A $25 Web Server

“See the world’s smallest web server. There are no age or height restrictions, but we’re talking real excitement. Step right up and see how these guys put together a stand-alone web server that includes an Ethernet controller, real-time networking kernel, TCP/IP stack, and a PCB smaller than a business card.

The recipe is easy. Take an Atmel single-chip microprocessor, hook it up to an off-the-shelf PC network card, add a little code, and presto—you’ve got the web server shown in Photo 1. This circuit is so simple that you can build it in an evening and add worldwide web access to your favorite embedded application.

The PicoWeb server provides web access to digital I/O and serial I/O signals without the need for assistance from external PCs or Unix computers. It’s a stand-alone device with a real-time networking kernel, a TCP/IP stack, and an HTTP web server. Plug the device into an Ethernet cable connected to the Internet, and you can control your sprinklers from any place on the planet.

This project started partly as an excuse to use a new microprocessor and partly to settle a long-standing argument about the possibility of delivering web pages with a commodity microcontroller. The Atmel AT90S8515 microprocessor looked exciting with its low-power RISC processor, 8 KB of flash program memory, 512 bytes of EEPROM, 512 bytes of RAM, 32 I/O lines, and a built-in UART. The realization that we could attach the Atmel part to an inexpensive PC ISA-bus network card with zero glue logic gave us a test vehicle to finally settle our argument (see Figure 1).

With an execution rate of one instruction per clock and a clock rate of 8 MHz, the AT90S8515 can transfer data over the ISA bus at full speed (about 2 MBps). All the hardware we needed to make a web server was quickly put into place. For debugging purposes, we hooked up the micro’s serial port to a cable with an RS-232–level converter embedded in the DE-9 connector’s hood. We also added a single LED for feedback.

After all that, we still had a little bit of money left in our $25 budget, so we threw in a $2 16-KB serial EEPROM chip to hold things like GIF and JPEG images (web pages need pictures, don’t they?). With all the hardware together, it was just a matter of programming!

NETWORK ADAPTER

We chose an inexpensive “NE2000-compatible” PC Ethernet ISA-bus adapter for our breadboard setup. We needed to find an ISA-bus Ethernet adapter that could be configured to not use plug-and-play mode. Typically, disabling plug-and-play mode is done with an MS-DOS diagnostic program supplied with the network adapter.

True NE2000-compatible adapters will work out-of-the-box with the Novell/Anthem NE2000-compatible device driver supplied with Windows 95. Source code for MS-DOS packet drivers for NE2000 Ethernet adapters have long been available on the web, making such cards an excellent choice for projects like this.

We used an SN2000CT card that we picked up at our local Fry’s Electronics store, but any true NE2000-compatible adapter will work if you can manage to turn off plug-and-play mode and set the adapter to a fixed I/O address. We
chose an I/O base address of 0x300 for our project. Because we are not using interrupts, it doesn’t matter how the card’s interrupt request (IRQ) is configured.

Those familiar with the PC/AT ISA-bus will note that a 16-bit ISA card can use up to 88 unique bus signals, excluding power and ground connections (see Table 1). The Atmel microcontroller only has 32 I/O lines, so how do you connect a 16-bit PC adapter card to the micro?

First, the network adapter card isn’t wired to all of the possible ISA-bus signals. We can easily determine which ones are used by looking at the copper traces coming from the circuit board’s connectors.

Second, the network card doesn’t need a number of other ISA-bus signals because of the mode in which we are using it (i.e., no DMA, no interrupts). And lastly, we can hardwire a number of the card’s input signals (i.e., to +5V or ground).

Because we aren’t using DMA or interrupts, we can ignore all of those signals, except AEN, which we wire to ground to indicate that DMA is not active. We need to hardwire *SMEMR to +5V to inhibit reads from the network card’s optional onboard ROM and hardwire the upper 15 ISA-bus address bits (SA5–SA19) to match the I/O base address to which the adapter has been configured (i.e., 0x300).

This arrangement leaves us with only 25 signals that need to be wired up to the Atmel microcontroller—the 16 bidirectional ISA-bus data lines (SD0–SD16), the remaining five ISA-bus address lines (SA0–SA4), RESET DRV, *IOW, and *IOR. (By adding a latch for the remaining five ISA-bus address bits, we could have shared those lines with the data bus signals, freeing up five more general-purpose I/O pins on the Atmel part.)

An NE2000-compatible Ethernet adapter is optimal because our processor is memory-limited. Such adapter cards have 16 KB of onboard SRAM—32× more than the Atmel microcontroller’s 512 bytes.

