz_reg(Sn_SR(0)) != SOCK_UDP);


//RECEIVE*****
do{
//look for incoming UDP datagrams
i16 = rd_wiz_reg(Sn_IR(0));
}while(i16 == 0);
wr_wiz_reg(0x04,Sn_IR(0));
//get the datagram size
hi_byte = rd_wiz_reg(Sn_RX_RSR0(0));
lo_byte = rd_wiz_reg(Sn_RX_RSR1(0));
get_size = make16(hi_byte,lo_byte);
//get the datagram's buffer offset
hi_byte = rd_wiz_reg(Sn_RX_RD0(0));
lo_byte = rd_wiz_reg(Sn_RX_RD1(0));
get_offset = make16(hi_byte,lo_byte) & gSO_RX_MASK;
//calculate the datagram's starting buffer address
get_start_address = gSO_RX_BASE + get_offset;
//UDP header size
header_size = 8;
//store the UDP header information
addri = get_start_address;
for(i8=0;i8<header_size;++i8)
{
packet[ip_destaddr+i8] = rd_wiz_reg(addri++);
++get_offset;
}
//store the UDP data
get_start_address = gSO_RX_BASE + get_offset;
udp_data_size = get_size - header_size;
addri = get_start_address;
for(i8=0;i8<udp_data_size;++i8)
{
packet[UDP_data+i8] = rd_wiz_reg(addri++);
++get_offset;
}
//update the receive buffer pointers
wr_wiz_reg(Sn_CR_RECV,Sn_CR(0));

