CIRCUIT CELLAR®
THE MAGAZINE FOR COMPUTER APPLICATIONS

Build An AVR Programmer

FEATURE ARTICLE

Stuart Ball

If you've been looking for an inexpensive way to program Atmel's AVR microcontrollers, then Stuart's project is just what you need. With a little work and only six ICs, you can build a programmer that is perfect for working with AVR parts.

If you've wanted to experiment with the Atmel AVR-series microprocessors but needed a cheap way to program them, this project is for you. Using only six ICs, you can start programming and experimenting with the AVR parts.

The AVR 90S-series microprocessors from Atmel are RISC-based microcontrollers. There are currently four parts in the family, the AT90S1200 and 'S2313 (in 20-pin packages), and the AT90S4414 and 'S8515 (in 40-pin packages). Table 1 shows the key differences between the parts. For a brief description of the AVR architecture, see "What's the Count" in *Circuit Cellar* 112.

All AVR devices have 32 general-purpose registers. On the AT90S1200, this is the only RAM available. The other devices in the series have additional RAM, as indicated in Table 1.

The AVR parts use flash memory to store programs. The flash memory can be electronically erased and reprogrammed, so there's no need to have a UV eraser sitting by the programmer.

Atmel provides two methods to program the flash memory—serial and parallel. The serial method uses three pins on the device and does not require any power supply other than the normal 2.7–6-V supply voltage. The serial method is suitable for programming the devices in-circuit. However, there is a fuse (SPIEN) in the device that can be set to disable serial programming. If this fuse is programmed, serial programming and reading of the device cannot be performed.

The parallel programming method uses eight data lines and seven control

	AT90S1200	AT90S2313	AT90S4414	AT90S8515
Pins	20	20	40	40
Flash memory	1 KB	2 KB	4 KB	8 KB
EEPROM	64 bytes	128 bytes	256 bytes	512 bytes
RAM	32 bytes	128 + 32	256 + 32	512 + 32
I/O Pins	15	15	32	32
8-bit timers	1	1	1	1
16-bit timers	0	1	1	1
UART	0	1	1	1

Table 1—*The four AVR devices have different amounts of on-chip RAM and flash memory. The 40-pin devices have more internal memory than the 20-pin devices, as well as more I/O pins.*

lines. This method also requires that +12 V be applied to the reset pin to enable programming. The parallel method requires more hardware but works regardless of the state of SPIEN. The programmer design for this project uses the parallel method.

Programming the AVR in parallel mode redefines the normal pin functions as shown in Table 2. Unlike a PROM, the AVR processor doesn't have separate address lines to select the location to be programmed. Instead, the address is written as two 8-bit data values using the data lines. The eight data lines get the programming address, programming data, and commands that tell the AVR what programming operation to perform.

The AVR drives –BUSY low when it starts to program a byte, and back high when programming is finished. This provides a means for the programmer to determine when a byte is finished programming.

The –OE signal goes low to enable the AVR outputs for reading. The –WR signal tells the AVR to start programming a byte and is pulsed low after the command, address, and data have been loaded.

BS selects either the low or high bytes of address and data. When BS is low, the low address or data byte is read or written, and when BS is high, the high byte is read or written. XA0 and XA1 determine whether the data on the data lines is a command, address, or data byte. XA0 and XA1 are defined as:

XA1	XA0	
0	0	Address
0	1	Data
1	0	Command

Finally, the clock pulse on the crystal input pin clocks the command, data, and address bytes into the chip.

The AVR devices also have an EEPROM that can be written by the processor under program control or programmed externally. There is also a device signature that identifies the device type and a pair of lock bits. The lock bits can be used to prevent anyone from reading the contents of the device. The lock bits and SPIEN fuse can be cleared only by erasing the entire device.