Part of the network adapter’s SRAM functions as a ring buffer to allow unattended reception of back-to-back Ethernet frames in case the controlling processor is busy doing other things. The rest of the Ethernet controller’s SRAM can be used to assemble transmitted Ethernet packets.

The net result is that very little of the Atmel microcontroller’s meager 512 bytes of SRAM is needed for the sending and receiving of Ethernet packets. After all, a single Ethernet packet can be as large as 1500 bytes! The real clincher is that this board can be purchased for as little as $8.88.

**FIRMWARE DESCRIPTION**

The PicoWeb server’s demonstration firmware supports a simple kernel, a tiny debug monitor, a p-code interpreter, a network adapter driver, a TCP/IP stack, and an HTTP server (i.e., web server). Let’s look at what kinds of network messages the PicoWeb server responds to and what the responses are.

At the lowest level, the PicoWeb server responds to network ARP requests identifying it as an active device on the network with an assigned IP Address. Every PicoWeb server is assigned a unique Ethernet address that is sent as part of the ARP reply packet.

Next, we’ll look at the BOOTP request. An IP address can be assigned to the device statically by storing the device’s IP address in the microcontroller’s flash memory or dynamically by using the BOOTP protocol.
protocol. When configured for
dynamic IP address assignment, the
PicoWeb server will begin sending
periodic BOOTP requests after
powerup until an appropriate reply is
received.

The PicoWeb server’s unique
Ethernet address is broadcast as part
of the BOOTP request packet. If a
valid BOOTP response is received,
the PicoWeb server uses the con-
tents of the response to set its IP
address. A Windows 95/98/NT
version of a BOOTP server program
is also available. This program uses
a text file that maps PicoWeb server
Ethernet addresses into assigned IP
addresses.

The PicoWeb server also re-
sponds to ICMP echo, or ping,
requests. The server responds to
ICMP echo requests by responding
with an echo reply. ICMP echo
requests are typically generated by
using a program called ping on a
remote host. The ping program
enables users to quickly test network
connectivity with the PicoWeb server
and evaluate the round trip time
(RTT) of the echo request/replies.

The PicoWeb server can send
and receive UDP packets. However,
the breadboard prototype’s demon-
stration firmware does not make use
of this capability.

At the TCP/IP level, the PicoWeb
server responds to HTTP GET
requests that are addressed to its IP
address. HTTP GET requests are
sent by web browsers such as
Netscape Communicator or Micro-
soft’s Internet Explorer. Our web
server responds to these requests
by sending back HTML documents,
text, images, and so on, just like a
“real” web server. The only difference
is that we don’t need a giant OS with
its attendant large memories, fancy
TCP/IP stacks, expensive micropro-
cessors, and high power consump-
tion to get the same results!

The demonstration firmware
purposely restricts the maximum
size of an HTTP GET response to a
single Ethernet packet (i.e., no more
than 1400 bytes of TCP/IP payload)
to conserve memory resources. In
the context of an embedded web
server using this class of low-cost
microcontroller, this restriction is not
an unreasonable tradeoff. A number
of HTML coding techniques can be
used to work within these limits,
including the use of HTML frames
and the “gluing together” of multiple
GIF and JPEG images using things
like HTML tables.

The demonstration firmware’s
basic response to an HTTP GET
request is shown in Photo 2. The
basic response is to return a web
page that shows a title, two radio
buttons, an update button, and a few
JPEG images. The two radio buttons
show the state of the LED on the
demonstration board.

The user can click the radio
buttons to change the state of the
LEDs (i.e., to on or off) and then
click the Set LED button to send the
new state information to the Pico-
Web server. The server responds by
setting the LEDs according to the
request and then updates the
returned web page to reflect the
current state of the LEDs. Those of
you familiar with server-side web
programming will recognize this as
the typical behavior of a cgi-bin script.

The images shown on the sample
web page are also supplied by the
PicoWeb server.

The demonstration firmware
contains a simple,
non-invasive debugger that
provides for things like
memory dumps, EEPROM
alteration, p-code
and network tracing
control, and more.

