



FROM THE BENCH

by Jeff Bachiochi



CONTEST PRIMER

## The Growth of the Atmel AVR Family

*The Atmel AVR family has been growing rapidly since its debut in the late 1990s. Today, you have several AVR products to choose from when preparing for a project. This month, Jeff delves deeper into the AVR story, and provides an example of how an AVR-based design allows him to control a thermostat.*

Each time I look at my lawn (I use the word “lawn” loosely), I can’t help but think that if I would treat it to a bit of weed killer, I would have a nicer-looking yard. I don’t do this for a number of reasons. First, I like the look of green, as opposed to brown dying weeds. Second, I’d have to mow it on a weekly basis rather than a monthly basis. My lush green weeds seem to grow at a much slower rate than my neighbor’s patchy tufts of green grass. On the sunny side of the yard, I have a crop of nice green vegetation (albeit not grass). On the shady side, I have a nice carpet of cushy green moss—and moss doesn’t need trimming at all! I feel sorry for the lawn-care folks who constantly call and want to help me rid the area of weeds. They just don’t get it.

I used to play golf. Yes, manicured courses look great. But it just isn’t high on my list of priorities to sink a lot of money into my lawn to bring it up to perfection. I know someone who hates grass enough to drop mulch all over it. Now, he doesn’t even need a lawn mower! Although I haven’t gone to that extreme, I’ll choose the hammock over the mower and live amongst my green weeds.

I keep with the ideas that seem to work for me. As you know, I’ve always been a big fan of flash memory-based micro-

controllers. Atmel began its AVR line of in-circuit programmable, flash memory-based processors in 1997. What were only four products just a few years ago have blossomed into a dozen or so today. You don’t get this kind of growth in a product line unless the first seeds have produced a bountiful harvest. Atmel credits this to a soaring price performance level.

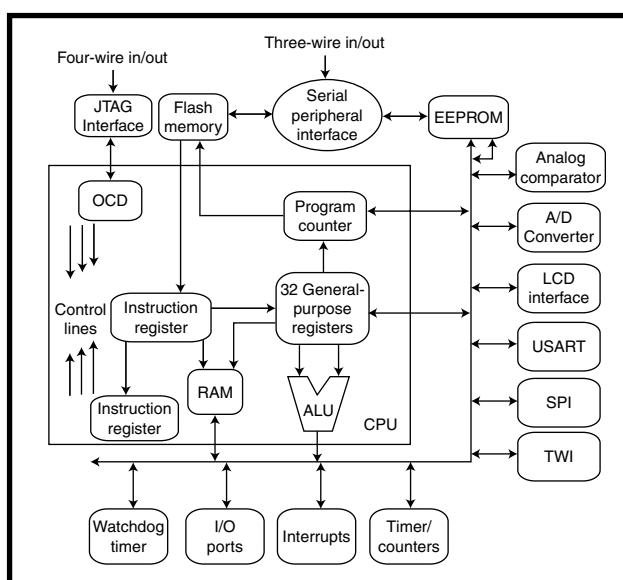
### WHAT IS AVR?

The CPU—which was developed by a pair of researchers in Trondheim, Norway prior to 1995—resembles most RISC processors, although it has smaller

registers with most of the instructions running in a single cycle. A well-defined I/O structure with an internal oscillator, timers, UART, SPI, pull-up resistors, pulse width modulation, ADC, analog comparator, and a watchdog timer limit the need for external components. AVR instructions minimize the size of your program whether you write in C or assembly. The AVR will optimize costs and get your product to market quickly by way of on-chip/in-system programmable flash memory and EEPROM.

There are actually three separate spaces mapped into an AVR microcontroller. The address space is a linear 64 KB in a 16-bit word format (16-bit instructions). The program counter covers the complete flash memory (word) range of each part. There are two data spaces, one for EEPROM (when available) and the other for SRAM. As you can see in Figure 1, 32 general-purpose registers (0x00–0x1F) combine with the ALU to make up the central core. The AVR uses Harvard architecture (separate memories and buses for program and data). Almost half of the instructions are executed in one clock cycle with the CALL and RET instructions requiring up to five cycles. (They must adjust the stack.)

