Automated BBQ Temperature Control

chapter 8

Before employing any captured artifacts or machinery, I will carefully read the owner's manual.

Peter Anspach,
The Evil Overlord List

in this chapter

- ☑ Controlling temperature for barbeque
- ☑ Closed loop temperature control systems
- ☑ Temperature sensors
- ☑ Fire control actuators
- ☑ Java processing and software
- ☑ Proportional-integral-derivative control algorithms

This is, unabashedly, a chapter for your inner carnivore. We did find nearly 300 hits on Google searching for "smoked vegetables", but that was out of curiosity more than anything. We like smoked barbeque a lot — sidebars in this chapter show you our favorite recipes for pulled pork and for beef brisket — but to be right it has to cook on a very low heat for a long time to let the fat render out. We also like the flavor we get by smoking with charcoal logs — an electric smoker just doesn't do the job right.

That combination, long, slow cooking with charcoal, is hard to do well. We keep the smoker at around 225°F, a temperature so low that it's easy for the fire to go out. You can do it if you're willing to tend the fire constantly, but we're not that patient.

Project Description

Instead of tending the fire yourself, a small control system can tend it for you, controlling the air flow to maintain a set temperature. You could enhance our design to warn you to add fuel during a very long smoke cycle.

Figure 8-1 shows the smoker we use. It's a slight enhancement of the common horizontal design, hanging a separate firebox on the left of the main smoker. The separate firebox keeps the direct radiant heat from the coals away from the meat, letting the hot smoky air do the cooking. The placement of the firebox lower than the smoker causes the upwards convection flow of the heated air to draw air into the smoker and then up the flue. The air inlet and flue damper are manually adjustable to control airflow and therefore the rate of combustion, which determines the internal smoker temperature.

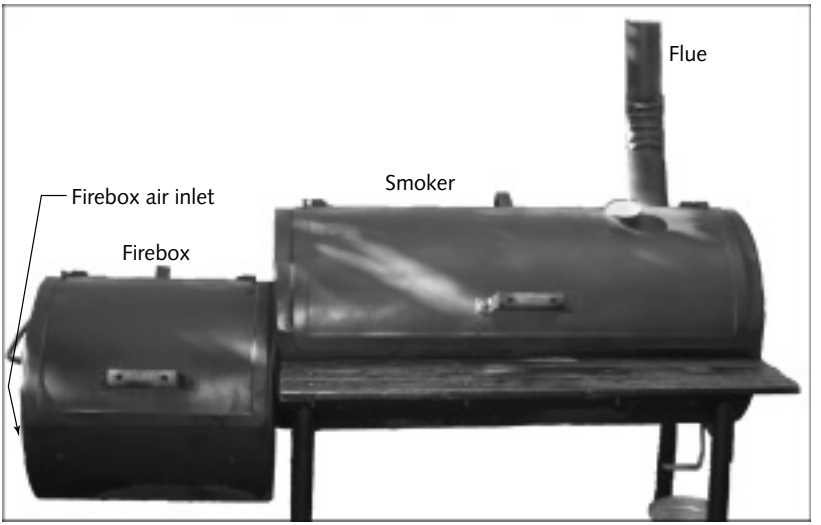

FIGURE 8-1: Horizontal smoker

Pulled Pork

Make a brine from 1 cup dark molasses, 1 ½ cups Kosher salt, and 2 quarts water. Use bottled water if your tap water tastes bad. Make the brine in a 2 to 2 ½ gallon food storage bag, then put a 6–8 pound Boston butt (it's part of a pork shoulder — ask your butcher) into the brine. Put the bag into a large bowl for support and to guard against leaks, and squeeze the air out of the bag so the brine completely covers the meat. You'll need to get the right size bowl to make this work easily. Let the meat soak in the refrigerator for 8–12 hours.

Remove the butt from the brine, dry it, cover with plastic or foil, and bring to room temperature. Make a rub by first grinding 1 ½ teaspoons whole cumin seed, 1 ½ teaspoons whole fennel seed, and 1 ½ teaspoons whole coriander to a powder. Add 1 tablespoon chili powder, 1 ½ tablespoons onion powder, 1 tablespoon paprika, and 1 tablespoon hot paprika. Sprinkle the rub evenly over the butt, patting it in and working to make sure it all sticks to the meat. We use a spoon to sprinkle the rub on, cover the meat and the rub with plastic wrap, and pat in the rub through the plastic. Turn the assembly over and repeat. Use all the rub for the spiciest results, less for a milder version.