//TRANSMIT*****
//load destination IP address
addri = Sn_DIPRO(0);
for(i8=0;i8<4;++i8)
wr_wiz_reg(packet[ip_destaddr+i8],addri++);
//load destination port address
addri = Sn_DPOR0(0);
for(i8=0;i8<2;++i8)
wr_wiz_reg(packet[UDP_srcport+i8],addri++);
//get transmit buffer offset
hi_byte = rd_wiz_reg(Sn_TX_WRO(0));
lo_byte = rd_wiz_reg(Sn_TX_WRI(0));
get_offset = make16(hi_byte,lo_byte) & gSO_TX_MASK;
//calculate transmit data buffer start address
get_start_address = gSO_TX_BASE + get_offset;
//load data into transmit buffer
addri = get_start_address;
for(i8=0;i8<udp_data_size;++i8)
{
wr_wiz_reg(packet[UDP_data+i8],addri++);
++get_offset;
}
//update transmit buffer pointer
wr_wiz_reg((make8(get_offset,1)),Sn_TX_WRO(0));
wr_wiz_reg((make8(get_offset,0)),Sn_TX_WRI(0));
//send data
wr_wiz_reg(Sn_CR_SEND,Sn_CR(0));
while(rd_wiz_reg(Sn_CR(0)));
```


to transmit. I then transfer the previously stored UDP datagram data from the PIC18LF8722's SRAM into the W5100's transmit buffer. The W5100 will transmit the data located between the socket's transmit read pointer and transmit write pointer. So, I must update the transmit write pointer by increasing it by the number of bytes I need to transmit. Once that's done, I issue the SEND command and wait for the send success signal, which is a cleared socket Command register. I can now issue a CLOSE command in the socket's Command register to close the socket or send or receive another UDP datagram.

CONGRATULATIONS!

You have completed W5100 bootcamp. In addition to the W5100 register I/O code, UDP transmit code, and UDP receive code, you have a basic and flexible W5100 hardware design to work with.

Here's a hint that will help you determine very early on where your W5100 design stands: Execute only the code through loading the IP address. Load the gateway address, the MAC address, the subnet mask, and the IP address. Don't open any sockets. At this point, you can PING your W5100 design. If you get good PING returns, your PHY hardware and your W5100 register I/O code are good to go. You've also tested and confirmed the operation of your W5100 address, data, and control signals. Bringing up UDP is a fun and easy way to get to know the W5100. The EDTP Internet test panel is a UDP application that runs on your PC. The EDTP Internet test panel is available for download from www.edtp.com. 

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

SOURCES

EDTP Internet test panel

EDTP Electronics, Inc.
www.edtp.com

HI-TECH PICC-18 C Compiler

HI-TECH Software
www.htsoft.com

PIC18LF8722 Microcontroller, REAL ICE, and MPLAB

Microchip Technology, Inc.
www.microchip.com

RDI-125BAG1A Pulse transformer

U.D. Electronic Corp.
www.ude-corp.com

W5100 TCP/IP Ethernet controller

WIZnet, Inc.
www.ewiznet.com

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2007/208.



WINNERS ANNOUNCEMENT

The WIZnet iEthernet Design Contest 2007 gave engineers throughout the embedded design community a chance to join the Ethernet revolution while competing for a share of \$15,000 in cash prizes and international recognition. Designers from around the world quickly stepped up to the challenge by incorporating WIZnet's W5100 hardwired TCP/IP Ethernet controller in innovative embedded projects. Within weeks of the contest launch, designers began submitting their exciting, next-generation, Ethernet-enabled embedded systems.

After spending many long days and nights closely studying the entries and judging them on their technical merit, originality, usefulness, cost-effectiveness, and design optimization, the judges presented their scores to the contest administrator. The results are now final, and we're proud to announce the winners.

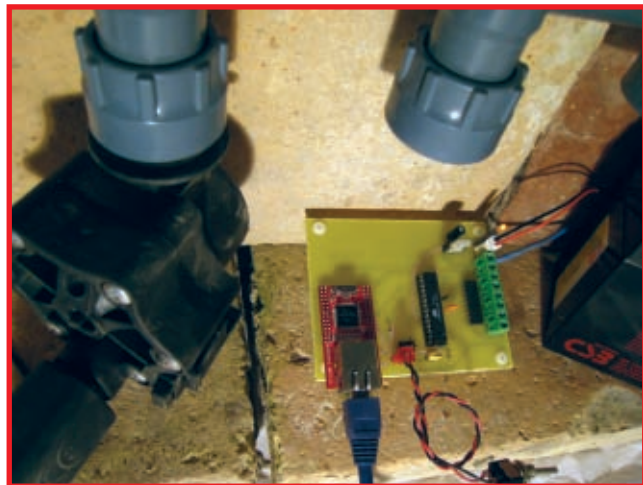
Congratulations to everyone who took part in the contest!

FIRST PLACE

Drip Irrigation Controller

The irrigation timer with advanced planning (ITAP) is a truly next-generation irrigation control system. Featuring a WIZnet WIZ810MJ network module and an Atmel ATmega168, the innovative controller provides user interaction through a standard web browser. As a result, the system doesn't have a keyboard or an LCD. The single-controller unit can manage up to eight zones. No software installation is required. Its functionality is split between the browser-based user interface and the hardware-based web server, data model, and control logic. The web server is used to read and write the internal data model. Its other function is to return files stored in internal program memory. Precision irrigation control is now a reality because the system provides useful information such as watering schedules and zone activity.

Thomas Bereiter
Italy
itimer@micaview.com



"My project is an irrigation timer. Timers are simple devices, but their user interfaces are unreasonably complex. An Ethernet connection made it possible to correct this imbalance by using a remote browser in place of a local LCD and keypad. Once browser-connected, it was possible to add 'what if' planning tools that would be unthinkable on a stand-alone device. The hardware is simply an ATmega168, a ULN2803 driver chip, the WIZnet module, and not much else. I had been looking at various ways of adding browser support to a USB-based design. Previously, I had rejected Ethernet for either cost or complexity reasons. When the design contest introduced the WIZnet module, it was instantly clear that it would greatly simplify the design. With the WIZnet module, I could keep the parts count down and not waste scarce flash memory on networking code."

— Thomas Bereiter

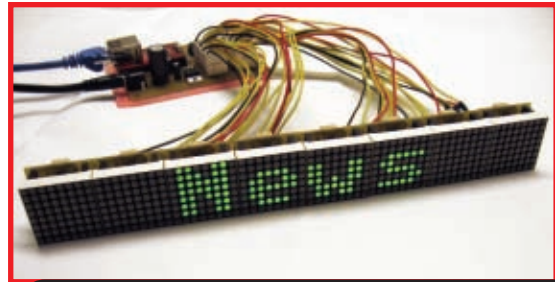
To see these projects and more, visit www.circuitcellar.com/wiznet/.

SECOND PLACE

LED News Ticker

The handy LED News Ticker brings the news to you by displaying up-to-date headlines in a scrolling format. The system features a main board and eight slave boards attached to dot-matrix LED displays. The main board features a Microchip Technology PIC18F2525 microcontroller connected to a WIZnet WIZ810MJ Ethernet module, which uses the W5100 to provide an easy-to-use interface to the Internet. The LED News Ticker requires no interaction to operate. Once powered up, the device immediately connects to the Internet and downloads news updates every 15 minutes. It handles all DHCP leasing and DNS resolving, allowing you to use dynamic IP addresses.

James Blackwell
U.S.
azoore@azosoft.com



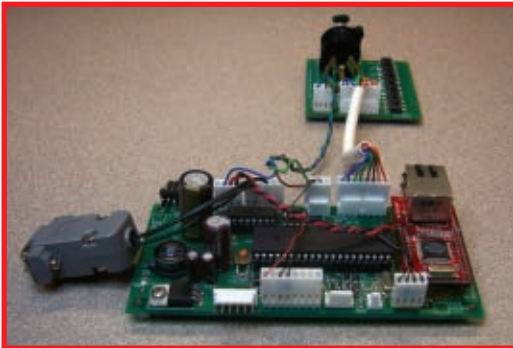
"The LED News Ticker consists of a main board that communicates to eight individual slave boards. The main board is fairly simple, using only a PIC18F2525 and a WIZ810MJ module to connect to the Internet. Each slave board piggybacks to a single 8 x 8 dot-matrix LED display, which is controlled by a PIC18F2221. The main board scrolls news headlines across the display by manipulating data in a frame buffer that is sent to the slave boards. The WIZnet modules really simplified the design of my system without sacrificing any usability. I plan on using them in future projects as well."

— James Blackwell

THIRD PLACE

DMX Portal

The well-designed DMX Portal is an affordable DMX lighting controller. You can use the novel system to remotely control up to 512 channels through an IP-based network or directly interface them to embedded systems with a serial connection. It was designed to be perfectly suited for designers who want to off-load DMX management and refreshes from the main system controller. It's also useful for distributed lighting systems where low-cost Ethernet wiring is more practical than expensive RS-485 wiring. The prototype includes an external EEPROM for scene storage and a Microchip Technology PIC18F4620 microprocessor. A WIZnet WIZ810MJ evaluation board is connected to the SPI on the PIC development board.



"The DMX Portal is a self-contained lighting controller for embedded systems and large distributed systems. I started this project because I wanted to be able to control moving lights and other special effects that use the DMX protocol from a system that was low cost and could change the lighting state based on digital triggers or simple commands from other systems. The WIZnet parts were a good fit for my design for several reasons. I was already using the PIC's hardware UART for my optional RS-232 interface, which required me to generate the DMX serial output completely in software. Since the DMX output requires precise timing to generate the correct bit rate, I need to disable all interrupts while the DMX data is being refreshed. The W5100 offers a very large buffer which is sufficient to store incoming commands arriving while the processor is unable to process the incoming data. Another reason the parts were a good fit was that the chip handles all the tasks of receiving and transmitting a UDP packet."

— Matt Ernst

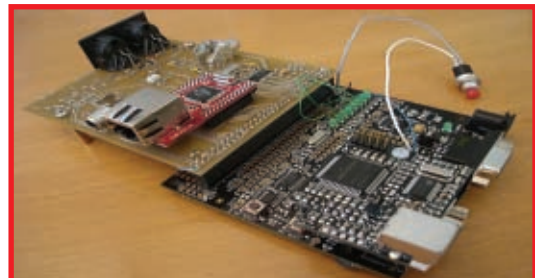
Matt Ernst
U.S.
nomadelectronics@gmail.com

FOURTH PLACE

Remote Real Virtual Instrument Interface

With the amazing Remote Real Virtual Instrument Interface, you can control any musical instrument with a MIDI input and capture its audio output over the Internet. You can also use the well-designed streaming media device to record audio if you don't need MIDI. It features a WIZnet W5100 hardwired TCP/IP chip, a Ramtron VRS31L3074 microcontroller, and a Texas Instruments TLV320AIC23B audio CODEC. The system's software is split into two parts: an embedded portion for the VRS31L3074 microcontroller and a PC portion for the VSTi plug-in. The PC-side software provides the interface to the virtual music studio software.

Clemens Valens
France
cvalens@yahoo.com | www.polyvalens.com



"My project is a networked audio and MIDI interface that integrates with virtual music studio software that supports the VST standard. The host sends UDP packets with MIDI data over the network to the processor and the processor outputs the MIDI data on its MIDI port. A synthesizer responds to the MIDI data by playing a sound. The audio CODEC samples the synthesizer output and transfers the samples to the processor. The processor fills UDP packets with these samples and sends them over the network to the host. The host then plays the sound. The W5100 allowed me to use simple hardware to build my project. I was actually looking into some kind of FPGA solution when this one came along. No need to sacrifice half of your processor power for a TCP/IP stack."

— Clemens Valens

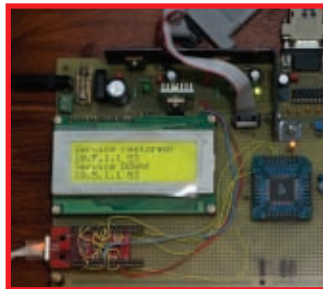
To see these projects and more, visit www.circuitcellar.com/wiznet/.

HONORABLE MENTION

Portable Network Service Monitor

This portable network service monitor was developed to help network administrators supervise datacenters. The handy monitor—which features a WIZnet WIZ810MJ module and an Atmel ATmega128 microcontroller—is equipped with a 4 × 20 LCD that can display important messages from any configured server in a local network. It also continuously checks connectivity to predefined services on different machines. When problems occur, it triggers an alarm.

Alexander Popov & Peter Popov
Bulgaria
sasho@popovbrothers.com



"We built a handheld network monitor that is not only an inexpensive solution, but also an extremely flexible one. The device consists of a WIZnet WIZ810MJ module, an Atmel ATmega128L microcontroller, a power supply, a 4 × 20 LCD, and a TTL-to-RS-232 converter for debugging. The WIZ810MJ module fits quite well in an 8-bit TCP/IP-enabled design without external memories, thus saving money and space. We enjoyed working with the module because of the easy hardware interfacing and good how-to documentation."
— Alexander Popov & Peter Popov



ThermoNet

The ThermoNet is a web-based, remote control, residential HVAC thermostat. The well-designed system includes an easy-to-use LCD front panel and a built-in web server. The front panel includes four buttons: Mode, Fan, Up, and Down. The hardware consists of a WIZnet WIZ810MJ module connected to an Atmel ATmega128 microcontroller external memory interface via direct memory mapping. An Atmel DataFlash chip provides ample storage for embedded web pages and is easily updated with new web pages and other files via a Windows-based program.

Kevin Houser
U.S.
thermonet@rocketfarmers.com

"ThermoNet is a web-enabled, dual-use project. It was designed and built to provide an easy-to-use web interface to manage heating and cooling energy usage, as well as water usage for irrigation. The primary parts used are an Atmel ATmega128, a WIZnet WIZ810MJ Ethernet engine, a Dallas RTC with 32 KB of nonvolatile static RAM, a 512 KB to 2 MB Atmel DataFlash memory, two temperature sensors, and a 2 × 16 LCD with LED backlight. The WIZnet parts were almost perfect for my project because they allowed me the freedom to implement any UDP or TCP/IP protocols I cared to develop or use. Many microcontroller-targeted Ethernet solutions are simply serial-to-Ethernet adapters or provide a canned web server implementation with limited expandability."
— Kevin Houser

FATE: Flexible Audio Transmission Over Ethernet

This project addresses the idea of digital audio for the masses. The purpose of flexible audio transmission over Ethernet (FATE) is to set up a simple dedicated wired Ethernet network. You can then use the network to coordinate the distribution of high-quality audio signals throughout a building and the area around it. The useful design uses a full parallel bus interface to a WIZnet W5100 IC. The IC is memory mapped on the "auxiliary bus" (expansion bus) of the processor core. Its register values can be seen and manipulated at any time. Interrupts aren't used.

John Clayton
U.S.
jclaytons@earthlink.net

"My project is an atypical way of looking at audio distribution to speaker boxes. It separates the audio signal from the raw power. That way, the power can be provided right at the speaker boxes, and the information signal can be transmitted to a set of surround-sound speakers directly, using dedicated CAT-5 wired Ethernet links. Digital audio is noise immune, and the mantra of my project becomes: 'No more MONSTER cables! Use these cheap CAT-5 cables instead!' I enjoyed the self-contained nature of the WIZ810MJ, especially the cool Ethernet jack with built-in pulse transformer. I also liked the 'auto-crossover-cable adjustment' feature of the W5100 chip."
— John Clayton



Travel WIZard

The incredible Travel WIZard is an embedded server application that helps you find airfare deals. The useful system uses Kayak, an online travel search engine, to explore the Internet. It then returns data that can be graphed to reveal the cheapest time of year to travel. The Travel WIZard features a WIZnet W5100 Ethernet controller and a Microchip Technology PIC24FJ128GA010 MCU, which resides on an Explorer 16 development board. The board includes a 32K × 8 serial EEPROM, LEDs, buttons, and a 2 × 16 LCD.

Matthew Pennell & Aaron Thomas
U.S.
alphantango22@gmail.com

"The Travel WIZard is an embedded server data-mining application. It allows the user to acquire airfare data to look for trends in ticket price. The software is written in such a way so as to support eventual expansion to other automated Internet tasks. Ultimately, we wanted to see if we could design a customizable platform that could be programmed to do any type of automated online data mining task. In addition, of course, to the WIZ810MJ with W5100, the system consists of a Microchip Explorer 16 development board with a PIC24FJ128GA010 MCU and 32K × 8 serial EEPROM. From a hardware perspective, we appreciated how easy it was to prototype with the WIZ810MJ plug-in module. From a software perspective, we found it easy to quickly learn to use the W5100. We found the part to be an ideal tool for learning the nuts and bolts of how the Internet actually works under the hood."
— Matthew Pennell & Aaron Thomas

To see these projects and more, visit www.circuitcellar.com/wiznet/.

HONORABLE MENTION



Web Camera

The versatile Web Camera system can take a picture at a resolution of 640×480 or 320×240 , pan the camera horizontally and vertically, and change its IP and gateway address to match a network. Photos are taken with a C328 JPEG compression module, which serves as a JPEG-compressed still camera. The host can send a snapshot command to capture a full-resolution, single-frame still picture. The picture is then compressed by the JPEG engine (OV528) and transferred to the host. After each photo is divided into 64-byte segments, a WIZnet WIZ810 Ethernet module transmits the packets over the Internet.

"My project is a Web Camera whose images can be accessed using the Internet, instant messaging, or a PC application. The project involves a PC application that enables the user to access and control the camera. The user can rotate the camera vertically and horizontally via the PC-based application. The heart of the Web camera is a Microchip dsPic30F4013 microcontroller, which puts the photos in packets and uses the WIZ810MJ module to send them to the Internet. The competition was a great opportunity to use the ready-to-use module with the W5100 chip on it."

— Minas Kalarakis

Minas Kalarakis
Greece
info@kalarakis.gr

Time Server

The well-designed Time Server keeps a master time and date clock that is synchronized to the U.S. WWVB time-code signal. It exists on the Ethernet network and serves time and date information according to the SNTP, DAYTIME, and TIME protocols. Client devices can connect to the system, request the time/date, and synchronize their local clocks. Because the Time Server doesn't rely on Internet servers, it can be used in secure networks that aren't connected to the Internet. A WIZnet W5100 provides the interface to the Ethernet network. A Freescale MC9S08QG8 microcontroller is used to decode the time-code pulse stream, update a real-time clock, and serve time/date information to clients on the Ethernet network.

Steven Nickels
U.S.
ssea000@gmail.com



"The Time Server is a fixed-function node on an Ethernet network that provides time and date information referenced from the NIST WWVB time-code radio signal. Once I developed the interface code that set up a socket connection, using the W5100 was very simple. I especially liked that I didn't have to compile a huge Ethernet stack. The SPI interface was especially important in the Time Server design since I wanted to use a microcontroller with a low pin count. Additionally, since the TCP/IP stack is embedded in the W5100, I don't have to worry about code integration issues, large flash and SRAM requirements, or license and royalty costs."

— Steven Nickels



Greener Lawn: A Sprinkler Control System

The well-made Greener Lawn system gathers historical weather data and forecasts and then makes intelligent watering decisions based on that data. The design features an ATmega128 processor connected to the Internet through a WIZnet WIZ810MJ. The weather forecasting and rainfall totals come from the National Weather Service's FTP server. Linux shell and Perl scripts gather the data and parse out the rainfall totals. This is stored as plain text files on the Linux web server. A PHP script running on an Apache web server enables you to configure the sprinkler controller. A second PHP script ties all of this information together into a single downloadable file that is requested by the WIZ810MJ.

Zack Clobes
U.S.
zack@custom-ds.com

"A long-time pet peeve of mine has been watching underground lawn sprinklers running during a rain shower, or when it's obvious that a shower is coming at any moment. It seems like such a waste to consume that energy and water. The Greener Lawn sprinkler controller system aims to be smart enough to make some basic decisions about whether or not it's a good idea to water the lawn. It consists of two primary pieces: the controller and a web server. An ATmega128 processor connected to the Internet through a WIZnet WIZ810MJ is the controller that actually controls the pump and solenoids. I was able to offload all of the physical Ethernet packet handling to a separate 'black-box,' thereby allowing me to use the relatively simple and inexpensive microcontroller. Knowing that I had a good network controller that, with just a few lines of code, would start responding to ping requests expedited the development process."

— Zack Clobes

NIETO: An NCID and NTP Client

The NIETO is an innovative network caller ID and NTP client. Featuring a WIZnet W5100 and an Atmel ATmega644, the system uses TCP to attach to an NCID server to retrieve and display caller ID information on an LCD. As an IP client, NIETO attaches to an NCID server and retrieves caller ID data over the Internet. The most recent call is always displayed.

Thomas Glembocki
U.S.
tomgle@yahoo.com



"I built NIETO in order to have a standalone box on my LAN to retrieve caller ID info without having to use a PC. The WIZnet W5100 provides four full TCP/IP sockets so that very little programming is needed to establish a TCP connection over Ethernet with a server. I was able to use the standard GCC C compiler WinAVR to open TCP sockets and send and receive data without any knowledge of what was taking place under the hood. The W5100 took care of all the Ethernet framing stuff and the TCP acknowledges, CRC checking and the like. It contains large enough buffers to handle TCP/IP frames without tying up precious CPU RAM space. In my case, all the CPU RAM was dedicated to storing caller ID data instead."

— Thomas Glembocki

To see these projects and more, visit www.circuitcellar.com/wiznet/.



The DMX Portal

Obtain Lighting Control Via Ethernet

Ready to build your own DMX lighting controller? Matt's design enables him to remotely control up to 512 channels through an IP-based network or directly interface them to embedded systems with a serial connection. It is perfect for distributed lighting systems where low-cost Ethernet wiring is a better option than RS-485 wiring.

The DMX Portal is a self-contained lighting control unit that you can use to control moving lights or special effects equipped with a DMX interface (see Photo 1). The project provides a low-cost, flexible way to interface embedded systems with DMX devices, and to allow DMX control to be distributed over long distances using Ethernet.

Most programmable lighting is designed for stage performances. Stand-alone lighting boards are the most common playback controllers/programming interfaces used in these applications. The programming interface provided by the lighting boards is implemented via slider controls and buttons similar to an audio mixer. This type of interface is most appropriate for stage performances

that change frequently or need to be adjusted on the fly. In applications such as permanent installations or automated applications, the programming doesn't need to change frequently, but the applications usually require the lighting controller to be controlled from another system instead of through physical controls. For this reason, the DMX Portal does not have a physical programming interface. Instead it offers RS-232 and Ethernet communication interfaces and two different protocols, which allow it to be used in a variety of different applications.

The system has an effects engine to automatically generate timed fades with simple commands. It also allows user-defined scenes to be saved and recalled when a command or digital trigger is received. These features enable the DMX Portal to fit into a variety of applications, ranging from a simple virtual lighting board emulated on a PC to a self-contained lighting control unit in an embedded system.

To keep project costs down, I built the DMX Portal around a Microchip Technology PIC18F4620 microcontroller and a WIZnet W5100 Ethernet interface. The complete system costs less than \$50, making it extremely cost-effective in comparison to other DMX controllers.

DMX PROTOCOL

To understand the DMX Portal, it is useful to first understand how DMX

works. At the highest level, DMX is nothing more than a serial transmission of 8-bit values. Data is transmitted at 250 kbps in frames that consist of the following sections: BREAK, MARK AFTER BREAK (MAB), START CODE data slot, and up to 512 channel data slots (see Figure 1).

The term BREAK means a low state where the voltage on the (+) DMX data line is lower than the voltage on the (-) DMX data line. The term MARK means a high state where the (+) line has a higher voltage than the (-) line. The START CODE and channel data slots each contain 11 bits, which are 4 μ s in length. The first bit is the start bit and is always low. The next 8 bits are the data portion of the slot, with the least significant bit first. The final 2 bits are stop bits and are always high. The START CODE can have different values for the 8 data bits, but it usually contains a value of 0x00 to indicate that the following data slots represent individual channel data. Delays may be added between any of the data slots as long as the data lines remain in the high (MARK) state and the delay does not exceed 1 s. The optional delay is useful because it allows time for the processor to attend to other tasks periodically during DMX transmission. The only limit on the delay is that the next frame must be sent no more than 1.025 s after the start of the previous frame. This

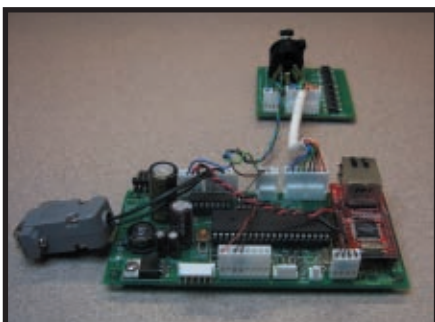


Photo 1—The DMX Portal is a compact lighting control unit. It's designed around a WIZnet 5100 Ethernet controller and a Microchip Technology PIC18F4620. The WIZnet development board is secured to the right side of the PIC development board. The five-pin XLR connector and RS-485 level converter are on the add-on board behind the PIC development board.

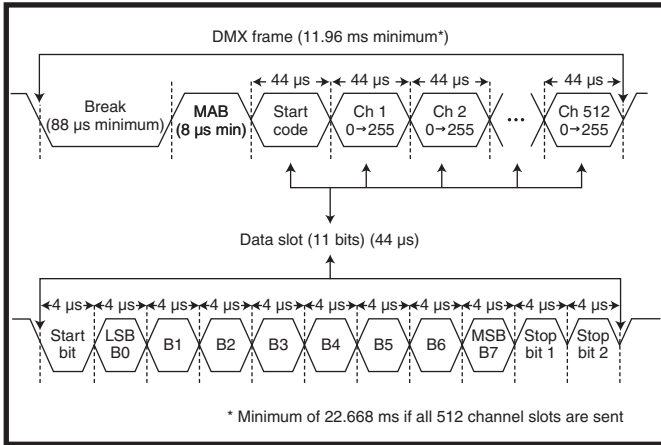


Figure 1—The DMX protocol transmits data in frames that consist of an 88-µs break, an 8-µs mark after break, and up to 513 data slots. The start code defines the type of data contained in the start slots following it, which makes the DMX protocol flexible enough to control different types of devices over a shared cable.

requirement ensures that data on the DMX line is always refreshed at a minimum rate. Many programmable lights are designed to go into a low-power state if no DMX data is received within a certain amount of time, effectively building an automatic power switch into the protocol.

The physical layer of DMX is a five-wire interface using a five-pin XLR connector. Pin 1 is Shield (ground), pin 2 is Data (-), pin 3 is Data (+), pin 4 is Optional Secondary Data (-), and pin 5 is Optional Secondary Data (+). The DMX specification requires the use of a five-pin XLR connector. The connector provides primary and secondary RS-485 data channels. The data lines (pins 2 and 3) use the RS-485 signaling specification also known as EIA-485. The secondary data channel (pins 4 and 5) is almost never used in practice and is the biggest source of differences among vendors.

Prior to the DMX512-A specification, vendors sometimes used a three-pin XLR connector and did not include pins 4 and 5 at all. Three-pin XLR connectors are commonly used for microphone connections and this practice allowed people to accidentally connect DMX and audio devices, potentially damaging equipment.

RS-485 also requires cable that has 120-Ω differential impedance. Standard microphone cable does not have this impedance, which can lead to poor signal integrity and data errors. Other manufacturers sometimes used pins 4 and 5 to deliver power, a practice that is also prohibited by the DMX512-A specification.

Common examples of multichannel receivers are dimmer packs with multiple AC outputs and moving lighting that uses multiple channels to control the intensity, x-axis, y-axis, and color of the light. Because DMX often involves long cable lengths and multiple receivers along the length of the cable, signal integrity is important to prevent bit errors. Using the proper impedance cable and terminating the end of the cable with a matched resistor will eliminate most signal integrity problems.

ADVANTAGES OF THE DMX PORTAL

Earlier in this article, I described the typical “lighting board” style of the DMX controller. When using this type of controller to create a light show,

RS-485 was designed to be a multidrop interface that allows multiple receivers to be on a single line. The DMX512-A specification allows up to 32 receivers on a single line without buffering the signal. Each receiver may respond to one or more of the 512 DMX channels that can be transmitted on a single line. Com-

the parameters of the show are typically entered into the controller’s memory as “scenes.” A scene is basically a snapshot of the current state of all the DMX channels in use. When finished, the show consists of a large number of these scenes, which can be recalled in a timed sequence to generate changing lighting similar to flip-book animation.

The aforementioned method works well for some applications, but there are situations where it has disadvantages. One example is architectural lighting in a building such as a restaurant. Restaurants typically have many rooms and tend to keep the lights lower for dinner than they do for breakfast and lunch. You may want to design a system that uses DMX-controlled dimmers for each room and automatically dims the lights in each room at the start of dinner. You may also want the ability to adjust the lighting in each room separately to compensate for the amount of light from windows or for special events. In this situation, scenes are a poor programming method because the exact level needed for each dimmer channel is not always the same. A standard lighting board would require you to have a scene defined for each possible combination of brightness in each room. Even if you allow only five discrete brightness levels per room and had four rooms, this would result in 625 scenes to cover every possible combination. Because the DMX portal gives you a way to programatically

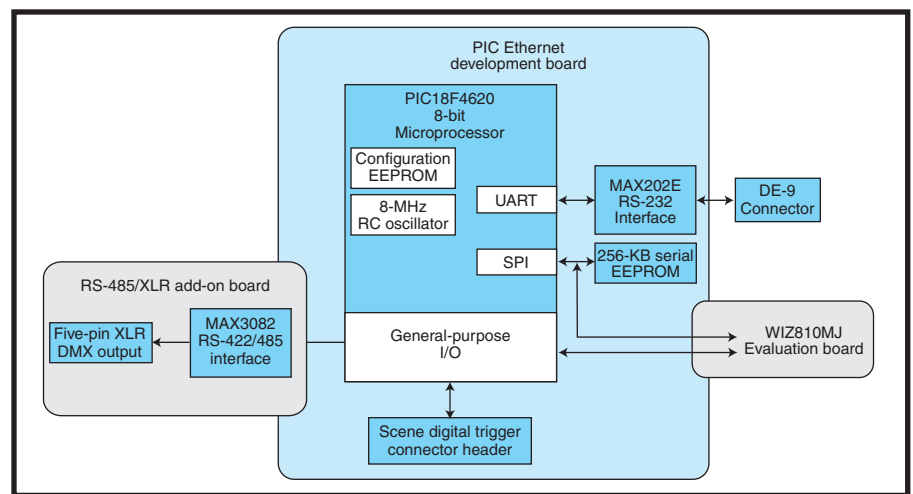


Figure 2—The DMX portal includes three boards. The PIC18F4620 microcontroller has more than enough power to handle the functionality of the DMX features, command processing, and the RS-232 interface. The WIZnet evaluation board offloads all of the processing requirements for an Ethernet interface and connects to the PIC18F4620 via the SPI port.

control the level of each channel and recall saved scenes, the task would be much simpler. A scene could be defined for the default levels of every room at the start of each meal period. If levels other than the defaults are required, the commands in the DMX Portal control protocols make it easy to modify the levels of individual channels on the fly. With this level of control, you can easily implement a system that provides many more manual brightness levels for each room without programming a ridiculous number of fixed scenes.

HARDWARE

As you can see in Photo 1, the DMX Portal prototype consists of three boards. Figure 2 shows how the boards are connected. The main board is a simple PIC development board, which contains power supplies, an RS-232 interface IC, and a 256-KB serial EEPROM. The board also has a few LEDs and headers for the PIC's debug lines and the remaining I/O pins (see Figure 3).

A PIC18F4620 microcontroller is the core of the system. It is responsible for maintaining the current state of all DMX

channels, generating the DMX output stream, processing incoming commands, and generating effects like fades. This project uses many of the PIC18F4620's peripheral hardware features. The internal EEPROM is used to store user-defined settings that are applied when the system is turned on (such as whether to enable DMX output, a value to initialize all channels to, and IP settings for the Ethernet interface). The PIC18F4620's serial communication hardware is used to implement the RS-232 asynchronous interface and SPI communication to the external EEPROM and the W5100 Ethernet controller.

The RS-485/XLR add-on board is a simple PCB that contains an RS-485 interface IC to translate the 5-V CMOS output of the PIC18F4620 to a differential output meeting the RS-485 fault-tolerance specifications (see Figure 4). This board also contains a five-pin XLR connector defined by the DMX512-A specification as the proper connector for DMX interfaces and eight tactile switches I used to debug the trigger functionality of the DMX portal.

The final board is the WIZnet

WIZ810MJ evaluation board, which contains the W5100 hard-wired TCP/IP stack IC. The board handles all of the low-level details of the UDP Ethernet interface for the DMX Portal. It also provides the passive components and the RJ-45 connector required for the Ethernet interface. Data is transferred to the microcontroller through the SPI port rather than the parallel interface, but this could be easily changed if high throughput via the Ethernet interface is required.

FIRMWARE

I wrote the firmware for this project completely in assembly with Microchip's MPASM compiler. Even though writing in assembly can be more time consuming and make complex algorithms more difficult to read, I had two reasons for using this language. The first was that I wanted full control to optimize the algorithms as much as possible for the PIC18F architecture and my specific application. The second reason was that I wanted this project to be usable by other people as a basis for PIC or DMX projects.

Using assembly requires no licensed

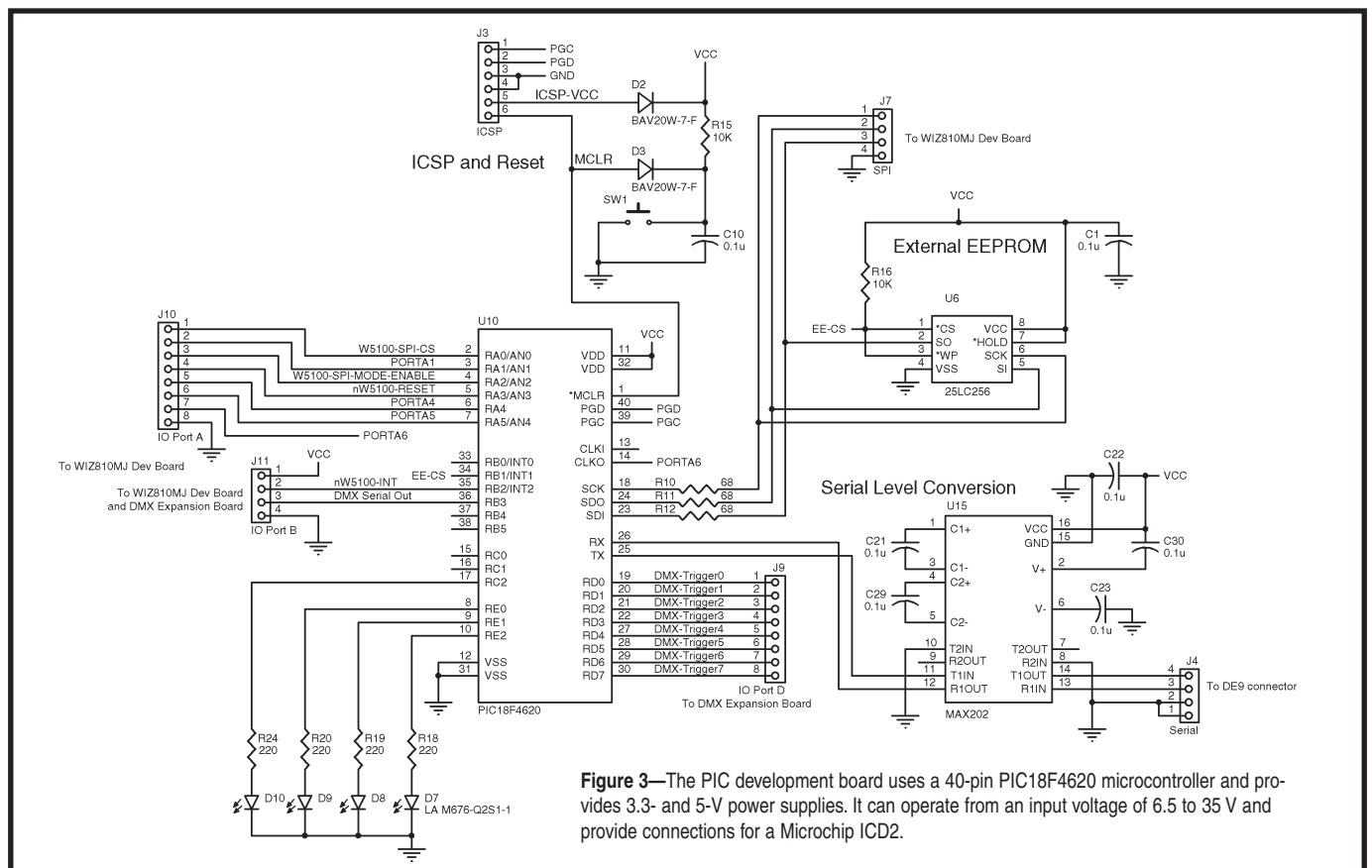


Figure 3—The PIC development board uses a 40-pin PIC18F4620 microcontroller and provides 3.3- and 5-V power supplies. It can operate from an input voltage of 6.5 to 35 V and provide connections for a Microchip ICD2.

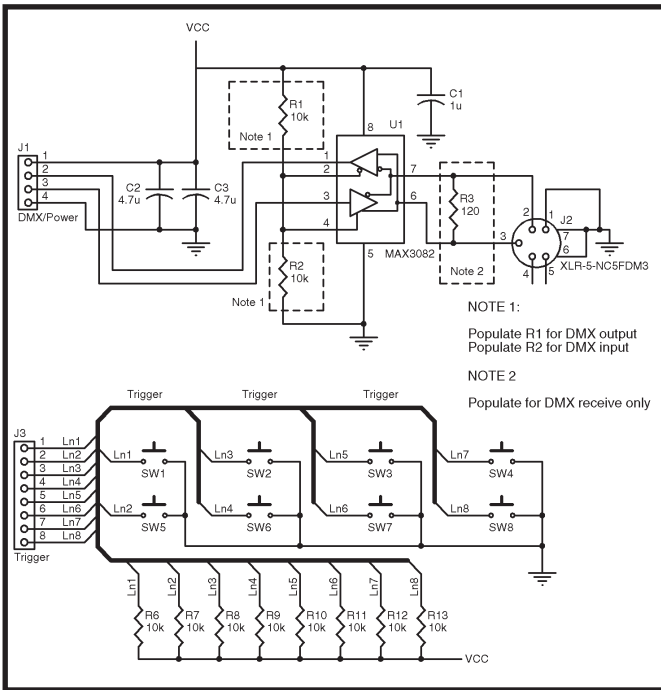


Figure 4—The DMX expansion board contains a standard DMX connector, a CMOS-to-RS-485 converter IC, and eight push button switches to debug the digital scene trigger functionality.

next command or refreshing the DMX output. Writing in assembly was particularly helpful here because it was straightforward to analyze how many CPU cycles were required for each command to execute

and optimize them as much as possible. The DMX output refresh was designed to be a blocking process like the command processor. This simplifies dealing with commands that change the contents of the DMX frame buffer. The DMX frame buffer is a 512-byte section of the PIC18F4620 data memory that stores the current 8-bit values for each of the 512 DMX channels. Because the DMX protocol allows for delays between each data slot, it would have been possible to write the code so the DMX refresh was not blocking. This would have improved the system's ability to deal with large amounts of incoming command data during refreshes. But it would also have required care in dealing with updating the frame buffer in the middle of a refresh.

Another reason the refresh was made to be a blocking process was that it effectively gave the highest priority to the DMX output. If one of the command interfaces was flooded with incoming data, it could have been

software compilers and facilitates a better understanding of what is going on at the hardware level. This understanding helps a lot when debugging code and also helps you write efficient code the first time. During the process of writing the firmware for the DMX portal, I developed a library of useful macros and subroutines for math, string, and utility operations. The time put into writing and debugging these code building blocks will help reduce the development time of future assembly-based projects and help bridge the readability gap between assembly and C.

Multitasking in this project is completely interrupt driven. When the device first powers up, all hardware peripherals and software variables are initialized before interrupts are enabled. The RS-232 and Ethernet interfaces have dedicated interrupts that call a process to collect the received data into a circular buffer within the PIC data memory. Once a complete command is detected inside the circular buffer, a blocking command processor routine is called that validates the command and executes the appropriate action. Because this command processor routine is blocking, it is important that the commands are executed efficiently so they do not interfere with receiving the

software compilers and facilitates a better understanding of what is going on at the hardware level. This understanding helps a lot when debugging code and also helps you write efficient code the first time.

Hardware timer interrupts are used to trigger the execution of the DMX refresh routines and provide timing

KEIL™
An ARM® Company

Software Development Tools

The Leader in Microcontroller Development Solutions

C/C++ Development Kit including best-in-class compilers, genuine Keil µVision®, and royalty-free RTX RTOS.

ULINK² Adapter for target debugging and Flash programming.

ARM
www.keil.com/arm

Cx51
www.keil.com/c51

C166
www.keil.com/c166

Out-of-the box support for more than 1,400 Microcontroller devices.

Keil RTOS and Middleware components are specifically optimized for embedded systems and include TCP/IP, Flash File system, USB and CAN support.

Call 1-800-348-8051 for a free demo CD.

www.keil.com

possible to starve the DMX refresh process of execution time. If this happened, and the DMX process was not blocking, the system might not have met the minimum DMX refresh rate and some lights could have gone into Auto Shut-down mode. Different applications are likely to have differing requirements for priority given to DMX output and command processing, but you can modify the code to suit your needs.

WORK WITH THE W5100

The W5100 was a good fit for this project because implementing the software required for a simple

TCP/IP stack with assembly would have been a time-consuming task. The W5100 handled all of the details and required me to write only a few sub-routines to read from the device and calculate offsets into the chip's buffer memory to find my data. Some of these calculations required 16-bit math, which is not natively supported by the PIC18F family. Luckily, I had already written a library of basic 16-bit math functions for use with the command processor.

The firmware stores all of the configuration parameters—such as the IP address, gateway, subnet mask, and MAC address—in the internal EEPROM so they can be used to configure the W5100 during power-up. These parameters can be modified via the ASCII command protocol and will be automatically saved to the EEPROM each time they are changed.

COMMAND INTERFACES

The DMX Portal provides Ethernet and RS-232 interfaces so it can be connected to a variety of systems. The Ethernet interface is useful if the system into which you want to integrate DMX functionality has Ethernet connectivity, or if it will be located a long distance away from the DMX portal. The RS-232 interface is useful for connecting to embedded systems



Photo 2—Using LabVIEW, it is easy to create utilities that communicate over standard PC I/O ports and have professional-looking graphical user interfaces. This utility allows the DMX Portal to be accessed through the RS-232 interface using the binary protocol.

because they frequently have serial output capabilities. Plus, the RS-232 interface offers a reliable, low-cost connectivity option with little software overhead.

Each of the command interfaces supports two different command protocols. The first protocol is ASCII text-based and provides a large set of easily readable commands to control every aspect of the DMX portal. This protocol is most useful if you want to control the DMX Portal through a standard terminal program. Because the commands are text-based, more processing overhead is required to transmit and process commands using this protocol.

The second protocol is a compact binary code based on a protocol developed by the open-source USB DMX project. (Refer to the Resources section of this article.) This protocol reduces overhead to a minimum and has all of the commands required to control the DMX output. It is useful if you have configured the non-DMX parameters of the device using the ASCII protocol and want to make changes to the DMX output as quickly as possible. A command can be sent that will switch between the two protocols while the device is running to allow flexibility between available commands and communication efficiency. Another

advantage is that the USB DMX protocol allows the DMX Portal to be controlled by software that supports the USB DMX protocol. I'll cover one such application, named FreeStyler, in the next section.

SOFTWARE INTERFACES

Because the DMX Portal has many combinations of communication interfaces and protocols, there is no ready-made application that can easily interface with it in all of the possible modes. I used the LabVIEW graphical programming language to build custom interfaces that could control the DMX Portal via the RS-232 or

Ethernet connections using the ASCII or fast binary protocols (see Photos 2 and 3).

Programming with LabVIEW was helpful because it has libraries for communicating via a serial port or TCP/IP. It also has many examples that can be easily modified. The other benefit is that LabVIEW enables you to build a functional GUI with little work. The code for the utility interfaces I wrote could easily be enhanced to behave like the final user interface of a project.

To demonstrate how to make a final user application, I wrote a virtual lighting board application based on the same code I used for my utility interfaces (see Photo 4). The virtual interface mimics the user interface provided by lighting boards by providing sliders that can be attached to sets of DMX channels. Buttons for other common features—such as temporarily setting all lights to off, also known as a blackout, or disabling the DMX output to enable the lights to go into Power Down mode—are also provided.

Earlier, I mentioned that the fast binary protocol was based on the protocol of the USB DMX project. A major reason for this decision was that there is a free lighting control program available called FreeStyler that provides excellent

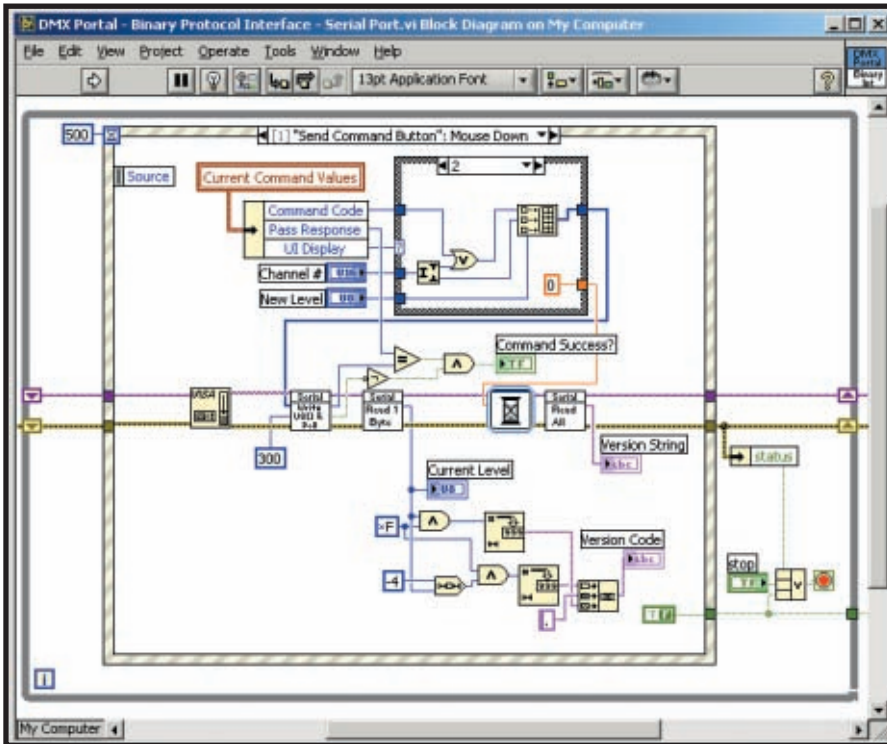


Photo 3—The code in LabVIEW is written graphically using a programming principle known as data flow. Each block in the diagram represents a function that will execute once data has arrived on each of its inputs. Outputs of a block are connected via wires to the inputs of other blocks or they are connected to control and indicator symbols, which correspond to graphical items on the GUI called the front panel.

support for many commercial programmable lighting fixtures and makes it easy to generate complex lighting effects. (Refer to the Sources section.) FreeStyler already supports the USB DMX binary protocol, so by

allowing the DMX portal to support the same commands, I effectively received support for this application with no extra work. The USB DMX project uses an FTDI chip that appears as a virtual COM port in Windows.

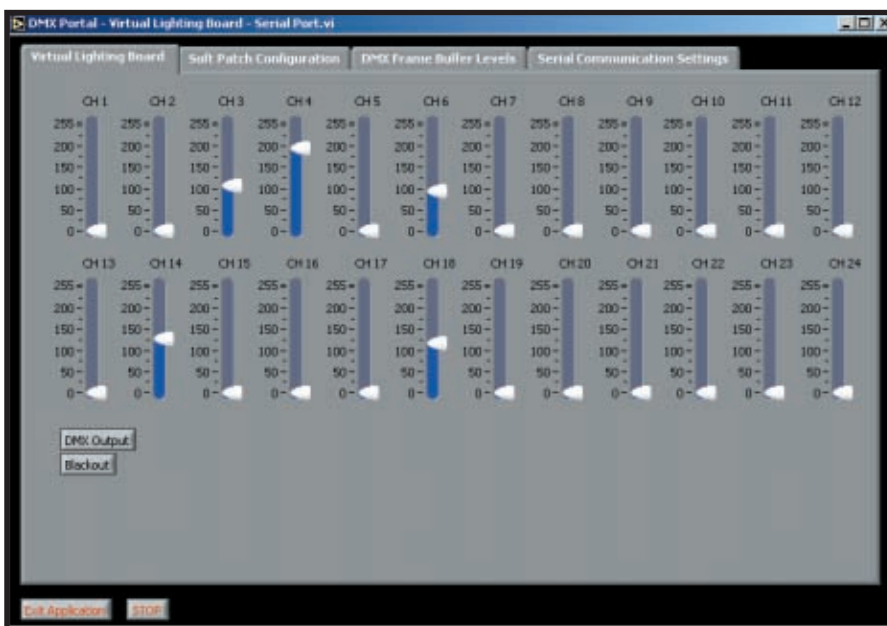


Photo 4—This application was written to demonstrate how a final user interface for the DMX Portal could be written using LabVIEW. It provides features commonly found on a standard lighting control board, such as level sliders and blackout controls.

USB Oscilloscope for \$169.50
 Logic and Spectrum Analyzers, Generator.
www.HobbyLab.us

imagecraft
C Compiler for Parallax Propeller

Eight processors, speed up to 80Mhz, Parallax's Propeller sets a new standard in innovative design.

- Large Memory Model allows programs much larger than COG RAM
- Native multiprocessing support in C
- Built-in "Propellent" program downloader

starts at **\$99!**

www.imagecraft.com
info@imagecraft.com
 (650) 493-9326 • FAX (650) 493-9329

FREE Monitoring Software

Applications
 HVAC Systems
 Data Centers
 Laboratories
 and more...

eSensors
www.eSensors.com (716) 837-8719

Environmental Monitoring Solutions
 Temperature + Humidity + Illumination
WebSensor EM01B

Get \$10 off with "cellar10" coupon code

All you need to do is put the DMX Portal into Binary Protocol mode and then point FreeStyler to the correct COM port.

FUTURE DEVELOPMENTS

Like any project, there is always room for improvement. I would like to improve the DMX Portal's RS-232 hardware buffer. The PIC18F4620 provides a 2-byte hardware buffer to help give the processor time to process incoming data if it can't get to it

immediately. As I discussed in the Firmware section, I chose to make some processes block the execution of other interrupt-driven processes to deal with time-sensitive requirements or race conditions involving modifying memory in the middle of these tasks.

There is a downside to this approach. When the DMX output is enabled, the processor is fully occupied for 22.668 ms while a frame is sent. Because the DMX output refresh

consists mostly of wait commands to generate the proper data rate timing, speeding up the processor would not reduce the amount of dead time where the processor could not handle incoming data. Even at the slow data rate of 9,600 bps, a single byte takes only about 1 ms to transmit. So, it is possible to send more than 2 bytes of data over the RS-232 link before one DMX frame refresh is completed. This can result in data being lost and commands that were sent not being processed.

The best way to maintain a constant DMX output refresh rate while still being able to tolerate large momentary bursts of command data is to implement the serial interface with a second small low-cost microcontroller. This would enable the data memory of the second microcontroller to act as a large command buffer. It would also make the serial interface as robust as the W5100 Ethernet interface, which already contains a large memory buffer.

Another approach would be to use an external hardware UART for the DMX output. This would eliminate the need for the processor to sit in wait loops to generate the correct data rate. It would also enable the processor to handle incoming data in the gaps between bytes. The downside to this approach would be that the DMX output refresh rate would become dependent on the amount of traffic received on the communication ports. This may not be a problem, but it would require a bit more care in implementation because the DMX standard requires a minimum refresh rate. If too many commands are sent during a frame refresh, this rate may not be met.

For the prototype, I used a relatively slow 8-MHz system clock because it could be generated from the PIC18F4620's internal RC oscillator. There is no reason why I can't run the PIC18F4620 faster with an external oscillator. Doing so would help improve the performance of some tasks such as command processing. Having a faster clock to reduce the processing time would help make the system run much smoother without

\$51 For 3 PCBs
FREE Layout Software!
FREE Schematic Software!

- 01 DOWNLOAD our free CAD software
- 02 DESIGN your two or four layer PC board
- 03 SEND us your design with just a click
- 04 RECEIVE top quality boards in just days

expresspcb.com

costing too much.

One improvement that could be made to the Ethernet interface would be to change from the UDP protocol to the TCP protocol. UDP has no mechanisms to guarantee packets are not lost. It also has a maximum payload size that can be a problem with some of my commands that generate large amounts of response data. A TCP-based link would automatically try to resend dropped packets, reorder packets that are received out of sequence, and detect a communications failure. TCP also behaves as a constant datastream, so it would not be subject to a maximum payload length like UDP. The W5100 supports TCP. Thus, it would not be difficult to make this transition. I think it would greatly improve the quality of the communication interface on the DMX Portal.

I also want to improve the data storage space for user-defined scenes. For simplicity, I chose to use a serial EEPROM for my storage, but it offers only limited space. An SD card would have been a much better choice because it

offers a low-cost storage solution that easily holds an entire show's worth of scene data. Because SD cards can be accessed through a SPI just like the EEPROM, supporting this feature would require only adding code to handle the FAT file system. An additional benefit to the SD card approach would be that scene data could easily be written or backed up to a PC without actually being connected to the DMX Portal. It is possible to write a software interface that performs these functions using the available communication interfaces, but it is not as simple as just reading a file from an SD card. ☒

Matt Ernst (mbernst@gmail.com) is a graduate of the University of Wisconsin-Madison School of Engineering. He has a strong interest in the automation and control systems used in the entertainment industries. Matt is a staff analog hardware engineer at National Instruments. He designs high-speed test and measurement hardware.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2008/217.

RESOURCES

FreeStyler, <http://users.pandora.be/freestylerdmx/>.

B. Suffolk, "USB DMX Project," www.usbdmx.com/protocol.html.

United States Institute for Theatre Technology, Inc., "DMX512-A Specification," www.usitt.org.

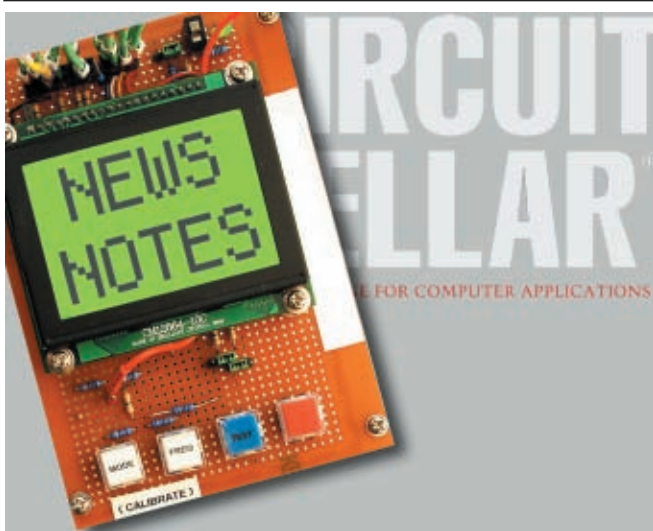
SOURCES

MPLAB IDE and PIC18F4620 Microcontroller

Microchip Technology, Inc.
www.microchip.com

W5100 Ethernet controller and WIZ810MJ evaluation board

WIZnet, Inc.
www.wiznet.co.kr/en



Make sure you're signed up to receive Circuit Cellar's monthly electronic newsletter. *News Notes* will keep you up to date on Circuit Cellar happenings. Stay in the loop!

Register now. It's fast. It's free.

www.circuitcellar.com/newsletter/

Expand Your I/O with Rabbit® RIO

Versatile Device with I/O Options

- Add functionality without costly processor changes
- Multiple Processor Interfaces
- Add 38 I/O
- Configure I/O for PWM, TRIAC, input capture, or decoder



Rabbit RIO™ Programmable I/O Application Kit for \$299

RABBIT 

Order Online At rsappkits.com



Content Collection and Display Build an Internet-Connected News Ticker

James no longer has to turn on a TV or computer to get news updates. His innovative design retrieves news headlines from RSS feeds and constantly scrolls them across a dot-matrix LED display. The system checks for updates every 15 minutes.

When I first heard about WIZnet's iEthernet 2007 design contest, I was really excited about all of the possibilities an integrated Ethernet chip would open up to smaller microcontroller-based systems. I knew there were tons of networking applications that I could create (e.g., web servers, remote sensor networks, and home automation/control systems). But many of the designs that first came to mind had already been done to death, and when it came down to finding a new project, I struggled a bit. Having missed the last few *Circuit Cellar* contests, I knew I had to get an entry done for this one.

At the time, I was messing around with interfacing dot-matrix LED displays to microcontrollers. I showed a friend how to make customized messages scroll across a display by programming a computer a certain way. (He wasn't exactly a *Circuit Cellar* reader.) He was really interested in the stock market, so he thought it would be nice to have a scrolling stock ticker similar to those shown on some financial television programs. I thought it was a really good idea, but rather than scrolling stock information, I decided to scroll up-to-date news headlines.

Luckily, there are many news organizations that provide free news updates via the Internet. Nearly all of the major ones provide really simple syndication (RSS) feeds for news headlines as well. RSS gives content

providers a way to distribute periodically updated material. Conveniently, RSS uses XML, so the relevant data is well-structured for processing by clients.

To retrieve the news headlines, I wrote a cut-down HTTP client and XML parser. I wanted the device to operate without any action by the user, so I implemented a domain name service (DNS) and dynamic host configuration protocol (DHCP) client in order to use dynamic IP addressing. Every 15 min., the device renews its

client IP, requests the IP of the BBC News RSS server, parses the XML for news headlines, and scrolls them across the display (see Photo 1).

The system features a Microchip Technology PIC18F2525, PIC18F2221, and a WIZnet WIZ810MJ Ethernet module. In this article, I'll describe how I used these building blocks to design the news ticker.

SOME CHALLENGES

I wanted the design to be as simple and inexpensive as possible. The WIZnet

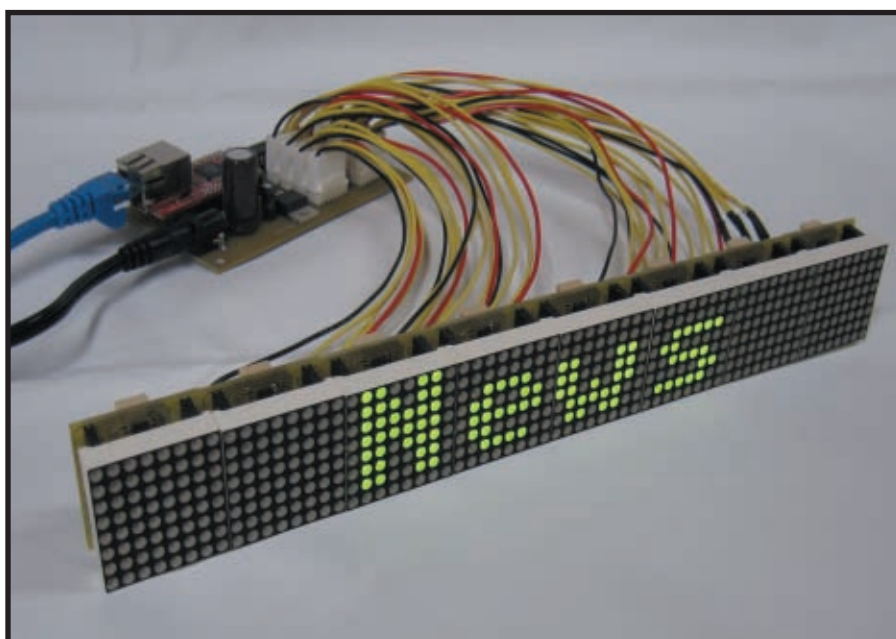


Photo 1—Check out the LED news ticker in action. Actually, you can't really see the full extent of the action. Still pictures don't show scrolling very well. Use your imagination!

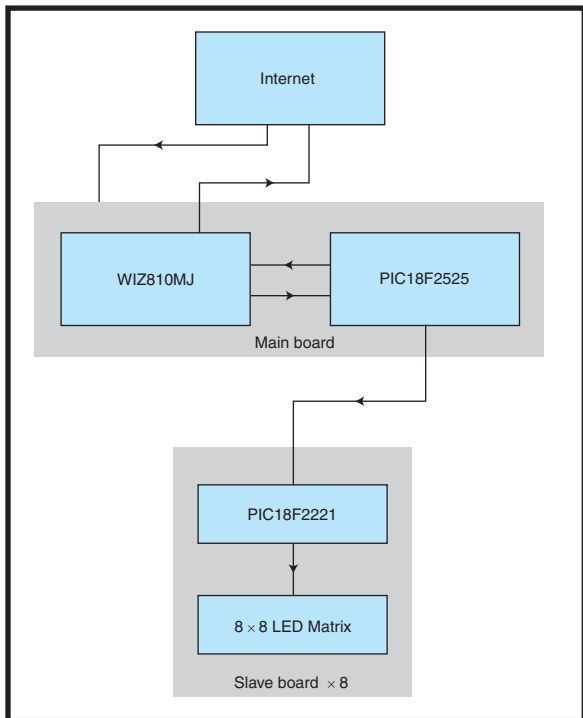


Figure 1—This is a block diagram of the system. On the main board, a Microchip Technology PIC18F2525 communicates with a WIZnet WIZ810MJ to access the Internet. The slave boards receive data from the main board while updating their LED display.

chip handles most of the network processing, so a cheaper, less complex microcontroller could be used for controlling the system. Knowing that these products have a limited amount of internal RAM, I considered adding some external storage for

column. Each row is then sequentially enabled while manipulating the columns of the matrix in order to illuminate the specified LEDs in that row. This process is called “scanning.” If the matrix is scanned at a high enough frame rate, persistence

buffering the incoming XML data; but ultimately, I decided against that. Without a buffer large enough to store the entire web page, I was limited to using only the integrated receive buffers of the WIZnet chip. Although the software would be more complicated, the need for an additional external component was eliminated.

Controlling all of the LEDs presented another challenge. Nearly all dot-matrix LED displays are common-anode or common-cathode. Each row of the matrix is connected to the anode or cathode of every LED in that row. The columns of the matrix are connected to the opposite end of every LED in that

of vision will be maintained. The purpose of scanning is to reduce the number of control pins required, along with the amount of power needed to illuminate the display.

Continuously scanning the display takes a lot of processing time, so a master/slave arrangement was used. In this setup, the master tells the slaves what image to draw while the slaves draw the image. This allows a large amount of I/O manipulation to be offloaded to the slaves, leaving the master free to do more important things.

A side effect of matrix scanning is a reduction in display brightness. Because each row of the display is on only once per frame refresh, and you have eight rows in the frame, the average current through an LED will be 12.5% of its instantaneous current. If the LEDs and their current-limiting resistors are specified for 20 mA (a pretty standard LED current), the average current through them will be only 2.5 mA due to the low duty cycle of matrix scanning.

Fortunately, the LEDs are still bright enough with the reduced current. I actually halved the recommended LED current from 20 to 10 mA. With the average current through each LED equal to 1.25 mA, the display was still easy to read in daylight.

A nice feature of LEDs is that they can withstand pulses of current much higher than their recommended average current. If the display is not bright enough using the 20-mA recommended average current, it can be raised according to the specification of the LEDs. For the LEDs in my displays, the maximum pulse current is 100 mA. With a 12.5% duty cycle, this gives an average current of 12.5 mA through each LED—still well within the recommended average range.

Make sure to select an appropriate current for your LEDs. There are 128 LEDs in each display, and I used eight displays in this system for a total of 1,024 LEDs. If 12.5 mA was selected as the average current through each LED, the maximum average current draw for the entire display would be 12.8 A! Using an average LED current of 1.25 mA

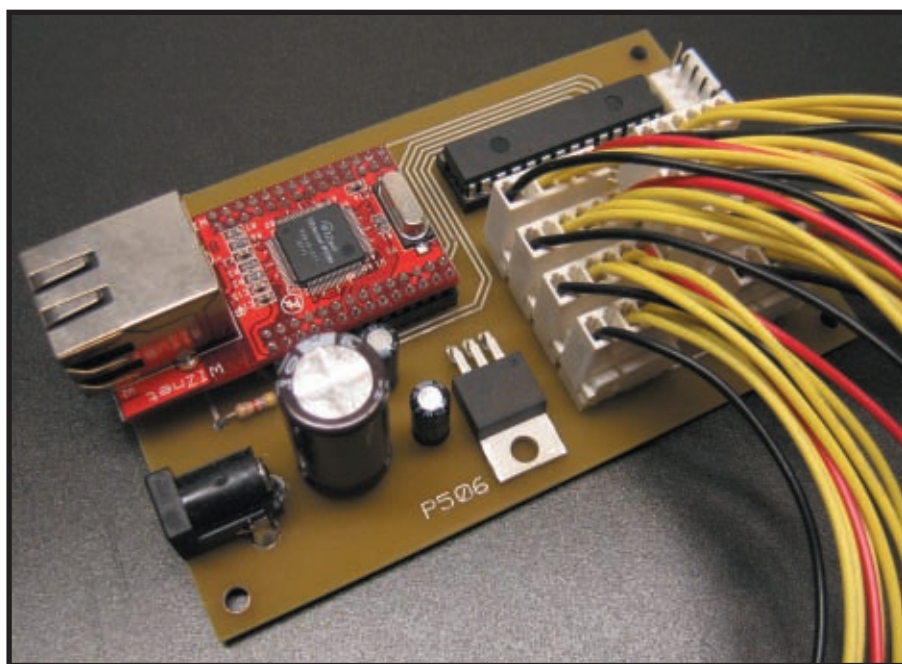


Photo 2—This is the main board. Can a PCB get any simpler?

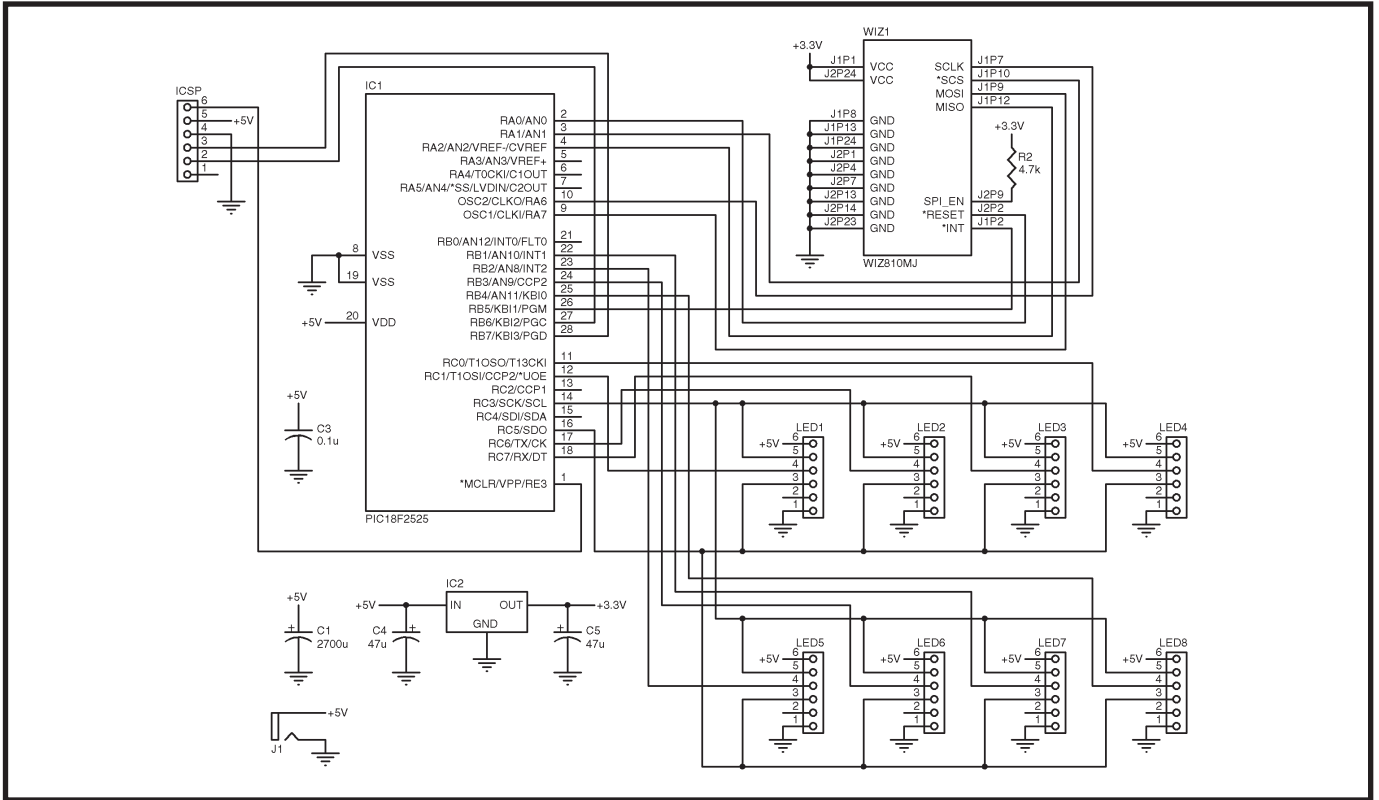


Figure 2—There's not much to the main board. The Microchip PIC18F2525 and WIZnet WIZ810MJ are the only major components.

reduces the total current required by a factor of 10.

You're now familiar with many of the gotchas I encountered. Let's move on to the hardware behind the project.

THE HARDWARE

As you can see in Figure 1, the design consists of a main board and eight slave boards that piggy-back to bicolor dot-matrix LED displays. The main board is responsible for handling networking-related tasks, updating the frame buffer for the display, and sending each slave the current frame it should be drawing. The slave boards continually paint their frame buffers to their respective LED matrix while waiting for the main board to send them updated display data.

Each LED matrix contains 64 pixels, organized as an 8 × 8 square. Each pixel is represented by a red and green LED, for a total of 128 LEDs per matrix. A maximum of eight characters can be fully displayed at one time.

The main board is fairly simple (see Photo 2 and Figure 2). It features a PIC18F2525 at 32 MHz, a

We Listen. Think. And Create.

Distributed I/O

Digital I/O

Serial I/O

Industrial Computing

HMI

SeaLINK USB serial adapters are the fastest, most reliable way to connect peripherals to any USB-equipped computer.

SeaLINK USB Serial Adapters Provide:

- 1, 2, 4, 8, and 16-Port Models
- RS-232, RS-422, and RS-485 Serial Interfaces
- Data Rates to 921.6K bps
- State Machine Architecture to Reduce Host Processor Overhead
- Operation as Standard COM Ports to the Host Computer
- Lifetime Warranty

FOCUS
On Success
Call Today!

SEALEVEL

sealevel.com > sales@sealevel.com > 864.843.8067

WIZ810MJ Ethernet module, power supply circuitry, and headers for connecting the slave boards. The WIZnet module houses a WIZnet W5100 integrated Ethernet controller and an RJ-45 jack with integrated magnetics.

The eight slave boards are also simple (see Photo 3 and Figure 3). Each one uses a PIC18F2221 at 32 MHz, a 1.32" 8 × 8 bicolor, common-anode dot-matrix LED display, a pair of Texas Instruments TPIC6C596 power shift registers, and four Fairchild Semiconductor FDC6312 dual-channel P-FETs. Shift registers are used to further reduce the number of I/O pins required to interface to the matrices. Each shift register is 8 bits wide, so two are used to control the 16 LEDs that make up the columns of each matrix (one green, one red, eight of each). They also provide a high-power current sink for the LED cathodes. The P-FETs act as high-current,

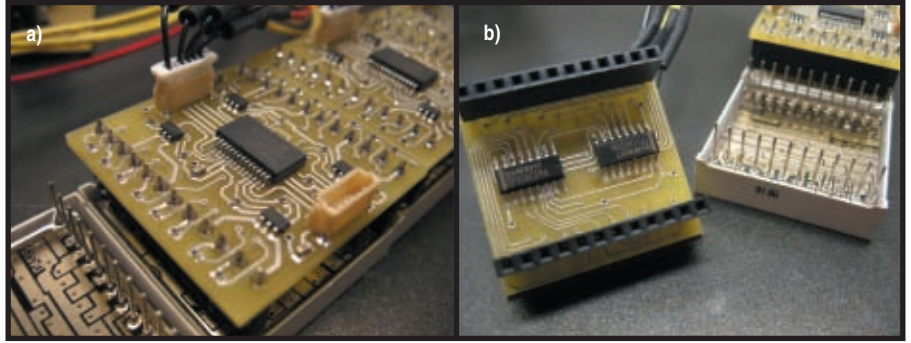


Photo 3a—Take a look at the slave boards. Soldering eight of these by hand took forever. **b**—This is a close-up of how an LED matrix and a slave board snap together.

high-side drivers for the common-anode rows.

A 5-V, 2.5-A, regulated switching power supply provides power to the system. Even with the current reduction from matrix scanning, a fairly large amount of current is required—roughly 1.3 A with the display fully illuminated. In reality, the device draws only around half that because I

don't use the bicolor capability. There are a lot of unlit pixels as well due to the shapes of each letter and spaces between words. Still, I have some wiggle room in case I ever decide to increase the display brightness.

The microcontrollers and LED circuitry are powered by the 5-V rail provided by the regulated switching power supply. A linear regulator on

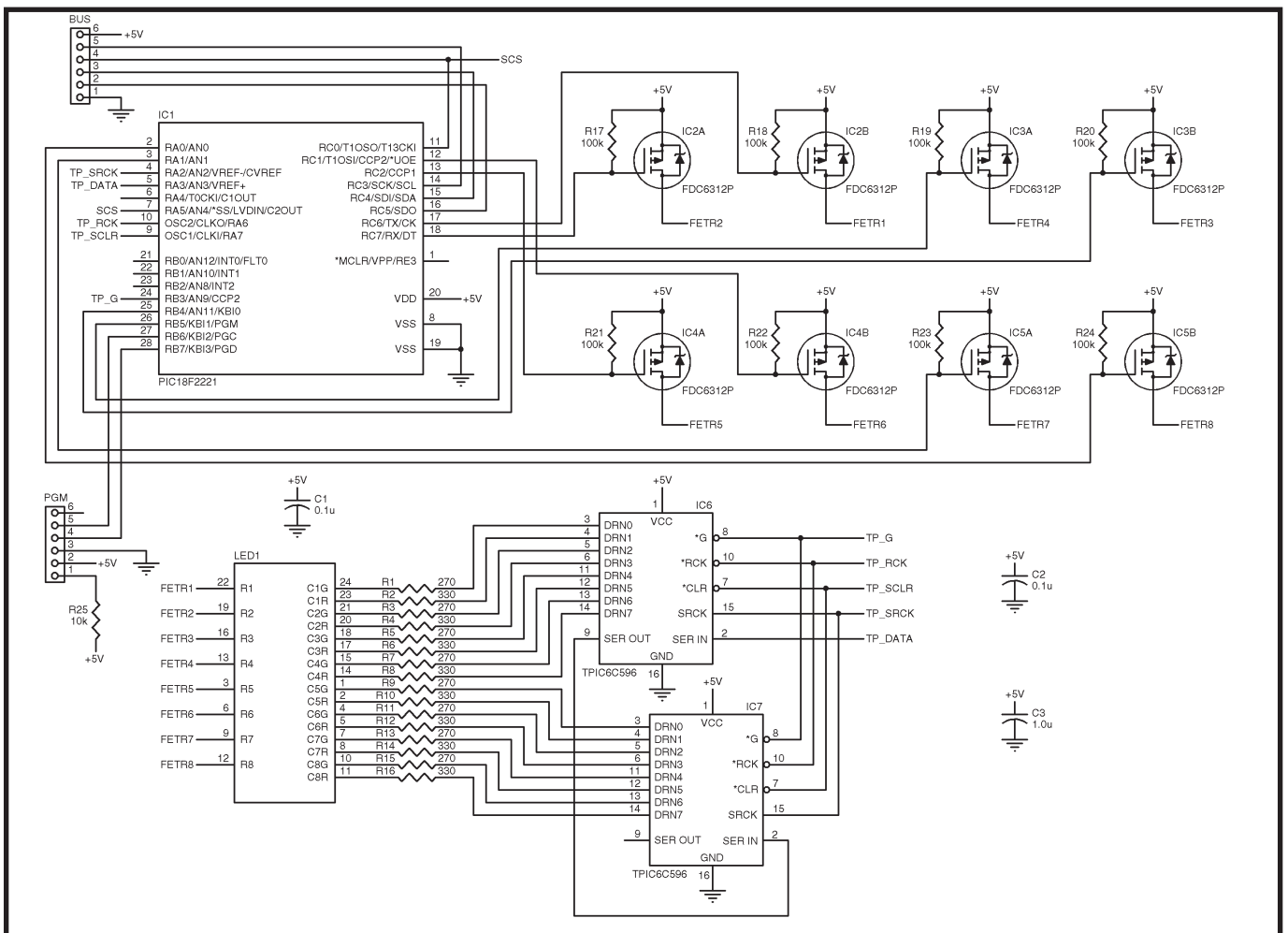


Figure 3—Here are the slave boards in all of their glory. Most of the components are for interfacing to the LED matrix.

the main board provides the 3.3 V required by the WIZnet module. The module and the PIC18F2525 are 3.3-/5-V-tolerant, so level-shifting circuitry isn't required.

The master communicates with the slaves via a SPI bus. Each slave has its own chip-select line. The bus's data rate is kept at a minimum due to the distance between the main board and the slaves. A SPI bus is also used to communicate with the WIZnet module.

WIZnet MAKES IT EASY

Having to deal with implementing a software TCP/IP stack with fairly limited resources is a pain, so a fully featured drop-in solution was attractive. There was also the added bonus of offloading a nontrivial amount of processing to an external chip, leaving the master to skip a lot of the grunt work. The WIZnet chip greatly reduced the complexity of the main board's firmware.

The WIZnet chip supports many protocols, including the well-known TCP and UDP among others. These two protocols are used for all of the networking services that the main board implements (DHCP, DNS, and HTTP). The W5100's socket interface enables connection to other nodes on the network.

You probably read Fred Eady's article, "iEthernet Bootcamp: Get Started with the W5100" (*Circuit Cellar* 208, 2007). Fred explained the socket interfaces better than I can, so refer to his article if you want to delve into the details of setting up the W5100 registers correctly. Suffice it to say, depending on the protocol used, all that's needed for a socket connection are the IP addresses of the client and host and the source and destination port. Once these are established, you can begin sending

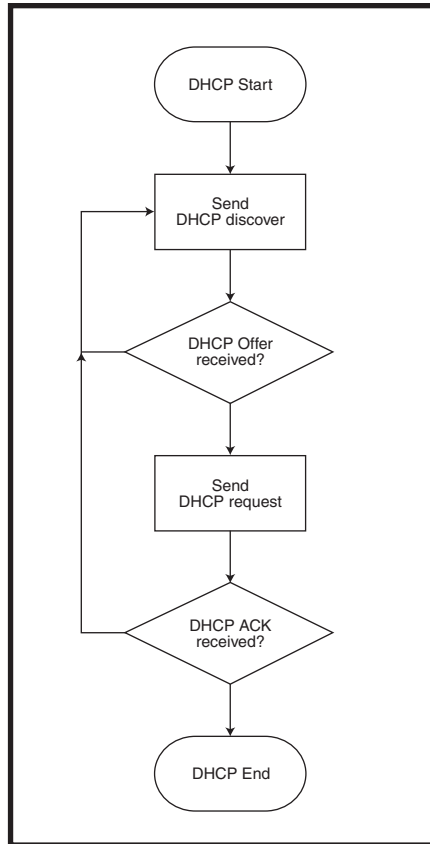


Figure 4—Packets are exchanged by the DHCP client and server to obtain local network information. The DHCP client will repeatedly request an IP until one is leased by the server.

and receiving data.

MAIN BOARD FIRMWARE

The firmware for the main board was written in C using Microchip's C18 compiler and MPLAB IDE. Nearly all 4 KB of RAM provided by the PIC18F2525 is used. Less than half of the 48 KB of program memory was used, so some functionality can be added.

The program itself is separated into several different tasks and is entirely interrupt driven. Short tasks are executed in the interrupt, while long tasks poll status flags and execute in the main loop. The main board's primary

tasks are the news update task, the display task, and the "updating message" task.

NEWS UPDATE TASK

The news update task is responsible for DHCP leasing, DNS querying, downloading news updates via HTTP, and formatting them for display. Three out of the four sockets are used on the WIZnet module (one for each of the previous services). The task is called approximately every 15 min. While it is running, it calls the "updating message" task to display a message while the headlines are downloaded from BBC News.

Every time the task is run, a DHCP lease is requested and a DNS query is made to resolve the IP address of newsrss.bbc.co.uk. Although caching and lease times are provided from the queries, these are ignored for the sake of simplicity.

DHCP

DHCP is a convenient feature to have on a device like this. It enables the automatic retrieval of local network information, including a free client IP address, the subnet mask, the gateway IP address, and the DNS server IP address. All of these are required to establish a connection on the Internet. Without DHCP, this information would have to be manually entered periodically because the addresses are dynamic. I wanted the device to operate without user interaction, so implementing a DHCP client was required.

DHCP uses UDP to facilitate client and server communication and is implemented on its own socket (socket #1). Figure 4 shows the process of obtaining local network information using DHCP.

The client first broadcasts a DHCP Discover packet, which alerts DHCP

Listing 1—The description tag stores the news headlines you want to extract. Everything else can be ignored.