In a typical ALU operation, two operands are output from



**Figure 1**—The AVR CPU core uses 32 general-purpose registers having direct access to the ALU. A generous portion of peripherals is interfaced through the system address and data bus. Total ISP gives access to both flash memory and optional EEPROM memories.

the register file, the operation is executed, and the result is stored back in the register file in one clock cycle. Six of the 32 registers (0x1A–0x1F) can be used as three 16-bit indirect address register pointers (X, Y, and Z) for data space addressing (enabling efficient address calculations). Address pointer register Z (0x1E–0x1F) also can be used as an address pointer for look-up tables in flash program memory.

The I/O memory space contains 64 addresses for CPU peripheral functions as control registers, SPI, and other I/O functions. The I/O memory can be accessed directly, or as the data space locations following those of the register file (0x20–0x5F). Larger devices have extended I/O addresses (0x60–0xFF) in data space.

Figure 2 shows the linear make up of the data space including the register file, I/O registers, extended I/O registers, and the internal data SRAM. On the ATmega169, there are 1024 bytes of available RAM beginning at address 0x100.

## INSTRUCTION SET

The RISC instruction set is broken into five areas: arithmetic and logic, branch, data movement, bit manipulation, and CPU control. The instructions that manipulate data use either direct addressing or indirect addressing. The former is to a working register, I/O or extended I/O register, or the SRAM. The latter is referenced by a 16-bit register, 16-bit register with offset, or 16-bit register with a pre-execution decrement or a post-execution increment of the register.

The instructions that manipulate the program counter load a new address either directly or indirectly (from a working register pair). When instructions deal with retrieving from or storing to the program memory (flash memory), the 16-bit register pair Z is used. Because program addresses are word-sized (16 bits), Z's LSB indicates which of the bytes within the word to deal with.

Most operations that deal with two registers can use any of the 32 working registers (0–31). The operations that include a constant (0–255) must use the upper 16 working registers (16–31). For those devices with a hardware multiply, the result of an 8-bit multiplication with two working registers is 16 bits. The 16-bit result is placed in working register

pair R0:R1. Branch instructions have a relative offset of 7 bits (63 to –64). Relative jumps and calls use a 12-bit offset (2047 to –2048). Indirect operations use one of the 16-bit register pairs X, Y, or Z (26:27, 28:29, or 30:31); therefore, they can cover a 64-KB address range. For devices with a larger address space, the EIND register in extended I/O space holds the upper bits of an extended indirect operation.

A large group of the instructions deals with bit manipulation and branching based on the outcome of an arithmetic, logic, or manipulation operation. This is ideal for control applications and makes tight code possible. For a more detailed description of the AVR's innards, refer to my December 1998 column, "Learning to Fly with Atmel's AVR," (*Circuit Cellar* 101).

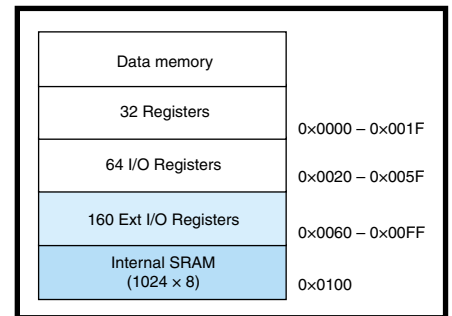
## BRANCHING OUT

It wasn't so long ago that the first AT90S part made its way onto the scene, followed by the ATiny and ATmega series. Today, the success of the AVR core is a reflection of the growing AVR family; it has enabled the product line to branch out in many directions.

The AVR core can be found in FPSLIC, the 5000- to 40,000-gate FPGA. It is also used in the DVD/CD storage chipsets (along with the ARM micro). Wireless support comes from the SmartRF series of AVR MicroTransmitters and MicroTransceivers. The secureAVR microcontrollers include a crypto-coprocessor for secure encryption and authorization functions. A range of USB controllers use the AVR core supporting full-speed USB 2.0. The ATmega series has expanded and now includes AVR micros with built-in LCD controller/drivers.