Remove the wrap and smoke the meat at 225°F for 8–12 hours, checking to see if it's done after 8 hours. It's done if it pulls apart easily with a fork. If you run out of time, smoke it at 225°F for 3-4 hours, then cover with foil and bake for 3 hours in an oven preheated to 300°F. Once it's done, let the meat rest and cool for at least an hour covered with foil, then pull apart into bite-sized chunks.

Serve with a mustard sauce made from 3 ounces Dijon mustard, 3 ounces sweet pickle juice, 1 ounce cider vinegar, and Tabasco to taste.

Beef Brisket

Make a rub from 4 tablespoons dark brown sugar, 2 tablespoons chili power, 2 tablespoons paprika, 1–2 tablespoons hot paprika, 2 tablespoons Kosher salt, 2 tablespoons garlic powder, 2 tablespoons onion powder, 1 tablespoon black pepper, 1 tablespoon white pepper, 1 tablespoon cayenne pepper, 1 tablespoon dry mustard, and 1 tablespoon finely ground cumin. Coat a 6–8 pound trimmed beef brisket with the rub, then wrap tightly in plastic and marinate in the refrigerator for at least 8 hours (overnight is best).

Bring meat to room temperature, then unwrap it and smoke at 225°F until it's done, which is when it reaches an internal temperature of at least 175°F. Do not cook too quickly or for too short a time, because the meat needs time at heat for the fat to drip out. Let the brisket rest for 20 minutes before slicing against the grain into thin slices.

If you replace the chili powder, cumin, and mustard with 4 tablespoons dried thyme, 4 tablespoons dried oregano, and a tablespoon of ground coriander, the rub works well on pork ribs or a pork shoulder.

System Diagram

Figure 8-2 shows the smoker system as a block diagram. Air enters the system at the left and exits at the top right. Heat is generated in the firebox and exits everywhere; the amount of heat generated is determined by the amount of charcoal in the firebox and the volume of air moving past the charcoal.

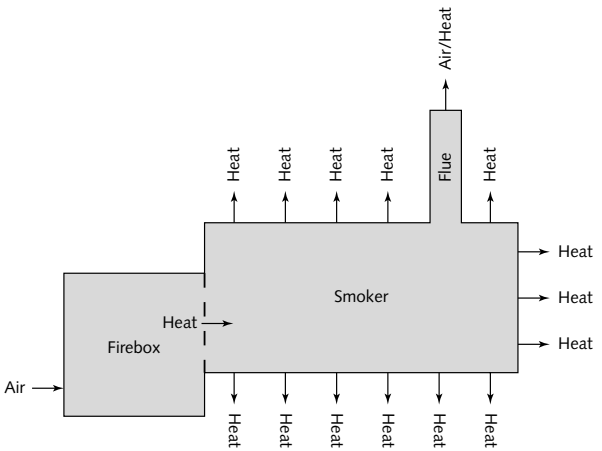

FIGURE 8-2: Smoker system block diagram

Under pure convection, the volume of air moving through the system is itself determined by the heat differential between the air in the system and the outside air, and by the size of the inlet and flue openings. One option to control the internal temperature is, in fact, to physically move one or the other of the dampers in the openings, but because that approach requires knowing the absolute position of the damper as well as being able to move it, we chose another approach — we decided to set the dampers so the system had slightly too little air flow, then use a fan to push air through the too-restrictive opening. That approach gives the control system the ability to reduce and increase heat, and requires only that we control the speed of a fan.

The overall control system has the block diagram shown in Figure 8-3. We used the Javelin Stamp processor to run the software, giving it a thermistor as a sensor and a variable speed fan as an actuator. There's a balance between the damper setting and the size of the fire you build in the firebox. Getting that setting right is a matter of trial and error, but the control software can help by telling you it can't stabilize the system and indicating whether the damper needs to be opened further or closed.

