

# F

or something so central to our comfort, temperature is rather ho-hum. Most of its mysteries were solved long ago. Nonetheless, Dallas Semi-

conductor has something new—the DS 1820. This 3-wire temperature sensor offers digital output. You don't need an A/D converter!

The DS 1820 is accurate to approximately 1 °F, and has a range of -67 to 257°F. In addition, it has high and low alarms, each one with a unique serial number.

It's a little tricky to do all that with only three pins, but the protocol is not too bad once you get used to it. All the action happens on one pin which is a bidirectional data bus.

First, a long pulse prepares the DS 1820 for a command. Then you send it bits: a short pulse for a 1 and a slightly longer pulse for a 0. Of course, some commands ask for data to be sent back. To read data, you send out short pulses like you used when you sent it a 1. But, since the data line is in a wired-OR arrangement, it stretches the pulse you sent when it wants to return a 0, and it does nothing to send a 1.

As long as it is above a certain minimum, it doesn't matter how long you wait between bits. The critical timing starts over with each bit, and small timing errors don't accumulate.

I immediately wanted to hook the DS 1820 to my PC since it seemed

# An RS-232 Thermometer

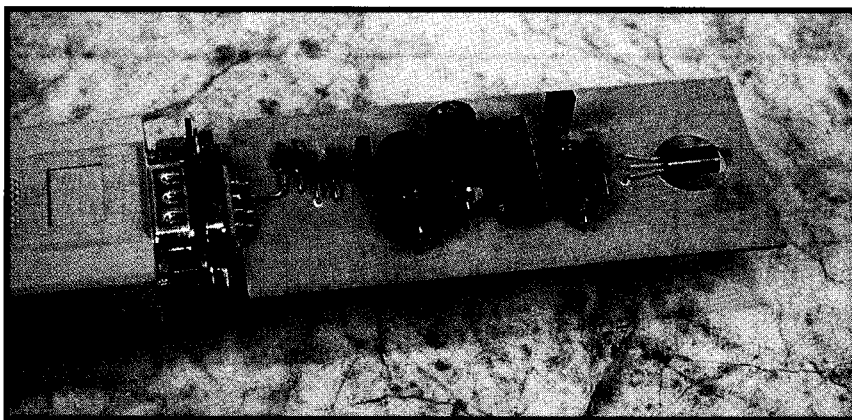
a perfect sensor for home-automation projects. I first thought I'd use a parallel port interface since I could practically solder it right to a DB-25 connector and start talking to it. But, as the old Russian proverb says, "the devil is in the details."

The DS1820 needs pulses of 15- $\mu$ s, which is just enough time for a few machine instructions. It's not enough to call a system time function and wait for results. Instead, you have to craft a delay loop for a specific CPU and clock speed or talk directly to the spare timer (if it's spare on your system).

And then there are interrupts. If an interrupt occurs while you're talking to the DS 1820, it messes up the timing. Although it wouldn't happen often, the possibility makes me nervous. You could disable interrupts while you talk to the sensor, but that's a software nightmare. Besides, I only have one parallel port and my printer is hooked to it.

DON **McLANE**

We've covered the Dallas Semiconductor **DS1620** digital thermostat chip before, but it's overkill if all you want is to read temperature. The DS1620 is a neat little **3-pin** chip that does the job nicely. Check out some tricks for interfacing it to a standard serial port.



**Photo 1:** Only one connection was needed to the top row of pins on the DE-9 connector. To avoid needing a double-sided PC board, solder D1 right to the connector as shown.



Serial ports also have problems, primarily because RS-232 uses such awkward voltages. But, if you set a serial port to the right baud rate and send the right character, you can produce a pulse that can be accurately controlled. The critical timing is done in the serial chip. Nothing can interrupt it. And, it's the same from system to system. So, even though it takes a little more hardware, I can interface the DS 1820 to a serial port. Since I have a free serial port, this solution is optimal.

Usually, I try to keep a design straightforward and avoid anything clever. But, since the details fell into place, I couldn't resist it. And, you're right-no serial port was intended to be used like this.

I'm doing this project in DOS so you can test and play with the circuit. Ultimately, I'll use a multitasking operating system since it makes more sense. I'll start home-automation tasks in the background and then go on to other things. Such a proposition is awkward in DOS, unreliable in Windows, but rock-solid with OS/2 and Linux. Since this is the direction I'm headed, I've designed the hardware to go with me.