```

height="49"
url="http://newsimg.bbc.co.uk/media/images/44531000/jpg/_44531986_usflag66i_ap.jpg"
"/></item> <item><title>Amnesty pressures IOC over
China</title><description>Amnesty International calls for condemnation of China's
handling of protests in Tibet.</description><link>http://news.bbc.co.uk/go/rss/-
/2/hi/uk_news/7325754.stm</link><guid

```

servers on the network. Multiple DHCP servers may respond with a DHCP Offer packet. This packet notifies the client of an IP if one is available for lease. The client responds to the first server that responds. The client then sends a DHCP Request packet to the first DHCP server found, requesting the IP that was previously received in the DHCP Offer. If the server accepts the request, a DHCP ACK packet is sent. This confirms with the client that the IP is now assigned. If an IP is not served, the process will repeat until a client IP is obtained.

DNS

DNS is another useful feature to have. Instead of having to remember the IP address of every web site you visit, DNS allows familiar hostnames (e.g., www.circuitcellar.com) to be used to connect to remote hosts. Many host IP addresses are dynamic, so DNS is another must-have (unless you want to update the IP manually, and that's no fun).

DNS uses UDP for communication between servers and clients. It is implemented on its own socket (socket #2). The process is straightforward and has only two steps. The client sends a DNS query packet to the DNS server that was identified through DHCP. The query packet contains the requested domain name (newsrss.bbc.co.uk in this case) in the form of questions. Multiple questions, and hence domain names, can be sent, but you need to look up only one domain name in this case. Following the question section is the answer section. This area will be used by the DNS server's reply, but it is unused in queries.

The DNS server sends a DNS response after receiving a query. It is noteworthy that the format for queries and responses is identical. The DNS response contains the original question sent by the client along with an answer. Multiple answers can be sent even if only one question is asked. Some of the answers may be redirected domain names, which can safely be ignored. The IP address of the server is the only data you're looking for. The

DNS client parses the answer section until it finds the required IP address of the host.

HTTP

HTTP is used to obtain the XML data from the BBC RSS news feed. It uses TCP and is implemented on its own socket (socket #0). HTTP supports several actions that enable data transfers from the host to the client, or vice versa. In this case, the only action you're concerned with is GET. There's nothing complicated here. The GET action retrieves a web page. The only data needed for the GET action is the URL of the web page you want. Once a GET action is received by the host, it starts transmitting the requested page to the client.

The XML data must be parsed to extract the desired news headlines. Fortunately, XML provides a nicely structured set of data by enabling providers to define their own elements. Elements are basically data grouped around "<>" tags. The BBC RSS feed provides the news headlines in elements named description. The tags will appear as <description> in the XML. Listing 1 shows a sample of a description element and the news headline it contains.

The XML data is parsed while it is still in the WIZnet chip's receive buffers. The parser steps through the buffer, looking for the beginning of a description tag. Once one is found, it stores the following text until the end of the description tag is found (</description> in this case). The news headlines are stored in the PIC18F2525's internal memory and most of it is used for that purpose. About 20 headlines can be stored, each with a maximum of 145 characters. BBC headlines are generally less than 120 characters. If a headline is longer than 145 characters, additional characters are discarded.

Once all of the headlines are stored, they must be parsed again. XML uses special sequences of characters to represent quotation marks, apostrophes, and ampersands (", ', and &, respectively). The parser must find them and convert them to

the correct characters.

DISPLAY TASK

The display task updates the slaves with display data and maintains the frame buffer. It is called approximately every 16 ms.

Compared to the news update task, the display task is relatively simple. It shifts all of the data in the frame buffer left one pixel every time it is run. When the frame buffer has been shifted eight pixels, it loads the next character into the buffer. Once the last headline is displayed, the task wraps around to the first headline.

The frame buffer holds pixel data for 10 frames. The display itself shows only eight of those frames (one frame for each LED matrix). An extra frame is on both ends of the buffer and they're there to facilitate scrolling. Each frame in the buffer stores 8 bytes of information. Each byte represents a column of the LED matrix. Each bit then determines whether that pixel is on or off. Sixty-four bits are used for each frame (8 bytes × 8 bits), with each bit corresponding to one pixel on the LED matrix.

To convert news headlines to pixels, an 8 × 8 font table is used. The table maps ASCII characters to pixel data for the LED displays. Eight bytes are used for each ASCII character. Like the frame buffer, each byte represents a column and each bit represents a pixel.

SLAVE BOARD FIRMWARE

The firmware for the slave boards was written in C using Microchip's C18 compiler and MPLAB IDE. Few resources are used on the slaves because their responsibilities are limited.

Compared to the main board, the slave firmware is pretty dumb. The program sits in a loop, continually refreshing the display, while waiting for the SPI interrupt to trigger. The SPI interrupt routine simply loads the data received from the master into a frame buffer.

The infinite loop continuously refreshes the LED matrix. Only one row is illuminated at a time, and rows are illuminated sequentially. Each byte in the 64-byte frame buffer holds

the color information for a single pixel. The data corresponds to the binary representation of which LEDs should be illuminated. For each row, a 16-bit number is generated holding the bit pattern for the LED columns. For example, the binary bit pattern B"11111111 11111111" would turn on every LED. The bit pattern B"01010101 01010101" would turn on every red LED. The pattern B"00000000 00000000" would turn off all LEDs, and so on. When the entire frame has been refreshed, the process begins again.

HAVE FUN

This project was a success. It is still sitting on my workbench, so I can read news headlines when I am not near a computer.

You learn a lot by doing a project like this because there's so much functionality involved. There's also a lot of room for new features that I didn't have time to implement. One that I will eventually integrate is the ability to upload a customized message through a web page that the main board will host. I may even add the ability to scroll stock information in the future. ☺

James Blackwell (j.blackwell@azosoft.com) graduated from Cleveland State University in 2007 with a Bachelor's degree in Electrical Engineering. He currently works as a design engineer for Delta Systems in Streetsboro, Ohio. When James isn't perfecting his perpetual motion machine, he enjoys playing bass guitar and drums.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2008/218.

RESOURCES

F. Eady, "iEthernet Bootcamp: Get Started with the W5100," *Circuit Cellar* 208, 2007.

WIZnet, Inc., "Application Note – DHCP," 2008, www.wiznet.co.kr/rg4_board/view.php?&bbs_code=en_pds&ss%5bst%5d=1&ss%5bsc%5d=1&kw=dns&bd_num=15833.

www.fairchildsemi.com

———, "Application Note – DNS," 2008, www.wiznet.co.kr/rg4_board/view.php?&bbs_code=en_pds&ss%5bst%5d=1&ss%5bsc%5d=1&kw=dns&bd_num=15833.

———, "W5100 Datasheet," 2008.

www.fairchildsemi.com

MPLAB C18 Compiler, PIC18F2221 microcontroller, and PIC18F2525 microcontroller

Microchip Technology, Inc.
www.microchip.com

TPIC6C596 Shift register
Texas Instruments, Inc.
www.ti.com

WIZ810MJ Ethernet module
WIZnet, Inc.
www.wiznet.co.kr/en

SOURCES

FDC6312 MOSFET
Fairchild Semiconductor Corp.

No other PCB-design tool gives you more value per dollar

Boards designed under EAGLE are developed in one-man businesses or in large industrial companies. Most of the top companies are our customers. The crucial reason for selecting EAGLE is not usually the low price, but rather the high-end functionality along with the ease of use. And EAGLE users appreciate the outstanding level of support, which at CadSoft is always free of charge, and is available without restriction to every customer.

These are the real cost killers!

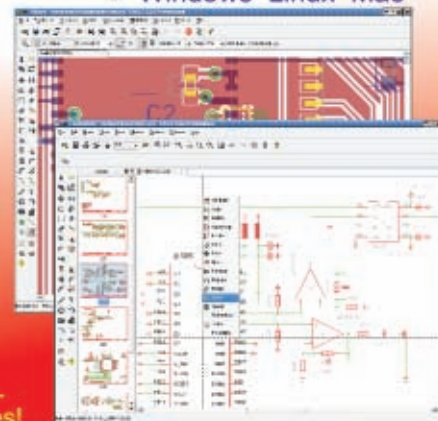
Version 5 is even easier to use, especially for beginners, due to an enhanced user interface.



EAGLE
Version 5

Schematic Capture • Board Layout
Autorouter

for Windows® Linux® Mac®



Version 5 Highlights

- ▶ Stand-alone schematic editor available.
- ▶ Automatic signal/contact cross references using frame coordinates.
- ▶ Right-mouse click for more consistent Windows UI.
- ▶ User-definable attributes for parts.
- ▶ Schematic sheet management.
- ▶ Hiding approved DRC and ERC errors.
- ▶ PRINT preview and text-searchable PDF output.
- ▶ Improved search engine for help.
- ▶ And much, much more.

Pick the level that is right for you — pay only the difference for upgrades!

	Light	Standard	Professional
Max. number of schematic sheets	1	99	999
Max. board size	4x3.2 inch	6.4x4 inch	64x64 inch
Max. # of signal layers	2	4	16
Layout or Schematic Editor		\$249	\$498
Layout and Schematic Editor		\$498	\$996
Layout Editor and Autorouter		\$498	\$996
Layout Editor and Schematic Editor and Autorouter	\$49	\$747	\$1494

Standard and Light Editions have full functionality except for the limitations mentioned in the table.

You can use EAGLE Light for evaluation and non-commercial applications without charge. Download it from our web site.

www.cadsoftusa.com
800-858-8355

CadSoft Computer, Inc., 19620 Pines Blvd., Suite 217, Pembroke Pines, FL 33029
Hotline (954) 237 0932, Fax (954) 237 0968, E-Mail: info@cadsoftusa.com

Windows® Linux® Mac® are registered trademarks of Microsoft Corp. Linux, Solaris® & Apple Computer, Inc.



Automated Data Mining

Build an Embedded Server Application

This well-designed embedded server application enables you to find airfare deals on the 'Net. The WIZnet W5100-based system uses Kayak, an online travel search engine, to search for flights. The data is then graphed to show you the best time of year to travel.

The bone-chilling wind and weekly ice storms that come with winter in central Illinois often leave us wondering where we can go to get away from the cold. Far away. Destinations on our short list this year include Jeju (South Korea), Buenos Aires, Acapulco, and Honolulu. As engineers, we didn't want to make a random decision with no logic behind it. We wanted to have a process. We wanted to travel to one of these far-off destinations, but how would we decide where to go and when?

As cost-conscious engineers, we wanted to locate the best airfare prices and travel dates. A few searches on one of the many available online travel search engines revealed that airfare can fluctuate significantly depending on the time of year. Airfare even fluctuates with the days of the week. We knew the only way for us to make a proper decision would be to collect about a year's worth of airfare data for our desired destination and then inspect the data for the cheapest time of year to travel. But manually executing a year's worth of airfare searches to a desired destination would have taken a long time. In fact, we would have spent the entire winter sitting in front of our computers, sipping hot chocolate, and running searches. But we had basketball to play and beaches to lounge on. Thus, we did what any engineers repulsed by the thought of endless repetitive work would do: we

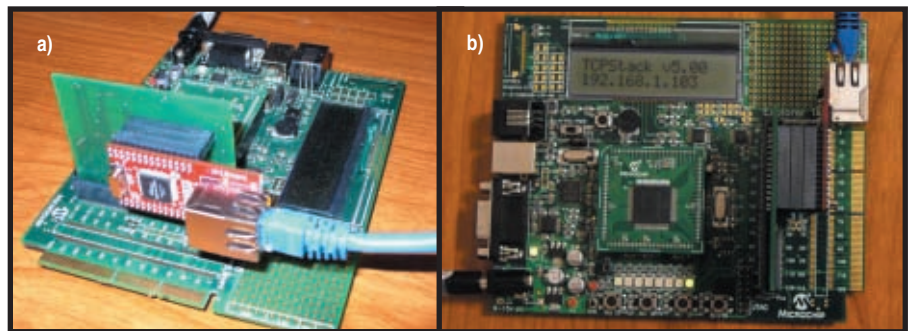


Photo 1a—The WIZnet WIZ810MJ network module plugs into a pair of female pin headers soldered to a 120-pin card edge connector board. The connector board plugs into the Explorer 16 development board. b—This is the top of the module.

automated the entire process with a novel design. We call our design the Travel WIZard. It is an embedded server

application featuring a WIZnet W5100 hard-wired TCP/IP Ethernet controller and a Microchip Technology Explorer 16

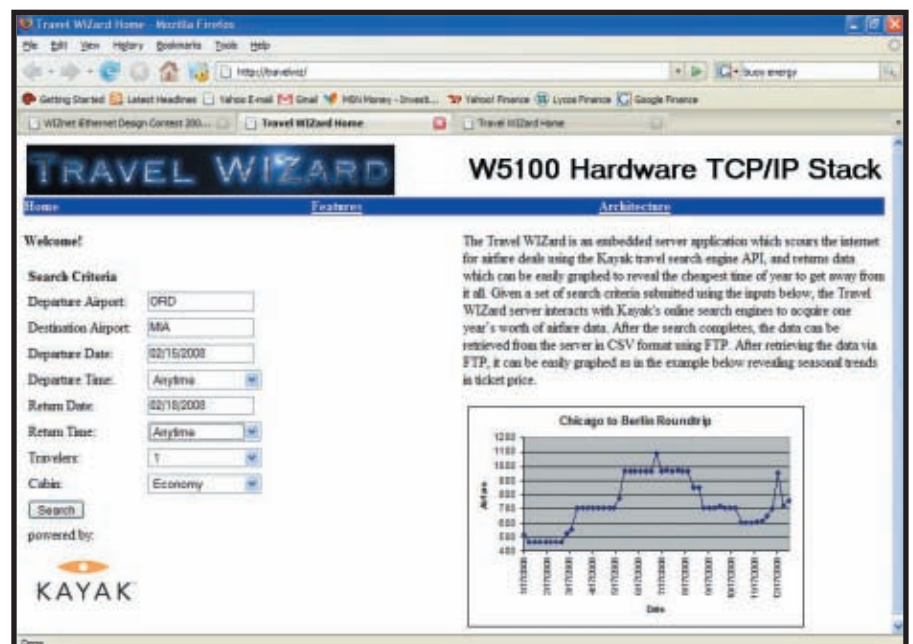


Photo 2—The Travel WIZard's server interacts with Kayak's search engine to gather a year of useful airfare information.

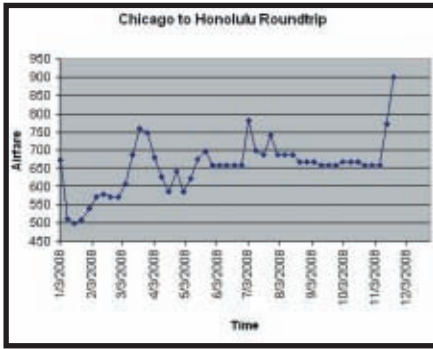


Photo 3—Data retrieved from the Travel WIZard is in comma-separated values (CSV) format. Flight search results retrieved via FTP are graphed to reveal seasonal trends in ticket prices.

development board (see Photo 1).

To use the Travel WIZard, simply enter your search criteria (e.g., departure airport, destination airport, and trip length) on the Travel WIZard web site (embedded in EEPROM on the Explorer 16). When you click the button to start your search, the embedded server interacts with the Kayak flight search engine (www.kayak.com) to acquire one year's worth of airfare data (see Photo 2). After the data is acquired, you can retrieve the results in a comma-separated value format via FTP. The resulting data can be easily graphed to reveal seasonal trends in ticket prices. Photo 3 is a screenshot of the results from a Chicago-to-Honolulu flight search.

HARDWARE

We used a Microchip Explorer 16 development board populated with a PIC24FJ128GA010 microcontroller as the basis for the project (see Figure 1). The Explorer 16 brings out many of the microcontroller's pins to a 120-pin card edge connector, including two SPI bus interfaces and the PIC24's parallel bus interface, the parallel master port (PMP). We acquired a WIZnet WIZ810MJ network module to handle our Ethernet interface. The WIZ810MJ is an excellent choice for quick Ethernet prototyping because it comes as a plug-in module with pin headers and it includes a Mag-Jack and supporting circuitry for the W5100. It also brings out the W5100's SPI and parallel bus interfaces to the pin headers, making

communicating with the device a snap.

The only problem was figuring out how to physically connect the WIZ810MJ to the Explorer 16 board (see Figure 2). Additional schematics, figures, and files are available at www.circuitcellar.com/wiznet/winners/001130.html.

After considering some kind of breadboard-fly-wire-solder combination, we decided there was no better time than the present to design our first PCB. Although it was intimidating at first, the PCB layout process turned out to be an enjoyable experience. We used the freeware version of CadSoft Computer's Eagle layout editor to design the board. The freeware version has some limitations with respect to board size, number of layers, and circuit complexity, but it was more than adequate for our purposes. Advanced Circuits fabricated the board. It took about two weeks to lay out the board, and then an additional week to receive prototype boards. The completed assembly is shown in Photo 1. The WIZ810MJ plugs into two pin headers, which are soldered to the adapter board. The adapter board plugs into the 120-pin card edge connector on the Explorer 16.

We included both W5100 bus interface options in our layout to maximize flexibility. We use the PIC24FJ128GA010's general-purpose I/O pin RF1 to select between the SPI and parallel bus interfaces via the WIZ810MJ's SPI enable control line. At build time, a preprocessor definition selects between the SPI bus driver and the parallel bus driver, so only the necessary code is included in the build.

The PIC24FJ128GA010 has a PMP peripheral specifically designed to

interface with parallel bus-enabled devices such as the W5100. The PMP is convenient because it includes special hardware to handle chip selects, read/write lines, and wait states. However, its many pins are multiplexed with various other peripherals of the PIC24FJ128GA010, including a second SPI module, which is used to interface with the EEPROM that stores our embedded web page. We didn't want to lose the ability to serve a web page from our board just because we switched to the parallel bus, so each time our W5100 parallel bus driver software functions are called, the PMP is enabled, data is transferred, and then the PMP is disabled. When the PMP is disabled, the associated pins revert to the function they had been previously assigned. In this case, that means the SPI module pins go back to SPI functionality. Enabling the PMP only when we need to access the W5100 incurs a slight run-time penalty because we have to execute more than the necessary number of instructions every time we access

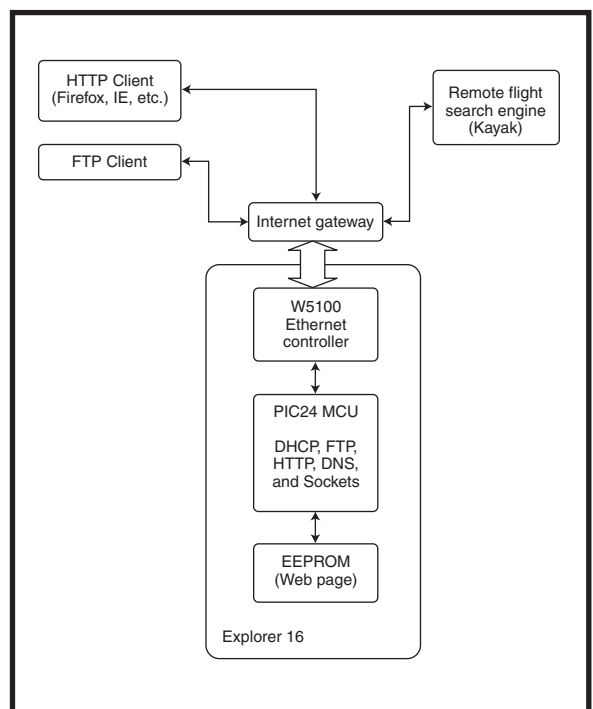


Figure 1—This block diagram shows the Microchip Explorer 16 development board populated with a PIC24 microcontroller at the heart of the system. You can connect to the system via HTTP or FTP to begin a search session or retrieve search results. Once search criteria have been entered, the Explorer 16 automatically interacts with the Kayak flight search engine to retrieve airfare search results.

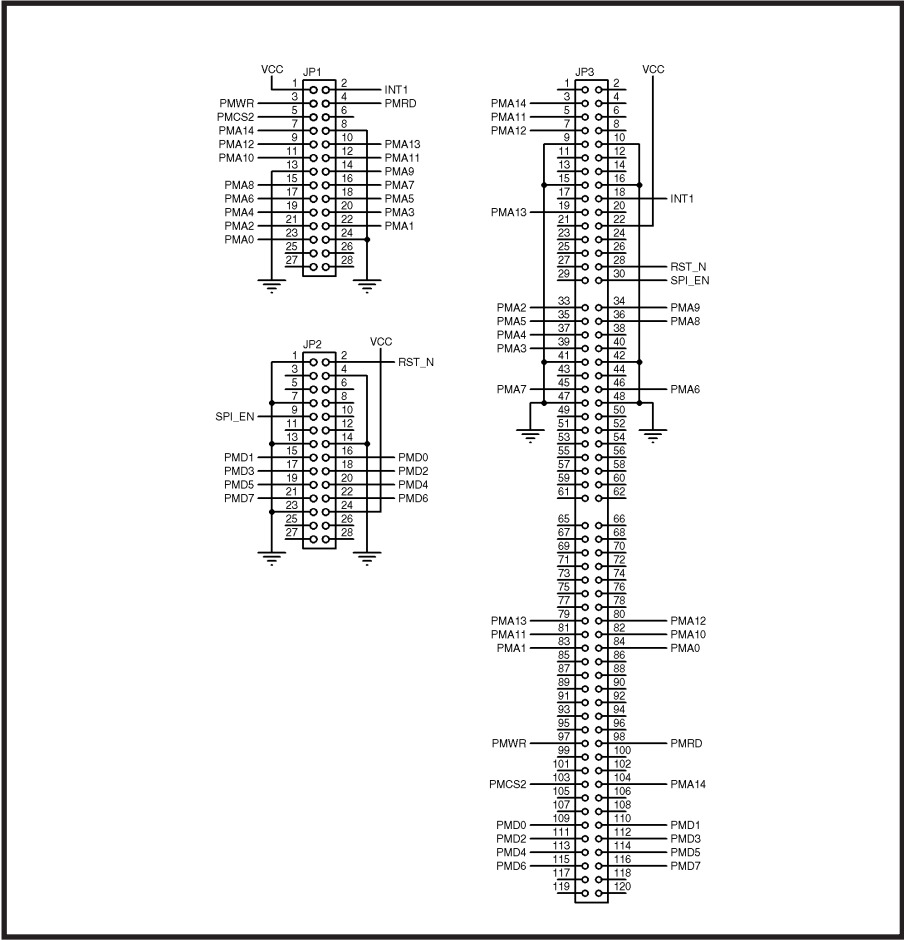


Figure 2—The two jumpers are located on the WIZ810MJ development board. On the left, you can see the connections from the WIZ810MJ 28-pin connectors to PIC24 processor pins. On the right, you can see the Explorer 16 board's 120-pin card edge connector. These pins go directly to the PIC24 processor.

the device. However, it enables us to maximize the PIC24FJ128GA010's available pin functions.

TCP/IP STACK

Microchip provides a free TCP/IP software stack designed to interface with its ENC28J60 Ethernet controller. The stack is a modular implementation of the five-layer TCP/IP model. ENC28J60 driver routines are at the lowest level of the stack. On top of the hardware drivers are the network layer ARP and IP protocols, followed by the transport layer TCP and UDP protocols, and finally the application layer protocols such as HTTP, FTP, and dynamic host configuration protocol (DHCP). In a layered software environment, the rule is that each layer can depend on only the functions in the same layer or lower. That means functions in the network layer should not be calling

functions from the higher transport layer. Layering software also means that data and implementation details can be encapsulated within a layer. The advantage is that implementation details may change in a given layer; but as long as the functionality stays the same, the upper layers will never be the wiser. Understanding the layered structure of the stack is important because it is the mechanism that enabled us to replace the software stack with the W5100's stack, which implements everything from the transport layer on down in hardware.

Let's clarify all of this abstraction with a specific example. Listing 1 is a code snippet from Microchip's HTTP server routine. The logic flow is not too informative when taken out of context in this snippet, but it is useful to notice that the routine is relying on transport layer functions—

such as TCPDisconnect, TCPGetArray, and TCPDiscard—to do its work.

Listing 2 reveals the internal details of the software stack TCPDiscard function, which is used to clear a socket's receive RAM. Compare that to Listing 3, which shows the same TCPDiscard function. This time it is implemented for the hardware stack. Notice that the function prototype is the same, as is the overall functionality (the socket's receive RAM is cleared). Under the hood, however, the implementation details have changed drastically. Nevertheless, from the perspective of the HTTP server routine, nothing has changed. It can call the same function and obtain the same result.

For another example, refer to Listing 3 for the hardware stack that we relied on for our W5100 driver routines W5100WriteReg and W5100ReadWord, which we used to access registers on the Ethernet controller. Doing so enabled us to swap out a SPI bus driver with a parallel bus driver without having to make a single change to any layer-accessing W5100 registers. This incredible technique enabled us to switch from using the SPI bus interface to using the parallel bus interface in a single evening. It was amazing considering it took several weeks to get the entire application functioning properly with the SPI bus.

The Microchip stack implements numerous application-layer protocols. Because of the modular implementation, we were able to rip out the guts of the lower layers of the stack and replace them with W5100 socket driver routines. We were careful to keep the functionality and external interface of the transport and network layers exactly the same. No modifications were necessary to the application layer software. This powerful technique enabled us to reuse all of Microchip's application protocols. At one time or another, we ran HTTP (server and client), DHCP, TELNET, FTP (server and client), DNS, and NBNS application protocols using the W5100. As we tested each protocol one by one, we had to learn enough about it to be able to

Listing 1—Microchip Technology’s application layer implementation of HTTP relies on transport layer functions to handle the details of TCP. The implementation details of TCP are hidden from the application layer.