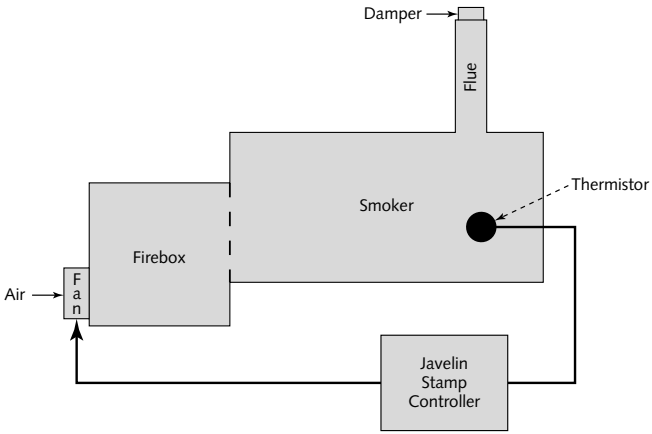

FIGURE 8-3: Control system block diagram

Sensor

The two most common sensor types for applications in this temperature range are thermocouples and thermistors. A *thermocouple* is a junction of two dissimilar metals, a bimetallic junction, which produces a voltage related to the temperature. Connecting wires to the thermocouple leads form additional bimetallic junctions between the metals forming the thermocouple and the copper wires, generating spurious voltages. Correcting out those additional voltages requires knowing the temperature at the additional junctions, increasing the hardware complexity.

A *thermistor* is a thermally variable resistor, a device whose resistance varies with temperature. We designed this project to use a thermistor because it's much simpler to implement in hardware, and because we could get one already packaged to survive barbeque conditions. There are two basic types of thermistor, ones whose resistance goes up with increasing temperature and ones whose resistance goes down with increasing temperature. The former are *positive temperature coefficient* (PTC) thermistors, the latter *negative temperature coefficient* (NTC) ones. Thermistors are relatively sensitive, producing a relatively large resistance change per degree of temperature change. Unfortunately, they're highly nonlinear over their full operating range, obeying the following equation:

$$R = R_0 e^{\beta\left(\frac{1}{T} - \frac{1}{T_0}\right)}$$

We're interested in temperatures at approximately 225°F, so we measured the resistance of the sensor we used over the range 210–240°F with the results shown in the *Resistance* column of Table 8-1. (We'll come back to the right-hand column shortly.)

Table 8-1 Measured Thermistor Sensor Resistance

Temperature Reading (Degrees F)	*Resistance (K Ohms)*	*Voltage Drop on 10K Ohm Resistor*
210	10.50	2.44
211	9.90	2.51
212	9.60	2.55
213	9.40	2.58
214	9.21	2.60
215	9.01	2.63
216	8.84	2.65
217	8.68	2.68
218	8.53	2.70
219	8.43	2.71
220	8.29	2.73
221	8.18	2.75
222	8.05	2.77
223	7.95	2.79
224	7.84	2.80
225	7.70	2.82
226	7.52	2.85
227	7.33	2.89
228	7.25	2.90
229	7.10	2.92
230	6.98	2.94
231	6.77	2.98
232	6.66	3.00
233	6.58	3.02
234	6.51	3.03
235	6.38	3.05
239	5.98	3.13
240	5.76	3.17

Circuits to measure the resistance of a thermistor are shown in Figure 8-4. In both circuits, the thermistor in series with a fixed resistor forms a voltage divider across the regulated 5-volt supply. The circuit on the left measures the variable voltage across the thermistor directly; the one on the right measures across the fixed resistor. The right-hand circuit helps linearize the measurements versus temperature, so it's the one you should use. The right-hand column in Table 8-1 is the voltage measured across a 10K Ohm fixed resistance using the right-hand circuit.