Pin	Definition
PB0–PB7	Data and command input/output
PD1	–BUSY feedback bit
PD2	–OE (Output Enable)
PD3	–WR (Write Enable)
PD4	–BS (Byte Select, selects high or low byte)
PD5, PD6	XA0 and XA1, address inputs
Crystal1	Clock pulse

Table 2—*In programming mode, the pins of the AVR device are redefined as shown. Program mode is entered by bringing the V_{PP} pin to +12 V.*

The Atmel parallel programming mode supports chip-erase, programming and reading of the flash memory and EEPROM, and reading the fuse and lock bits along with the device signature bytes.

Programming a location in the AVR involves placing a command on the data bits, setting XA0/XA1 to two, and toggling XTAL1. Then, the high address is placed on the data bits, XA0/XA1 is set to 0, –BS is set to 1, and XTAL1 is toggled again. A similar procedure is followed to load the low address byte and the data byte. Then –WR is pulsed low to start programming. When programming is complete, the AVR drives the –BUSY bit high.

Although the AVR has a program word that is 16-bits wide, the word is programmed one byte at a time. Figure 1 shows the waveform used for programming the low byte of the word. The same sequence is followed for the high byte except the BS line is high when the data byte is loaded.

The process of reading the AVR device is similar to writing, except that the read command is issued and the –OE signal (instead of –WR) is pulsed low to enable the AVR outputs. Before programming, the programmer software erases the device using the erase command. The AVR does not use the –BUSY bit to indicate when the erase is complete so erase timing is up to the programmer.

Command bytes are 0x80 to erase the chip, 0x10 to program the flash memory, and 0x02 to read the flash memory. The AVR commands for reading/programming the EEPROM and for reading/programming the fuse and lock bits are not implemented on this project.

You can download an assembler for the AVR microcontrollers from the Atmel web site.

PROGRAMMER FEATURES

The AVR programmer provides capability to program all four of the Atmel AVR devices. Two ZIF sockets are provided, one for 20-pin and one for 40-pin parts. The programmer plugs into the parallel printer port of a PC. Windows 95 software to use the programmer is available on the *Circuit Cellar* web site.

The programmer circuit connects to the parallel printer port of a PC. As you can see in Figure 2, U2 is a 74HC-374 that latches and buffers the data bus to the AVR. U3 is a buffer that allows the data bus to be read. U1 drives the control inputs to the AVR device. U4 decodes the control lines from the printer port to latch data, read data, or clock data into the AVR.

The printer signals –AF and –INIT

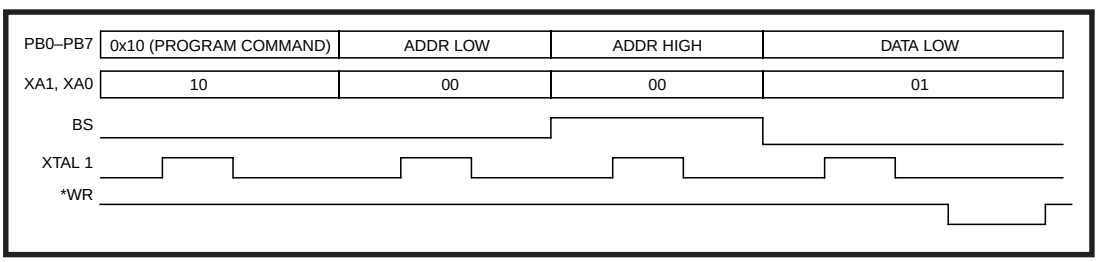

Figure 1—*The timing diagram for programming a single byte in the AVR device is shown here. Data, address, and commands are written to the device using the 8-bit data bus.*

are used to select which register on the programmer is written or read. These pins are defined in Table 3.

To perform one of these operations, the –AF and –INIT lines are set to the correct state, and the –STB signal (pin 1) is pulsed low. This causes the appropriate output of U4A to pulse low.

To read data, the programmer has to turn off the printer port output buffer and the data register on the programmer. If the data buffer on the port was left enabled, there would be a bus conflict with U3, and if the data register on the programmer was left enabled, there would be a bus conflict with the AVR device when it turns on its outputs.