## FPSLIC

You may remember Atmel's 2001 Design Logic FPSLIC contest. The AVR core in this SRAM-based FPGA gives your design a high degree of system level integration. The FPGA macro library has plenty of custom peripherals, which help augment the feature-packed fixed peripherals of the AVR micro. An important feature of the FPGA is the internal interrupt architecture allowing rapid response to that circuitry. A JTAG interface provides extensive



**Figure 2**—The data space consists of general-purpose registers and I/O registers. Devices stuffed with peripherals use an extended I/O register space. Note that 1 KB of internal SRAM begins at address 0x100 for the ATmega169.

on-chip debug support and boundary scan capabilities to the AVR ports.

The 3- to 3.6-V operating voltage assures low-power consumption while being 5-V I/O tolerant. Oscillator speeds to 40 MHz can provide a throughput of up to 30 MIPS. Both instruction code and configuration is loaded from EEPROM at power-up. The core's cache logic also allows for some dynamic (no data loss) configuration changes. FPSLIC can handle compute-intensive arithmetic functions such as those used in filters and transforms.

## DVD/CD STORAGE CHIPSETS

Let's face it, eight-tracks are dead. (Although some people say they never existed. When my son Kris asked why some of my records had this little plastic doohickey in the large hole, I knew I was teetering on the edge of being labeled Cro-Magnon.) The compact disk media has officially stolen the show.

Even the 12" laserdisc is nothing but junk because DVD formats push the envelope of storage. A DVD/CD ATAPI controller, servo controller, laser power controller, amps, and pre-amps are necessary to provide compact audio and video enjoyment. The AVR core is used in Atmel's DVD/CD ATAPI chipset to perform internal data path and buffer management control.

## SmartRF

Adding high-performance PLL-based RF circuitry to the AVR core creates a convenient and cost-sensitive solution for wireless control and telemetry. The transmitter solution has ultra-low shutdown current until a button is pressed. The microcontroller controls

power output while the PLL uses the crystal to maintain a stable carrier.

The transceiver solution brings secure communication allowing handshaking procedures. Its flexible configuration covers a best fit for bandwidth, selectivity, immunity, and range.

The RF receiver can supply a wake-up signal to the microcontroller for power-conserving measures when idle. Covering the 400- to 950-MHz bands, the SmartRF data throughput can reach 64 kbps.

## SECURITY FOR SMARTCARDS

As we become more comfortable with a cashless society, the risk of identity theft continues to rise. We dislike being held captive by a simple multidigit number. In the future, transactions will be based on more than just this fragile key. Smartcards are becoming the new cache. Without knowledge of the data held securely within, it becomes useless to everyone but the owner. Thus, the security of a smartcard is its most valuable asset. The AVR architecture becomes the hidden strength of the new plastic via ISO 7816 external interfacing.

Security features must protect data from not only direct software access but also from hardware probing. The secure AVR-based micros employ voltage and frequency control, secure layout, and bus encryption security features. Atmel's commitment to support is reflected in the application development tools they make available for all of the AVR products.

## USB CONTROLLERS

Atmel understands the importance of USB to the future of computing. It has an extensive line of USB hubs, microcontrollers, and hosts. This guarantees that the company will be part of that future.

Many products are based on AVR microcontrollers offering low- and full-speed USB peripherals. Low-power AVR micros can run directly from the USB bus thanks to internal 3.3-V regulators. The array of on-board peripherals simplifies designing even the most I/O-intensive user devices.

Software support includes the USB

Command type	Command
Fan	F0 or F1
Mode	MA, MH, MC, or MO
Heat set point	Hxx (where xx = HEATSETPOINTH, HEATSETPOINTL)
Cool set point	Cxx (where xx = COOLSETPOINTH, COOLSETPOINTL)
HVAC Status	SI, SH, SC, or SO
Temperature	Txx (where xx = TEMPH,TEMPL)
E-mail	E0, E1, or E2

**Table 1**—The thermostat sends these commands whenever a change occurs (i.e., you change the heat set point or the temperature falls below the heat set point).

library and applications Wizard. This tool produces code, based on user input, including USB descriptors, reports, and even complete applications.