Debugger com-
mands such as
those in Table 2 can
be entered via the
serial port, or via
the network using a
web browser and a
URL that refer-

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>I/O CH CK</td>
</tr>
<tr>
<td>A2</td>
<td>SD7</td>
</tr>
<tr>
<td>A3</td>
<td>SD6</td>
</tr>
<tr>
<td>A4</td>
<td>SD5</td>
</tr>
<tr>
<td>A5</td>
<td>SD4</td>
</tr>
<tr>
<td>A6</td>
<td>SD3</td>
</tr>
<tr>
<td>A7</td>
<td>SD2</td>
</tr>
<tr>
<td>A8</td>
<td>SD1</td>
</tr>
<tr>
<td>A9</td>
<td>SD0</td>
</tr>
<tr>
<td>A10</td>
<td>I/O CH RDY</td>
</tr>
<tr>
<td>A11</td>
<td>AEN</td>
</tr>
<tr>
<td>A12</td>
<td>SA19</td>
</tr>
<tr>
<td>A13</td>
<td>SA18</td>
</tr>
<tr>
<td>A14</td>
<td>SA17</td>
</tr>
<tr>
<td>A15</td>
<td>SA16</td>
</tr>
<tr>
<td>A16</td>
<td>SA15</td>
</tr>
<tr>
<td>A17</td>
<td>SA14</td>
</tr>
<tr>
<td>A18</td>
<td>SA13</td>
</tr>
<tr>
<td>A19</td>
<td>SA12</td>
</tr>
<tr>
<td>A20</td>
<td>SA11</td>
</tr>
<tr>
<td>A21</td>
<td>SA10</td>
</tr>
<tr>
<td>A22</td>
<td>SA9</td>
</tr>
<tr>
<td>A23</td>
<td>SA8</td>
</tr>
<tr>
<td>A24</td>
<td>SA7</td>
</tr>
<tr>
<td>A25</td>
<td>SA6</td>
</tr>
<tr>
<td>A26</td>
<td>SA5</td>
</tr>
<tr>
<td>A27</td>
<td>SA4</td>
</tr>
<tr>
<td>A28</td>
<td>SA3</td>
</tr>
<tr>
<td>A29</td>
<td>SA2</td>
</tr>
<tr>
<td>A30</td>
<td>SA1</td>
</tr>
<tr>
<td>A31</td>
<td>SA0</td>
</tr>
</tbody>
</table>

Table 1—The PC/AT ISA-bus signals in red and green are required
by the NE2000 network adapter. The signals in green are connected
to the Atmel microcontroller and the signals in red are “hardwired”
to power and ground. We were able to connect the Atmel micro to the
ISA bus with no extra logic.
finds a special TCP port (i.e., port 911).

The format of a debugger command URL is http://
IPaddress:911/command[+
parameter1]+parameter2] Any results from executing a
debug command will be returned as a
web page. For example, http://
IPaddress:911/dm+60+80 will list
the contents of the first 128 bytes of
the microcontroller’s SRAM. New
debugger commands can easily be
added (or deleted to save program
code space).

The PicoWeb server’s prime
function is to return web pages and
images in response to HTTP GET
requests directed to URLs targeting
its HTTP server. The PicoWeb
server’s firmware responds to URL
queries directed to TCP port 80 in a
conventional manner. Because the
PicoWeb server doesn’t have a true
file system, URLs trigger dedicated
routines in the firmware as opposed
to simply returning the contents of a
disk file.

A summary of the standard URLs
implemented in the PicoWeb
server’s demonstration firmware are
shown in Table 3 (note that the
http://IPaddress part of the
URL does not appear in the table). Either http://IPaddress or
http://IPaddress/i00.html
will retrieve the default web page (home page) from the PicoWeb
server.

SOFTWARE DEVELOPMENT

You’re probably wondering: “How
does all of this fit in an 8-KB micro-
processor?” The answer: “Very
carefully!” We wrote the software to
efficiently implement the necessary
network protocol layers by using a p-
code technique to conserve code
space in exchange for somewhat
reduced execution speed.

Time-critical network code is left
in native Atmel RISC code. As a
result, the actual space used by the
PicoWeb server demo firmware is
under 7000 bytes, including debug-
ging code. Therefore, space remains
for developers to roll their own code and add even more functionality to
the PicoWeb server.

Because p-code can be run from
the serial EEPROM chip (at a greatly
reduced execution rate), a substan-
tial amount of added functionality is
possible. The present development
environment allows custom applica-
tion development without the need
for access to the underlying real-time
networking kernel source code. The
source code for the PicoWeb server
real-time networking kernel is
available for license by serious
developers.

The software development envi-
ronment used for the project made
use of the Atmel free assembler,
which can be downloaded from
Atmel’s web site.

To enhance the capabilities of the
assembler, a Windows version of the
GNU C preprocessor was used to
drop certain macro/
include file capa-
bilities that were
conspicuously missing from the
Atmel product. Atmel sells a $49
development kit for
the AT90S8515 that enables users to
quickly get up to speed and down-
load programs into
the AT90S8515’s

flash memory.