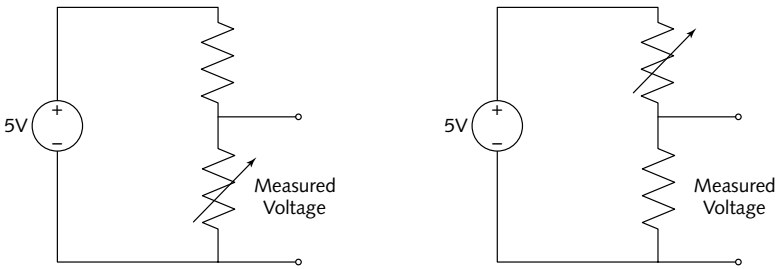

FIGURE 8-4: Thermistor resistance measurement circuits

Figure 8-5 graphs the data from Table 8-1. The dashed, downward-sloping line is the calculated curve fitting the thermistor equation for the unit we used, and has the parameters T_0 = 210°F, R_0 = 10.5K Ohms, and β = 1008.74. The straight downward-sloping line is a linear curve fit to the measured data (the equation parameters are just below the line on the right), while the points overlaying the linear curve fit are the measured data itself. The 10K Ohm resistor isn't too far off the measured resistance at the 225°F target temperature, erring on the low temperature side, and is a readily available value.

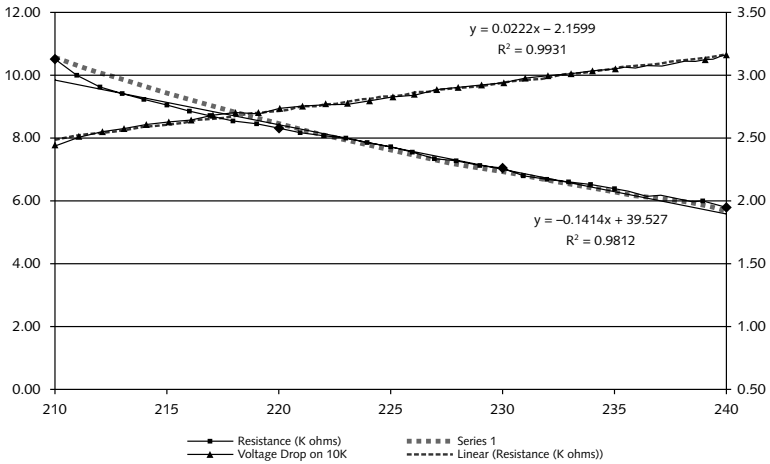

FIGURE 8-5: Thermistor data plot

The rising line and points in Figure 8-5 are the voltages measured using the right-hand circuit in Figure 8-4 using a 10K Ohm fixed resistor. The corresponding linear curve fit equation is shown in the upper right corner.

The plots are relatively linear over the operating region we care about — the measurements across the fixed resistor even more so — so we didn't bother to include any other logic or circuitry to address the nonlinear properties of the thermistor. (We also clamped the measured voltage between 2.5 and 3.2 volts to ensure that our computations stayed in the linear region — see the software section — and did a quick calculation of the load current, less than a third of a milliamp, to be sure it wouldn't overload the Javelin Stamp supply.)

Actuator

Even though we discarded the idea of moving either the flue damper or the firebox damper (Figure 8-6) in favor of using a fan to control the airflow through a restricted port, we still had engineering challenges to solve in creating the actuator our control system requires. Foremost among them were how to mount the fan on the barbeque and how to keep the fan within its maximum heat rating.