## THEORY OF OPERATION

RS-232 voltage levels are nominally  $\pm 12$  V. In actuality, they range from  $\pm 8$  to  $\pm 12$  V, most often coming closer to  $\pm 8$  V. So, when I talk about  $\pm 12$  V here, interpret it loosely.

As you can see in Figure 1, I'm stealing power from a couple of the RS-232 status lines. If I set RTS high, I can get +12 V from it. If DTR is low, I have a -12-V source. Diodes D1 and D4 protect the circuit at the times that these signals are incorrect. Most of the load comes in pulses, and capacitors C1 and C2 smooth out the sudden changes.

The DS 1820 requires a +5-V supply, which I obtain from the +12-V supply with a 78L05 regulator. As any good temperature sensor, it draws minuscule current (to prevent self-heating).

R1 is the pull-up for the wired-OR data line. Most of the time it holds the line at 5 V. The data line can be pulled down by either Q1 or the DS1820. A wired-OR is invoked when they both pull down at once.

The computer's data line (TD) is usually at -12 V, thus Q1 is off. Q1 is an N-channel enhancement device and doesn't turn on until its gate goes positive by a few volts. When the computer sends data, the signal swings to +12 V, turning Q1 on.

## Commands for the DS1820 Digital Thermometer

- 33h Read ROM-read the unique 48-bit serial number. Follow by reading 7 bytes.
- 55h Match ROM-address a single DS1820 when there are multiple devices on the 1-wire bus. Follow with 8-byte address.
- CCh Skip ROM-skip addressing. If there's only one device on the bus, you don't need to use an address.
- F0h Search ROM-identify devices on bus.
- ECh Alarm Search-search for a device where the temperature is out of limit.
- 4Eh Write Scratchpad-write to the locations that store the low- and high-alarm thresholds. Follow with 2 bytes of data.
- BEh Read Scratchpad-read memory. Follow by reading (up to) 9 bytes (see memory map).
- 48h Copy Scratchpad-copy thresholds (memory locations 2 and 3) to nonvolatile memory.
- 44h Convert Temperature-take a temperature reading.
- B8h Recall E2—take the values from the nonvolatile memory and put them into the alarm trigger registers (memory locations 2 and 3).
- B4h Read Power Supply-power-supply mode. Read one bit.

Memory map:

| Offset | Register             | Offset | Register            |
|--------|----------------------|--------|---------------------|
| 0      | Temperature LSB      | 5      | Reserved            |
| 1      | Temperature MSB      | 6      | Counts remaining    |
| 2      | High alarm threshold | 7      | Counts per degree C |
| 3      | Low alarm threshold  | 8      | CRC                 |
| 4      | Reserved             |        |                     |

In addition, there are 2 bytes of nonvolatile memory.

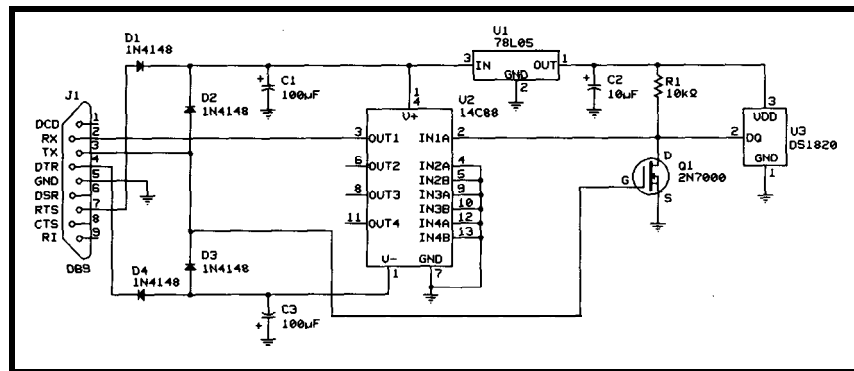
Since Q1 is a CMOS device, I don't want its gate hanging open when the circuit is unplugged from the computer since a static discharge could zap the transistor. That's why I clamped the gate to the power supply through diodes D2 and D3. The data line then helps power the circuit.

U2 is an RS-232 driver, providing a CMOS version of the more common MC-1488. Since it's CMOS, it doesn't use much power, which is critical. It also pulls rail-to-rail (i.e., its output goes all the way to the power-supply voltages). Since I'm stealing power from signal lines, my output needs to be about the same as my power. I've already lost a diode drop from D1 and D4. I don't want to give up any more voltage.

U2 actually has four drivers in it, although I only use one. While throwing three drivers away seems like a waste, it was the best part I could find for the job. Since CMOS gates only draw current when they're switching, the unused drivers draw insignificant current. I need to ensure the unused inputs are grounded so the gates don't oscillate and draw substantial current. (My first prototype demonstrated this, but don't tell anybody I did something so foolish.)