The printer port buffer is disabled by writing to a bit in the printer port control register (different from the programmer control register, U1). To disable the programmer data register, bit 5 of the programmer control register is set high, driving the output-enable pin (pin 1) of the programmer data register (U2) and tristating the outputs.

The read data buffer, U3, is a bidirectional device, but the direction pin (pin 1) is grounded, allowing transfers in only one direction. Because the AVR programmer uses the printer port data lines for both reading and writing, a bidirectional printer port is required.

The bits in the control register (U1) are defined in Table 4. The –BUSY bit from the AVR is buffered by U5B (pins 3 and 4) and drives pin 11 of the printer port connector. This is the printer BUSY signal and is monitored by reading the printer port status register.

SO1 and SO2 are 40- and 20-pin ZIF sockets for programming the AVR device. Q1 supplies +5 V to the AVR socket and the LED (D2) lights when +5 V is applied.

The input power supply is a 12-VDC wall-wart transformer. Be sure to use a supply with a DC output, not AC.

Twelve volts is enabled by Q2. When output D7 of the control register (U1–19) is high, Q2 is saturated and the voltage at the 12-V pin of the

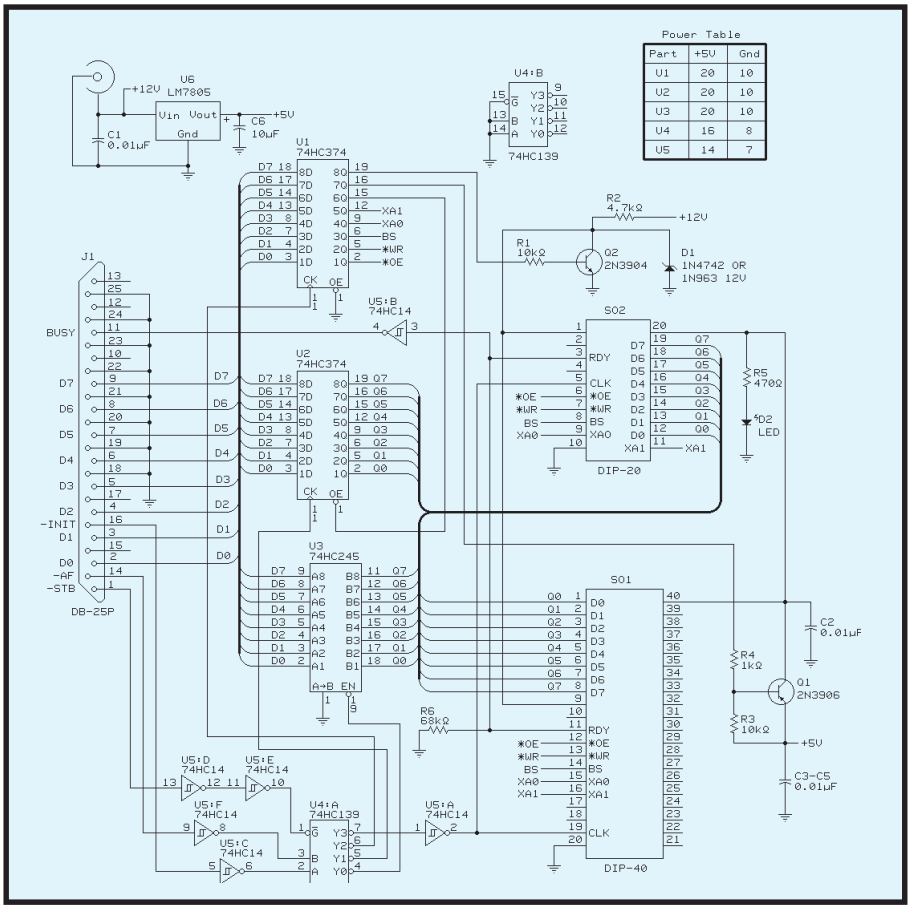

Figure 2—*Because the AVR programmer requires only six ICs and two transistors, you can build this programmer and start experimenting with AVF parts in no time.*

AVR is 0 V. The actual voltage is the saturation voltage of Q2, but that will typically be 200 mV or so.