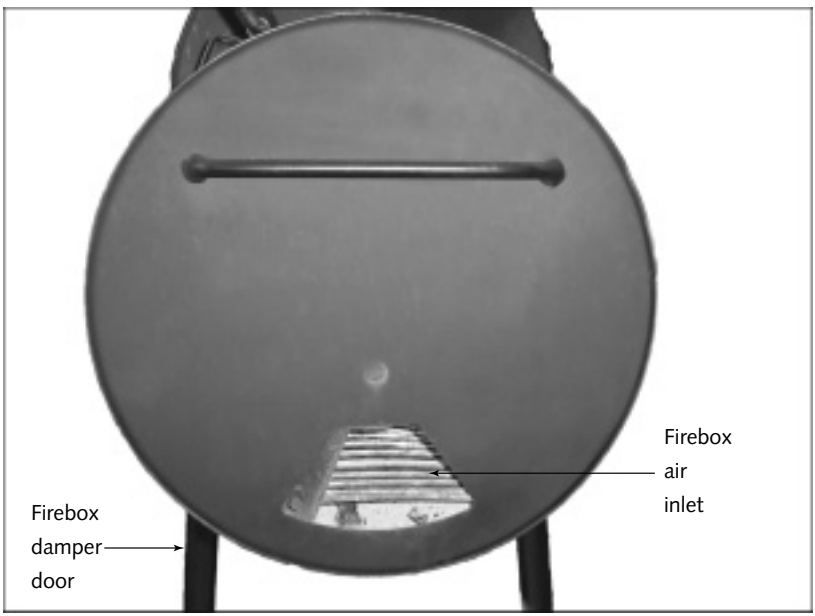

Figure 8-6: Firebox inlet

Those two challenges are related, of course, because mounting at the irregularly shaped firebox inlet is harder than mounting on the vertical, cylindrical flue, but the firebox inlet lets the fan operate in ambient temperature air, while the flue requires that the fan operate in the high temperature, smoke, and possibly grease-laden exhaust gases.

We searched the Web for a long time for high-temperature DC fans — we chose a DC fan to make it simple to interface and control with a microprocessor — and simply could not find any that were both sized for the 3-inch flue diameter on our smoker and priced reasonably. A Google search for *fan dc "high temperature"* returned the best results we found, and we then searched both the Thomas Register and the W.W. Grainger site (`www.grainger.com`) without success. We found a wide variety of fans such as you'd use to cool a computer case, but they all had maximum temperature ratings of 158°F (70°C). Even running as cool a smoker as we planned, we have to handle air temperatures up to 300°F (149°C), so one of these fans mounted on the flue won't last long.

That problem, more than any other factor, drove us to mount the fan at the firebox inlet, where the fan would operate at ambient outside air temperatures and the only temperature problems we'd face would be thermal isolation from the fire itself. The design also has to ensure the airflow remains slow enough that it doesn't blow ashes into the smoker.

The remaining actuator design element is how to drive the fan over a range of speeds. It's simple to output a pulse width modulated (PWM) signal from the Javelin Stamp — there's code built in to do the job — so all that's required is to provide a driver transistor to handle the power levels required by the fan motor. The fans we looked at typically required less than 210 ma at 12 volts. We searched a catalog of transistors readily available at RadioShack and chose the IRF510 MOSFET, rated for up to 60 volts at 3 amps; 20 watts. We wired a diode in parallel with the fan, wired backwards to the current flow, to snub any reverse switching spikes. Figure 8-7 shows the fan drive circuit; the PWM Drive signal connects directly to the output pin on the Javelin Stamp.

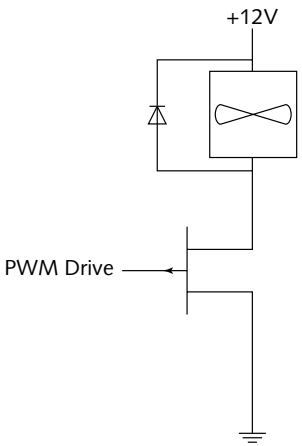

FIGURE 8-7: Fan drive circuit

Parts and Designs

While we were developing this project, we found a complete product that did much the same thing, the BBQ Guru from HomeBBQ.com. It's no longer available on their site (`shop.homebbq.com/pd_bbq_guru.cfm`), but you can see what it was using the Wayback Machine (`web.archive.org/web/20040224101508/http://shop.homebbq.com/pd_bbq_guru.cfm`).

Our initial plan for the project used a PC, simplifying the software by making a highly capable platform available for processing and display. A laptop would be convenient to take outdoors, and could use a plug-in probe and fan. PC-interfaced thermometers are available, so the remaining electronics design work would be what's needed to drive the fan. We thought a while about using an expensive laptop where it was likely to fall or get spilled on, though, and decided a standalone controller with its own embedded processor was a better idea.

Sensor

We wanted a sensor that we could use as either an ambient air temperature probe within the smoker or as a meat probe, looking forward to other projects in which we'd extend the calibrated range of measurements beyond the 210–240°F range. We happen to have a wireless barbeque thermometer with an appropriate probe (see Table 8-2), so we designed our circuits around it. We've also seen probes as part of oven thermometers and other equipment at Bed Bath and Beyond, and at Super Target, so look around.