## CONSTRUCTION

I chose to use a regular DE-9 connector and tit the circuit board



**Figure 1:** Interfacing to the awkward voltages of the RS-232 interface is difficult. Thanks to low-power components, this circuit is powered directly from the signal voltages.

**Listing 1:** Notice the striking simplicity C++ brings this main() function. You just declare an instance of the class (i.e., tempor) and then use it. All the details are "under the hood" of the class.

```
class DS1820 {
  char save_IER, save_MCR, save_LCR, save_speed;
  void DScommand(const uchar *to_sensor, uchar *from_sensor);
  void DSreset(void);
  DSsend(const uchar *to_sensor, uchar *from_sensor);
  double tempcalc(uchar *buff);
  cksumOK(uchar *buff);
public:
  DS1820(void); // constructor
  ~DS1820(void); // destructor
  double Ctemp(void); // return Centigrade temp
  double Ftemp(void); // return Fahrenheit temp
};

... ugly code deleted

void main()

  DS1820 tempor;
  printf("Hit a key to exit\n");
  while (!kbhit())
    printf("temp = %5.1f \r", tempor.Ftemp());
  getch(); // swallow char
}
```

between the two rows of pins. A right-angle connector might have been stronger mechanically, but I found the symmetry more attractive. I also wanted to use a single-sided PC board, but I needed one connection to the top row of pins. So, one end of D1 is soldered directly to one of the top pins (see Photo 1).

The bottom row of pins on the DE-9 connector is soldered directly to pads on the PC board. To fasten the top side, glue a couple of pins to the PC board (epoxy or the thick cyanoacrylic glue works well).

## SOFTWARE

I've written a program to read and display temperature (see Listing 1). I've included a compiled binary, which executes with **RUN\_1820**. It's compiled for COM2. However, if you're using a different COM port, a comment in the source code tells you how to change the source code. Just recompile and run.

The source is written in C++ and compiled with Borland's Turbo C++. Initially, I wrote it in C, but I found myself saying things like "this stuff should be put in a constructor" and

"these variables should be private to a couple functions."

Finally I gave in, renamed the source to **.CPP** and made a quick conversion. I'm glad I did since the resulting code is cleaner. While I've always appreciated C++ in large systems, I never expected to like it for low-level stuff like this. But, C++ seems well suited. I guess I'm coming around.

DOS doesn't provide any operating system calls to set the serial port to 115.2 kbps, so I need to program the 8250 serial chip directly. Here's how you talk to the sensor.

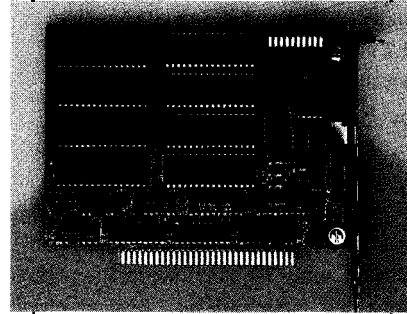
You first send it a reset pulse lasting 480-960  $\mu$ s. To do this, I set the serial chip to 9600 bps and sent a zero. An RS-232 data line idles in the 1 state when no characters are being sent. A character always begins with a start bit at the 0 state. The RS-232 drivers invert this, so the 1 state is -12 V and the 0 state is +12 V.

We have a start bit and eight 0 bits, which adds up to one long pulse. At 9600 bps, it gives us a pulse of 938  $\mu$ s. The pulse is echoed to the computer, so I receive the 0 that I sent. Just read the character to clear the register in the serial chip. You don't need to do anything with the received data.

The DS 1820 responds with a presence pulse lasting 60-240  $\mu$ s.



**VMAX**<sup>®</sup>  
**QUALITY PRODUCTS**  
**RESPONSIBLE SERVICE**  
**RELIABLE DELIVERY**



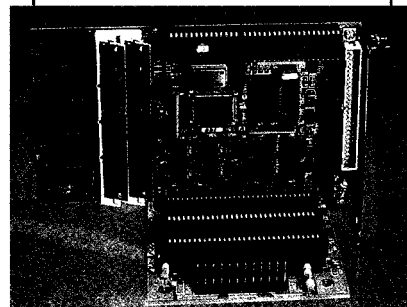
### ENHANCED SOLID STATE DRIVE — \$164\*

4M Total, Either Drive Bootable  
 1/2 Card 2 Disk Emulator  
 Flash System Software Included  
 FLASH & SRAM, Customs too