## LCD CONTROLLERS

Almost every product that requires a user interface takes advantage of an LCD for rendering important information. Whether the display has graphic glyphs or alphanumeric characters, it's just a matter of segment control. Empowering an AVR core controller with an LCD controller simplifies your job. This peripheral can driver up to 25 segments and four back planes. To support the necessary multiplexing, the controller supports various duty and biasing, as well as flexible frame frequency control.

Atmel's ATSTK500/502 starter kit is a great platform for helping you design applications for an AVR microcontroller with a built-in LCD peripheral controller. As I was working on last month's column, I thought virtual peripherals were fine for demonstrating a point, but I'm the kind of guy that likes hardware. So naturally I was looking for an alternative to the virtual thermostat used to demonstrate an application of the Lantronix XPort Ethernet-to-serial 'Net-enabling device. The ATmega169 has the right features to demonstrate a true hardware application of last month's virtual thermostat.

## STARTS WITH THE DISPLAY

The display supplied on the ATSTK502 has many glyphs as well as seven characters as seen in Photo 1a. I chose to use a subset of this (see Photo 1b). The up- and down-pointing arrowheads indicate various enabled modes and functions, while only four characters are needed to show temperature.

The LCD controller is first initialized for quarter duty, 3.0-V contrast, with a

54-Hz refresh rate (based on a 3.58-MHz system clock). This allows the proper output signals to be generated for the LCD's segments and four back planes (commons). Segments are mapped into 20 data registers (LCDDR0–19), with five registers for each com-

mon (LCDDR0–5 for COM0, etc.). Only 25 bits are needed (one for each segment output) in each group. That's three full bytes and the LSB of the fourth byte.

(The remaining bits are unused.)

Character data consists of 16 segments (14 for character segments and two for glyph segments), with 4 bits in each of the four back plane groups. This puts the same 4 bits for character 1 and 2 into LCDDR0, 3 and 4 into LCDDR1, and 5 and 6 into LCDDR2. Therefore, the look-up table data can be masked directly into 1 of 3 bytes (with the data nibble swapped for even characters).

When a bit in one of these registers is written with a one, the corresponding LCD segment is enabled (masking the reflectivity of the background and showing up as black). Bits written with a zero allow the segment to remain transparent. The glyphs share bit locations with character segments with the same byte, which requires masking bits when updating data. This means that updating the data for a character must not alter the data for any glyphs and vice versa.

The LCD\_Update routine tests variables to determine which glyphs and characters need to be updated. The mode byte has four possibilities: H (heat), C (cool), A (auto), and O (disabled). The Fan byte indicates whether the fan is a one (on) or a zero (auto). A byte value for HVACSTATUS is dependent on mode, temperature, and set point; it indicates when the furnace/air-conditioner is required to run. The values are "H" (call for heat), "C" (call for cool), "I" (idle), and "O" (disabled). Although the character data comes from a look-up table consisting of characters 0 through 9 followed by "-", "H," "C," "degree sign," and "F," one of three displays can be requested.

The DISPLAYMODE byte can be one of three values: 0x00 (temperature), 0x01 (heat set point), and 0x02 (cool set

point). The temperature (DISPLAY-MODE) displays the measured temperature of a thermistor displayed as “## °F” (more on this later), while the set point (DISPLAYMODEs) display “H- ##” (heat set point) and “C-##” (cool set point).