When output D7 of the control register goes low, Q2 is turned off and resistor R2 pulls the AVR 12-V input up to the 12-V supply. The AVR doesn't draw significant current from the +12-V input, the high voltage is used to put the device into parallel programming mode. Most 12-VDC transformers produce an output of 15 V or so when lightly loaded, so D1 clamps the voltage to 12 V at the AVR.

Transistor Q1 is a PNP type, 2N3906. When output D6 of the control register (U1–16) is high, resistor R3 pulls the base of Q1 to +5 V, turning Q1 off. When control register D6 goes low, the base of Q1 is pulled toward ground through R4. This saturates Q1 and applies +5 V to the AVR sockets. This also turns on the LED.

R6 ensures that the –BUSY bit will be low if no AVR device is installed in either socket, which will produce a device error. Without R6, an attempt to program an empty socket may not detect the error until verify.

Finally, U6 (a 7805) regulates the +12-VDC input to produce 5 V for the logic and the Atmel devices.

ABOUT THE SOFTWARE

The software was written in Microsoft Visual C++, as a Win 32 app for Windows 95. Photo 1 shows the dialog box for the programmer software. Two clickable buttons are provided, one to Erase and one to Erase

Pin 14 (–AF)	Pin 16 (–INIT)	Function
0	0	Pulse AVR clock input
0	1	Write control register
1	0	Write data register
1	1	Read data from AVR

Table 3—*The –AF and –INIT pins on the printer connector (pins 14 and 16) select which programmer register will be written or read. The –STB signal (pin 1) is driven low to actually read the data or clock the register.*

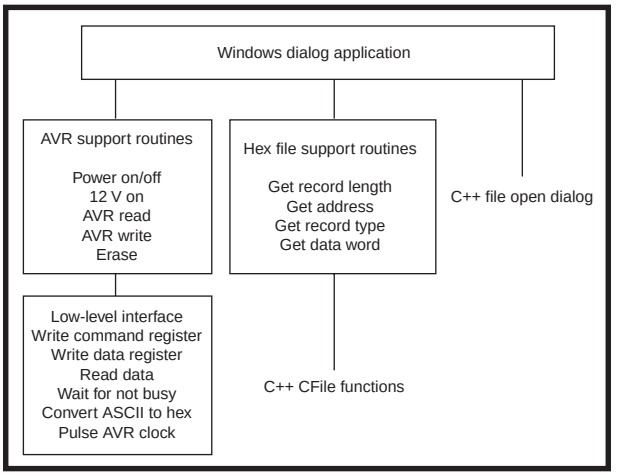

Figure 3—*The software interfaces uses a standard Windows dialog box. The software is partitioned into low-level code that communicates with the port, AVR-specific code that performs device-level functions, and code that communicates with Windows.*

and Program. The Erase button issues an erase command to the device. No erase verification is performed.

To program a device, a file must be selected. The programmer software requires an Intel-format hex file (see the File Formats sidebar for more information). The Atmel assembler enables you to select either Motorola S-record or Intel-format files. The filename may be typed into the edit field or you can click the Browse button to bring up the standard Windows 95 open-file dialog box to select a file.

The Erase and Program button is grayed out until you select or enter a filename. Clicking this button will erase the device and then program it with data from the selected file. If you enter a file that doesn't exist, you will get a file error.

The software provides radio buttons for selecting the port address (0x278 or 0x378) and a speed-compensation value for the speed of your CPU. The speed compensation controls the amount of settling time between successive operations to the printer port and the timeouts for detecting device errors and erasure. The compensation value isn't critical, but if you get device errors while programming, try a different value.

Finally, the software provides a 4-line status box where messages are displayed. The box will display the current address as each hex record is programmed and the location of any errors that occur. Messages scroll up during operation and the last four are displayed.