```
static void HTTPProcess(HTTP_HANDLE h)
{
...
if(w == 0xFFFFu)
{
    if(TCPGetRxFIFOFree(ph->socket) == 0)
    {
        // Request is too big, we can't support it.
        TCPDisconnect(ph->socket);
    }
    break;
}

1bContinue = TRUE;
if(w > sizeof(httpData)-1)
    w = sizeof(httpData)-1;

// Populate httpData array with w bytes
TCPGetArray(ph->socket, httpData, w);
httpData[w] = '\0';

// Discard remaining data from this socket
TCPDiscard(ph->socket);

...
}
```

Listing 2—This code snippet reveals the implementation details of the TCPDiscard function from Microchip’s software stack.

```
void TCPDiscard(TCP_SOCKET hTCP)
{
    TCB_STUB *ps;

    if(TCPIsGetReady(hTCP))
    {
        ps = &TCBStubs[hTCP];
        LoadTCB(hTCP);

        // Delete all data in the RX buffer
        ps->rxTail = ps->rxHead;

        // Send a Window update message
        SendTCP(hTCP, ACK);

        SaveTCB(hTCP);
    }
}
```

know whether or not it was working correctly. Doing so enabled us to acquire a mountain of knowledge on the inner-workings of the Internet during our testing.

FLIGHT SEARCH PACKAGE

After mastering the low-level details of Internet communication, it was time to move on to the task at hand: automating airfare data mining.

We implemented a custom TCP client software package for retrieving airfare information from a remote server. The client software uses the transport layer driver interface to communicate over Ethernet. The package is a group of related files that enable the flight search TCP client to be easily integrated into the top-level application. The centerpiece of the package is the flight

search task, which actually implements the API for communicating with the remote flight search server.

The flight search task waits for a software flag to be set, indicating that search parameters have been entered and a search may begin. The flag is set when a button is clicked on our board’s web page. We added code to the Microchip HTTP server to kick off a flight search session. When the flag is set, the next time the flight search task runs, it connects to the remote flight search server Kayak.com using DNS to resolve Kayak’s IP address. After the IP address of Kayak’s server was resolved, we configured a W5100 socket to be used in TCP client mode. The TCP client mode socket establishes the connection with Kayak, and then uses that connection to acquire airfare data.

The first step toward interacting with Kayak is to acquire a search session ID, which is needed to submit search queries. The Kayak server uses an HTTP interface so all communication is done using TCP. Our client software calls the transport layer TCP functions to transmit to and receive data from Kayak.

Kayak returns all data in XML format. For us, that meant writing a couple of simple parser functions in our client to retrieve data from within Kayak’s XML tags. We learned that the equivalent of a function call is made over the Internet by using a web address as the function name followed by CGI-formatted function arguments. Furthermore, the data transmitted over the Internet is simply ASCII text. For example, to acquire a session ID from Kayak, we make the “function call” by writing “api.kayak.com/k/ident/apisession?token= cRGMxxxxxxxhT7kSEZE2Dw” to the W5100’s transmit RAM and flushing our TCP socket. The “token” argument is actually a developer key, which must be acquired from Kayak in order to use the API. Kayak responds to the function call with an XML document. One of the elements within the XML document is the session ID. We parse through the socket’s receive RAM to

locate the session ID. We then store it safely away into processor RAM and clear the receive RAM buffer.

The session ID, after being acquired once, is used with all ensuing flight search queries. The session ID expires after 30 min. of inactivity (no search queries). To manage the load on its servers, Kayak restricts the frequency of search queries. Developers are limited to about 41 queries per hour. Beyond that, Kayak responds with a message indicating that you have run out of search quota and you need to wait (a message we received frequently during development).

We submit a search query using data supplied from the user entered into our web page via HTTP. We then wait 90 s before polling for results. After retrieving the results from Kayak's XML document, we wait another 90 s, update our search dates to a week later, and run another query. The process continues until 52 searches—covering one year from the initial search date—have been performed. At our current quota limit of 41 queries per hour, it takes about 156 minutes to complete a search run.

We save the top result (the cheapest result from each flight search query). The result is parsed out of Kayak's XML data in the W5100 socket's receive RAM. A snippet of an XML reply from Kayak is shown in Listing 4. This is the actual data that the W5100 buffers into its receive RAM. From this, we parse out the airline and the price (in bold) and store them into an array in microcontroller RAM along with our calculated date.

During a flight search run, all four sockets on the W5100 are active simultaneously. We have one socket in TCP Server mode listening to TCP port 80 for HTTP connections. Another TCP server socket is listening on TCP port 21 for FTP connections. A third socket is configured for UDP to listen for NetBIOS name service requests. The fourth socket does double duty. It manages our DHCP lease, and it is used in TCP Client mode to connect to Kayak and

Listing 3—This code snippet illustrates how implementation details of the TCPDiscard function were changed to support the W5100's hardware stack. The interface and functionality remain the same, so no changes are required to higher level software. With this new implementation, the function relies on lower-level driver routines to communicate with the W5100.

```
void TCPDiscard(TCP_SOCKET hTCP)
{
    WORD ReceiveDataLength, ReadPointer;

    if((ReceiveDataLength = TCPIsGetReady(hTCP)))
    {
        /* Retrieve the current read pointer */
        ReadPointer = W5100ReadWord(Sn_RX_RD0(hTCP));

        /* Increment by the size of available data */
        ReadPointer += ReceiveDataLength;
        W5100WriteReg(SnRXRD0(hTCP),((ReadPointer&0xFF00)>>8));
        W5100WriteReg((SnRXRD0(hTCP)+1),(ReadPointer & 0x00FF));

        /* Delete all data in the RX buffer */
        W5100WriteReg(Sn_CR(hTCP),W5100_SOCKET_RECV);
    }
}
```

Listing 4—The Kayak flight search API returns results in XML format. Key off of the tag names to parse out results of interest such as airfare price and the airline offering the trip.

```
<?xml version="1.0"?>
<searchresult>
  <searchinstance></searchinstance>
  <searchid>s_c7Lu3a6QgVzrppeNtQ</searchid>
  <count>15</count>
  <morepending>>true</morepending>
  <trips>
    <trip>
      <price url="/book/flight?code=2-
zEGArPtH5_L$r14jZSJ1.s_c7Lu3a6QgVzrppeNtQ.19-
ZGnLUYLfGuVsthbFkgeW.F.AA.20880.0&_sid_=19-ZGnLUYLfGuVsthbFkgeW"
currency="USD">209</price>
      <legs>
        <leg>
          <airline>AA</airline>
          <airline_display>American Airlines</airline_display>
          <orig>ORD</orig>
          <dest>MIA</dest>
          <depart>2008/01/17 20:10</depart>
          <arrive>2008/01/18 00:05</arrive>
```

retrieve flight search results. We can use a single socket to manage DHCP and connect to Kayak because, in both cases, the socket does not need to stay connected all of the time.

FINDINGS & IMPROVEMENTS

We learned a lot about the Internet as we worked on this project. Sure, we were already familiar with terms such as "MAC address," "DHCP," "gateway server," and "client application," and we knew enough to get by. But we didn't know too much

about how the Internet really functions under the hood. This project taught us a lot about the underlying details of Internet communication.

If you're planning to put your current embedded project online, we recommend obtaining a network protocol analyzer. It will do wonders for your debugging capabilities. We used an excellent open-source application called Wireshark (formerly Ethereal). Wireshark captures Ethernet packets in real time from your PC's wireless or LAN cards. Packet contents are

displayed in a user-friendly, easy-to-read format. Depending on the packet's protocol, the contents are broken out byte-by-byte according to the different fields contained in the packet. The program also has numerous filtering options to limit the number of captured packets to only what you are interested in. Wireshark was essential to the development of our project.

PLANNING AHEAD

We have used the Travel WIZard to acquire airfare data to destinations all over the world. It is interesting to observe seasonal trends in ticket prices to different locations. However, the application has plenty of room for improvement. The once-per-week sample rate is rather sparse, allowing opportunities for price variation that we could miss if we happen to sample on the wrong day. We plan to implement a more dense mechanism for storing results so we can pack a greater number of results into the same amount of

RAM. (Currently, we store in ASCII format.)

We also want to be able to view the search progress from the embedded web page. Google offers a graphing API (Google Chart), in much the same way that Kayak offers an API, which can be used to graph data. Using the Google Chart API, we could graph the flight search results as they come in, right on the embedded web page.

Many web sites offer easy-to-use APIs just like Kayak. Using the W5100 and the Internet application layer protocols, the number of applications is limitless. We are anxious to add more automated packages to the Travel WIZard to make it an even more productive machine. Doing so should enable us to spend more time on the beach while the system does the dirty work for us. ☒

Editor's note: To learn more about this project, you can review the schematics, photos, diagrams, and

additional files posted at www.circuitcellar.com/wiznet/winners/001130.html.

Matt Pennell (mgpennell@gmail.com) is an embedded software engineer for Caterpillar's Electronics & Connected Worksite Division. He specializes in hardware drivers, TPU microcode, and eTPU microcode for engine control units. Matt earned a B.S.E.E. at the University of Illinois at Urbana-Champaign. His technical interests involve designing hardware and software for real-time embedded systems. Outside of work, Matt enjoys weight lifting and sports.

Aaron Thomas (thomaal@gmail.com) is an engineer for Caterpillar's Electronics & Connected Worksite Division. He specializes in FPGA development for electronic control modules. Aaron earned a B.S. in computer engineering from the University of Toledo in Ohio. His technical interests include computer design, programming, and automation using embedded systems. In his spare time, Aaron enjoys traveling, sports, and video games.

Windows CE Touch Controller
CUWIN™

CUWIN4300K ▶
\$499 / Qty 1

- 7" / 10.2" wide color TFT display
- 800 x 480 resolution, 260K colors
- SD card & Ethernet support
- RS232 x 2 / RS485 x 1 or RS232 x 3
- Mono Speaker and Stereo Jack
- Real time clock (Battery backup)
- Visual Basic and Visual C++ support
- USB I/F (ActiveSync)
- ARM9 32bit 266MHz processor
- Keyboard or Mouse support

CUWIN3200 ▶
\$399 / Qty 1

CUWIN020 7" CUWIN030 7" CUWIN040 10.2" CUWIN050 15.2"

CUWIN060 7" Base type case
CUWIN080 7" Cover Case (Front waterproof)
CUWIN090K 10.2" Silver Bezel type case
CUWIN100K 10.2" Silver Bezel type case

COMFILE TECHNOLOGY www.cubloc.com
Toll-Free: 1.888.928.2562

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2008/219.

SOURCES

EAGLE Light Edition
CadSoft Computer, Inc.
www.cadsoftusa.com

Wireshark Network analyzer
Gerald Combs
www.wireshark.org

ENC28J60 Ethernet controller, Explorer 16 development board, and PIC24FJ128GA010 microcontroller
Microchip Technology, Inc.
www.microchip.com

WIZ810MJ Network module and W5100 Ethernet controller
WIZnet, Inc.
www.wiznet.co.kr/en



Time Server Design

Synchronize with the WWVB Time Code Signal

Coordinate your Ethernet applications with Steven's Time Server. The system keeps a master time and date clock that is synchronized to the U.S. WWVB time code signal. It connects to an Ethernet network and sends time and date information according to the SNTP, DAYTIME, and TIME protocols. Now, no matter their locations, your devices can connect to the system, request the time and date, and synchronize their local clocks.

Back in the 1980s, characters in television shows like *The A-Team* always counted on a well-executed plan of explosions and traps set to go off at just the right time. You would often see Hannibal synchronizing his watch, although the others seemed to make things up as they went along. If you have a "team" of devices on your

Ethernet network, they may need to be time synchronized. You wouldn't want data from two data collection devices to have significantly different timestamps, and you especially wouldn't expect to see data from the future.

My Time Server design provides a reference time and date for an Ethernet network (see Photo 1). It's synchronized to the WWVB time signal so you can have separate physical networks in different locations around the country that will be synchronized to the same time and date. You can use it in secure Ethernet networks where connection to the Internet is not allowed, restricted, or unavailable. Because it's inexpensive, it can be used in the home to keep your Ethernet-enabled irrigation controller, security system, and weather station all working from the same time reference.

In an industrial Ethernet network, the design becomes another node on the network helping to keep multiple process controllers synchronized and data timestamps accurate.

Figure 1 shows the project's three main parts. The time code receiver demodulates the WWVB signal into a pattern of pulses that indicate the time and date. A Freescale Semiconductor MC9S08QG8 microcontroller decodes the pulse pattern to get the time and date, maintains a local real-time clock, and manages the high-level Ethernet protocols for serving the time and date. A WIZnet W5100 controller handles the interface to the Ethernet. It can manage Ethernet data transfer up to the TCP/IP level, so the design is simple.

RECEIVING THE TIME CODE

The WWVB is a radio station managed by the National Institute of Standards and Technology (NIST). Don't bother trying to tune to it on your AM or FM radio dial because the frequency is 60 kHz and the content is about as exciting as the daily hog futures report. But it's one of the most popular radio stations in the United States because millions of devices are receiving its broadcast to set the time and date in home alarm clocks, school

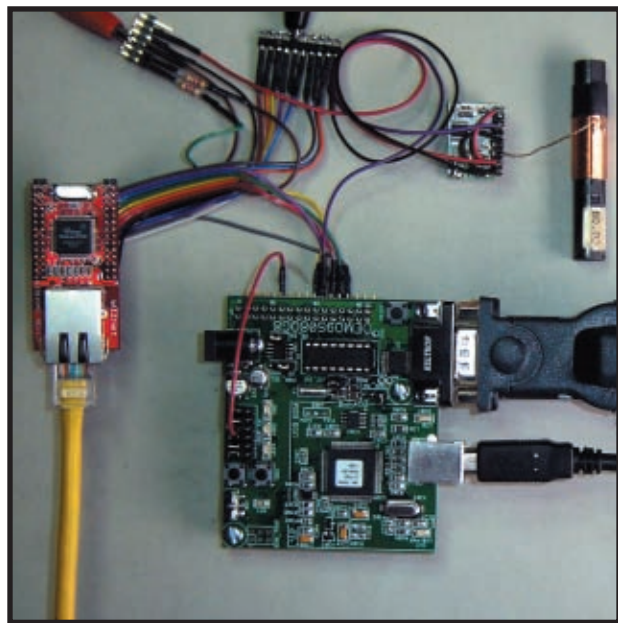


Photo 1—Using hardware modules provided by the manufacturers of the three main components greatly accelerated the project's development. The WIZnet WIZ810MJ module, Freescale Semiconductor DEMO9S08QG8, and C-MAX Time Solutions CMMR-6 are wired according to the main schematic. Power is provided by a 3.3-V bench supply.

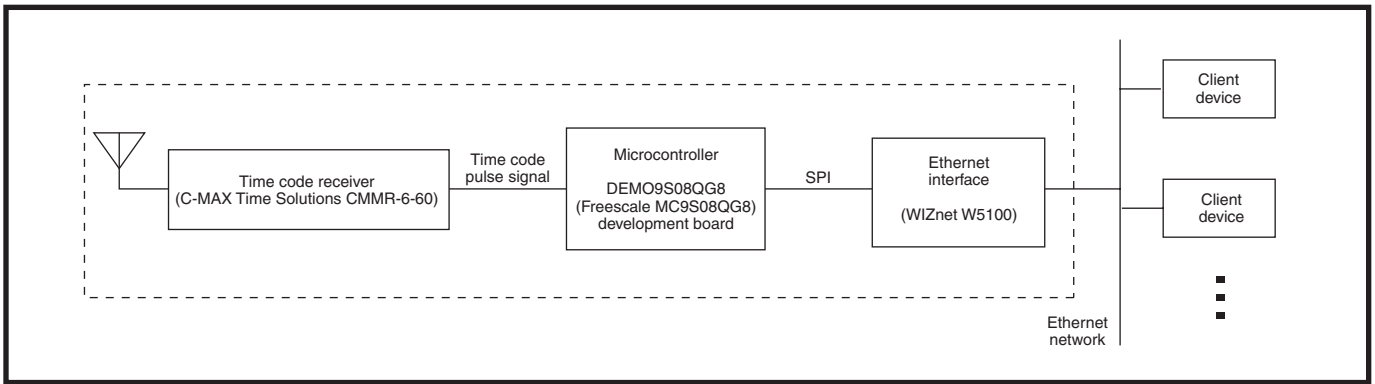


Figure 1—The server has only three main components: a C-MAX Time Solutions WWVB time code receiver, a Freescale MC9S08QG8 microcontroller, and a WIZnet W5100 Ethernet interface.

and government clocks, and even wristwatches.

The time and date information is encoded into the broadcast by reducing the power of the carrier frequency for a specific time in a 1-s period. For a binary 0, the power is reduced for 0.2 s. For a binary 1, the power is reduced for 0.5 s. There is a special indicator called the “position marker,” which is created by reducing the power for 0.8 s. The position marker is used to indicate the start of the frame of data. There are various markers within the frame at defined positions so that the device decoding the stream can verify that it is synchronized.

A frame contains 60 bits, where a bit is a 1, 0, or position marker. Because each bit has a 1-s period, it takes 60 s (1 min.) to send the frame. Figure 2 shows a sample frame. You might think “seconds” information is not in the frame, but it is. The first bit is a position marker. The start of the bit also indicates “seconds = 0.” The

next bit indicates “seconds = 1,” and so on.

When a number is part of the information, such as the minutes value, the bits are arranged using binary coded decimal (BCD). There are flags and values that indicate daylight savings time, leap year, leap second adjustment, and correction offset to Coordinated Universal Time (UTC). Position markers occur at various places in the frame, as I mentioned earlier. There is a position marker at the start of the frame and another at the end. When two consecutive position markers (end of previous frame and start of current frame) are detected, the start of frame has been found. For more information, refer to the NIST’s web site.

To demodulate the WWVB signal, the server uses a C-MAX Time Solutions CME6005 time code receiver. The antenna and receiver IC chosen for the server are specific for the U.S. WWVB signal; however, C-MAX offers

parts for other frequencies, as well as a multifrequency solution. I purchased the C-MAX CMMR-6 receiver module (antenna included) from Digi-Key for approximately \$10.

The CME6005’s output is a signal that represents the WWVB time code signal, but at a level that is suitable for interfacing to a microcontroller or perhaps an application-specific integrated circuit (ASIC). The microcontroller, with its input capture feature, detects a negative edge and measures how long the signal remains low. Per the WWVB signal, if it remains low for 0.2 s, a 0 bit is decoded. If it remains low for 0.5 s, a 1 bit is decoded. If the signal is low for 0.8 s, a position marker is decoded.

After power is applied, it may take up to 2 min. before the project will have a time and date set into its local clock. The worst case would be if power is applied and the first bit of the time code it sees is actually the second bit of a frame. In that case, you would

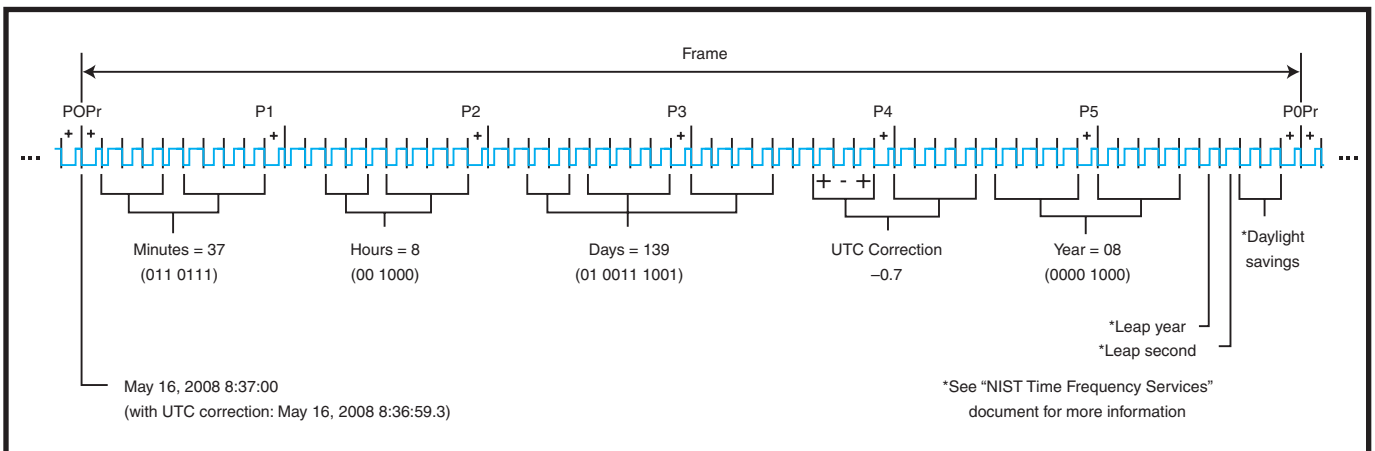


Figure 2—The time code signal consists of a frame of 60 bits with a 1-s period. The year, day, hour, minute, and other informational flags are encoded in the frame.

have to wait for the rest of the partial frame to complete and then wait for the next full frame to be decoded.

When a full frame of 60 bits has been received, the microcontroller's firmware checks to ensure that all seven position markers are set. Then it parses through the bits to decode the years, day, hour, and minute to compute a value that represents the number of seconds since January 1, 1970 0:0:0. For some additional debug assistance, a flag is set that will cause a date and time string to be sent out of the microcontroller's UART.

SERVING TIME (THE GOOD WAY)

The design uses the MC9S08QG8 microcontroller to decode the time code signal, maintain a local real-time clock, and run the design's Ethernet protocols. While the MC9S08QG8's built-in real-time clock feature is handy, you could easily swap in your favorite microcontroller. I had Freescale's DEMO9S08QG8 development board handy. All it needed was a few resistors, capacitors, and a 32.768-kHz crystal added to the unpopulated area of the board that is specifically designed for these components. For firmware development, I used the free version of Freescale's CodeWarrior development studio.

Listing 1 is a high-level view of the firmware. (The full source code is posted on the *Circuit Cellar* FTP site.) The "main" function starts off by setting up the time code decoder and the interface with the WIZnet W5100, then the "forever" loop continuously checks to see if any Ethernet client devices are requesting the time and date. The server supports three high-level protocols for serving the time and date: Simple Network Time Protocol (SNTP), DAYTIME, and TIME.

SNTP is implemented according to request for comment (RFC) 1361. A network client makes a connection to port 123 of the server and sends a message using UDP. The server responds with a similar message that includes the time and date information.

DAYTIME and TIME protocols are even simpler. The DAYTIME protocol

Listing 1—This pseudocode shows a high-level view of the server's firmware. After the hardware is initialized, the forever loop continuously checks to see if a client is requesting the time and date.

```
//Setup:
Initialize MCU registers (set up real-time interrupt)
Initialize the WIZnet W5100
    Set each Tx and Rx buffer for 2K size
    Set Gateway address (hardcoded)
    Set Subnet mask
    Set MAC address (hardcoded)
    Set IP address (hardcoded static IP)
Turn on MCU's input capture feature
Open the sockets for the three servers

//“Forever” Loop:
//SNTP Server (UDP, port 123)
If SNTP message received,
    Read the message into a local buffer
    Update the time and date fields
    Send the message to the client

//DAYTIME Server (TCP, port 13)
If client is connected,
    Format time and date string.
    Send time and date string to client
    Reset the connection

//TIME Server (TCP, port 37)
If client is connected,
    Send the 32-bit time value to the client
    Reset the connection
```

is used when a client makes a TCP connection on port 13. A string with the following format is sent to the client:

```
<Weekday name>, <Month name>
dd, yyyy hh:mm:ss-UTC
```

If a TCP connection on port 37 is made, the TIME protocol server sends a 32-bit number representing the number of seconds since January 1, 1900 to the client.

The project must always be able to serve the time and date, even when RF interference prevents the WWVB signal from being received. Therefore, a local real-time clock is maintained by incrementing an integer variable that represents the number of seconds since January 1, 1970. The MC9S08QG8 supports a real-time interrupt feature that is set to fire an interrupt every second based on an external 32.768-kHz reference crystal. When a client device makes a request, the time and date is formatted according to the local real-time clock. When the interference clears and a full-time code frame is received, the local clock is updated with the new time and date

information.

Let me discuss one last item concerning Ethernet time servers. The Network Time Protocol (NTP) is a popular protocol that this project does not support. The firmware is significantly more complicated because it adjusts for latencies that are the result of the Ethernet data transfer. The NTP is used in networks that require precise timing and where the reference clock is not dependent on the RF reception from the WWVB radio station, but rather directly from a cesium clock.

SIMPLE ETHERNET

The CME6005 makes receiving the WWVB time code signal simple by combining all of the discrete demodulation hardware into a single chip. The microcontroller makes the firmware simple because the SPI, real-time clock, and input capture features are all integrated into the hardware. And, last but not least, the WIZnet W5100 makes the Ethernet interface simple because it manages all of the Ethernet protocols right up to the TCP/IP level.

The W5100 has both a parallel

interface bus and a SPI. I used the SPI because the server's data throughput requirements are low. The MC9S08QG8 microcontroller includes a SPI master as one of its built-in peripherals, so writing and reading bytes to and from the W5100 is as simple as writing and reading a register. Each SPI transaction involves sending 4 bytes: a write or read command, the MSB of the register address, the LSB of the register address, and the byte to write to the register (or 0, if reading). When reading, the final byte written is merely used as a dummy byte so that the SPI clock causes the W5100 to place the contents of its register on the MISO line. The MC9S08QG8's SPI data register is then read to get the byte.

The W5100 contains multiple registers that control its configuration and access to the incoming and outgoing data buffers. The first set of registers is called the "common registers." They are used to set up the gateway address, subnet address, IP address, MAC address, and several configuration

Listing 2—This code shows how to read the W5100's Socket 0 Rx buffer. The W5100's registers must be used to locate where in the buffer new data can be read. Then, after reading the data, the registers must be updated so the buffer location pointer will be ready for the next read operation.