## THERMISTOR

A negative temperature coefficient (NTC) thermistor is used as the lower leg of a voltage divider with a 10-kΩ resistor to  $V_{REF}$ . Because the thermistor has a 10-kΩ resistance at 25°C, the voltage at the junction of the two devices (and the input to an A/D channel) will be  $0.5 V_{REF}$  at room temperature. The thermistor’s equation for temperature is:

$$\text{Temp.} = \frac{\beta}{\ln\left(\frac{V_{ADC}}{V_{REF} - V_{ADC}}\right) + \frac{\beta}{T_{AMB}}} - T_{ZERO}$$

where  $\beta$  is 3450,  $V_{REF}$  is 1.263 V,  $T_{ZERO}$  is 273°K,  $T_{AMB}$  is 298°K (273° + 25°), and  $V_{ADC}$  is the analog-to-digital conversion voltage.

To allow this project to proceed without some serious calculation routines, I used my trusty Radio Shack scientific calculator to solve for the  $V_{ADC}$  values for all the Fahrenheit temperatures between 40° and 99°. The  $V_{ADC}$  values were then converted to the 8-bit conversion equivalents and stored in a look-up table.

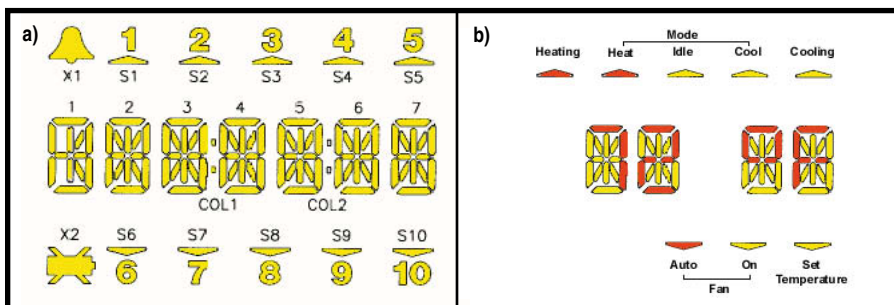
The ADC is initialized for continuous single ended conversions of channel 0 using a division factor of 32 on the system clock. The ADC actually does a 10-bit measurement conversion and stores the value either left or right justified into a 16-bit extended I/O register. By using left justification, the high byte of the conversion is the high 8 bits of a 10-bit conversion (or

an 8-bit conversion value). Eight of these conversions are added together, and the total is shifted right 3 bits to divide the total by eight and end up with an average. A low conversion value of 99 corresponds (coincidentally) to a high temperature of 99°F. A high conversion value of 180 corresponds to a low temperature of 40°F.

To find the actual temperature, the conversion average is compared to the first value in the TEMP\_TABLE (with the first entry 180 = 40°F). If the table entry is greater than the conversion average, then the temperature variable, which is initialized to 40, is incremented and the next table value is grabbed and the comparison repeats until the TEMP\_TABLE entry value is less than or equal to the conversion average. The temperature variable has now incremented to the temperature in degrees Fahrenheit that corresponds to the appropriate average A/D conversion value. To stay compatible with the thermostat conversion protocol that I discussed last month (“Global XPortation,” *Circuit Cellar* 162), the temperature is finally converted into two ASCII digits as TEMPH and TEMPL. You’ll notice that all variables are stored as ASCII values to remain compatible with the protocol.

## BUTTONS

Five user inputs provide local mode changes to the thermostat. The first button, Mode, increments the MODE variable through four possibilities: H, C, A, and O. Heat places the thermostat in Heat mode and asks the furnace for heat if the temperature falls below the heat set point by setting the HVAC-STATUS to “H” (heating). HVACSTATUS returns to “I” (idle) when the tem-



**Photo 1a**—The LCD used on the STK502 adapter has a number of glyphs as well as 12-segment alphanumeric digits. **b**—I used the arrowheads as function indicators and only four of the available digits.

perature rises to heat set point + 4°F (an arbitrary delta I added here, but is not present in the virtual thermostat).