The software doesn't check for the correct device size, so if you try to program 8 KB of data into a 1-KB device, it will let you. Of course, you'll get a verify error. The software also doesn't check that the file you select is a valid hex file, although almost any other type of file will produce file or device errors.

The software was created using the Microsoft C++ wizards to format the dialog box and buttons. The programming code was originally developed as a console application, then the programming routines were incorporated into the Windows dialog shell.

Figure 3 shows the software structure. The Windows dialog application passes control to other, lower-level functions when you click on a button. The AVR support routines provide functions such as programming a single location, reading a location, etc. These functions call lower-level routines that actually read and write the hardware registers on the programmer via the parallel port.

Bit	Definition
D0	Drives AVR –OE signal
D1	Drives AVR –WR signal
D2	Drives AVR BS signal
D3	Drives AVR XA0 signal
D4	Drives AVR XA1 signal
D5	1 disables data register U2
D6	0 turns on +5 V to AVR
D7	0 turns on +12 V to AVR

Table 4—*U1 controls various programmer functions. Five bits connect directly to the AVR sockets, one enables the data register, and two turn the +5-V and +12-V on and off.*

CIRCUIT CONSTRUCTION

The prototype was constructed on perfboard and wired point-to-point. Figure 4 shows the parts layout that was used and the parts list is available on the *Circuit Cellar* web site. Use a grid of copper EMI tape under the ICs, or some other means to get a good, low-impedance ground. SO1 and SO2 are located on the bottom of the board to make final mounting easier. The completed project was mounted in a plastic case with holes cut in the top for the ZIF sockets and LED.

The power supply is a 12-VDC wall-wart transformer. Anything that can supply 300 mA or more will be adequate. A 2.1-mm coaxial jack is mounted on the plastic case for connection to the supply. The schematic shows 74HCxxx parts. You can use 74HCT or 74ACT as well.

The two ZIF sockets, SO1 and SO2, are mounted on the back of the board to simplify mounting the board in the case. The 40-pin ZIF sockets are manufactured by 3M, Aries, and other were common when a lot of PLDs were in 20-pin DIP packages. 3M/Textool still makes them and they are listed in the Digi-Key catalog.

If you can't find the 20-pin socket, or you want a lower-cost alternative, you can substitute a 24-pin socket and use only the lower 20 pins. If you do this, be sure to route the connections

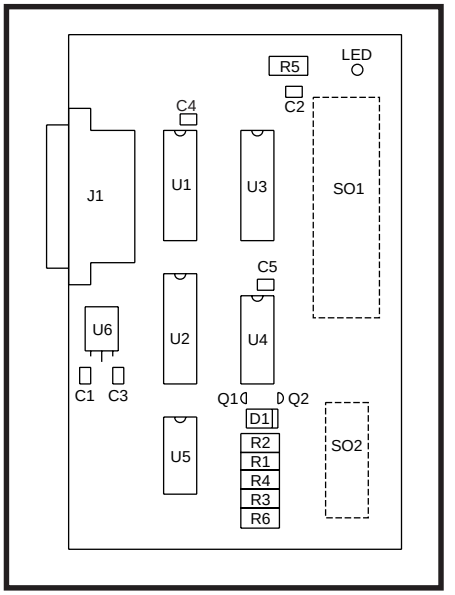

Figure 4—*Note that SO1, SO2, and the LED are mounted on the back of the board to simplify mounting.*

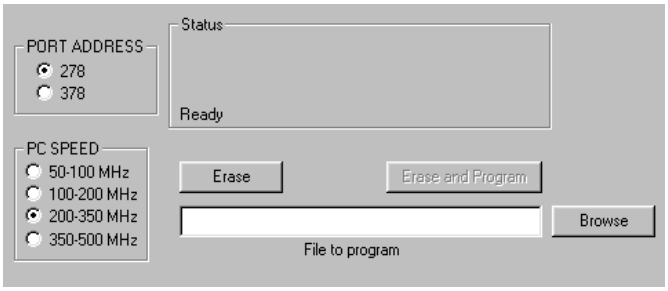

Photo 1—*There are only five user functions—selecting a file, erasing a device, erasing/programming a device, selecting the parallel port address, and selecting the compensation value for CPU speed.*

to the right pins and select a socket (such as the Aries24-65*xxx* series) that accepts an IC that's 0.3" wide.