```
int nb, addr, wizptr, i, numbytes = 0;
nb = (WIZNET_rdwByte( WIZNET_READ,
                    WIZNET_Sn_RX_RSRO(0) + 0, 0 ) << 8);
nb += WIZNET_rdwByte( WIZNET_READ,
                    WIZNET_Sn_RX_RSRO(0) + 1, 0 );
//Get location of data
addr = (WIZNET_rdwByte( WIZNET_READ,
                    WIZNET_Sn_RX_RDO(0) + 0, 0 ) << 8);
addr += WIZNET_rdwByte( WIZNET_READ,
                    WIZNET_Sn_RX_RDO(0) + 1, 0 );
wizptr = addr & 0x07FF; //2K - 1 = 0x800 - 1
wizptr += 0x6000; //2K size
for( i = 0; i < nb; i++ )
{
    if( numbytes < 80 )//prevent overrun of local buf
    {
        buf[numbytes++] = (char)WIZNET_rdwByte( WIZNET_READ, wizptr, 0 );
    }
    ++wizptr;
    //It's a circular buffer, so roll at the top.
    if( wizptr == 0x6800 ) //2K size
    {
        wizptr = 0x6000;
    }
}
addr += i;
//Store pointer (only lower 16-bits)
WIZNET_rdwByte( WIZNET_WRITE, WIZNET_Sn_RX_RDO(0) + 0,
                (unsigned char)(addr >> 8) );
WIZNET_rdwByte( WIZNET_WRITE, WIZNET_Sn_RX_RDO(0) + 1,
                (unsigned char)(addr & 0xFF) );
//Reset flags
WIZNET_rdwByte( WIZNET_WRITE, WIZNET_Sn_IR(0), Sn_IR_RECV );
WIZNET_rdwByte( WIZNET_WRITE, WIZNET_Sn_CR(0), Sn_CR_RECV );
```

solder by numbers™

Finally, a social network for you, the electronics enthusiast.

Design Circuits
 Make New Friends
 Chat with VoIP and Webcam
 Write Blogs
 And Much More!
 Join Clubs
 Find a Job
 Earn Commissions

Your hobby has finally entered Web 2.0 status.
Sign up for your free membership today!
www.SolderByNumbers.com

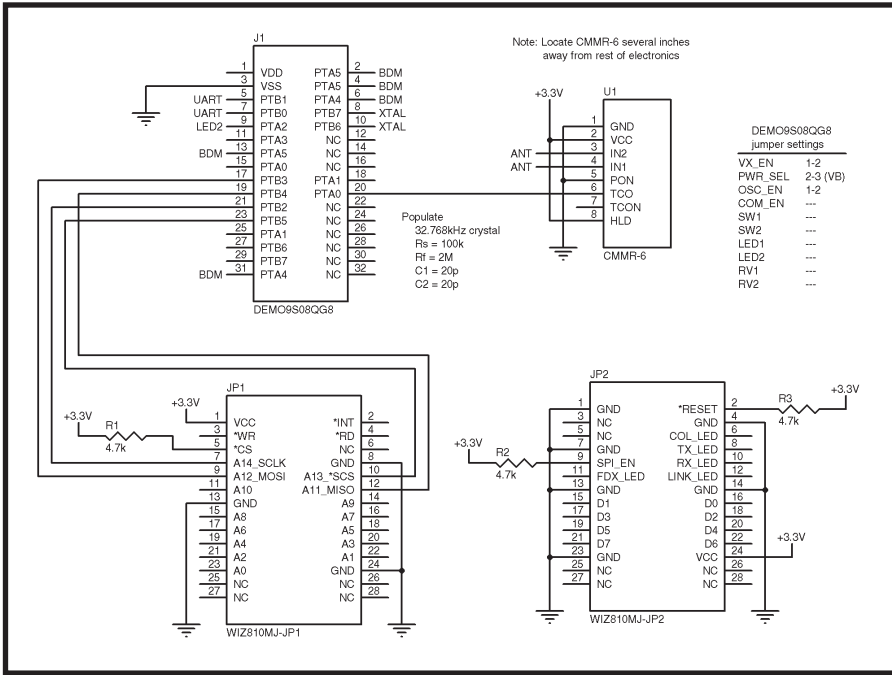


Figure 3—The design's hardware was assembled using modules available from the manufacturer of each of the three main components of the device. Take a look at how the C-MAX CMMR-6, Freescale DEMO9S08QG8, and WIZnet WIZ810MJ are wired.

items that control how the Ethernet interface is managed. In the server's firmware, these registers are set with

hard-coded values to keep things simple. The MAC address is set with random values that are unique to my Ethernet

network. A finished product would need to reserve a block of MAC addresses from the IEEE to guarantee that it has a globally unique MAC address for whatever network it is installed in.

The W5100 supports four sockets. Think of it like an office phone with four lines. Up to four people can call and you can answer them, service their calls, or place them on hold and service them as needed. When a conversation is over, the connection is closed. Each socket has a set of registers that is used to set up the connection. The design uses only three of the four sockets, one for each of the three servers supported. Instead of a physical "line" (per the office phone analogy) data from the Ethernet is routed to a socket based on the socket's "port" identification number. To get all of the data transferred back and forth, the server and the client (the device calling into the system) must use the same language or, in Ethernet terms, protocol. We are using the user datagram protocol (UDP) and transmission

FlashPro430
GangPro430

USB Flash Programmers for Texas Instruments' MSP430 microcontrollers.

Reliable and the fastest programmer on the market.
Perfect for production usage.

- ✓ 60 kB Flash can be programmed in about 2 seconds
- ✓ supports JTAG, Spy-BI-Wire and BSL interfaces
- ✓ can assign unique serial numbers
- ✓ up to eight programmers can be connected to one PC and program target devices simultaneously

New: FlashPro-CC and GangPro-CC
USB Flash Programmers for CC series devices (ChipCon) from TI

Programmers

- USB-FPA-1 JTAG / SBW or BSL MSP430Fxx
- USB-FPA-2 MSP430Fxx
- USB-FPA-3 MSP430Fxx
- USB-FPA-8 MSP430Fxx

One PC and 8 programmers

Elprotronic Incorporated
www.elprotronic.com

We Build Tiny Web Servers

Lowest-Cost Designs
Complete Software
Fast Delivery

Low-Cost Web Servers

Web-Enable Almost Anything for Almost Nothing

By the Staff of Geis Technology

FREE BOOK!

GEIST Technology

Request your free book at (512) 331-8676 or FreeBook@GeisTek.com

control protocol (TCP), which are fully managed by the W5100.

While the W5100 manages the transfer of data with the client, you need to further define what time and date information should be transferred. The servers use protocols defined in request for comment (RFC) documents that are managed at www.rfc-editor.org. I briefly covered these protocols in the previous section.

When you know what data will be transferred, you need to get the WIZnet W5100 to send and receive it. Each socket has registers that are used for reading bytes from an incoming data buffer and registers for writing bytes to an outgoing data buffer. While you could just reference the W5100 datasheet and other examples from the WIZnet web site to see how to use the registers, sometimes it's difficult to pick up the nuances of the interface. Listing 2 is a sample of the code I used to read the bytes from socket 0's incoming data buffer.

For testing, I created a Java application that runs on a Windows PC with

NetBeans6 development software. I specifically wanted to borrow the parts of the code that act as the client side of the protocols so I could confirm that my servers are compatible with implementations made by other people. For the SNTP client, I credit Adam Buckley with his contributions to the NTP Public Services Project. And for the TIME client, I referred to *Java Network Programming* by Elliotte Rusty Harold.

I'll now share a couple of frustrating experiences. This way you can prevent them from happening to you. In my network setup, I had the project, a PC running a time client software application, and a PC running Ethereal to monitor network data transfer, all connected to an Ethernet switch device. I wasn't seeing the proper data packets going back and forth between the project and the PC running the time application. After spending several hours trying to debug my code that drives the WIZnet W5100, I focused on my network setup. An Ethernet switch associates an IP address with

the devices plugged into its RJ-45 hardware ports, and it routes data packets to only the appropriate device. After replacing the Ethernet switch with an Ethernet hub, I was able to monitor the data packets using the PC running Ethereal (a hub sends all data packets to all devices). I also had a problem when I changed to a new static IP address in the design. The Ethernet switch device still had the old IP address associated with the hardware port. So, after several hours of head scratching, I power cycled the Ethernet switch so it would learn the IP address. This issue was also fixed when I started using the Ethernet hub.

MODULAR HARDWARE

I always try to set up a first-level prototype using hardware modules, development boards, or evaluation boards offered by the manufacturer. I am more of a firmware engineer than a hardware engineer, so I like getting something up and running first to demonstrate feasibility.



As low as...

\$9.95
each!

Two Boards
Two Layers
Two Masks
One Legend

Unmasked boards ship next day!

www.apcircuits.com



Expand Your I/O with Rabbit® RIO

Versatile Device with I/O Options

- Add functionality without costly processor changes
- Multiple Processor Interfaces
- Add 38 I/O
- Configure I/O for PWM, TRIAC, input capture, or decoder



Rabbit RIO™ Programmable I/O Application Kit for **\$299**



Order Online At **rsappkits.com**

08033

Figure 3 shows the server. Photo 1 shows the modules wired together. The CMMR-6 is a module (antenna included) for the CME6005. All that is required is power, ground, and a line from the time code signal output to the timer input capture pin of the microcontroller. I observed that moving the CMMR-6 several inches away from the other electronics decreased the RF interference while receiving the time code signal. Also, the time code signal would not be received at all if the fluorescent lamp on my bench magnifier lamp was on (even though I live only 50 miles from the WWVB transmitter in Fort Collins, CO).

The DEMO9S08QG8 is a demonstration board for the MC9S08QG8 microcontroller. It includes the Freescale Background Debug mode (BDM) for quick programming and debugging. I use the UART port to send out the time and date whenever the time code signal is received and decoded correctly. A simple connection to one of the LEDs becomes a heartbeat indicator. The DEMO9S08QG8 includes unpopulated pads for the 32.768-kHz external reference clock and support components. These components were easily stuffed so that a local real-time clock feature could be supported.

I used the WIZ810MJ module for the W5100. It includes the required crystal and Ethernet RJ-45 jack with built-in magnetics. A SPI is used between the microcontroller and the W5100. This keeps the amount of wiring down to just a few digital signal lines plus a few for power and ground. The WIZ810MJ uses 2-mm headers, so I found compatible receptacles, soldered wires to the pins per the schematic, and terminated the other end with pin sockets that can be inserted on the 0.1" header of the DEMO9S08QG8. A bench instrument set for 3.3 V powers this first-level prototype.

FINISHED DESIGN

A finished hardware design could easily be realized with a PCB measuring about 3 square inches. Consider keeping the C-MAX time code receiver

chip and antenna on a separate board and connected through a 4" to 6" cable so that it can be positioned for the best reception. With a plastic box, a wall transformer, and a 3.3-V power circuit, the cost should stay less than \$100.

To take the project to the next level, add some robustness to the firmware. The time code data does not contain a checksum or CRC, so make sure at least three time code frames are received where the "minutes" value is one higher than the last before the local real-time clock gets changed. The project's firmware could also be enhanced using all the regular tricks: enable the watchdog, error check the data, blink the heartbeat LED differently for errors, and more. I recommend that you periodically read one of the W5100 registers during times of few client requests just to make sure the Ethernet interface is still active.

A few of the settings for the Ethernet interface are hard-coded in the firmware. You'll need your own MAC address, and you must set up your own static IP address appropriate for your network. I gained some experience creating code to run DHCP client functions while I was working on a different project—which I named the Internet Weather Display—so I know porting it to the project's firmware wouldn't be too difficult. WIZnet's web site provides several application notes on the topic of implementing a DHCP client. 📄

Steven Nickels has a B.S. in electronic engineering technology from Minnesota State University at Mankato. He works as a senior software engineer for Medtronic Navigation in Louisville, CO. Steven's main area of interest is firmware development, but he picks up a soldering iron every now and then. Several of his interesting projects are posted at <http://ssea000.googlepages.com>.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2008/220.

RESOURCES

A. Buckley, "Section 13.3 Java SNTP Client," The NTP Public Services Project, <http://support.ntp.org/bin/view/Support/JavaSntpClient>.

C-MAX Time Solutions, "CME6005 Datasheet," 2007.

———, "CMMR-6 Receiver Module Datasheet," 2007.

Freescale Semiconductor, Inc., "Application Module Student Learning Kit Users Guide Featuring the Freescale MC9S08QG8," DEMO9S08QG8, 2006.

———, "HCS08 Family Reference Manual," Rev. 2, 2007.

———, "MC9S08QG8 MC9S08QG4 Data Sheet," Rev. 4, 2008.

E. R. Harold, *Java Network Programming*, Third edition, O'Reilly Media, Inc., Sebastopol, CA, 2004.

M. Lombardi, "NIST Time and Frequency Services," NIST Special Publication 432, National Institute of Standards and Technology, 2002, <http://tf.nist.gov/general/pdf/1383.pdf>.

RFC Editor, "RFC 1361," "RFC 867," and "RFC 868," www.rfc-editor.org/rfc.html.

WIZnet, Inc., "W5100 Datasheet," Version 1.1.6, 2008.

———, "WIZ810MJ Datasheet," Version 1.1, 2007.

SOURCES

CME6005 Time code receiver and CMMR-6 receiver module
C-MAX Time Solutions
www.c-max-time.com

DEMO9S08QG8 Development board and MC9S08QG8 microcontroller
Freescale Semiconductor, Inc.
www.freescale.com

NetBeans6 IDE
NetBeans Community
www.netbeans.org

W5100 Ethernet Controller and WIZ810MJ module
WIZnet, Inc.
www.wiznet.co.kr

Networked Timing

Build a Timer With Advanced Planning Tools

Precision irrigation control is now a reality. Thomas's irrigation timer with advanced planning (ITAP) is a truly novel irrigation control system. The easy-to-use system, which directs user interaction into a standard web browser, provides useful information such as watering schedules and zone activity.

Electronics are supposed to simplify our lives, but all too often, the reverse is true. While researching irrigation timers, I was struck by how the evolution from an electromechanical design to an electronic design had actually made the device less usable. Early irrigation timers were based on the simple rotary motor design still popular in plug-in appliance timers. A geared motor slowly turned a wheel with on/off pins. If you wanted water at 6 A.M. in zone 2, you simply pushed in the pins for zone 2 and 6 A.M. The wheels and pins also served as crude analog gauges. A glance at the wheel would give you a pretty good idea of when things would next turn on and for how long.

In the electronic age, the wheels and pins are gone. They have been replaced by a tiny LCD and a small keypad. The electronics have added significant new capabilities, but at the cost of requiring significant data entry with a tiny keypad. The visual feedback about what you've programmed is also gone. To ensure that you have not accidentally programmed a 4-h flood, you need to page through identical screens, taking note of each setting.

As you can see in [Photo 1](#), I designed

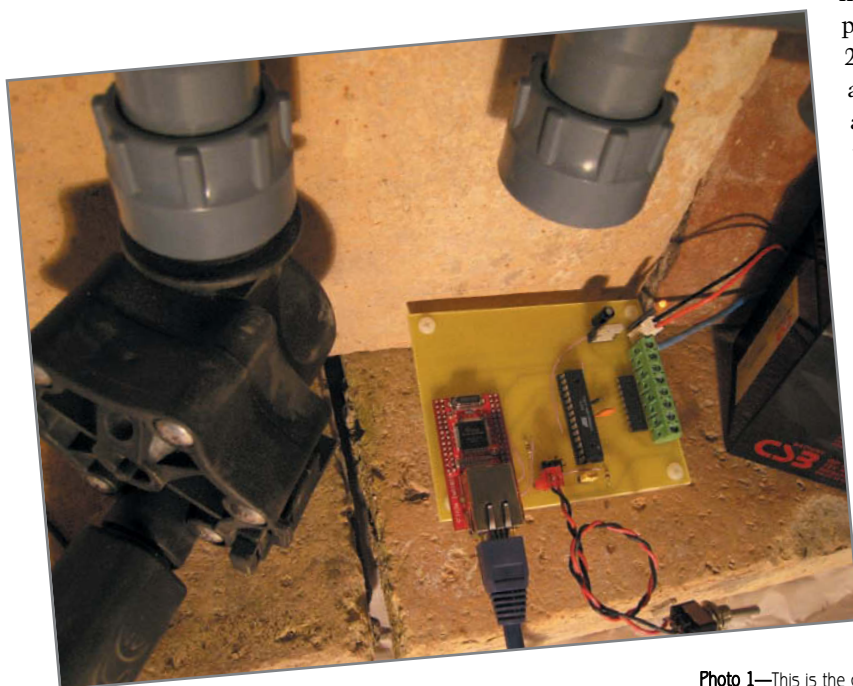


Photo 1—This is the completed circuit board. The WIZnet WIZ810MJ is the red board with the RJ-45 connector. A solenoid valve connected to a manifold is to the left of the circuit board.

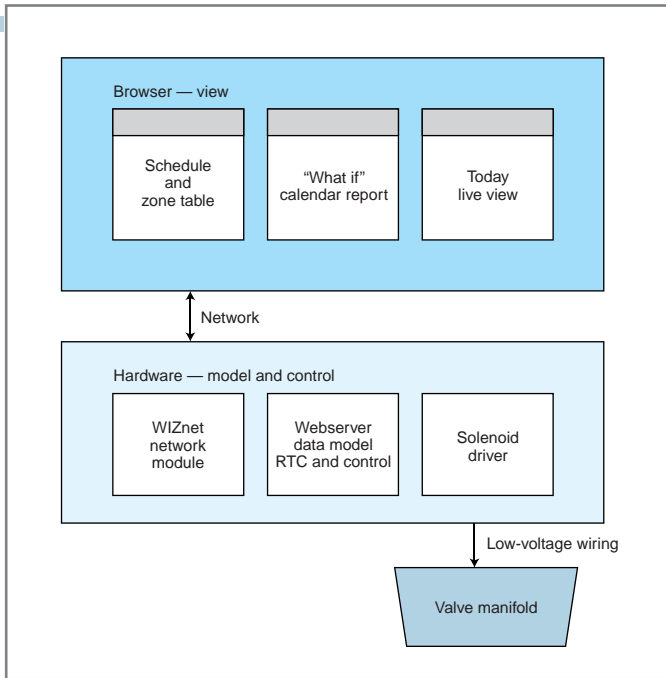


Figure 1—This shows the interaction between the browser pages, the hardware, and the physical zone valves.

electromechanical design. Because the biggest problem was entering and displaying large amounts of data, I selected a web browser for the user interface. I considered various USB-based ways of connecting to a browser, but they invariably required some type of installation on a host PC. Around that time, I saw the WIZnet iEther net Design contest 2007 announcement and decided to build my timer design around the WIZnet WIZ810MJ network module.


A potential design trap in a network-enabled device is the temptation to do too much with the network. An irrigation timer should be a simple device. It is, after all, just a timer and some solenoid valves. The network interface is present in the design because it is the best way to connect to a web browser. I have always been drawn to simple, low-cost designs, so I used the network interface to save money by eliminating an LCD and keyboard. Others might use the network interface as an excuse to jam in seldom-used features and boost the end product's price. Once I started building a web browser interface, I realized that this was a good opportunity to show that web technologies, which are normally associated with large enterprise deployments, can also be used effectively on micro devices.

an Irrigation Timer with Advanced Planning capability (ITAP). The challenge was to incorporate modern functionality, yet keep the design as simple to use as the old

ITAP DESIGN

The design philosophy was to not consider the ITAP as a networked or web-enabled device, but rather to use the

Cellular and GPS capable
Data Mover



- Flash file System
- 4 Chan 12-Bit A/D
- 1MB SRAM
- 512KB FLASH
- 4 Isolated Inputs
- 4 Hi Current Outputs
- 2 External RS-232
- 2 Internal RS-232
- Bat Backed Clk/Cal
- Cell Modem Option
- Internal GPS Option
- Metal Case Option
- 1ma Standby Option

It's easy and cost-effective to do mobile or solar-powered data collection and asset monitoring with the JK micro's **Data Mover**. With the ability to integrate a Cellular Modem, GPS and DOS-based embedded controller in a single rugged enclosure, you can capture and transmit your data quickly, easily and at low cost. Inexpensive development kits including Borland C/C++ and PowerBasic are available now. Call or email us for more details.

Call **530-297-6073** Email sales@jkmicro.com
 On the web at www.jkmicro.com

JK microsystems



AP CIRCUITS
 PCB Fabrication Since 1984

As low as...
\$9.95
 each!

Two Boards
 Two Layers
 Two Masks
 One Legend

Unmasked boards ship next day!

www.apcircuits.com



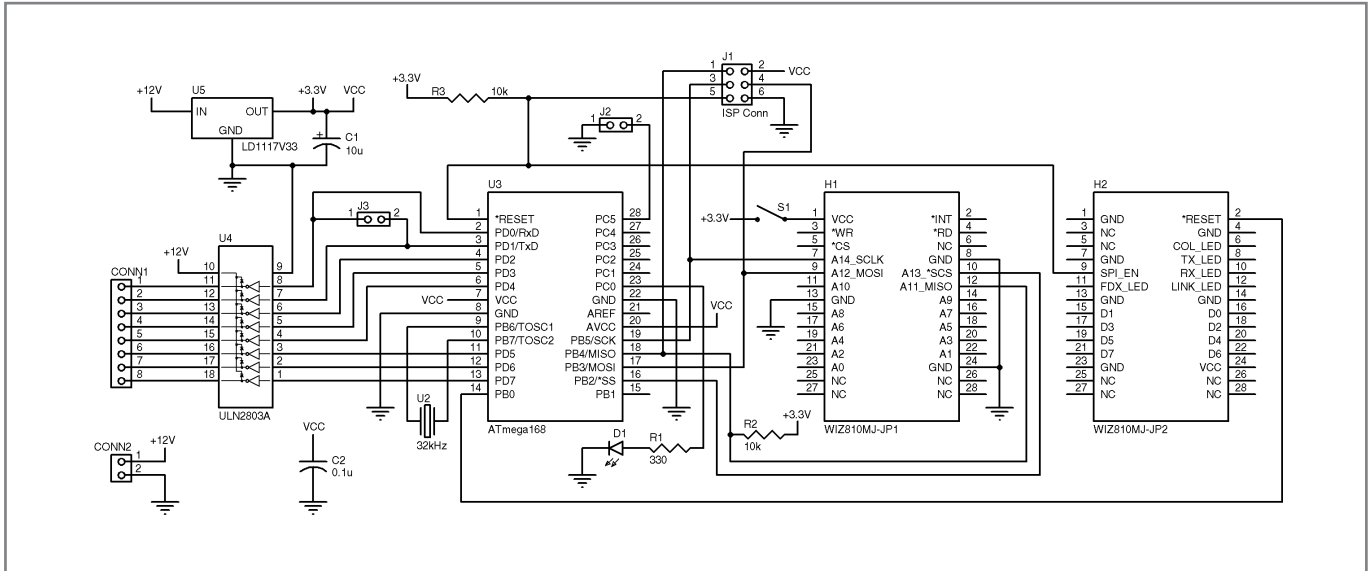


Figure 2—Connectors H1 and H2 are for the WIZ810MJ module. In SPI mode, few of the WIZ810MJ's pins need to be connected.

network as a better serial line. Conceptually, the ITAP has two boxes. Box one is a headless control unit that mounts in the garden and is hard-wired to zone solenoids. It also has an internal calendar and timer that run the watering schedule. Box

two is a fancy LCD and keyboard that plugs into box one whenever a change of schedule is required. Box two, of course, is really a laptop with a web browser.

The hardware box represents the headless control unit (see Figure 1).

A microcontroller maintains a data model in EEPROM that is used by the timer-control firmware to determine when to turn on and turn off solenoid water valves. A WIZ810MJ network module is used by the web server firmware to accept requests

Easy Embedded Linux

\$169
Qty 1

16MB FLASH / 32MB RAM
200Mhz Arm9 CPU
16 Digital I/O
Watchdog
10/100 Ethernet
Battery backed Clock/Calendar


Audio In/Out
2 USB
2 Serial Ports

*We brought you the world's easiest to use DOS controllers and now we've done it again with Linux. The **OmniFlash** controller comes preloaded with Linux and our development kit includes all the tools you need to get your project up and running fast.*

*Out-of-the-box kernel support for USB mass storage and 802.11b wireless, along with a fully integrated Clock/Calendar puts the **OmniFlash** ahead of the competition.*

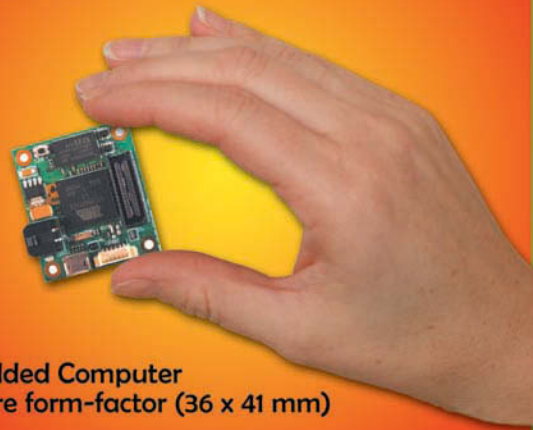
Call **530-297-6073** Email sales@jkmicro.com
On the web at www.jkmicro.com

JK microsystems



Embedded & Network Computing Technologies

Tiny, Light & Powerful All In One!



Embedded Computer
Tynycore form-factor (36 x 41 mm)

ATMEL AT91SAM9G20 @ 400MHz

256MB NAND Flash (8bits),
64MB SDRAM (32bits @ 133 MHz)
USB Device, Serial DBGU & 2 Expansion Ports.

www.calao-systems.com

Listing 1—This shell script compresses the HTML and JavaScript files as part of building the firmware image. The compressed files reside in the Atmel ATmega168's flash memory.

```
#!/bin/sh
FILES="error.html index.html edit.html common.js cal.html
day.html"
(
for f in $FILES; do
    id=`echo $f | tr '.' '_'`
    echo "prog_uchar $id[] = {"
    gzip -9 -v $f | xxd -i
    echo "};"
done
) > gz_data.inc

SCK1
__builtin_write_OSCCONL(0x40); //lock
```

from a standard web browser. The browser pages provide data entry and display capabilities of the watering schedule, as well as calendar-based planning tools. The entire design is self-contained. There are no extra files, scripts, or drivers that need to be installed on a PC.

An irrigation timer is not something you reprogram often. After an initial period of fine-tuning, the unit is expected to work, unattended, for the rest of the season. I did not need to add any daily reporting, only a minimal status page. This emphasizes the design point that the network connection is only there for reprogramming the device. Fancy reports and real-time status pages are engaging for the first few weeks of ownership, but then for most people the excitement of watching the grass grow begins to fade.

The long periods of time between reprogramming make it so you don't have to install any information on a PC. It is usually not a problem to pop in an installation CD when you first buy a product. Assuming there are no missing drivers or system conflicts, the first install is easy. The problem comes six months later, after you have upgraded your PC and cannot find the installation CD. For the same reason, it is important that all HTML pages live on the device

and not on a PC. If the device requires a PC installation, it stops being a stand-alone device and becomes yet another peripheral ready to break when you modify your PC.

HARDWARE

The hardware was designed to include the fewest number of parts. The essential parts are an Atmel ATmega168, a WIZ810MJ network module, and a ULN2803 Darlington array (see Figure 2). I do all of my development work in Linux. The excellent AVR toolchain available for Linux is one of the reasons I prefer Atmel microcontrollers. With Linux command line tools, I can do all of my firmware development at my desk, not hunched over the lab bench. An old laptop sits on the lab bench and serves as a network-to-USB gateway for the in-circuit programmer. I chose

the ATmega168 largely because of its 16 KB of flash memory and because it is easily available in a DIP package, simplifying prototyping. Timer 2 is used with an external 32-kHz crystal in real-time clock mode. The SPI master controls the WIZ810MJ. Other than a handful of I/O lines, none of the microcontroller's other features are used. Although it seems like a shame to use so few features, it would be pointless to complicate the design.

The WIZ810MJ is a network module that takes care of most of the complexity of adding TCP/IP networking to a design. In SPI mode, only a handful of the 56 pins are needed. The rest can be left as no connects. The version of the module that I worked with has a known problem in that it continues to drive the SPI lines even when its *SS is unasserted. The work-around is to drive the module pin SPI_EN low, which will free up the other SPI lines. The ATmega168 uses the SPI lines for serial programming, so I had to do the SPI_EN trick even though there are no other SPI devices. The WIZ810MJ draws a fair amount of current, so I did not want to leave it powered up continuously. I toyed with the idea of adding circuitry to power the module on and off. Because the network link LED status is available on a module pin, I could periodically power on the module, check for an active link, and turn it off. In the end, I just put in a toggle switch, which was crude but effective.

A ULN2803 is used as the solenoid driver. It is an eight-driver package with internal clamping diodes. The inputs are logic-level and were connected directly to port pins on the microcontroller. Each driver can sink 500 mA, which is ample for a typical irrigation solenoid valve drawing about 200 mA at 12 VDC. The solenoids are 24 VAC, but are quite content running on 12 VDC. Conveniently, inputs and outputs

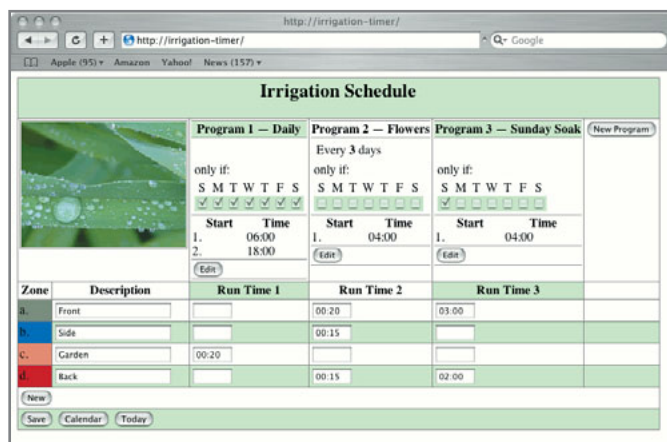


Photo 2—The irrigation schedule is the main browser page. Four solenoid valves are represented by zones a-d. The three different programs determine at which time and on which days the zones will be active.

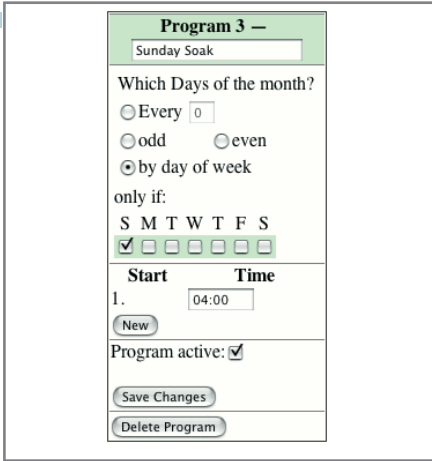


Photo 3—The program details on the main page are read-only. Clicking “edit” brings up an editable version.

are on opposite sides of the DIP package, making layout much easier on a single-sided board.

The only user feedback from the hardware are the WIZ810MJ’s two network status LEDs and an LED driven by the microcontroller. The microcontroller-driven LED blinks out either a hint at the IP address or an indication of solenoid status.

DATA MODEL

There are a number of ways to organize a watering schedule. Several vendors have agreed on a single model, which is the one used here. First, there are zones, which correspond to a physical run of pipe controlled by a solenoid valve. Then there are programs, which indicate which days of the week and at what time of the day to start an action. Make a table and put the zones in the rows and the programs in the columns. At the intersections, decide if and for how long a zone should be on for each program. This can get somewhat complex, but it provides a good deal of flexibility for things like long, slow soakings and for following community odd/even watering restrictions.

The ITAP’s internal data model consists of an array of program structures that keep all of the specifics for a

program, including start times and run length times, for each zone. All times are kept in minutes rather than hours and minutes. Internally, this simplifies storage and comparisons. A running count is kept of the day of the year in addition to the day of the month. The former is used for computing every N day cycles. The latter is used for odd/even days of the month.

SOFTWARE

The software divides into microcontroller support, network code, data model access, file storage, and timer logic. The microcontroller support code is minimal. The only interrupt is the Timer 2 overflow, which is configured as a 4-Hz RTC. Everything else is done by polled I/O. Every couple of seconds the main loop checks if the WIZ810MJ has been powered up. A simple read of the last byte of the IP address is performed. If the module is off, the byte will read as 0xFF. If the byte matches the expected value stored in EEPROM, the WIZnet module is assumed to be present and it is reinitialized.

The ATmega168’s PORT D connects pin for pin to the ULN2803 solenoid controller. Thus, pin zero of PORT D corresponds to solenoid zero, pin one to solenoid one, and so on. At each program step, there is the possibility that you might request all zones to switch on. To

limit current inrush problems, the software sets the values bit by bit with a small delay in between.

In a similar software rather than hardware role, the WIZ810MJ’s reset line is controlled by a port pin rather than RC logic. In addition to parts savings, this allows for a full part reset.

The network code is designed for the specific task of communicating with the one browser that it expects to find on a private network. The main loop polls the WIZnet sockets for a page request. Only HTTP GET requests are honored. The request can be for either a file stored in the flash memory file system or for a data page built on the fly. Changes to the data model are accomplished with arguments of the page request.

In a system where a browser request causes some action, such as writing EEPROM or switching on a sprinkler, minimize the chance that a user unintentionally repeats the action by clicking the REFRESH button. The method used here is to clean up the location line by returning a 307 REDIRECT result code. With no arguments to the page request, pressing REFRESH will simply reload the page with no additional actions.

When using the WIZ810MJ as a web server, there is a potential trap if the pages being served are complex. The W5100 can support a maximum of four simultaneous connections.

If the HTML page rendered by the browser has more than four subelements (e.g., <image> or <script> tags), the browser will likely issue simultaneous requests. If a WIZnet socket is in LISTEN state, it will accept a connection. If the browser attempts a connection for which there is no listening socket, the browser will get a connection-refused error. Depending on the browser, this could result in either a reported error or a noticeable delay before a retry. The workaround is to keep the pages simple, with few

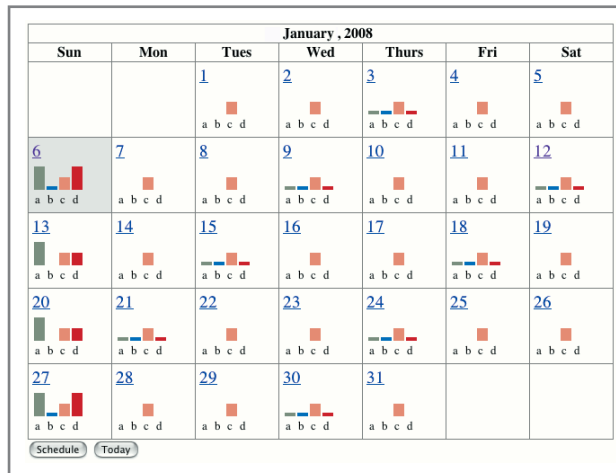


Photo 4—The calendar view shows what will happen this month. The bars in the bar graph correspond to the zones. The height of the bar shows the relative time that the zone will be on that day.

subelements. Newer chips, such as the WIZnet W5300, support up to eight sockets.

An early design trade-off in a project like this is deciding how much flash memory to devote to storing HTML page data. It would have been easier in many ways to just wire in a 1-GB SD flash memory card and fill it with the output from a web site builder tool. However, this would have negated the strong desire I had to show by example that an attractive, useful, and feature-complete UI could be implemented on a small-footprint device.

Accepting the limitations of finite storage, the question was how to stay as small as possible. Early in the design process, I considered keeping HTML templates in some sort of compressed form. The plan was to decompress the templates, fill in the actual values, and then serve this modified data to the browser. The final design is better in every way.

If the pages are not modified but instead have static content, they can be highly precompressed with the gzip utility and simply copied, unmodified, from flash memory to the network. No compression code is needed. The pages are just a payload. Modern browsers already accept gzipped data, so nothing additional is required.

Using JavaScript makes static content possible. The JavaScript language has constructs for dynamically building pretty much anything that can be done with HTML. All of the ITAP tables and forms are dynamically built. The data needed to fill in the tables and forms is formatted using JavaScript Object Notation (JSON) and resides in a separately loadable JavaScript file. The file is

built on the fly each time a server request is made for "pdata.js." In a larger system, data would likely be passed as XML. XML is a bulky format intended for data exchange between unrelated systems. XML really has no place in point-to-point micro applications. The nice thing about using JSON is that it is automatically parsed as it is read by the browser. Additional parsing is not required.

Another important benefit associated with putting all of the UI building code in JavaScript is that there is a clean separation between the UI view and the data model. The data model is maintained by the ITAP firmware. The firmware knows nothing of UI layout. A change to the UI does not require a firmware change.

COMPRESSED FILES

Looking through the WIZnet sample code, I noticed copyright notices around the code that accesses flash memory-based files. Thus, I thought it would be useful to show my standard tools approach to compiling files into flash memory (see Listing 1). This Linux shell script takes each file, compresses it, converts it to ASCII hexadecimal, and wraps the result in a data declaration ready for a C language `#include` statement.

USER INTERFACE

Photo 2 shows the ITAP's main schedule page. To keep the size small, only standard buttons and standard fonts were used. The schedule page is implemented as one large HTML FORM. Inside the FORM is a TABLE built dynamically with JavaScript. Only a small bit of HTML is used to define the basic




MACH64
PROGRAMMABLE LOGIC
STARTER KIT

Based on the Lattice ispMach 4064.

Includes 250 page lab manual.

Learn CPLDs the fun way with the MACH64! This complete kit comes with everything you need to take you from mystery to mastery with CPLDs and programmable logic. Learn to turn *software* into *hardware*!

www.XGAMESTATION.COM




PIC-SERVO
MOTION CONTROL

MOTION CONTROLLERS FOR BRUSH, BRUSHLESS AND STEPPER MOTORS.

- controller chips
- controller boards

www.picservo.com
JEFFREY KERR, LLC



THE SERIAL PORT LIVES!

Everything you need to know about COM ports, USB virtual COM ports, & asynchronous serial ports for embedded systems.

Hardware & software for RS-232 & RS-485. Wireless options and more.

Serial Port Complete
Second Edition
Jan Axelson

ISBN 978-1-931448-06-2 \$39.95
Lakeview Research LLC www.Lvr.com

From the author of USB Complete

prev **TODAY — January 6, 2008 6:06** next

Zone	Description	Set	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
a.	Front	On																								
b.	Side	off																								
c.	Garden	On																								
d.	Back	off																								

Schedule Calendar Today

Photo 5—Clicking on a specific day in the calendar view brings up this page, which shows when the zone will be active.

table and headings. Each page in the UI contains the same first line:

```
<script language="JavaScript"
  type="text/javascript"
  src="pdata.js"></script>
```

This code loads the current program and zone data. Based on the number of programs and zones, columns and rows are dynamically added and populated with their values. JavaScript functions take care of details like converting between minutes and hour and minute values. There are three programs and four zones in Photo 2. The empty fields indicate that the corresponding zone is off during that program.

To keep the main schedule page from getting too cluttered, only a summary of the program values is shown, and they are all read-only. Clicking an Edit button brings up a program edit page (see Photo 3). Both program tables are generated by the same JavaScript code. A flag tells the JavaScript to hide or desensitize certain fields in read-only mode.

The basic program interval can be set to either a fixed number of days or odd/even days. The check box by day of week is just an aid. It is no different than selecting a one-day interval. The days of the week check boxes are used to modify the basic interval. In an earlier design, I had two rows of check boxes, one for days when the program should run and a second for days when the program should not run. This was not necessary because a "not Friday" program is logically the same as selecting every day except Friday. There is always a trade-off between convenience and UI clutter. The Program active check box is used with the planning tools. To see what a particular program contributes to the overall totals, it can be temporarily disabled.

As you can see, there is plenty of flexibility, but also plenty of opportunity to make a mistake. The planning tools are a simple, but effective innovation that gives a glimpse into the future. Based on the currently active programs, the firmware runs

the clock forward to determine how much water each zone will get each day. This summary data is used to construct the calendar page (see Photo 4). Each day in the calendar contains a small bar chart. The letters and colors of each column correspond to a zone from the main schedule page. For simplicity, the height of the bar is limited to one of four discrete steps. The bar graph is not a GIF or JPEG image. Instead, it is implemented in JavaScript as a four-row table with variable ROWSPAN elements. As with the rest of the UI, the firmware reports only the data. The JavaScript makes all display and layout decisions.

The final page is the Day page, which drills down from a calendar bar chart to show when (during the day) the water will run (see Photo 5). Each tick mark corresponds to a 15-min. interval. The tick marks are constructed from TABLE rows with a small white border to highlight the individual tick. Together the Day and Calendar pages make for accurate "what if" planning.

I did not originally plan to have status view or zone on/off override buttons because I did not expect to have the network connected except during reprogramming. I later added these features mainly for debugging and

demonstration purposes. The status and override are implemented with the JavaScript XMLHttpRequest() facility, the cornerstone of AJAX.

IMPROVEMENTS

The ITAP was a fun project to build. It was much more of a software project than a hardware project. The design has proven to be simple to use and easy to explain to others. There is still a learning curve to understand what zones and programs are all about, but this knowledge is also required for any timer. The fact that there is no expensive LCD sitting idle in the garden is continuously comforting.

One serious drawback with the current design is having a static IP address for the ITAP. I did not want the ITAP to get its address by DHCP, because I expect to have a laptop plugged into the ITAP while standing in the garden. A reasonable alternative would be for the ITAP to play DHCP server for a laptop client.

WIZnet has released two additional modules, the WIZ830MJ and WIZ812MJ. I have not worked with either, but both appear to be better choices for new designs. The WIZ830MJ has the W5300 chip. The WIZ812MJ is a redesign of the WIZ810MJ. Both new modules have 2.54-mm headers. ☐

Editor's note: This project won First Place in the 2007 WIZnet iEther net Design contest. For more information about this design and the other winning projects, go to www.circuitcellar.com/wiznet.

Thomas Bereiter (itimer@micaview.com) has written software for everything from microcontrollers to huge distributed systems. He has a B.S. in computer science from the University of Illinois. Thomas currently designs prototype systems in Umbria, Italy.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/224.

SOURCES

ATmega168 Microcontroller
Atmel Corp. | www.atmel.com

WIZ810MJ Network module and W5100/5300 Ether net controller
WIZnet, Inc. | www.wiznet.co.kr/en/

Wireless Mobile Robotics

A Wi-Fi-Enabled System With a Mounted Webcam

Scott used a microcontroller, an embedded Ethernet board, and a wireless router in an innovative control system for a compact mobile robot. The robot features a mounted webcam that transmits real-time pictures to a remote laptop. Scott explains how he planned the project, assembled the pieces, and created the control software.

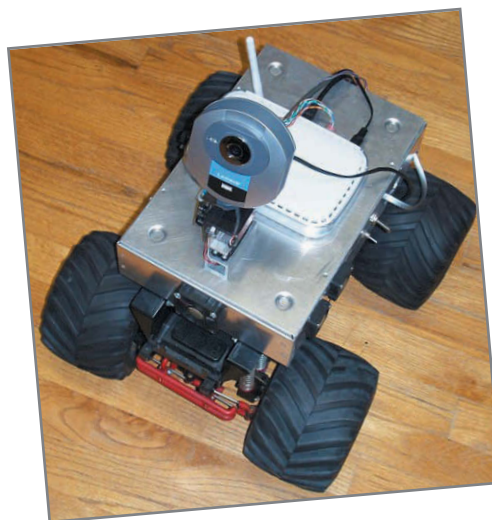


Photo 1—This is the assembled WiFi-PIC-Bot with a webcam. Pan and tilt servos are located just below the camera.

Robots are everywhere. They are used to build cars. They are sent to poke at rocks on Mars. Small rovers can vacuum your pool or your house, and there are several different models at the local toy store. I've played with RC cars, boats, planes, and helicopters for many years, but after seeing TV shows like *BattleBots* and *Junkyard Wars*, I had an itch to build something bigger and better to play with.

In 2001, I volunteered as a mentor for a FIRST Robotics Competition at Penn High School in Mishawaka, IN. It was an incredible experience. (If you like building mechanical devices, I recommend that you consider becoming a mentor.) The FIRST robots use controllers similar to standard RC types, which are capable of telemetry and feedback, but they are a bit pricey for the average designer. These days, however, you can pick up a Wi-Fi router for approximately \$30 and a used laptop for as little as \$100. The two of these, along with a fast Microchip Technology PIC microcontroller and a WIZnet embedded Ethernet board, make a nice platform for a robot project.

In this article, I will describe how I built a robotics system—which I call the “WiFi-PIC-Bot”—along with the control software and interface electronics (see [Photo 1](#)). I entered this project in the WIZnet iEthernet Design Contest 2007.

But since then, the robot has gone through a few upgrades. You can view the project as it was then (www.circuitcellar.com/wiznet/winners/DE/001106.html).

The WiFi-PIC-Bot is a modified RC ClodBuster (dual-motor, four-wheel drive with four-wheel steering) monster truck. I replaced the RC servo receiver with a WIZnet WIZ810MJ embedded Ethernet board controlled by a Microchip Technology PIC24FJ64GA002. The WIZnet board is connected to a Netgear WGR614 wireless router that transfers steering and throttle servo data as UDP packets back and forth to a remote laptop. The PC program reads a joystick, sends the servo commands out through its Wi-Fi card, and displays a real-time picture from a Linksys WVC54GC Internet-ready webcam mounted on the robot. One of the latest upgrades was pan and tilt servos for the Linksys camera.

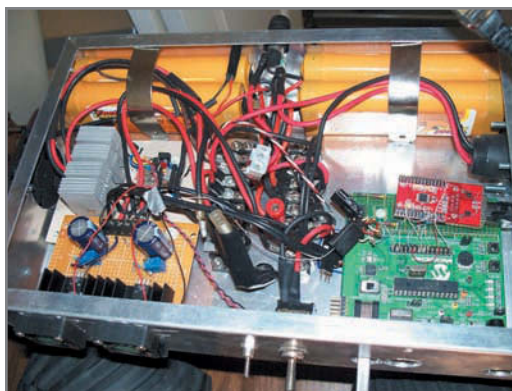


Photo 2—In the control box, battery packs are at the top, power supplies are at the bottom (left), and speed control is in the middle (left). A Microchip DM300027 development board with a WIZnet WIZ810MJ Ethernet board is in the lower right.

MODIFICATIONS

Making an RC monster truck into a robot isn't a simple task. Each issue had to be addressed separately. Many of the original design components had to be modified to

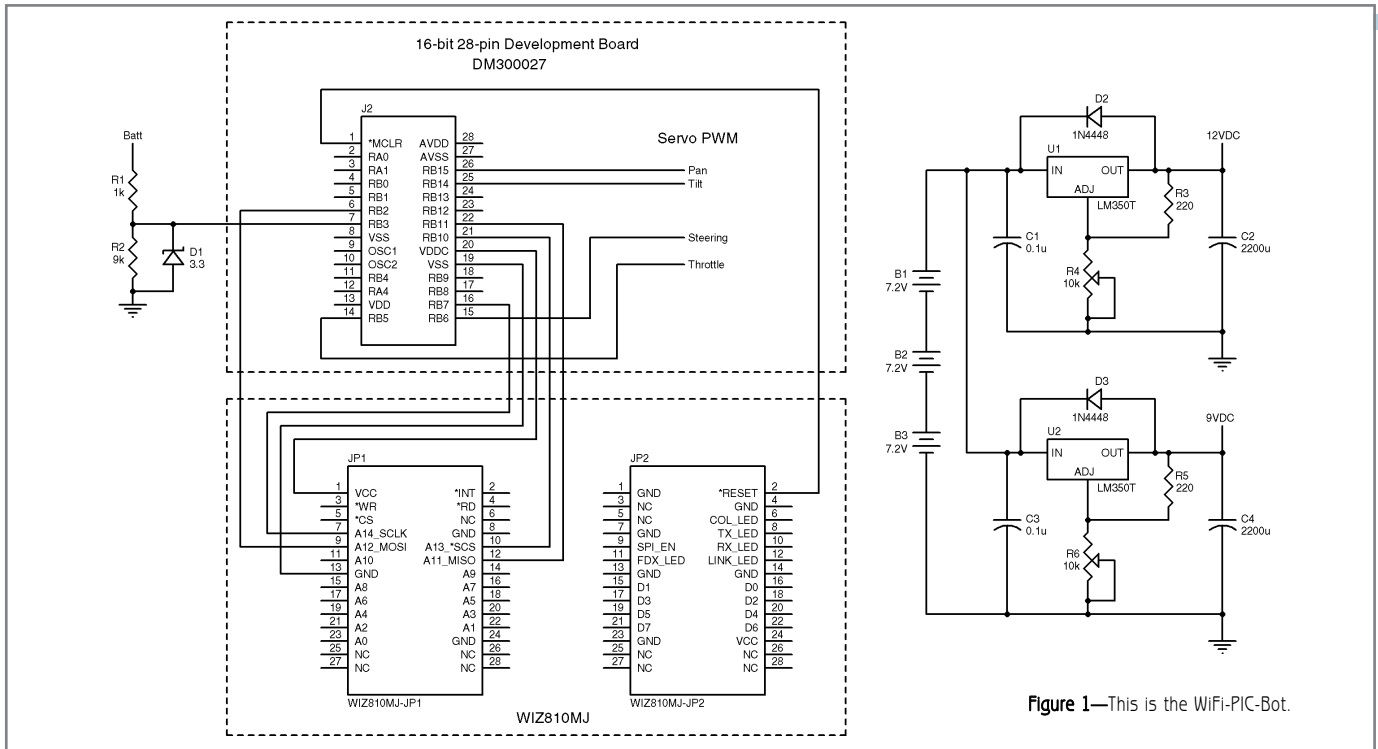


Figure 1—This is the WiFi-PIC-Bot.

make the toy into a reliable and sturdy robot platform. The ClodBuster's single 7.2-V battery wasn't sufficient to run the vehicle for more than a few minutes, and I couldn't use it for running the new processor, adding servos, the webcam, and the wireless router. The stock body and suspension system couldn't carry much weight. The original motor gears were fine for zipping around the yard, but were a bit too fast for indoor use. The stock motor control was just a four-position mechanical switch operated by a small servo. Slow speed was accomplished by switching a resistor in series with the motors. Each of these issues had to be addressed separately.

The first parts to go were the wimpy plastic body, the RC receiver, the mechanical speed control, and the stock battery packs. An aluminum 8" x 12" x 2.5" box fit nicely on top of

the frame. It had enough room inside for most of the electronics and batteries. I mounted power supply and speed control cooling fans, power LEDs, and switches to the outside of the box. I drilled for servo wire access, a programming connector, and network cable ports. I added a multi-terminal, high-amperage connector for on-system battery charging with bank selection. I attached the Internet camera with its pan and tilt servos along with the Wi-Fi router to the top of the box.

Both stock drive motors were already changed to high-performance models, so I left them as they were. I changed the transmission gears to the highest gear ratio I could find to slow down the robot for indoor use and to add torque for moving the extra weight around. I added hard rubber supports inside the springs to the suspension system shocks to accommodate the added weight of all the batteries and new electronics.

The main motor drive speed control had an interesting set of issues to resolve. Two parallel high-performance motors operating at high torque require a high-current controller with reverse, and I didn't have one. However, I had a Novak T-4 electronic speed control in my stash of RC parts and a collection of high-current MOSFETs. It was time for a bit of reverse engineering. Under the hood of the original T-4, the circuit had six small FETs in parallel for forward speed, and one FET for braking, with no reverse. After a day of head scratching and PCB probing with my oscilloscope,

Listing 1—This is servo control output compare 1 setup code.

```
// Output compare pin setup
RPOR3bits.RP6R = 18; // Make Pin RP6(RB6) OC1
OC1CONbits.OCTSEL = 0; // Use Timer 2 data
OC1CONbits.OCM2 = 1; // Use PWM mode
OC1CONbits.OCM1 = 1; // OCFA fault detection disabled
OC1CONbits.OCM0 = 0;
OC1R = 0x0400;
OC1RS = 0x0c00; // Set timer compare value for servo center position
IFS0bits.OC1IF = 0; // Clear interrupt flag
IEC0bits.OC1IE = 1; // Enable the interrupt

// Timer 2 setup
TMR2 = 0; // Clear the timer
PR2 = 0x9000; // Set the servo update time for 20 ms
T2CONbits.TCKPS1 = 0; // Set the timer prescale 1:8
T2CONbits.TCKPS0 = 1;
T2CONbits.T32 = 0; // Set the timer for 16 bit mode
IFS0bits.T2IF = 0; // Clear interrupt flag
IEC0bits.T2IE = 1; // Enable the interrupt
T2CONbits.TON = 1; // Turn the timer on
```

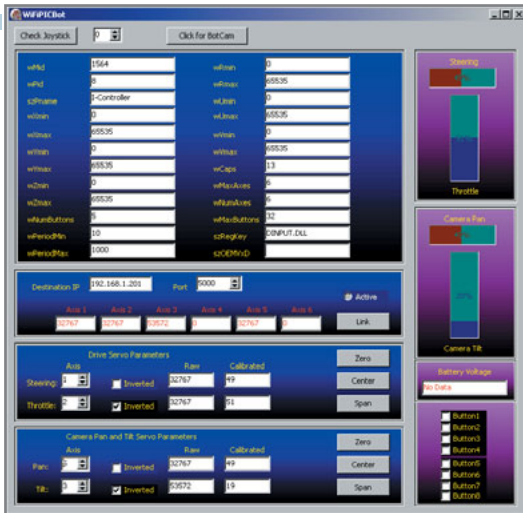


Photo 3—This is a PC control program joystick setup and calibration form.

I determined that I could remake this controller into a full-bridge speed controller without too much effort. I carved some custom heatsinks for the new International Rectifier IRFP048 MOSFETs from an old Pentium heatsink and added a cooling fan on the outside of the box to complete the new speed control system.

Three stacks of standard NiCd RC car packs power the robot. The first supply is for the main drive motors. It is a parallel stack of two 7.2-V packs. Those enable the robot to run for about 30 min. at slow speeds before needing a charge. The second supply is for the router and the PIC/WIZnet electronics. I didn't want the control electronics and motors to share the same battery packs for a reason. I knew the motor batteries would most likely be the first to run down, and I still wanted control over the servos at all times. If you've ever had an RC car or plane "run away" because the servo power went dead before the drive power, you know what I mean. I also didn't want to worry about conducted EMI from the motors into the other power supplies. The Netgear router requires 12 V, and the rest of the control electronics need 5 VDC. I used a stack of three 7.2-V NiCd packs in series with two LM350T regulator circuits to supply these voltages.

The addition of the webcam and two more servos was too much current draw for the 5-V supply because the camera alone requires 2 A. So, I

constructed an additional 5-V supply from two 9.6-V RC car NiCd packs and another LM350T regulator circuit similar to the one for the PIC24FJ64GA002. A smart charging station with temperature probes and maybe some Lithium polymer battery packs will have to go on my WiFi-PIC-Bot "upgrade-someday" list. **Photo 2** shows the inside of the control box with some of the battery packs, speed control, PIC24FJ64GA002 PCB, and WIZ810MJ PCB installed. Cooling fans are mounted on the sides of the box for the power supplies and speed control

circuits and run off of the 12-V supply. A switch for the speed control along with switches for the power supplies and fans are also mounted on the sides of the box. I left a connector inside to disconnect the motor supply batteries while I fiddle around with software and such so the robot won't drive off the table unexpectedly from a code error. A table-top robot hoist will also have to go on the to-do list. I left the original RC connectors on the speed control and steering servos so I could still drive the robot around with the RC transmitter and receiver and test the speed control by itself, if needed.

EMBEDDED CONTROLLER

The embedded Ethernet board, speed control, and servos for controlling

steering, camera pan, and tilt are wired to a Microchip DM300027 16-bit, 28-pin development board (see **Figure 1**). I replaced the original crystal with an 8-MHz crystal to gain some speed and installed a PIC24FJ64GA002 processor. The WIZnet board is connected to the DM300027 development board by wire wrapping the header connector pins directly. The two boards communicate via the SPI bus. RB10 is used as the slave select pin. RB5 and RB6 are the PWM outputs from OC1. OC2 is used for throttle and steering servo control. OC3 and OC4 are used for camera pan and tilt servo control. I also wired RB3 with a resistor divider (10:1) and a 3.3-V Zener clamp for a battery voltage monitor. The resistor divider is required because the three battery packs in series at full charge can be up to 27 V.

SOFTWARE

There are two separate software packages in this system. The first is the embedded code for the PIC, and the second is the UDP client on the external laptop. The RC servos operate from pulses produced by the PIC24FJ64GA002's timers. They are available on the output compare register pins. The published RC servo pulse width specification is 1 to 2 ms, although I've found that some servos need a larger range of pulse widths for control of their full span capability. If you send a pulse that is too small or too large, however, some servos will keep their internal motors on and

Listing 2—This is the main loop timer routine (executed every 250 ms).

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  LJoyEx.dwSize:= sizeof(LJoyEx);
  LJoyEx.dwFlags:= LJoyEx.dwFlags and JOY_RETURNALL;
  joygetpos(JoystickID,@LJoy);
  joyGetPosEx(JoystickID,@LJoyEx);
  UpdateJoystick;
  UpdateCalibration;
  UpdateTelemetry;
end;
```

Listing 3—This code is for servo axis data calibration, scaling, and inversion.

```
if checkbox9.Checked=True then
  SteeringPercent:=100-(trunc(SteeringValue/(StMax-StMin)*100))
else
  SteeringPercent:=trunc(SteeringValue/(StMax-StMin)*100);
```

eventually cause internal damage to the motor or gears, so keep that in mind when writing RC servo code. If

the servo is humming, but not moving, you may have gone too far. The servo control pulse must also be

repeated every 20 ms for the servo control IC to stay awake.

Controlling one servo is rather simple. But when you have several servos, the timer interrupts can conflict, so it's best to have one timer interrupt routine resetting all the others for the main 20-ms loop. An oscilloscope can be handy for checking pulse widths and timing. The PIC24FJ64GA002 has five internal 16-bit timers. Section 14 of the document, "PIC24 Family Reference Manual," shows all of the registers involved with the timers. This microcontroller has programmable I/O pins, so you must also tell the device what pin is to be used as what function during code initialization. The set-up code for one of the servo control output pins and the associated timer registers is in Listing 1.

WIZnet & Wi-Fi

Controlling the PICBot is achieved by sending UDP packets back and forth from a remote laptop to a WGR614 wireless router. The Linksys Internet camera and the WIZnet PCB

Listing 4—This is the UDP packet format and transmit routine.

```
if (UDPLinkActive=True) then
  begin
    UDPString[1]:= char(170); //10101010;
    UDPString[2]:= char(SteeringPercent);
    UDPString[3]:= chr(ThrottlePercent);
    UDPString[4]:= chr(PanPercent);
    UDPString[5]:= chr(TiltPercent);
    UDPString[6]:= char(85); //01010101
    IdUDPCliet1.Send(UDPString);
  end;
```

Listing 5—Receiving battery voltage telemetry from the robot.

```
if (UDPLinkActive = True) then
  begin
    VBATT_Value:=IdUDPCliet1.ReceiveString(IdTimeoutDefault);
    if (VBATT_Value[1]='V') then
      begin
        Vbyte1:=byte(VBATT_Value[6]);
        Vbyte2:=byte(VBATT_Value[7]);
        VBATT_Number:=3.3*((Vbyte1*256)+Vbyte2)/1024;
        VBATT.Text:=format('%2.1f VDC',[VBATT_Number]);
      end;
    end
  else
    VBATT.Text:='No Data';
  end;
```

USB WITH NO STRESS

DESIGNING OF ELECTRONIC DEVICES * CNC MACHINING(CAD/CAM) * COMPOUND OF ELECTRONIC COMPONENTS * DESIGNING AND MANUFACTURING OF PCB * CONTRACT MANUFACTURING SMT & THT * SOFTWARE DEVELOPMENT * EVALUATION BOARDS * KITS

PROPOX presents various series of low-cost integrated modules for data transmission via USB interface. Easy steps for projects that should be done in the short time. HOSTS, SLAVES, HUBS available.

FEATURES OF AVAILABLE SOLUTIONS

- * Single chip embedded USB host/slave controller **MMusbVNC1L**
- * Dual Port TX Buffer Dual Port RX Buffer **MMusb232RL**
- * USB 2.0 High Speed (480Mbps/Second) compatible **MMusbX232**
- * On-chip SIE and USB transceivers **MMusbSL811**
- * synchronous serial protocol (USB to JTAG, I2C, SPI or bit-bang) **MMusbX232**
- * 1.8V (chip core) and +3.3V I/O interfacing (+5V Tolerant) **MMusbX232**
- * Two independent USB 2.0 Low speed/Full speed USB ports **MMusbVNC1L**
- * Fast Opto-Isolated Serial Interface Mode option **MMusb2232**
- * Integrated Level Converter on UART interface and control signals **MMusb232**
- * Improved Power Management control for USB Bus Powered **MMusb232RL**
- * UHCI/OHCI/EHCI host controller compatible **MMusbX232**
- * USB Host/Slave with a standard microprocessor bus interface **MMusbSL811**
- * USB Dual Channel Serial/Parallel Ports **MMusb2232**
- * 300-3m Baud (TTL), 1M Baud (RS232), 3M Baud (RS422/RS485) **MMusb232**
- * Auto-address increment mode, saves memory READ/WRITE cycles **MMusbSL811**

USA (609)-323-7568 EUROPE (0048)-58-712-8058

www.proprox.com

PROPOX USB SOLUTIONS

Many ideas one solution

Factory on Your Fingertip

From Idea to Reality

- > Design
- > Prototyping
- > Production

See Our Differences in

- > Quality
- > Service
- > Price

- *Milling *Turning *Grinding *CNC
- *Wire Cutting *Laser *Plasma
- *Water Jet *Plastic Injection
- *Sheet Metal *Gear *SLA
- *FDM *SLS *LOM

CLICK TO MAKE

MachinePIER

Tel: 408-421-9840 Email: sales@machinepier.com
Website: http://www.machinepier.com

are connected to the router using short Cat 5 cables. The router is also used as the DNS server, even though the camera could operate as its own Wi-Fi server, if needed. The UDP datagram is arranged in the UDP Datagram.doc file posted on the *Circuit Cellar* FTP site.

When the WIZnet board receives a UDP message, the PIC24FJ64GA002 first checks byte 1 and byte 6 to determine if the packet is valid. It then adjusts each servo pin timer value corresponding to the associated PWM channel. The steering and throttle timer values give the correct range of servo movement span from 1,472 to 4,672 (0x05C0 to 0x1240 hex) for a total of 3,200 counts. So, the equation for the steering servo timer value is ((percent × 32) + zero offset) and the equivalent PIC24FJ64GA002 code would be $OC1RS = ((Rx_Buffer[1] \times 32) + 0x05C0)$.

PC SOFTWARE

The UDP client software for the laptop control is written in Borland Delphi Pascal. The code uses the Microsoft multimedia library routines in Mmsystem.dll to poll the joystick driver for available joysticks and their parameter tables. I have collected several joysticks over the years, and all of them worked for the robot until I added the camera pan and tilt servos. Not all joysticks have four axis controls; however, the controller for my RealFlight simulator has five axes and several switches with a convenient USB connector. I've found that the older analog joysticks were a bit noisy for robot control, and it's just about impossible to find a laptop with an analog game port on it these days. I haven't used any of the joystick switches to control anything yet, but I can imagine the next robot upgrade will have something to do with lasers or maybe some bottle rocket launchers or Nerf guns (safer for work).

Once a joystick is selected from the main set-up form, the controls for various servos are mapped to the output channels, inverted if

Listing 6—This code is used to open an Internet browser window to show the robot camera video.

```
if (panel4.Visible = False) then
begin
panel4.Visible :=True;
Webbrowser1.Visible := True;
Button2.Caption := 'Click for Setup';
OpenDialog1.Title:= 'Open BotCam HTML File:';
OpenDialog1.FileName:= 'capperbotcam2.html';
If OpenDialog1.Execute then
WebBrowser1.Navigate('file://'+ OpenDialog1.FileName)
else
WebBrowser1.Navigate('about:Error Opening File');
end
end;
```

required, and calculated/calibrated from the raw data to obtain a set of 0–100 percent values (see [Photo 3](#)). The destination IP address and port is selected to match the WIZnet configuration. When the Link button is activated, UDP datagrams are sent every 250 ms with all of the data to control the robot servos. At the same time, an HTML page is loaded into an integrated browser panel that polls the camera for a real-time video stream. The camera's IP address and video setup information is located in the HTML file.

In [Listing 2](#), the first four lines poll the Microsoft multimedia driver for the current status of all the potentiometers and switches on the selected joystick. The UpdateJoystick procedure then updates the raw data to each servo control variable and shows the status of the joystick buttons and switches. The raw joystick axis values are displayed with edit

boxes, and the status of the joystick switches are shown using checkboxes. The mapping of joystick axis to the robot control servo variable is done with up/down spin buttons and a case statement.

The UpdateCalibration procedure takes this raw data and changes it into percentage values for the UDP client (inverted if needed). If the client is active, it sends the data out over the network to the robot at the designated IP address and port (see [Listing 3](#) and [Listing 4](#)). The UpdateTelemetry procedure checks for any data sent back from the robot (currently just one battery voltage) and refreshes the edit box for that value (see [Listing 5](#)).

Just as important as having an oscilloscope handy for diagnosing electrical hardware issues, a good network analyzer is a must for watching packet data. Ethereal (now called Wireshark) is a GNU open-source protocol analyzer program that works great for watching every byte that slips into and out of your network ports.

The camera video is shown on a panel that pops up over the joystick settings panel when you press the Click for BotCam button at the top of the form (see [Photo 4](#)). A file-open dialog box is used to pick the HTML file to use for the camera panel (see [Listing 6](#)).

The HTML file downloads and runs a JavaScript file from the camera that streams video at a maximum of 30 frames per second at a maximum size of 640 × 480 pixels. The fun part is that I can



Photo 4—This is a runtime form with a PICBotCam video screen.

let anyone on the 'Net log in and watch the video while the robot is running. I just need to give a viewer the IP address and log-in credentials. I can also set the Wi-Fi security to keep hackers out.

AM I DONE YET?

No way! There's a never-ending list of gadgets and features for a "Future-Bot." Now that the robot has a camera, it needs some sort of VoIP to communicate with the people it encounters during its travels. Have I mentioned all of the control buttons that need to control something? A high-resolution (approximately 8 megapixels) still camera could be useful for getting a good shot of something the Internet camera finds interesting. Maybe I'll rent it out to the fire department for search-and-rescue operations in tiny crawl spaces. It could putz up and down the beach with an attached metal detector and tiny treasure scoop. I need a job at NASA so I can do this full time.

I hope you enjoyed reading about the PICBot. Have fun building your own. (I know you want one.) ☑

Scott Coppersmith (rscopper@aol.com) holds a BSEE from Michigan Technological University and is currently working as a senior engineer for Robert Bosch LLC. He also teaches evening classes at Ivy Tech Community College in South Bend, IN. Scott's hobbies include Tesla coils, fusors, lasers, embedded systems, and Delphi programming.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/224.

RESOURCE

Microchip Technology, Inc., "PIC24 Family Reference Manual," DS39704A, 2006.

SOURCES

IRFP048 MOSFETs

International Rectifier | www.irf.com

WVC54GC Wireless Internet video camera

Linksys | www.linksys.com

DM300027 Development board and PIC24FJ64GA002 microcontroller

Microchip Technology, Inc. | www.microchip.com

WGR614 Wireless router

Netgear | www.netgear.com

WIZ810MJ Network module

WIZnet, Inc. | <http://wiznet.co.kr/en/>

Expand Your I/O with Rabbit® RIO

Versatile Device with I/O Options

- Add functionality without costly processor changes
- Multiple Processor Interfaces
- Add 38 I/O
- Configure I/O for PWM, TRIAC, input capture, or decoder



Rabbit RIO™ Programmable I/O Application Kit for **\$299**

RABBIT™



Order Online At rsappkits.com

08033

 **Electronicstalk**

} Keep abreast of the latest news from your industry, delivered **free** to your desktop

Sign-up today at www.electronicstalk.com

With a library of more than 49,000 articles from more than 3,000 companies, we are the number one destination for people making purchasing decisions!

Electronicstalk matching buyers with sellers
www.electronicstalk.com

Web Camera Design

This versatile web camera system can take a picture at a resolution of 640 x 480 or 320 x 240, pan the camera horizontally and vertically, and change its IP and gateway address to match a network. After each photo is divided into 64-byte segments, an Ethernet module transmits the packets over the Internet.

You can use cameras for everything from recording celebrations to building surveillance. One of the most exciting new developments in camera technology is the webcam. At the heart of a webcam is a microcontroller that controls peripheral devices, such as the camera module (camera chip, lens, etc.) and the communications. Due to my interest in cameras, the Internet, and embedded technology, it made sense for me to design my own web camera (see [Photo 1](#)).

I built my web camera around a Microchip Technology dsPIC30F4013 16-bit microcontroller, a COMedia C328-7640 serial camera module, a WIZnet WIZ810MJ module, and two standard servos for rotating the camera

module (see [Figure 1](#)). In this article, I will describe how I did it.

NETWORK MODULE

The WIZ810MJ dictates the way the camera communicates with the other devices in the system (see [Figure 2](#)). It is a network module that includes a W5100 TCP/IP hardwired chip (including PHY) and mag-jack (RJ-45 with transformer) with other glue logic.