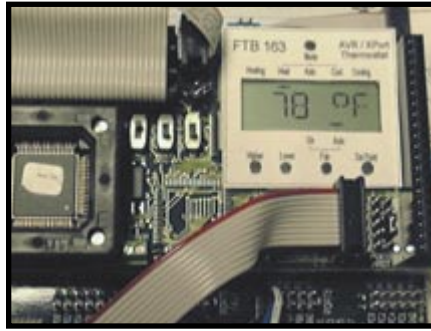
Cool places the thermostat in to Cool mode and asks the air conditioner for cool if the temperature rises above the cool set point by setting the HVACSTATUS to "C" (cooling). HVACSTATUS returns to "I" when the temperature falls to cool set point - 4°F (an arbitrary delta).

Auto places the thermostat into both Heat and Cool modes (i.e., it can call for heating or cooling as necessary). Obviously, if the heat set point and cool set point are too close, the systems will fight. Although this wasn't implemented in the virtual thermostat, it is prevented here in the Change\_Plus and Change\_Minus routines by comparing the heat and cool set points. They are automatically adjusted to keep a two times delta distance between them. Off, or disable mode, shuts down the call for heat/cool operation by setting HVACSTATUS to "O."

The second button, Fan, sets the FAN variable to either a one (on) or a zero (auto). The one value forces the furnace/air conditioner fan on; the zero value allows the furnace/air conditioner to determine the state of the fan based on its own criteria.

The third button, Set point, allows you to change the heat and cool set points. This button sets the display mode to one of three values: 0x00, 0x01, and 0x02. A display mode value of 0x00 indicates the LCD should display the actual temperature. A value of 0x01 indicates the LCD should display the heat set point. A value of 0x02 displays the cool set point.

After the heat set point or cool set point has been selected, the fourth button increments the set point by one. The fifth button decrements the set point by one.



**Photo 2**—I placed a small front panel over the LCD on the STK502 adapter. The ZIF socket on the left holds the ATmega169 device. Black dots on the front panel indicate the positioning of push button switches. The physical switches on the STK500 demo board are actually along the bottom edge of the PCB just out of view.

## TALK AND LISTEN TO XPORT

The thermostat conversion protocol is based on a serial data rate of 9600 bps using a data format of 8N1. The ASCII commands for this protocol make debugging simple. For the most part, every time something changes (e.g., the temperature), a command is sent to the XPort. All commands are two or three characters (see Table 1). E-mail notices can be enabled and disabled only through the XPort.

The XPort is responsible for passing on thermostat commands to anyone on the Ethernet who has opened a connection with it. This might be you and your browser. After all, you might want to adjust the heat set point of the thermostat from work so it's toasty when you get home. Or, this might be the XPort located at the furnace/air conditioner in the basement where a command from the thermostat to warm up the house could be carried out.

The thermostat must not only send changes as commands to the XPort; it must also respond to commands from the XPort (or the Ethernet). Many commands are similar to those the thermostat may send itself (see Table 2). These commands can change variables in the

Command type	Command
Fan	F0 or F1
Mode	MA, MH, MC, or MO
Heat set point	Hxx (where xx = HEATSETPOINTH, HEATSETPOINTL)
Cool set point	Cxx (where xx = COOLSETPOINTH, COOLSETPOINTL)

**Table 2**—These commands are sent to the thermostat whenever a change occurs via Ethernet (i.e., you change the heat set point from a browser).

thermostat like a user when thermostat buttons are pushed. Five additional commands also change variables in the thermostat, but they do not have similar local input (see Table 3).

The Sx command updates the variable STATE to enable/disable the thermostat's ability to request heat/cool from the furnace/air conditioner. The IN command requests that the thermostat send the status, mode, fan, heat set point, cool set point, and actual temperature.

The low and high temperature thresholds are similar to the heat set point and cool set point; however, instead of requesting heat/cool from the furnace/air conditioner, e-mails can be automatically sent by the XPort to signify that the thermostat has reached an unsafe temperature. The thermostat indicates this to the XPort using the Ex command (0 = safe, 1 = high temperature warning, and 2 = low temperature warning). The Nx command updates the TRIG variable to enable/disable the reporting of Ex e-mail commands.