If you don't plan to program many devices, you can use machined-pin sockets for SO1 and SO2. They obviously won't last as long as the ZIF sockets. And, if you plan to work only with the 20-pin devices, you can leave off the 40-pin socket and vice-versa.

TESTING

Three test files are provided—*TST4414.HEX*, *TST2313.HEX*, and *TST1200.HEX*. *TST4414.HEX* is for '90S4414 and '90S8515 devices, *TST2313.HEX* is for '90S2313 devices, and *TST1200.HEX* is for the '90S1200. All of the files just blink an LED to verify that the programmer works.

Figure 5 shows a circuit that connects a 40- and 20-pin socket to use the test software. The crystal on the test board isn't critical—anything from about 3 MHz up to 10 MHz will work. If you're confident in your wiring abilities, you can skip the test circuit.

Using the programmer is fairly simple. Plug the programmer into the parallel port and plug in the 12-V supply. When you connect the programmer to the computer, use a short cable, 6' or less. Avoid ribbon cable because the crosstalk between the wires tends to be high and may cause errors.

From Windows Explorer, double click on the *AVRBURN.EXE* file to start the programmer software. Select the file you want to program and install the AVR device into the appropriate socket. Click the Erase and Program button and the programmer will erase, program, and verify the part. If you just want to erase a device, don't select a file.

Even though there are two sockets, you can only program one part at a time. Using two sockets eliminates the need for a set of jumpers that would be required to configure a single socket for both device types, but it doesn't allow programming of

File Formats

The Intel-format hex file was developed by Intel in the early days of microprocessors, when data was typically read from a paper-type reader attached to a teletype machine. The file consists of a series of one-line records, terminated by an end-of-line character. All the data is in ASCII, and each record uses the following format:

:LLAAAATTDDDDDDDDDD....CC

where the colon character (:) starts each line, LL is two hex-ASCII characters that define the number of data bytes, AAAA is the starting address for the data on the line (four hex-ASCII characters), TT is the record type (00 for data record, 01 for the last record in the file), DD is the data (two characters per byte), and CC is a one-byte checksum.

The AVR microcontrollers use a 16-bit wide flash memory, so the data is ordered as pairs of bytes to make a word. The low byte of each pair is sent first, followed by the high byte.

A line from a hex file for the AVR devices looks like:

:100020000EBD00E00BBD08EC0ABD00E000E00DBD18

This line describes a hex record with a length of 16 bytes (10h), starting at address 0020 (hex). The record type of 00 indicates that this is a data record. This is illustrated by:

10 (length) 0020 (starting address) 00 (record type)

The first few bytes of data for this line are 0E BD 00 E0 0B BD 08 EC. These would be programmed into an AVR device as words, like:

BD0E E000 BD0B EC08

The one-byte checksum at the end was an important feature when data was sent using electromechanical paper tape readers. In a disk-based system, a bad read will result in a CRC error from the disk controller. The checksum in the file is redundant, and the software for the AVR programmer ignores it.

The four character address value limits the Intel format to a maximum of 64k addresses. There is also an extended Intel hex format that provides a larger address space by defining a third record type.

Although the programmer software doesn't support it, the Atmel assembler is capable of generating Motorola-format hex files. The Motorola format looks like:

S1LLAAAADDDD...CC

where S1 is the start sequence, LL is the length of the record (in bytes), AAAA is the address DD is the data, and CC is a two-byte checksum.