Table 8-2 Temperature Probe

Sensor	*Source*

Pal Products
Nu-Temp model NU-701 Simple Wireless BBQ Thermometer
	Nu-Temp
	`www.nu-temp.com/701.htm`
	
	Probes available at
	`http://www.nu-temp.com/replace.htm`

The probe (Figure 8-8) is a negative temperature coefficient (NTC) thermistor encased appropriately for food use and with a high-temperature cable.

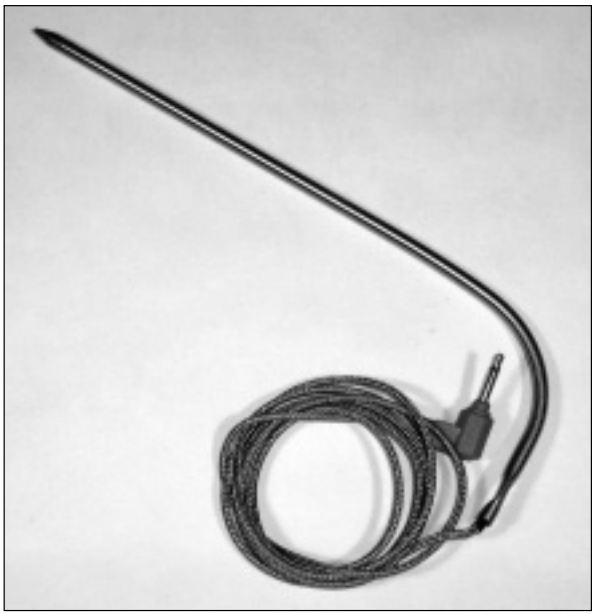

FIGURE 8-8: Probe

Table 8-1 records our measurements using this probe. We used the sigma-delta analog-to-digital converter (ADC) capability of the Javelin Stamp processor to read the probe, connecting them together in the circuit shown in Figure 8-9. The probe itself is in series with a 10K Ohm resistor, as in Figure 8-4; the junction of the two is the measurement point for the ADC. The sigma delta converter works by summing the two voltages fed to the integrating capacitor in Figure 8-9. One voltage is the output of the resistive divider formed by the thermistor and fixed resistor; the other is the direct 0 or 5-volt output of a pin on the Javelin Stamp. The software in the Javelin Stamp modulates the pulses on that output pin to cause the input pin, connected directly to the integrating capacitor, to settle at the 2.5 volt logic threshold.

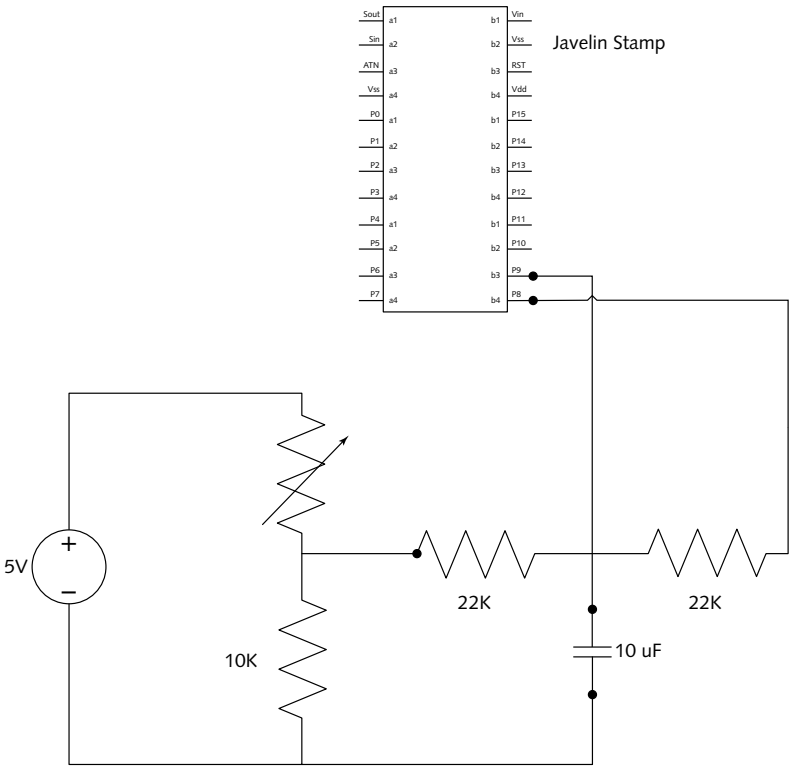

FIGURE 8-9: Temperature probe measurement

The result of the sigma delta ADC software processing is a count from 0 to 255, corresponding to the duty cycle on the output pin. A value of zero corresponds to zero volts from the divider; a value of 255 corresponds to 5 volts.