The AT90S8515 processor
allows in-circuit programming of its
flash memory via a four-wire SPI
interface. To eliminate the need for
the $49 development kit, a C pro-
gram was written to enable a PC to
program the Atmel microprocessor
on our breadboard in-circuit via a
cable attached to a PC parallel port.

The HTML-like code in Listing 1
displays a web page that provides
the status of the breadboard’s
onboard LED and provides an HTML
form that enables the user to control
the LED.

Special tags beginning with a
back-tick (‘) are embedded in
standard HTML code. These tags
invoke firmware routines when a
web page containing them is returned
to the requestor. The tags can be
used to dynamically insert variable
data and text into a web page when
referenced. Table 4 shows a
number of the tags implemented in
the PicoWeb server demonstration
firmware.

The tag `t outputs a standard
HTML text header. The tag `000
outputs the string input type=
and is used as an example of how to
save EEPROM storage space. `001
is an example of a dynamic tag. It is
part of a kind of “if-then-else”
construct. In Listing 1, if the value of
output bit 1 is zero (i.e., the LED is
on), the first radio button on the web
page will include the CHECKED
string; otherwise the string is omit-
ted. If the output bit 1 is one (i.e.,
LED is off), then the second ratio
button will include the CHECKED

Table 2—Debugger commands like these can be sent over the network using a web
browser or via the PicoWeb’s serial port.
option to execute program code out of EEPROM (including external serial EEPROM), and in many cases, a reduction in program code size when compared to native code.

Code simplification is achieved because the p-code makes no reference to native registers, and because the p-code “virtual machine” uses 16-bit-wide data types for most operands. P-code execution from EEPROM is possible because the p-code instruction pointer is 16 bits wide, with the uppermost bit indicating (when set) that the next p-code instruction should be fetched from EEPROM and not from the Atmel microcontroller’s 8-KB flash program memory. Program size reduction results from a combination of factors, including efficient application-specific p-code routines and flexible p-code operand addressing modes.

A description of the many p-code routines is beyond the scope of this article, but user p-code documentation is supplied with the board.

The build procedure is controlled by a simple batch file.

Atmel's PicoWeb server firmware routines can be activated by referencing special URLs. This arrangement provides simple remote control of the Atmel micro’s digital I/O lines.

Table 3—Certain PicoWeb server firmware routines can be activated by referencing special URLs. This arrangement provides simple remote control of the Atmel micro's digital I/O lines.

<table>
<thead>
<tr>
<th>URL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Return document i00.</td>
</tr>
<tr>
<td>/iihh</td>
<td>Return document number hh to user. Documents numbers are two-digit hex values. Anything after hh in the URL is ignored.</td>
</tr>
<tr>
<td>/uihh</td>
<td>Call firmware routine number hh. Mostly useful for testing “html include” routines.</td>
</tr>
<tr>
<td>/x%n-value</td>
<td>Set digital I/O port bit n to value where value is either 0 or 1. Because the breadboard LED's anode is connected to bit 0, the command “x?2=0” turns on the LED.</td>
</tr>
</tbody>
</table>

Listing 1—This HTML-like code delivered the web page shown in Photo 2. The special tags beginning with a back-tick (') activate PicoWeb firmware routines that insert text into the HTML document stream when the page is retrieved by a web browser.
script that processes a text file with a list of web pages and images to be included in the build and produces a binary load image file suitable for use with our network-based serial EEPROM loader (netprog.pl). The complete build procedure only takes a couple of seconds on a Pentium II–based PC.

A full software build produces the following files that need to be downloaded into the PicoWeb server:

- **picweb.rom**—Atmel flash code/data in generic ROM format
- **picweb.ep**—Atmel EEPROM data in generic ROM format
- **picweb.el**—serial EEPROM p-code/data in EEPROM loader format
- **images.dat**—serial EEPROM HTML, GIF, and JPEG data

The first two files are used to program the flash memory in the Atmel microcontroller, and the last two files are used to program the PicoWeb server’s onboard serial EEPROM.

Programming of all flash memory and EEPROM data is controlled via a batch file that invokes the programming utility to program the microcontroller’s flash memory with the contents of **picweb.rom** and (if nonempty) programs the on-chip EEPROM with the contents of **picweb.ep**.

The batch file also invokes the Perl-based netprog.pl script to program the serial EEPROM with the contents of **picweb.el** and **images.dat** (via the network).