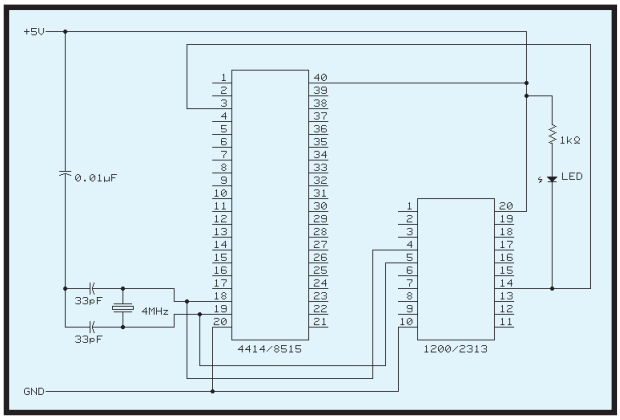

Figure 5—*If you program the test files,* TST2313.HEX *or* TST4414.HEX, *into the appropriate device and plug it into this test circuit, the LED will blink. This test will show you if the programmer is wired correctly.*

two devices at once. If you try to do that, you'll get bus contention on the –BUSY bit as both devices try to drive it, and bus contention on the data bus when the software tries to verify. And besides, what practical program would you ever want to put in both 20-pin and 40-pin devices?

ABOUT PARALLEL PORTS

The programmer requires a bidirectional parallel printer port to operate. In bidirectional mode (also called PS/2 mode), one bit of the control register is used to turn off the data buffer so the data lines can be used as inputs.

Most IBM PC-compatible motherboards include an integrated printer port, and these usually support advanced modes such as ECP and EPP, as well as the simpler bidirectional mode. Some motherboards (e.g., the one I used to develop this project) ignore the bidirectional control bit when in ECP or EPP mode. These ports must be configured (in the BIOS) as bidirectional to use them with the programmer, and this disables the ECP/EPP modes.

Another drawback to using the motherboard-based printer port is the possibility of damaging the port while debugging. Damage can occur if you leave the project's read buffer turned on while the parallel port outputs are turned on. If the output drivers on the printer port are destroyed, the motherboard is ruined.

I've debugged the programmer project and you may not plan to experiment with other parallel-port projects, but it's still possible for a wiring error to cause bus contention. You can also get bus contention if you plug the programmer into a port that isn't bidirectional.

That's why I connected the programmer to a separate printer port board that costs about $20 and plugs into an ISA slot. The motherboard printer port is addressed at x378 and the add-in card is addressed at 0x278. If one of my experiments damages the output driver on the add-in card, I can just throw it away and put in a new one. The add-in card doesn't support ECP or EPP modes, but it does support bidirectional mode, which is all this project requires.

PARAxx.ZIP (*xx* is the version number) is a program that checks the parallel ports and tells you what modes they operate in. The program is available from the Parallel Technologies web site. You can also get it from any of the Simtel sites. (Simtel is a collection of Internet shareware.)

That's all there is to it. Now you can begin experimenting with the AVR-series microcontrollers. ■

Stuart Ball works at Organon Teknika, a manufacturer of medical instruments. He has been a design engineer for 18 years, working on projects as diverse as GPS and single-chip microcontroller designs. He has also written two books on embedded-system design. You may reach him at sball85964@aol.com.

SOFTWARE

The software and parts list for this project are available for download via the *Circuit Cellar* web site.

RESOURCES

Parallel Technologies, ftp.lpt.com/parallel
Simtel, www.simtel.net

SOURCES

AT90S*xxx* micros
Atmel Corp.
(408) 441-0311
Fax: (408) 436-4200
www.atmel.com

ZIF sockets
Digi-Key Corp.
(218) 681-6674
Fax: (218) 681-3380
www.digikey.com

3M
(800) 364-3577
(651) 737-6501
Fax: (800) 713-6329
www.3m.com

Aries Electronics, Inc.
(908) 996-6841
Fax: (908) 996-3891
www.arieselec.com