Only USART interrupts are used in this application. Although the normal program flow can place data to be sent in a ring buffer, it must set the USART data register empty bit to allow the transmit interrupt routine to empty the buffer. The receiver, however, is always enabled, and fills a second ring buffer with received characters until a <CR> is received. The ring buffer is searched for the first character less than ASCII zero. The next three or four characters are placed in temporary variables and verified for legal data. If all passes muster, then interrupt routine branches to handle each command. Most of these commands just update a variable, but the IN command requests

information. I wanted to leave this interrupt routine to handle that, so I tagged the need for this using the T bit (user bit) in the status register.

Normal program execution proceeds as follows. Do an ADC loop for temperature conversion and do a table look-up of the conversion result to degrees Fahrenheit. Then, based on the selected mode and current temperature, make any changes necessary and send any command changes to the XPort. Next, look for button pushes, make any changes necessary, and send any command changes to the XPort. Finally, update the LCD and check the T bit in the status register to see if the XPort has requested information (IN). If necessary, send the XPort all of the commands indicating the current status and go back to begin the loop once again.

## PLUG IT IN

Now, it's time to make the magic happen. Both demo PCBs have a DE-9F for serial communication. Of course, using the XPort and AVR microcontroller in a design does not require any RS-232 converters or connectors; but, jury-rigging the demo boards requires a cable. Try and find a DE-9M-to-DE-9M cable when you need one. Forget it. Fiddling around in the junk box I found two crimp-on DE-9Ms and a short piece of ribbon cable. I plugged it in and, voila, no magic. Doh! This needs to be a null modem cable.

After the physical connections are completed correctly, things started to look up. Using my browser, I contacted the XPort and got a pop-up comment on installing the Java environment. Then up came the thermostat layout that I showed you in my "Global XPortation" column (*Circuit Cellar* 162).

This time, however, the XPort talks to the AVR microcontroller, and the information displayed on the LCD is reflected on the display in the browser window (see Photo 2). I can click on the browser's buttons and the changes are sent back to the thermostat. Yes.

## EXPANDING UNIVERSE

From the standard eight-pin ATiny controllers to the ATmega with high-density in-system programmable flash memory, there's an AVR solution tailored to meet your most specific needs. And note that both the instruction set and architecture are the same for all AVR products. So, when your code increases, you can easily and quickly port to a larger device. AVR comes three ways: as a standard package product, an application-specific standard product, and an ASIC core integrated into a System-on-a-Chip solution.

You can depend on Atmel to continue expanding its collection of successful AVR products. The RISC core's single-cycle instructions result in serious code efficiency. The AVR's flash memory technology provides in-circuit programming thanks to a separate "boot flash."

You can smile, knowing that Atmel is constantly improving its suite of program and system development tools. It's the kind of support that ends up saving you precious design time. After all, if your design time decreases, then you've earned a little free time. And what better way to make use of it than to lie in a hammock and watch the grass (or weeds) grow. ☑

*Jeff Bachiochi (pronounced BAH-key-AH-key) has been writing for Circuit Cellar since 1988. His background includes product design and manufacturing. He may be reached at jeff.bachiochi@circuitcellar.com.*

## PROJECT FILES

To download the code, go to [ftp.circuitcellar.com/pub/Circuit\\_Cellar/2004/163](http://ftp.circuitcellar.com/pub/Circuit_Cellar/2004/163).

## SOURCES

**ATmega169 Microcontroller**  
Atmel Corp.  
(408) 441-0311  
[www.atmel.com](http://www.atmel.com)

Command type	Command
State	S0 or S1 (0=off, 1=on)
Request information	IN
Low temperature threshold	Axx (where xx = LOWWARNINGH, LOWWARNINGL)
High temperature threshold	Zxx (where xx = HIGHWARNINGH, HIGHWARNINGL)
E-mail triggers	N0 or N1 (0=disable, 1=enable)

**Table 3**—These commands can be sent to the thermostat via Ethernet. They adjust variables that cannot be changed via local control (i.e., the e-mail triggers are only enabled and disabled through the Ethernet). The thermostat has no local control over this function, although it is responsible for sending an Ex command if e-mail triggers are enabled.