Programming the Atmel microcontroller takes less than 20 s. Programming the serial EEPROM via the network connection takes a similar amount of time, depending on the amount of p-code and web page data placed in the serial EEPROM.

We developed the parallel port programming tool (PPPT) so that it would not be necessary to remove the Atmel microcontroller chip from the breadboard and plug it into a separate programmer to change the firmware. Our program was inspired by the BASIC program from Jeff Bachiochi’s “Learning to Fly with Atmel’s AVR” article (Circuit Cellar 101).

The PPPT is a 16-bit MS-DOS program written in C that will run under Microsoft Windows in an MS-DOS Prompt window. The program requires a simple cable that you can build yourself (see Figure 1 for the cable pinouts). By plugging the cable into the programming connector on the PicoWeb server, the AT90S8515 can be programmed in-circuit using a PC’s parallel port.

We like our program better than the similar program provided with the Atmel AVR demo kit because, besides being free, PPPT reprograms Atmel parts about twice as fast as the AVR demo kit. PPPT also has both command-line and menu-driven interfaces (see Table 2 for a summary of the commands). Here’s a sample command line we use (in a batch file) to erase, download both program memory and EEPROM, and then reset the PicoWeb server:

```bash
pppt -ce -lp test.rom -le test.eep -en
```

Our programming tool allows the AT90S8515’s program memory and EEPROM to be dumped, modified, saved, or loaded via standard ROM files produced by the Atmel assembler.

**THE NEXT GENERATION AND BEYOND**

Eventually we got the PicoWeb server breadboard working and settled our argument about the
feasibility of putting cheap microcontrollers on the Internet. We proved that cheap web-enabled embedded microcontrollers are indeed feasible, but our breadboard wasn’t pretty and it was way too big!

The next step was to take the PicoWeb server breadboard’s components, combine them with the ISA-bus Ethernet adapter’s components, and lay everything out on a single PCB. To save on circuit board area, the surface-mount version of the Atmel AT90S8515 was used along with a low-cost highly integrated version of an NE2000-compatible Ethernet controller, the Realtek RTL8019AS. The result is a simple two-sided 1.4×3” circuit board (smaller than a business card) that fits into an extended DB-25 connector hood (see Photo 3).

The Realtek RTL8019AS Ethernet controller is a single-chip NE2000-compatible device with on-chip RAM that only needs a transformer, a single resistor, and a few capacitors to implement a complete 10BaseT Ethernet network connection. Therefore, an Atmel AT90S-8515 microcontroller, a Realtek RTL8019AS Ethernet chip, two crystals, a transformer, RJ-45 and DB-25 connectors, and a few resistors, capacitors, and LEDs are the only components needed to construct this ultra-small PicoWeb server.

While we were at it, we added a +5-V regulator and a Maxim RS-232–level shifter. We kept the breadboard’s 16-KB serial EEPROM chip. Because the Realtek chip supports an 8-bit data bus, we were able to free up eight more I/O pins on the Atmel microcontroller. As a result, the DB-25 connector has 16 free digital I/O lines, as well as an RS-232 serial port. The total parts cost remained under our $25 target.

The printed-circuit version of the PicoWeb server supports all the functions of the breadboard version plus a few additional features. There are now 16 bits of external digital I/O available over the DB-25 connector. Two of the DB-25 pins carry the AT90S8515’s serial port transmit and receive lines using standard RS-232 levels. In-circuit programming of the Atmel microcontroller also is via the DB-25 connector.

The onboard regulator accepts either AC or DC power in the range of 6–30 V. Typical current consumption is under 30 mA from the +5-VDC supply.

What we seem to have created is the world’s smallest, cheapest, and lowest power web server. However, given the rapid pace of technology, these records surely won’t stand for long!

Steve Freyder telecommutes from home for Science Applications International Corp. (SAIC) working on automated toll-collection systems. He lost his office at SAIC in San Diego many years ago by never visiting it. Steve has been programming since he first discovered computers in high school in 1970. You may reach him at steve@freyder.net.

David Helland works for SAIC, most recently on portable electronics for military training range systems. Dave has been building hardware and software systems for several decades now, and in his spare time, he restores vintage fiberglass dune buggies. You may reach him at dhelland@worldnet.att.net.

Bruce Lightner works from home for Lightner Engineering in La Jolla, CA. He too discovered computers several decades ago and has been building hardware and software for them ever since. You may reach him at lightner@lightner.net.

Collectively, the authors are often referred to as “Frey n’ Hell Light” by their friends.