### 486 SLAVE PC — \$895\*

Add up to 4 Boards to One Host PC  
 Fast Data Transfer and I/O  
 PC-I 04 Port, IDE & Floppy Control  
 Independent Processors on One Bus  
 No Special Compilers Needed



### TURBO XT w/FLASH DISK — \$266\*

To 2 FLASH Drives, 1 M Total  
 DRAM to 2M  
 Pgm/Erase FLASH On-Board  
 CMOS Surface Mount, 4.2"x6.7"  
 2 Ser/1 Par, Watchdog Timer

All Tempustech VMAX<sup>®</sup> products are  
 PC Bus Compatible. Made in the  
 U.S.A., 30 Day Money Back Guarantee  
 \*Qty 1, Qty breaks start at 5 pieces.

**TEMPUSTECH, INC.**

**TEL: (800) 634-0701**

**FAX: (941) 643-4981**

Fax for 295 Airport Road  
 Fast response! Naples, FL 33942

Read the serial chip again to clear the character. It won't be a 0 because it's too short, but it doesn't matter what it is.

Send the DS 1820 commands a bit at a time. To send a 0 bit, I need to send a pulse lasting 60–120  $\mu$ s. If the serial chip is set at 115.2 kbps, a zero byte (again, a start bit and 8 data bits, all low) gives a pulse of 78  $\mu$ s.

To send a 1 bit, I send a shorter pulse lasting 1–15  $\mu$ s. If I send a character with all bits high at 115.2 kbps (i.e., FFh), then I only send a start bit of about 9  $\mu$ s. The commands are sent a bit at a time, least significant bit first. After each bit, I read the serial chip to clear the receive register (I still don't care about what's read). Typically, I then send the command CCh and then 44h.

You could have more than one DS 1820 connected in parallel. In our case, there's only one and the CCh command tells the chip to skip the addressing protocol. The 44h command tells the chip to begin a temperature conversion.

Within 2 s, the DS 1820 completes a temperature reading. While this is not fast, temperature doesn't change quickly. After reading the temperature, start with a reset

pulse and presence pulse sequence as before. Then send a CCh to skip addressing and a BEh to read the temperature.

To read back values, I send short pulses as though I were sending it a 1. However, I need to look at the character when I read the serial chip. If the DS 1820 wants to send back a 0 bit, it stretches out the pulse to at least 15  $\mu$ s, and if it wants to send a 1 bit, it does nothing. So, if I look at the first bit of the received character, I can get the bit that the sensor is sending. It's tricky, but the timing works out elegantly.

For RS-232, 115.2 kbps is pretty fast. The maximum length of cable you can use is limited to 20'. If you want to push this limit, look for a low-capacitance cable.

## CONCLUSION

Three things have been accomplished:

- 1) you have a circuit you can use or modify
- 2) I've demonstrated a method for interfacing a serial port to a one-wire bus. Serial ports are available on most PCs
- 3) I've included software which provides a starting place to develop your own applications. In particu-



lar, the C++ class may be a helpful example.

Clearly, this project is only a beginning. The rest is up to you.

*Don McLane is an engineer with Syscon International, where he designs real-time data acquisition and control systems for industrial applications. He may be reached at dmclane@delphi.com.*

## SOFTWARE

Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" for downloading and ordering information.

## I R S

- 422 Very **Useful**
- 423 Moderately Useful
- 424 Not **Useful**

# ENGINEERING PROFESSORS

## LISTEN UP!

The reason you read *Circuit Cellar INK* is obvious.

Like thousands of others, you have come to realize that *Circuit Cellar INK* presents technically relevant computer-based applications in a comprehensive and instructive manner. In fact, our projects are presented just the way you'd do them if only you had the time.

Since 1994, we have provided thousands of college students with industry-quality development tools and information. This program is **FREE** and because of the continued success we have experienced, we'd like to extend our offer to the 1995-1996 school year. The fall semester is upon us...

If you're an engineering professor or teach qualified technical students, we'd like to give you and all of your students free subscriptions to *Circuit Cellar INK*. Please tell us how we can contact you directly so you don't miss a single issue.

**WE ARE HELPING AMERICA REGAIN THE COMPETITIVE EDGE. WE'RE HOPING YOU WILL TOO!**

Rose Mansella, Circulation Coordinator  
Circuit Cellar INK  
4 Park St.  
Vernon, CT 06066

Tel: (860) 875-2199  
Fax: (860) 872-2204  
Internet: [rose.mansella@circellar.com](mailto:rose.mansella@circellar.com)