A network interface card (NIC) must have a unique MAC address. Where can you find a MAC address? You can buy 1,000 to 1,500 MAC addresses from the IEEE, but it can be expensive. Thus, the best option is to use

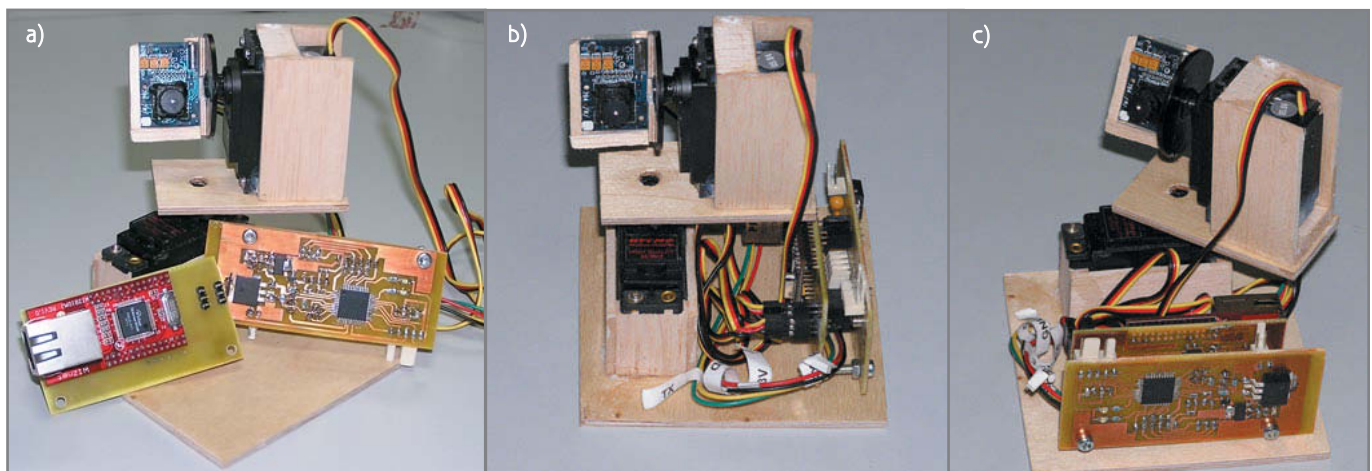


Photo 1a—This is the complete web camera design. **b**—The wiring is fairly simple. **c**—The design features two single-layer boards.

an old Ethernet card. I had one on hand, so I installed it in my PC to get its MAC and then I threw it away. As a result, my camera is recognized as an Intel NIC.

DHCP OR STATIC IP

In addition to a MAC address, the camera needs an IP address. IP addresses can be assigned statically or dynamically. Dynamic assignments are handled by a Dynamic Host Configuration Protocol (DHCP) server. For example, my home network gets IP addresses from the ADSL router's DHCP server, which uses the range of 10.0.0.xx to 10.0.0.137. The router itself has a static IP address of 10.0.0.138. However, if the camera had a dynamically-assigned IP address, none of the other machines would know what that address is and they wouldn't be able to communicate with the camera. Therefore, the camera has a statically-assigned IP address stored in the microcontroller's EEPROM at address 0x7FFC00. The default value for this address is 10.0.0.50 (defined in the source code) and the port is 50000. Of course, this IP address can be changed through the firmware.

Listing 1 is the code that fetches the stored IP address from the EEPROM. Each location in the EEPROM is 16 bits wide and can store 2 bytes. As you can see in Listing 1, the IP address, the SubNet mask and the gateway address are copied to `addr_param[]`, a global variable of type `char`.

Other Functions like `Send_a_UDP_Packet()` can access this array to get the IP address to include with the data that the WIZ810 will send to the host computer. Later in this article, I will describe the function that

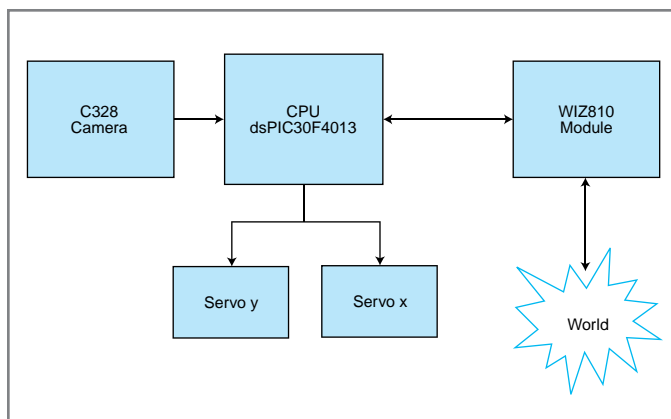


Figure 2—The design is fairly straightforward. A dsPIC30F4013 sits at the center of the design.

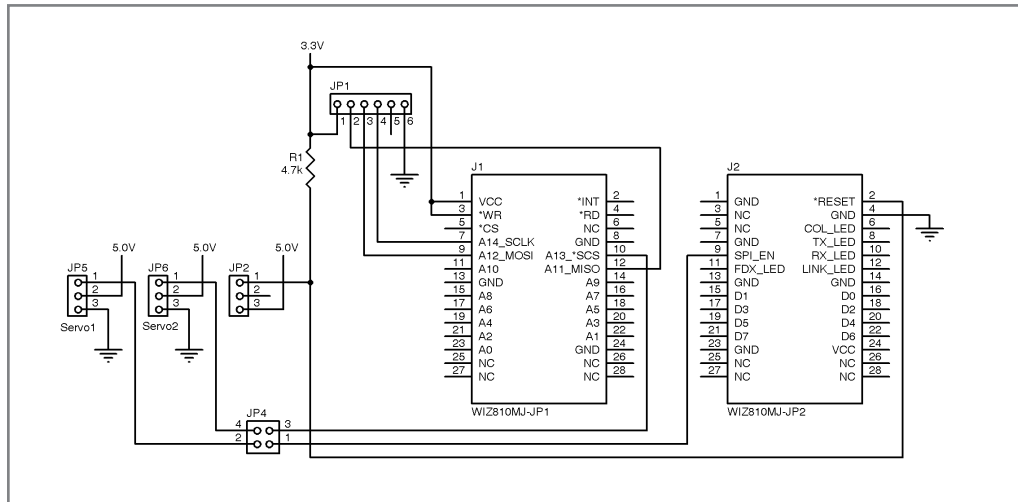


Figure 1—Here you see the WIZnet WIZ810MJ module and two standard servos that are used to rotate the camera module.

acquires these parameters and stores them in EEPROM.

TCP OR UDP?

This was my first project using embedded Ethernet. I had to decide between the the transmission control protocol (TCP) or the user datagram protocol (UDP). TCP establishes a connection in advance and delivers the data reliably and in the sequence it was sent. UDP features an unreliable connectionless datagram transmission structure. It processes data without establishing a connection. Therefore, lost or out-of-sequence packets are not hidden from the application. Also, there is no flow control, which means that packets can arrive faster than the recipient can process them.

I transmit photos with this system. So what if I lose one? I can catch the next one.

On one hand, today's computers are fast enough to process multiple applications at the same time. On the other hand, the UDP algorithm, as described in the W5100 datasheet, is simpler than TCP. That's why I chose the UDP protocol.

The WIZnet module can be interfaced via a parallel bus or via SPI. I chose SPI for its low pin count and simplicity. Although the dsPIC30F4013 microcontroller has two serial ports, I may develop future projects with microcontrollers with only one UART. However, UART1 on the '3014 shares pins with the SPI interface, so it is necessary to use a command like `U1MODEbits.ALTI0=1`; to move the UART1 function to alternate pins. I also issued a command for a 200-ms delay because it seemed to need some time to do the job! The SPI port is initialized in 8-bit master mode (see Listing 2).

A Microchip Technology TC2117-33 LDO regulator supplies the WIZ810 with 3.3 V. It draws 146 mA. I should also mention that the WIZ810 draws more current when the cable isn't connected. It gets really hot. All W5100 inputs are 5-V-tolerant, which means that they can be connected directly to the microcontroller. However, for compatibility, I reduced the microprocessor's

Listing 1—IP, Subnet, and Gateway static values are stored in EEPROM starting at location 0x7FFC00. For fast execution, cache all values to the microcontroller memory.

```
for (i=0;i<11;i++){
    Temp=Eeprom_Read(0x7FFC00+i);
    addr_param[i]=(Temp & 0xFF00) >>8;
    addr_param[i+1]=Temp & 0x00FF;
    i++;
}
```

output signal to 3.3 V with a resistor divider (see [Figure 3](#)). A 33-Ω resistor is connected in series to eliminate the SDI current. The SPI_EN and the *CS signals are connected to Port B's pins 11 and 12, respectively. The RESET signal is pulled high using a 2.2-kΩ resistor and connected to pin PB10.

CAMERA MODULE

The C328 camera module provides a serial interface (UART) and a JPEG compression engine (see [Figure 4](#)). The module consists of three main parts: an OmniVision Technologies OV7640/8 VGA color digital camera chip with an 8-bit YCbCr interface; an OV528 serial bridge, which is an embedded controller chip with a JPEG CODEC that can compress and then transfer image data from the camera chip to external devices; and a program

in EEPROM that provides a set of user-friendly commands for interfacing to external host. (This program supports 11 commands for interfacing to the host.)

RESOLUTION

The module can produce pictures in Normal mode (no compression) or Compressed mode (picture compressed with the JPEG algorithm). The maximum resolution in Compressed mode is 640 × 480 pixels. In Normal mode, the resolution is 160 × 120 pixels. For each picture, a total of 57,600 bytes (i.e., 160 × 120 × 3 bytes per pixel) must be transferred to the host.

In practice, the size of a compressed picture file with a resolution of 640 × 480 pixels rarely exceeds a total of 90 KB. For a picture taken inside a room, the file size is about 60 KB. For a half-size picture with a 320 × 240 resolution, the file size will be 30 KB or less. The bytes must be transferred with a slow UART interface at 57,600 bps.

VGA resolution is 640 × 480 pixels with 16 or 256 colors (the display standard for the PC). It was introduced in 1987 with IBM's PS/2 line and was popular in PCs with 14"

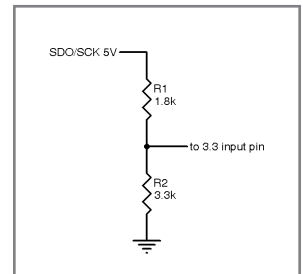


Figure 3—I reduced the microprocessor's output signal to 3.3 V with a resistor divider.

Flash Memory SUMMIT

"The NAND market has grown faster than any technology in the history of semiconductors."

— Jim Handy, Objective Analysis

Attend Flash Memory Summit for the latest practical information on flash memory and the most recent developments in flash memory applications.



Learn to make your products
**Fast, Rugged
and Mobile**
at the only conference
dedicated to flash memory!

**4th Annual Flash Memory
Summit & Exhibition**
August 11-13 2009
Santa Clara, California
FlashMemorySummit.com

Exhibit Space & Sponsorship Information:
Alan@FlashMemorySummit.com

or 15" monitors. Many new flat panels have a resolution of 1,280 × 1,024 or more. A picture with 640-pixel horizontal resolution covers about only half the monitor.

Modern desktop programming systems have functions that can resize an image. A picture with a 320 × 240 pixel resolution is balanced in quality and file size.

CAMERA INTERFACE

The host (a dsPIC30F4013 microcontroller) must initialize the C328 module after powering it up. Initialization involves transmitting the

Listing 2—In the dsPIC30F4013, UART1 and the SPI share the same bus. Force UART1 to use an alternative bus. The SPI is initialized in 8-bit master mode.

```
Uart1_Init(57600);
U1MODEbits.ALTIO = 1; // Clear the way for SPI
Delay_ms(200); // It needs some time to do the job
Spi_Init_Advanced(_SPI_MASTER, _SPI_8_BIT, _SPI_PRESCALE_SEC_2,
_SPI_PRESCALE_PRI_1, _SPI_SS_DISABLE,
_SPI_DATA_SAMPLE_MIDDLE, _SPI_CLK_IDLE_LOW, _SPI_IDLE_2_ACTIVE);
```

SYNC command (AA 0D 00 00 00) via the UART until the module sends an acknowledge command (ACK). An ACK command is usually received by the time the SYNC command is sent 60 times. A 10-ms delay

must be used between SYNC commands. The best synchronization occurs at 57,600 bps (see Listing 3). At a high speed of 115,200 bps, synchronization can't be achieved. Thus, the best for the host is to power off and on the module. The BSS22 transistor on the PCB acts as a switch to power on and off the module. At the speed of 57,600 bps, the module synchronizes at 60 SYNC commands and never fails. Thus, the firmware doesn't wait to receive the ACK command.

The C328 implements two different communication modes, depending on which command the host sends to get a snapshot picture

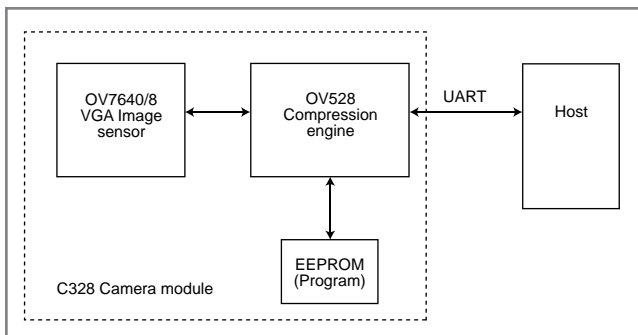


Figure 4—The C328 camera module includes a VGA color digital camera chip and an OV528 serial bridge. A UART is the intermediary between the module and host.

Cellular and GPS capable
Data Mover

- Flash file System
- 4 Chan 12-Bit A/D
- 1MB SRAM
- 512KB FLASH
- 4 Isolated Inputs
- 4 Hi Current Outputs
- 2 External RS-232
- 2 Internal RS-232
- Bat Backed Clk/Cal
- Cell Modem Option
- Internal GPS Option
- Metal Case Option
- 1ma Standby Option

It's easy and cost-effective to do mobile or solar-powered data collection and asset monitoring with the JK micro's Data Mover. With the ability to integrate a Cellular Modem, GPS and DOS-based embedded controller in a single rugged enclosure, you can capture and transmit your data quickly, easily and at low cost. Inexpensive development kits including Borland C/C++ and PowerBasic are available now. Call or email us for more details.

Call **530-297-6073** Email **sales@jkmicro.com**
 On the web at **www.jkmicro.com**

JK microsystems

WIRELESS MADE SIMPLE®
 BRING YOUR PRODUCT QUICKLY AND LEGALLY TO MARKET

RF Modules Add INSTANT wireless analog / digital capability to your product.

Low-Cost TX, RX & TRX Modules Multi-Channel Modules

Long-Range Modules

OEM Products FCC PRE-CERTIFIED & ready to customize for your application.

Handheld TXs Keyfob TXs Function Modules

Feature Products A closer look at Linx innovation

LOW-COST • LONG-RANGE TRANSCIVER

- Direct serial interface
- Low power consumption
- PLL-synthesized architecture
- RSSI and power-down functions
- Compact surface-mount package
- No external RF components (except antenna)

REMOTE CONTROL TRANSCODER IC

- Up to 8 inputs
- Bi-directional control
- Transmitter ID output
- Automatic confirmation
- Secure 2nd possible addresses
- Latched and/or momentary outputs

LINX TECHNOLOGIES **800-736-6677**
 159 Ort Lane · Merlin, OR 97532
www.linxtechnologies.com

Listing 3—The C328 module must synchronize at the host UART speed. The host must send a special packet several times. An LED connected at pin 13 blinks to indicate this.

```
void camera_connect(){
char i;
  for(i=0;i<70;i++){
    send(0xAA , 0x0D , 0 , 0 , 0 , 0 ); // Sending SYNC packets
    PortCbits.RC13 ^=1;
    delay_ms(10);
  }
  send(0xAA , 0x0E , 0x0D , 0 , 0 , 0 ); // Confirm SYNC with an ACK packet.
  Connected=1;
}
```

(uncompressed or a JPEG picture). In snapshot mode, the host sends a Get_Picture() command and the C328 replies with AA A0 01 xx yy zz (3 bytes hold the length of data following these bytes) and all the bytes in the picture's file. This means about 900 KB of data or 160 s for a picture with a resolution of 640 × 480. This mode is unacceptable because 640 × 480 × 3 bytes per pixel = 921,600 bytes × 10 bits/byte = 9,216,000 b/57,600 bps = 160 s per photo, or 2.67 min per photo.

In JPEG mode, the C328 uses the packet method. Before the Get_Picture() command, the host issues a command to determine a packet's number of bytes. The default value is 64 bytes long, and the maximum is 512 bytes. After some tests, I found that the best performance was achieved with the default values. Figure 5 illustrates this point, and the camera_snapshot() function implements it in code. As you can see, this function requests the picture data. The resolution must be set prior to this function. During the main procedure, the Get_A_Photo (char resolution) function is called to set the resolution. But first, it establishes the connection to the camera with the camera_connect() function. It sets the resolution with camera_setup(vgaResolution) and finally calls the camera_snapshot() function to get the data. camera_snapshot() is responsible for gathering the packet data and passing them to the array character packet[256]. After a packet is received from the camera, it is acknowledged, and the function passes the data to the WIZ810MJ's

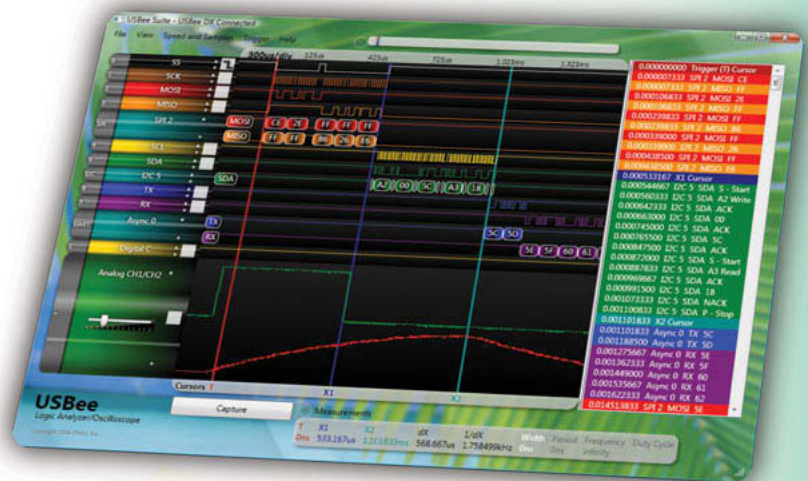
socket 0 buffer and does everything required to send it as a UDP packet. It transfers the value of remote_ip and remote port to the Socket 0's registers SO_DIPR and SO_DPORT, respectively. Subsequently, it calculates the start address of data and passes the values to the SO_TX_WRO and SO_TX_WR1 registers. Finally, the host issues a SO_CR_SEND command for the data to be sent to the output and clears the send flag to be ready for the next packet. At that point, an enhancement can be made to write the data directly to the

socket 0 buffer. To do so, the length of the packet must be set at 512 bytes long (maximum) and the UART speed must be 115 kbps to increase performance. The overall process will increase the frames captured by the host from three to five per minute to three to six per minute. One frame more

isn't important at this time.

On the PCB, there are two red LEDs. One is connected on pin RC13 and flashes on SYNC commands. The other is connected on RC14 and flashes once when the camera takes a picture.

The C328 requires 3.3 VDC to work and its I/O is not 5-V-tolerant. Therefore, a resistor divider (R3 and R5) at the dsPIC30F4013's UART2 TX pin is used to convert the 5-V signal to 3.3-V levels. A 33-Ω resistor (R4) was added to the dsPIC30F4013's UART2 Rx pin to



See It & Solve It

with
USBee Test Pods

- Logic Analyzers
- Oscilloscopes
- Signal Generators
- Protocol Analyzers
- I2C, SPI, ASYNC, CAN
- 1-Wire, PS/2, USB
- I2S, SMBus, Serial
- Configurable
- Programmable
- High Speed USB
- PC-Based
- And start at \$139

USBee[™]
USB based Electrical Engineer

www.USBee.com

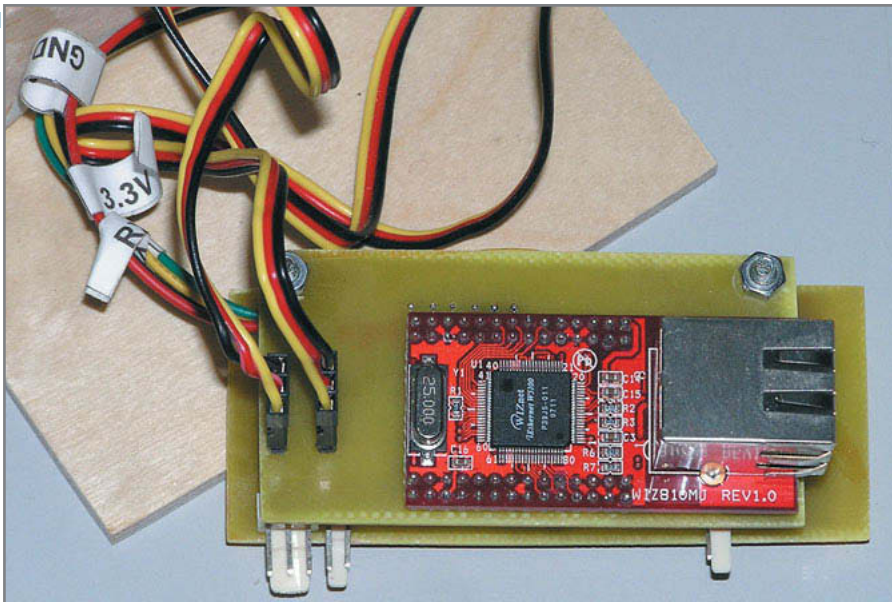


Photo 2—One board holds the dsPIC30F4013. The WIZ810MJ module is mounted on the other.

limit the current.

CAMERA ROTATION

Two standard servos—the second is attached to the shaft of the first servo—rotate the camera horizontally

and vertically. The dsPIC30F4013 communicates with servos via pulses. As the host, the dsPIC30F4013 generates a pulse of various lengths approximately every 20 ms.

The pulse's duration applied to the

control wire determines the angle of the shaft. This is called pulse width modulation (PWM). The servo expects to see a pulse every 20 ms or so. If the pulse spacing is greater than about 50 ms (manufacturer-dependent), the servo will enter Sleep mode in between pulses. It will move in small steps and the output will be jerky. The off time can vary. This has no adverse effects as long as its value is between approximately 10 to 30 ms. It is only the on time that determines the position of the output arm.

The pulse is normally between 1 and 2 ms long. The length of the pulse is used by the servo to determine the position to which it should rotate. Note that different servos will have different constraints on rotation. However, they all have a neutral position that's always around 1.5 ms (e.g., a Futaba S3003 servo's neutral position is 1,520 μ s and its maximum rotation is 1,900 μ s). When a pulse is sent to a servo that's less than 1.5 ms, the servo

Easy Embedded Linux

\$169
Qty 1

16MB FLASH / 32MB RAM

200Mhz Arm9 CPU

16 Digital I/O

Watchdog

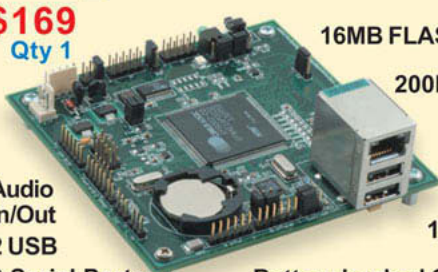
10/100 Ethernet

Battery backed Clock/Calendar

Audio In/Out

2 USB

2 Serial Ports



*We brought you the world's easiest to use DOS controllers and now we've done it again with Linux. The **OmniFlash** controller comes preloaded with Linux and our development kit includes all the tools you need to get your project up and running fast.*

*Out-of-the-box kernel support for USB mass storage and 802.11b wireless, along with a fully integrated Clock/Calendar puts the **OmniFlash** ahead of the competition.*

Call **530-297-6073** Email sales@jkmicro.com
On the web at www.jkmicro.com

JK microsystems



AP CIRCUITS

PCB Fabrication Since 1984

As low as...

\$9.95

each!

Two Boards

Two Layers

Two Masks

One Legend

Unmasked boards ship next day!

www.apcircuits.com







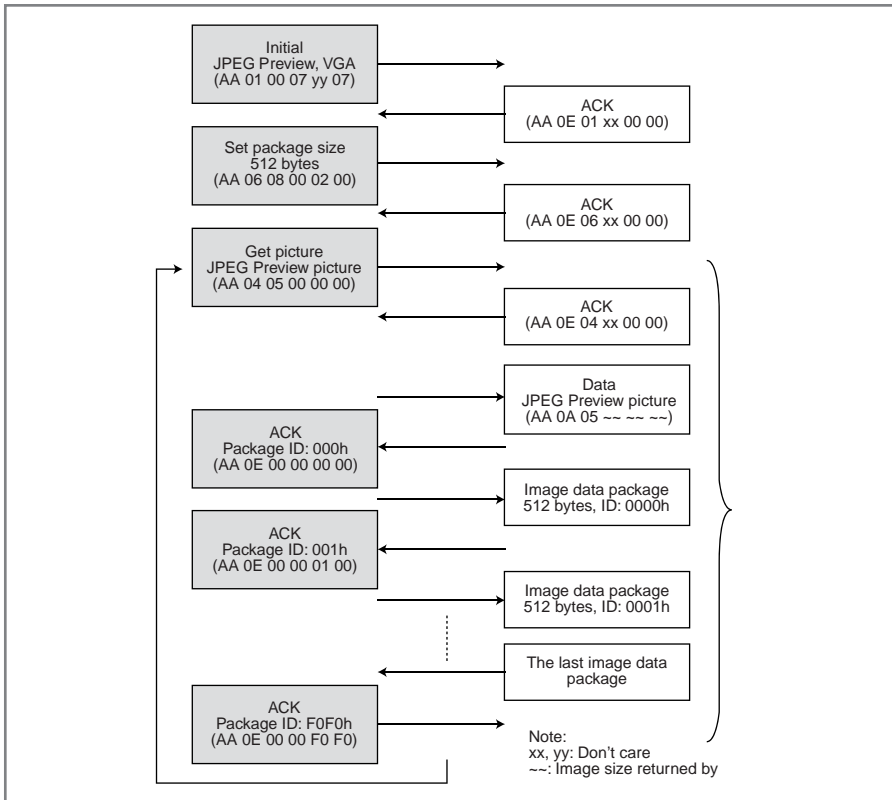


Figure 5—This is the command protocol to get a snapshot. The host is on the left. The host requests the image as packets of a known size. At the end, it acknowledges with a special packet. (Source: COMedia, "C328-7640 User Manual," 2005, www.comedia.com.hk)

rotates its output shaft a number of degrees counterclockwise from the neutral point and holds it there. When the pulse is wider than 1.5 ms, clockwise rotation occurs. Generally, the minimum pulse is about 1 ms wide, and the maximum pulse is 2 ms wide. Because of the hardware (e.g., motor and gears), the servo cannot rotate instantly to the instructed position with one pulse. The host has to issue some pulses to the servo until it reaches the final position.

The nominal supply voltage for the servo is 4.8 to 6.0 V at 7.2 to 8 mA. A Microchip Technology MCP1826S regulator supplies the entire device with 5.0 V at 1,000-mA maximum current. The supply voltage for the servos is settled at 5 V, directly connected to MCP1826S.

The dsPIC30F4013 drives the servos with pulses. This sounds like a good application for the comparator module of the microcontroller. The comparator module is driven by Timer2. The dsPIC has an 8-MHz crystal and the PLL enabled with a multiplier of four, giving a core clock

frequency of 32 MHz. Therefore, Timer2 is initialized through the

InitTimer2Interrupt() function with the following parameters: dual compare mode, continuous pulses output (OC1CON = 0x0005), rising edge start (OC1R) = 5, falling edge start (OC1RS) = 52. The register PR2 is set at 600. Changing the value of OC1RS affects the "on" time. This can be considered a duty cycle that can be changed at will. The value of 25 corresponds to 0 degrees, and the value of 85 to 180 degrees, with a resolution of 3 degrees per step. Channels 1 and 2 are initialized for the two servos.

HARDWARE

As you can see in [Photo 2](#), the design consists of two single-layer boards. One holds the dsPIC30F4013 and the second is piggy-backed to WIZ810MJ module. The main board is simple. It features a dsPIC30F4013 with an 8-MHz crystal, power supply circuitry, and headers for connecting the board holding the WIZ810MJ (see [Figure 6](#)).

A 5-V, 1-A MCP1826S regulator provides power to the system. A TC2117-3.3 linear regulator on the main board provides the 3.3 V

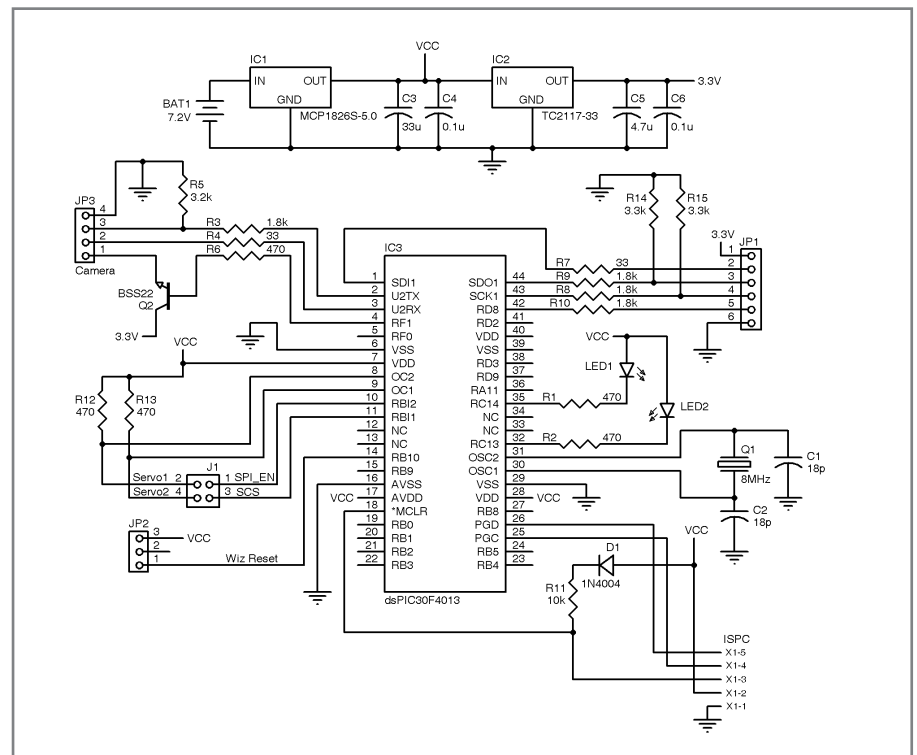


Figure 6—The processor PCB features a dsPIC30F4013, an 8-MHz crystal, power supply circuitry, and headers for connecting to the other PCB.

required by the WIZ810MJ module and the C328 camera. Thus, the WIZ810MJ module and the dsPIC30F4013 are 3.3-/5-V-tolerant, the C328 module is 3.3-V-only so level-shifting circuitry is required. I used level-shifting circuitry for both.

The host communicates with the WIZ810MJ via the SPI bus. It uses the UART port for the C328. A BSS22 transistor acts as a switch to power on and off the C328 camera module.

MAIN BOARD FIRMWARE

The main board's firmware (see Figure 7) was written in the C language. The MikroC compiler was used. The total size of the code is less than 6 KB. MikroC has an evaluation version that works fine with code less than 6 KB, so it is easy to experiment with.

I first read about the WIZ810MJ in Fred Eady's 2007 article "iEthernet Bootcamp: Get Started with the W5100" (*Circuit Cellar* 208). He introduces the W5100 and covers the topic of sockets. In an example, Eady uses the UDP protocol. After I made a decision about the protocol, I started looking for code. The well-written W5100 manual details the process of sending a UDP packet. Searching the Internet, I also found code for Atmel's micro-processors. I keep some header files from that code and the same alias for further reference.

The firmware isn't interrupt-driven. The main function looks for data that have arrived in the WIZ810MJ's Rx buffers by reading the Sn_RX_RSRx register. It loops until the arrival of data (see Listing 4). When data arrives, it passes them to a global parameter Packet[]. It moves the buffers pointer to the new location and writes a 0x04 to S0_IR to clear the Receive flag.

Action is taken according to the first byte in the packet. A switch statement

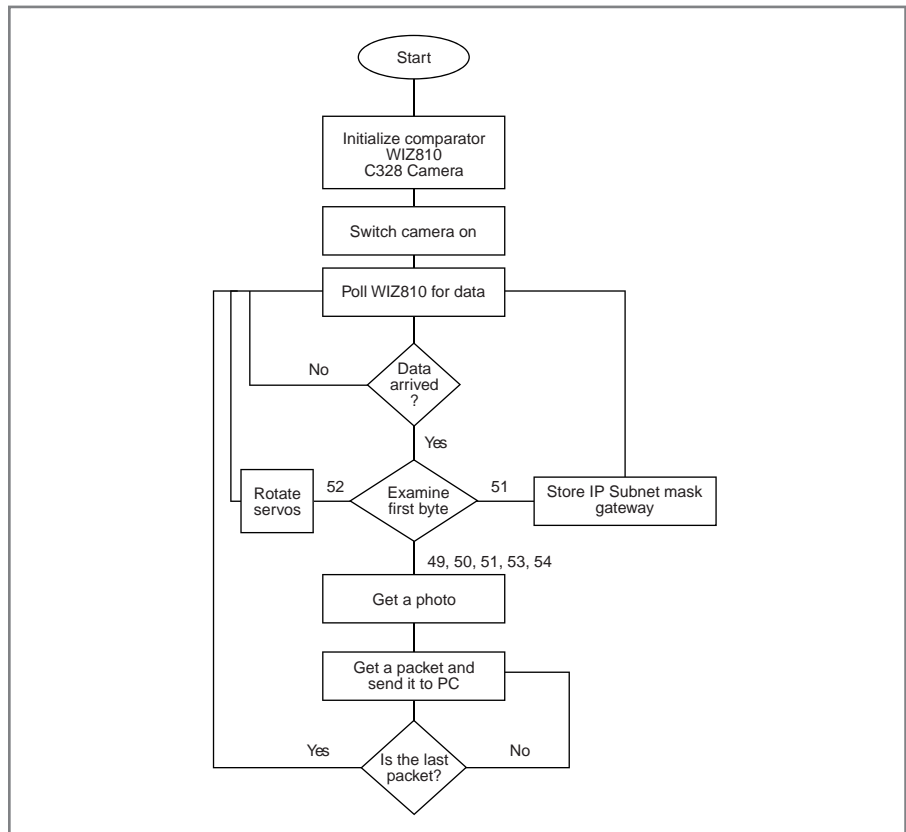


Figure 7—The firmware flow chart is fairly straightforward. It polls the WIZ810 for data. When data arrives, action is taken according to the first received byte.

takes care of this. There are six different cases, and among them are some that combine actions. For instance, case 53 (ASCII "5") means "Rotate camera then get a photo at 320 × 240 resolution." The PC client can change the web camera's IP Subnet mask value. Case 52 (ASCII "4") takes care of storing the new values to dsPIC30F4013 EEPROM.