Actuator

We tried to find a 12-volt DC fan designed for outdoor use, but without success, so ultimately we settled for a simple 120 mm fan originally designed for cooling PC cases. Although most any 120 mm fan would do (smaller fans likely won't move enough air), we really like the RadioShack unit we list in Table 8-2 because of its metal body. The fan blades are still plastic, though, so it can't get too close to the firebox even though it's working at the inlet. No matter what fan you use, they're inexpensive enough that should the fan fail after being used beyond its intended temperature ratings, the replacement cost is small. Put connectors in the wiring between the fan and controller to simplify replacement.

We mounted the fan in a foot-long, square cross section metal pipe we made using four pre-bent galvanized step flashing pieces we found at The Home Depot (Figure 8-10). We used high-temperature aluminum foil tape to hold the sections together and sprayed the assembly with high-temperature paint. The pipe guides the air from the fan to the inlet while spacing the fan back from the hot zone around the inlet. You'll want to keep the pipe close around the fan and near the inlet to make sure all the available air flow goes to the smoker. If you're building a permanent brick barbeque, consider building in an air pipe to conduct the airflow from a fan to the fire pit.

Figure 8-10: Fan mounting

We used the power transistor (and heat sink) shown in Table 8-2 to drive the fan, as in Figure 8-7, coupling the gate on the MOSFET to the PWM signal pin on the Javelin Stamp through a 10K Ohm resistor.

We needed a 12-volt supply for the fan. The Javelin Stamp demo board accepts from 6 to 24 volts, so by ensuring we provided enough current from the power adapter, we were able to use a common source. Don't use just any 12-volt AC-to-DC power adapter, because most of the ones you'll find are unregulated and likely put out 13 to 16 volts. We smoked a couple of fans before we thought to actually measure the adapter voltage output. The RadioShack unit in Table 8-3 is regulated to a selectable voltage, making it useful for a variety of projects, and outputs 0.8 amps — enough to power the fan shown (0.32 amps) and the Javelin Stamp (0.3 amps).

Table 8-3 Actuator

Component	*Source*
RadioShack Catalog #: 273-238 4″ Cooling Fan	RadioShack www.radioshack.com/product.asp?catalog%5Fname=CTLG&product%5Fid=273-238
RadioShack Catalog #: 276-2072 IRF510 Power MOSFET Transistor	RadioShack www.radioshack.com/product.asp?catalog%5Fname=CTLG&product%5Fid=276-2072
RadioShack Catalog #: 276-1368 Heat Sink	RadioShack www.radioshack.com/product.asp?catalog%5Fname=CTLG&category%5Fname=CTLG%5F011%5F002%5F008%5F000&product%5Fid=276%2D1368&site=search
RadioShack Catalog #: 273-1667 3-12VDC/800mA Regulated AC-to-DC Adapter	RadioShack www.radioshack.com/product.asp?catalog%5Fname=CTLG&product%5Fid=273-1667 You'll also need power leads to connect to the project; see these: Hobby Power Leads Adaptaplug http://www.radioshack.com/product.asp?catalog%5Fname=CTLG&product%5Fid=273-1742

Processor

We'd far rather program in reasonable, object-oriented languages, so given that the Parallax Javelin Stamp offered both that and the peripheral device support we needed — the sigma delta analog-to-digital converter and the pulse width modulation output — it was a simple choice to make. We hosted the prototype on the Parallax Javelin Stamp Demo Board, identified in Table 8-4 and shown in Figure 8-11. The solderless breadboard area on the right is convenient for prototyping, and the spacing is such that you can plug in the LCD directly across a set of five rows.