I want to bring special attention to the SPI routine. When you try to use the W5100 in SPI mode, even if the *SCS is High, the W5100 (or WIZ810MJ) drives the MISO. To avoid doing so, the functions `wr_wiz_reg(char reg_data, unsigned int reg_addr)` and `rd_wiz_reg(unsigned int reg_addr)`

enable `SPI_EN` and then pull down the *CS signal. They write or read from the module and then they disable `SPI_EN` and pull up the *CS.

WEBCAM PROGRAM

Software is required for the desktop PC to communicate with the camera. A PC program was needed to display the photos on its screen. The language is Visual Basic version 6.0 and standard controls were used to enable everyone to experiment with the code.

It all begins with the PC. The client program is responsible for requesting a picture, collecting the packets and checking the photo's integrity, and then displaying it (see Photo 3). In the program's main window, there are two buttons marked 320 × 240 and 640 × 480. After pressing a button, the program sends a UDP packet using the control WinSoc. The packet consists only of 1 byte. This is the character "2" for button 320 × 240 and the character "1" for button 640 × 480. The WinSoc control uses the camera's static IP address and port number to send the packet.

Listing 4—The firmware polls the WIZ810 for data arrived. The Sn_RX_RSRx register holds the size of data arrived. If there is no data, then poll the WIZ810 again.

```

do{
    hi_byte = rd_wiz_reg(Sn_RX_RSR0(0));
    lo_byte = rd_wiz_reg(Sn_RX_RSR1(0));
    get_size = make16(hi_byte,lo_byte);
}while(get_size <=0x0000); // if no bytes received --> loop
  
```

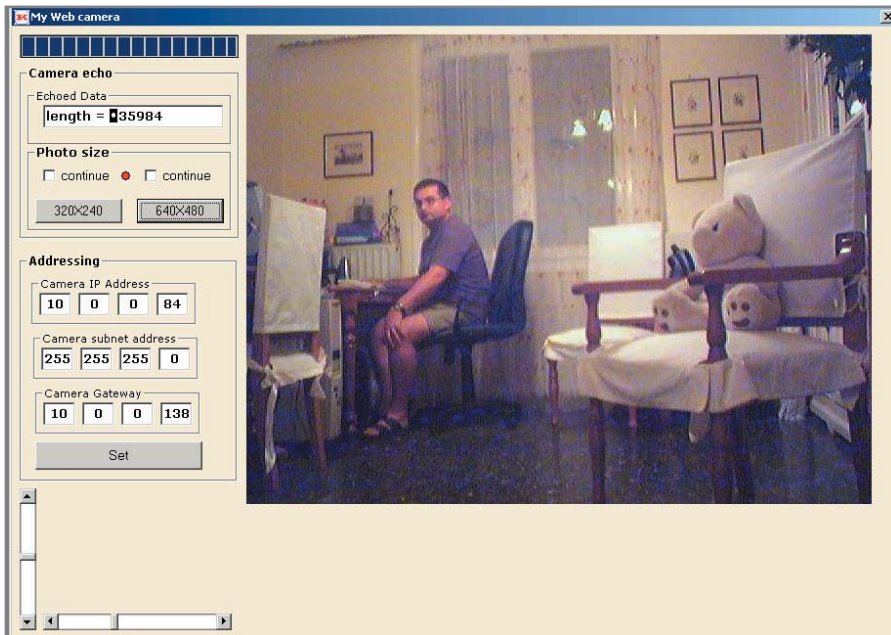


Photo 3—Here you see the client program's main window.

As I already mentioned, action is taken according to the first byte in the packet. A switch case takes care of this. A click on button “320 × 240” sends the character “2” (ASCII 50). The switch case calls the function to get and then send a photo in CCTV resolution.

The WinSoc control listens to the default port, 50000, and collects the data. The WinSoc control's DataArrival event will be raised when data arrives in the default port, 50000. To inform the program how many bytes it has to collect, the first packet includes the logo length=xxxx, where xxxx is the size of the picture. Every packet that arrives adds its bytes on a RAM buffer. When all bytes arrive at the buffer, a LoadPicture() function writes them to disk, and then loads the file to a picture control. The picture control is configured to double a picture's width and height. After that, the procedure checks to see if the corresponding “continue” box over the button is checked. If so, it issues a packet with the same character to the camera to get a new photo.

A progress bar on the top shows the progress of the received bytes. There is a Set button in the main window. By clicking this button, you send the camera a packet with all the values shown above. The packet is 13 bytes long. The sliders at the bottom left corner are used to rotate the camera. Every time you move the slider, the slider control fires

the change event and a packet with new values are transmitted to the camera.

IMPROVEMENTS

There's more work to be done on the firmware. It isn't interrupt-driven.

A new packet can arrive, but it has to wait until an entire photo is sent. This requires a new signal to be added on the PCB to connect the WIZ810MJ interrupt to the microcontroller.

The UART speed should be set to 115 kbps to improve the camera's connection. This requires the firmware to poll the UART RX for the proper answer before moving to the next step. Right now, this design operates with my home ADSL connection at 1024/128 kbps. I can send five frames per minute to my office computer. 📷

Minas Kalarakis (info@kalarakis.gr) holds a B.S. in marine communications from The Naval Marine School of Crete. He is a network administrator and computer technician for The Man Power Organization. Minas's main areas of interest are software and hardware development for embedded systems. In addition his interest in electronics, he enjoys flying RC model aircraft and cycling with his kids.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/228.

RESOURCES

F. Eady, “iEthernet Bootcamp: GetStarted with the W5100,” *Circuit Cellar* 208, 2007.

WIZnet, “WIZ810MJ Datasheet,” Ver.1.2, 2008, [www.wiznet.co.kr/en/pro02.php?&ss\[2\]=2&page=1&num=23](http://www.wiznet.co.kr/en/pro02.php?&ss[2]=2&page=1&num=23).

SOURCES

COMedia C328-7640 Serial camera module

COMedia | www.comedia.com.hk

Electronics123.com (distributor) | www.electronics123.com

dsPIC30F4013 Microcontroller, MCP1826S regulator, and TC2117-33 regulator

Microchip Technology, Inc. | www.microchip.com

C compiler

mikroElektronika | www.mikroe.com/en/compilers/mikroc/dspic/

OV528 Serial bridge and OV7640/8 VGA Color digital camera chip

OmniVision Technologies, Inc. | www.ovt.com

WIZ810MJ Ethernet module

WIZnet, Inc. | www.wiznet.co.kr/en

Internet Weather Display

The Internet Weather Display is a weather station that operates without external sensors. The design gathers weather data and alerts from the Internet and displays it on a color TFT monitor. An LED flashes when alerts are transmitted.

It seems like a weather station is one piece of equipment that every electronics enthusiast has to have. But for people who live in apartments, condominiums, or townhomes, mounting exterior sensors is typically difficult or prohibited. Even some single-family home neighborhoods have strict homeowner association rules against “unsightly” objects outside the home. My Internet Weather Display is a weather station you can operate without exterior sensors. The project gets its data from professional weather stations located in your neighborhood, most often at schools or other government buildings. Like a backyard weather station, it shows current conditions. An added bonus is that you get forecasts from professional meteorologists and alerts issued by the U.S. National Weather Service (NWS).

Photo 1 shows the project in action. The system retrieves weather data from the Internet and then displays it on

the 5” color TFT monitor. The simple user interface includes a push button to select current conditions, the forecast, and active alerts. The TFT monitor includes

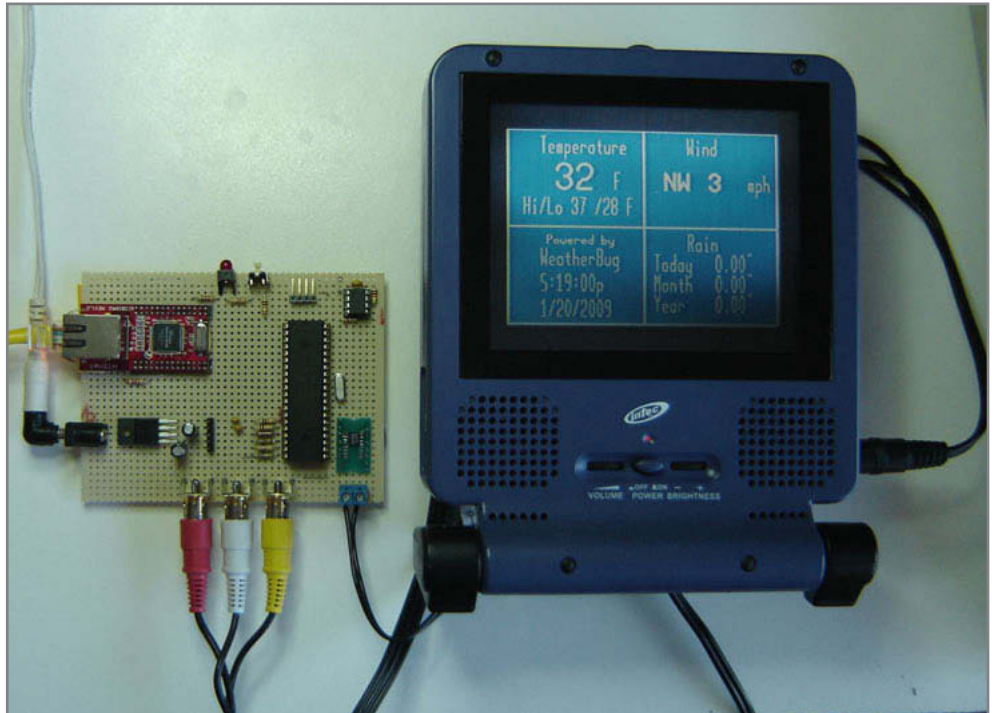


Photo 1—The Internet Weather Display gathers weather data from the Internet and presents it on a monitor. It displays current conditions, the forecast, and alerts issued by the U.S. National Weather Service.

built-in speakers for sounding an alert when new weather alerts are received. An LED also flashes while alerts are active to notify people who are hard of hearing.

WEATHERBUG API

During the past few years, several weather data providers have popped up on the Internet. WeatherBug is one such provider. It offers an application that resides in the task tray of your Windows PC and constantly shows the temperature of a local weather station. It can even alert you when an NWS message has been issued. While this application keeps you informed of the weather while you are actively using your PC, many people do not keep their PC fully powered on all day and night, so they could potentially miss an important weather alert. It's also inconvenient to wait the 30 seconds to 2 minutes for a PC to power-up just so you can check the forecast. A "real" weather station must remain on and be able to show wind direction immediately.

If you go to the WeatherBug website and get past the pages for the general consumer, you'll find the WeatherBug Labs page. There, you learn how to install WeatherBug on your Linux PC, cell phone, or personal webpage. A simple device like my Internet Weather Display has limited resources, so it needs a way to get just the raw data, and this is provided through a service called the "WeatherBug API." I recommend you review its terms of use. As users of the Windows PC application have seen, WeatherBug is supported by revenue from advertisements or through a yearly subscription. It does not charge for access to the API, and there are few restrictions if you use it for noncommercial

purposes. But if you plan on selling a device that uses the WeatherBug API, your device must be able to open links to the WeatherBug website and you may need to compensate WeatherBug with a portion of your revenue.

The first step in getting access to the API is to register on the WeatherBug Labs website. Upon successful registration, you are assigned a unique access code that must be included with the request messages that are sent to the server. The server watches how often your device requests data and may refuse to respond if your device is polling for data too often. Watch out for bugs in your code that may cause a loop to send out a request message even though your intent was a 1-minute polling period.

There are two message formats available: XML and "pipe delimited." The former is a bit more difficult to use because you must format headers and keep track of special strings of characters used to identify the data. The pipe-delimited format is simpler because it uses readable characters where the data is separated by the "|" (ASCII 0x7c) character. The Internet Weather Display uses the pipe-delimited format. Refer to the WeatherBug API website for the complete documentation.

To request weather information, a message like this is sent:

```
http://a111111111.isapi.wxbug.net/WxAlertISAPI/WxAlertIsapi.cgi?GetAlert60&Magic=160&ZipCode=80234&Units=0&RegNum=0&Version=7&t=1005&lv=0
```

Note the use of the HTTP high-level protocol. Only the "GET" command is necessary. As I already mentioned,

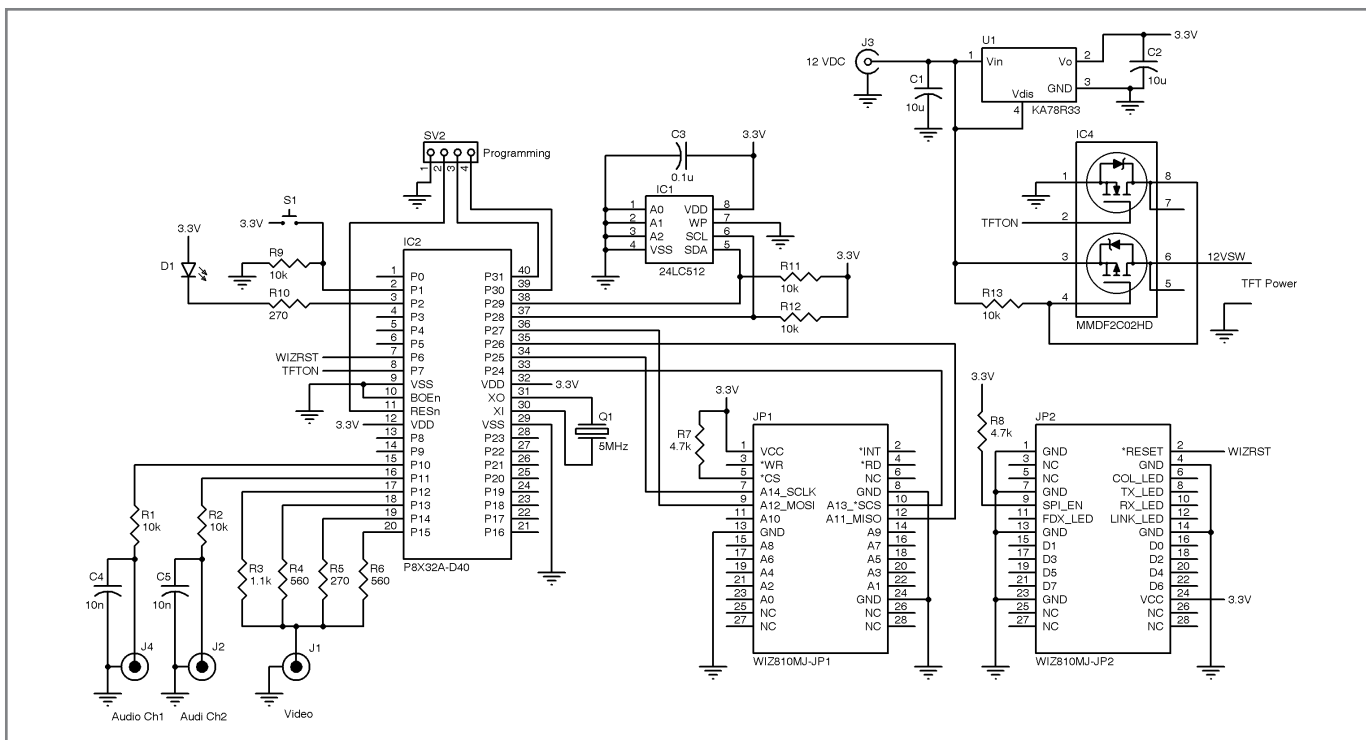


Figure 1—The Internet Weather Display uses a Parallax Propeller microcontroller to drive the video and audio interface. A WIZnet W5100 module handles all the Ethernet messaging up to the TCP/IP level.

the request message includes the unique code assigned to you when you registered at the WeatherBug API website. The number following "Magic=" identifies the data you're requesting. Use "10991" to request the current/live conditions, "10992" to request the two-day forecast, and "160" to request the active alerts. The three-day forecast and a list of weather station IDs are also available. The number after "ZipCode=" identifies the area for which you want weather information. The WeatherBug server will pick a station near the specified zip code. You could also specify "StationID=####," where "####" is a unique station number from the aforementioned list of station IDs.

The response is a stream of readable characters that includes HTTP protocol information followed by the weather data values. At the front of the weather data information is the "magic" ID. Verify this value to run the proper parsing routine. The data values that follow are separated by the "|" character. The following is an example of a response that contains alert information:

```
160|5|3|1|2|1197848582|2|3|11978486
42|3|2|1|1197848702|
```

To extract the data, use a simple string-parsing routine to get the characters delimited by the "|" character. To reduce the number of bytes that the server must send, some fields use a number value to index into a locally defined string table. For example, the response for the alerts message includes an "alert type" value. The text for the alert is stored in a table of strings in the Internet Weather Display's memory. The table is searched for the matching "alert type" number, and the text to display is extracted from the table.

VERSION 1.0 TO 2.0

I entered the first version of my Internet Weather Display project in the 2007 WIZnet iEthernet Design Contest. In that version, I used an NXP Semiconductors ARM processor

with a monochrome LCD. The "i" consumer devices have set a new standard in user interface, so a character or monochrome LCD just doesn't cut it any more. For this updated version, I went with a bit more color. Color TFTs are available, but few larger than 2.8" have a built-in controller, and I definitely wanted something bigger than 4". I also needed a microcontroller that could handle a large color display. The Parallax Propeller seemed to be a perfect fit for this application.

A while ago, I saw the ads for the Propeller microcontroller and thought, "Finally, something new in the area of microcontrollers." But because most of my projects used a microcontroller costing less than \$10, the Propeller's \$25 price tag at the time restricted my desire to learn more about it. Fortunately,

Parallax has since lowered the price to a more comfortable \$12, so I took another look and found it could easily generate the signals for either composite video or a VGA monitor. I decided to go with composite video because medium-size TFT monitors with composite video inputs are readily available for around \$55, thanks to the in-car entertainment market. I could even connect the project up to my HDTV and have my own "weather channel."

The Propeller P8X32A-40 microcontroller is laid out in much the same way as the demonstration board offered by Parallax (see Figure 1). The video signal is generated by three output pins that set up a resistor digital-to-analog converter. Two audio channels are used to create an attention-getting, warble-tone alert sound when new alerts are received.

Listing 1—The main loop handles the user interface and polls the WeatherBug server for updated weather data.

```
Set I/O
Initialize display driver
Initialize WIZnet W5100
Initialize variables

Main Loop (1ms)

  If button is pressed (debounce),
    If sleeping,
      Turn on the display
  If new alerts,
    Show alerts
  Else
    Show current/live
  Else
    Show next screen
  Reset alert sound
  Set flag to update screen
  Reset sleep timer

  If sleep timer expired
    Stop the display driver
    If no new alerts,
      Power off the display
      (We need power to the speakers for alert sound)

  If screen update needed & not sleeping
    Show current/live, forecast, or alert(s)
    Reset screen update flag

  If new alerts, sound alert

  If 5 min expired,
    Poll for weather data
    Set flag to update screen
    Check for new alerts
```

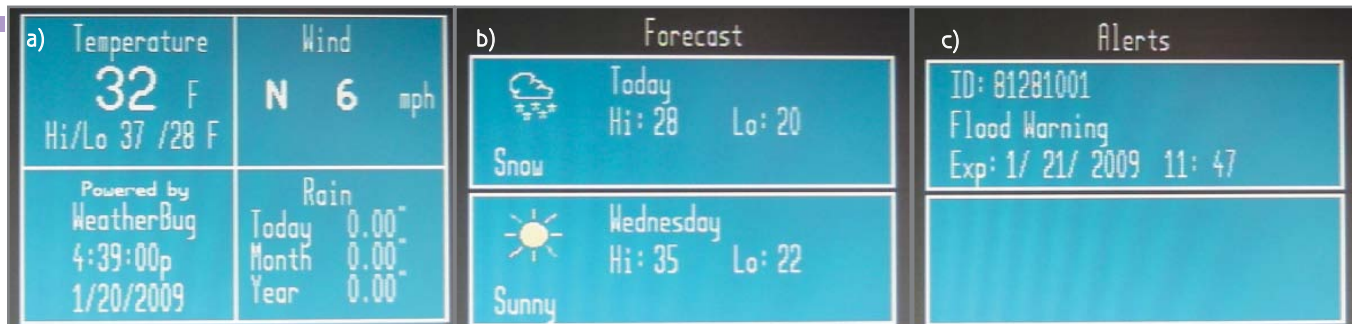


Photo 2a—The Current Conditions screen shows rainfall totals, temperature, and average wind direction and speed. **b**—The Forecast screen shows the high and low temperatures, as well as the expected conditions for the next two time periods. **c**—The Alerts screen displays active weather alerts and updates.

An LED flashes while there are active alerts, and a push button is used to select a new screen image. The program code is stored on the EEPROM. When the Propeller powers up, its resident bootloader copies the first 32-KB from that device into its internal RAM, starts an operating system driver called the “SPIN interpreter,” and then starts executing the program code from RAM.

The WIZnet W5100 Ethernet controller is the perfect companion chip for the Propeller microcontroller. Most other 32-bit microcontrollers have a version that includes integrated Ethernet. The Propeller does not; in fact, it doesn’t come with many peripherals at all. An external MAC+PHY could be used, but a TCP/IP stack would eat into the limited program memory. A better solution is to use the W5100 to handle all the message passing on the Ethernet up to the TCP/IP level. The W5100 chip has both a SPI and a parallel interface. I chose the SPI to keep the design simple, and because I didn’t need the high throughput rate available in the parallel interface.

The project was constructed using soldered wire connections on a perf-board. RCA connectors were used for video and audio lines just in case I wanted to try out different TV monitors. I could’ve used direct wire connections if the board had been mounted inside the same plastic case as the monitor.

The only required external connections were the Ethernet cable and power from a 12-VDC wall

transformer. If you use a small monitor like the one in Photo 1 and you can connect it to the same 12 VDC, be careful that you have the right size wall transformer. The Internet Weather Display board components require less than 100 mA, but my monitor required 500 mA, so a good choice was a 1-A transformer. If you’re using a monitor or TV with its own power source, a 250-mA transformer is sufficient. To add a bit of “green” to the project, the software turns the TFT monitor off after a sleep period using a MOSFET switch pair.

SOFTWARE

This project was the first time I used the Propeller, so I had a little bit of a learning curve to overcome. I had three options for code development: Parallax’s custom SPIN language, C language using an Image-Craft compiler, and assembly language. I chose the SPIN language because it appeared simple to learn and it didn’t cost me anything. Parallax provided the Propeller Tool IDE that enabled me to create the SPIN source code files and download them to the chip through a small USB programming adapter. The Propeller did not offer an emulator, nor did the chip have any on-chip debug, so I had to be a little creative while debugging. Luckily, the included library code that drove the composite video signal worked right away and I was able to use the TFT screen to watch values.

Listing 1 presents pseudocode for the main loop. If you press the push

button, the show variable is incremented to select a different display screen. Pressing the push button also stops the alert sound if it’s on. Every 5 minutes, the WeatherBug server is polled for the weather data. The alert information is checked to see if there are any new alerts. If there are, the alert sound is turned on. To be “green,” the display is turned off 20 s after the last button press. You may want to remove this feature if you use a standard television set instead of a small TFT monitor or if you want the display to be on all the time. Also, if you manually turn off the TV to save power, then connect the audio outputs to separate amplified speakers so that you’re alerted to new weather statements.

Photo 2 shows the three screens. The Current Conditions screen includes temperature, average wind direction and speed, and rainfall totals (see Photo 2a). The Forecast screen shows the high and low temperatures and expected conditions for the next two time periods (see Photo 2b). The Alerts screen shows active weather statements (see Photo 2c). The display can show only two alerts at a time. You can view additional alerts, if they exist, by pressing the push buttons. The screens are drawn using the Graphics library included with the Propeller Tool. For more information about how the Propeller microcontroller draws images, refer to Chris Cantrell’s article “Tile Graphics” (*Circuit Cellar* 209, 2007).

Listing 2 shows pseudocode for the steps needed to get data from a



Development Solutions for
ARM, 8051 & XE166
Microcontrollers

Microcontroller Development Kits

C and C++ Compilers

Royalty-Free RTX Kernel

µVision Device Database & IDE

µVision Debugger

Complete Device Simulation

Examples and Templates

Keil PK51, PK166,
and MDK-ARM
support more than 1,700
microcontrollers
www.keil.com/dd

RTOS and Middleware Components

RTX Kernel Source Code

TCPnet Networking Suite

Flash File System

USB Device Interface

CAN Interface

Examples and Templates

Keil RL-ARM and ARTX-166
highly optimised, royalty-free
middleware suites
www.keil.com/rtos

www.keil.com
1-800-348-8051

“ The W5100 does most of the ‘heavy lifting.’ It handles the entire Ethernet interface up to the TCP/IP level. It has a simple command interface to load data for sending outgoing data and for reading received data. ”

WeatherBug server. Since there are multiple servers and the IP addresses of those servers may change, the first step is to perform a domain name system (DNS) transaction. After the response is received, the WeatherBug server’s IP address is known, and the request messages for live/current conditions, forecast, and alerts can be sent. After each request is sent, we get the responses and save them to specific buffers.

The DNS transaction consists of sending a query message to a DNS server. The server then responds with a message that includes the IP address that must be used. In my

network setup, my DSL modem acts as a gateway to a DNS server out on the Internet. The query message is sent to my DSL modem, and then the modem forwards the message to a DNS server on the Internet. The modem also forwards the response back to my device. The project’s software extracts the WeatherBug server IP address from the response. To keep things simple, the DNS transaction is performed each time the weather data is updated. If you want a higher update rate, it would be better to parse the DNS response for the “time-to-live” parameter and perform only the DNS transaction after

Listing 2—This pseudocode shows the sequence needed to get weather data from the WeatherBug server. First, DNS must be used to obtain the IP address of one WeatherBug server. A request is made to the server to send back the current/live conditions, a two-period forecast, and a list of active alerts.

```
-DNS
Set destination IP to 192.168.0.1 (my DSL modem)
Set destination port to 53
Open UDP socket
Send DNS query message
Wait for response
Get the response
Parse for the WeatherBug server IP

-Connection Setup
Set destination IP to WeatherBug server IP
Set destination port 80
Open TCP socket

-Get Live/Current Weather Data
Connect
Send request for live data, HTTP “GET”
Wait for response
Get response
Parse for weather information, save to specific buffer
Disconnect

-Repeat above for Forecast

-Repeat above for Alerts

-Connection teardown
Close socket
```

Standards Make Sense

Standards improve quality and enable designers to share components across different projects. Today, ARM[®] Cortex™-M profile processors, combined with the Cortex Microcontroller Software Interface Standard (CMSIS) and optimized middleware from the industry's largest ecosystem, are setting the hardware and software standards for microcontrollers.

These standards enable leading vendors such as Luminary Micro, NXP, and STMicroelectronics to supply advanced microcontrollers, while maximizing code reuse across multiple platforms.

Cortex-M3 Microcontrollers Make Sense

"The strengths of ARM processor-based NXP microcontrollers are fundamentally changing digital products by combining ease-of-use with high connectivity and low power consumption."



Geoff Lees
Vice President and General Manager,
Microcontroller Product Line



For more information visit
www.onARM.com

ARM

The Architecture for the
Digital World[®]

© ARM Ltd. AD158 | 01.09

that time expires.

If you look at the code that's posted on the *Circuit Cellar* FTP site, you'll notice that little code is required to support the W5100. The W5100 does most of the "heavy lifting." It handles the entire Ethernet interface up to the TCP/IP level. It has a simple command interface to load data for sending outgoing data and for reading received data. I ported the W5100 driver code from the first version of the project without too much trouble. The Propeller's SPIN language is similar to C, but it has some interesting nuances, such as the strict use of indentation rather than braces to define statement blocks.

FEATURE CREEP

There are many ways to customize this project to your liking. First, there are other weather data providers. *The Weather Channel* provides data, but its terms of use are a bit strict. The NWS provides data with very few rules, but the interface is more complicated. The advantage of the NWS's data is that it is detailed, so you can see exactly how much snow is predicted and which direction a storm is moving.

Weather is just one type of information the project can display. Other data providers support news and stock updates. Be prepared to learn a new protocol like XML, SOAP, or RSS because most services don't provide simple data interfaces like WeatherBug's pipe delimited format.

My design uses a 64-KB EEPROM, where only 32 KB are used to support the Propeller's 32-KB RAM copy of the program code. You can use the remaining 32 KB to log data or store configuration values. You can also put additional devices on the I²C bus for more storage. In addition, you can use the EEPROM to store image data so that more RAM is available for program code. Images are currently used on the forecast screen and the raw data is combined with the program code.

One nice feature would be to show radar images in a loop. Although it

isn't part of the WeatherBug API, you could "get" the image from the WeatherBug web site, store the last four images locally, and set up the display to loop the images. You may want to go with a different microcontroller and TFT to support this feature. The radar images are usually in JPEG format, so you'd need the code to convert the image file to a bitmap. There are a few open-source solutions available, but the memory requirements are significantly greater. The Internet Weather Display's graphics capabilities are somewhat limited, so it might take a lot of work to get a clean radar image. Consider using a higher-resolution TFT with a digital interface.

The Internet Weather Display uses simple tones to alert you when new alarms are detected. The Propeller can generate complex sounds with its StereoSpatializer and VocalTrack libraries. You can use voice announcements or musical tunes if you want sounds that are more pleasing to hear. There are few things worse than waking up to a loud monotone beep during the middle of the night. You can modify the software to play different sounds based on the type of alert. Loud, attention-getting sounds can be used for warnings. A quiet, single "ding" sound could be used for advisories.

Do you want every feature? First off, step away from the "dark side," because you're starting to think like a person in a marketing department. The current version of the Propeller microcontroller has memory limitations, and my code uses just about every byte. The graphics library in Propeller Tool normally uses the double-buffering of a 12-KB image buffer. This project doesn't use animation, so I was able to use only a single buffer and I got back a good amount of memory for program space. There are other Propeller hardware platforms that support larger memory configurations by swapping program code between the RAM and external EEPROM as needed. Also, Parallax is currently working on the next version of the Propeller chip, and it will undoubtedly

include more RAM.

If you want to try and make this into a sellable product, make sure you check the data provider's terms of use. They will likely require some sort of compensation. You will also want to make the code much more robust by adding features such as the ability to select the WeatherBug data source, and the ability for the device to get its IP address using DHCP. All Ethernet devices must have a unique hardware/MAC address. The IEEE administers these addresses, and you can purchase a range. Note that if you use the project's code, both the MAC address and the WeatherBug unique identifier code have been set to illegal values. You must obtain your own unique values. If building a version for your own use, then perhaps use the MAC address from an old PC Ethernet card that's sitting in

your junk box. You'll get a unique WeatherBug code when you register at the WeatherBug Labs website.

GO SENSOR-FREE

The Internet Weather Display project enables everyone to have a weather station no matter where they live. Even if you don't face the same restrictions associated with mounting exterior sensors as some other users, this design may be better than a backyard weather station because you receive accurate, professional forecasts and NWS alerts. You don't have to worry about ideal sensor placement or the cost of maintaining and replacing the sensors. You can even modify the design to connect with other data providers to display important data such as news headlines and sound stock market alerts. 📧

Steven Nickels (ssea000@gmail.com) has a B.S. degree in electronic engineering technology from Minnesota State University, Mankato. He is a senior software engineer at Medtronic Navigation in Louisville, CO. Steven has not yet received any complaints from neighbors about the odd-looking equipment around his house.

PROJECT FILES

To download the code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/228.

RESOURCES

Parallax, Inc., "Propeller Manual," V1.01, www.parallax.com/dl/docs/prod/prop/WebPM-v1.01.pdf

WeatherBug, "WeatherBug API," Pipe Delimited Format, <http://weather.weatherbug.com/desktop-weather/api-documents.html>.

W5100 and WIZ810MJ Datasheets and information, WIZnet, www.wiznet.co.kr/en.

SOURCES

Propeller P8X32A-40 Microcontroller
Parallax, Inc. | www.parallax.com

WeatherBug Labs API
WeatherBug | <http://weather.weatherbug.com/labs.html>

W5100 Ethernet controller and WIZ810MJ network module
WIZnet Co. Inc. | www.wiznet.co.kr

Pololu
Robotics & Electronics

Robot Kits
Line followers
Robot arms
Hexapods
Chassis

3pi Robot
\$99.95
High-performance, C-programmable, ATmega328-based robot (with Arduino support!)

Mechanical Components
Motors, servos
Wheels, ball casters

Motion Control
Motor controllers
Servo controllers

Robot Controllers *with Arduino support!*
voltage regulator
power LED (green)
red user LED
ATmega48/328 microcontroller
dual H-bridge
trimmer pot
programming connector
20 MHz clock

Solder Paste Stencils
Use our low-cost solder paste stencils to quickly assemble your surface-mount designs.
From \$25

Custom Laser Cutting
From \$25
Cut your own custom chassis, front panels, and more!
1-877-7-POLOLU
www.pololu.com
6000 S. Eastern Ave. 12D, Las Vegas, NV 89119