Table 8-4 Embedded Processor

Processor	*Source*
Parallax Javelin Stamp	Parallax `www.parallax.com/javelin/`

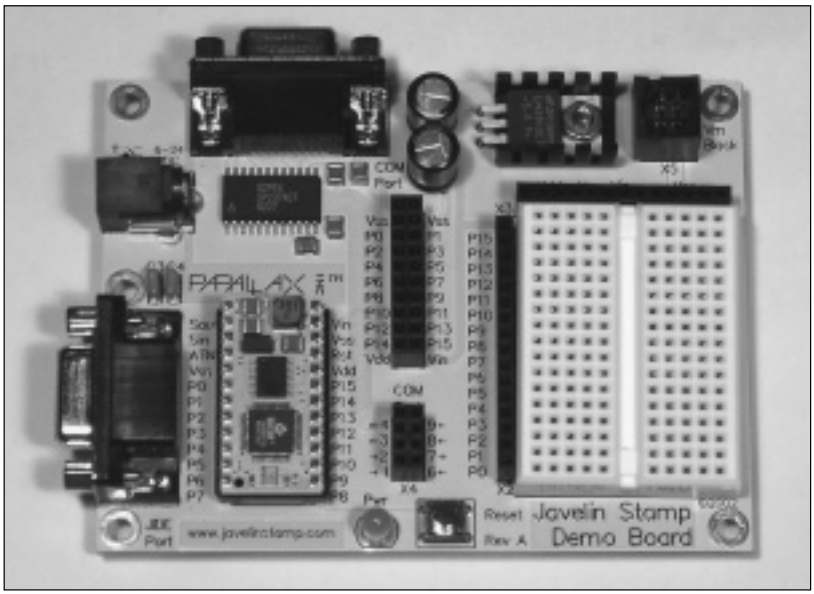

FIGURE 8-11: Javelin Stamp

Display

We used the Parallax BPI-216 Serial LCD Module for the display output (see Table 8-5). The LCD connects to power, ground, and any I/O pin on the Javelin Stamp, and not only provides a 2.4 or 9.6 Kbps, one-wire interface to the processor, it also responds to a number of commands to clear, scroll, and address the cursor to locations on the display.

Table 8-5 LCD Display

Display	*Source*
Parallax BPI-216 Serial LCD Module	Parallax `www.parallax.com/detail.asp?product_id=27910`

Figure 8-12 shows the LCD displaying output from our software. At the point we took the photo, we'd just started up the fire. The ADC was reading a very low value, below the linear region of the thermistor, and the computations were therefore driving the fan to the full speed state.

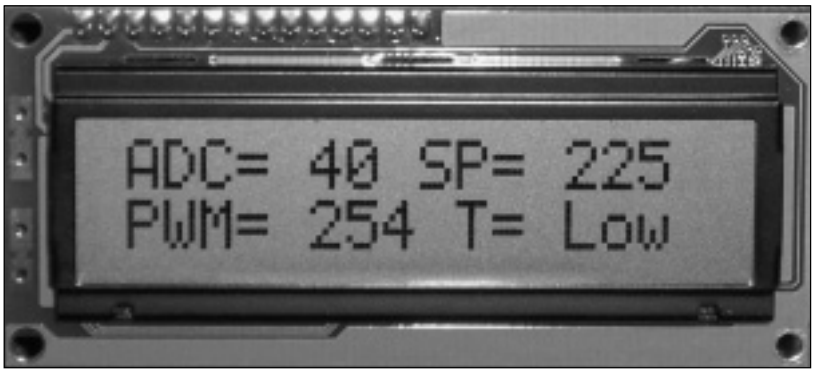

FIGURE 8-12: LCD display

Software

The software for the controller is all in Java for the Javelin Stamp, written to live within that processor's characteristics of no garbage collection (so you don't want to allocate memory or strings as the software runs) and 16-bit integer arithmetic (so you have to pay attention to ranges of values and order of computations).

Sensor

The values from the ADC correlate linearly to the voltage input to the sigma delta converter. Figure 8-13 shows a range of measurements we made, along with a linear trend line and the corresponding equation.

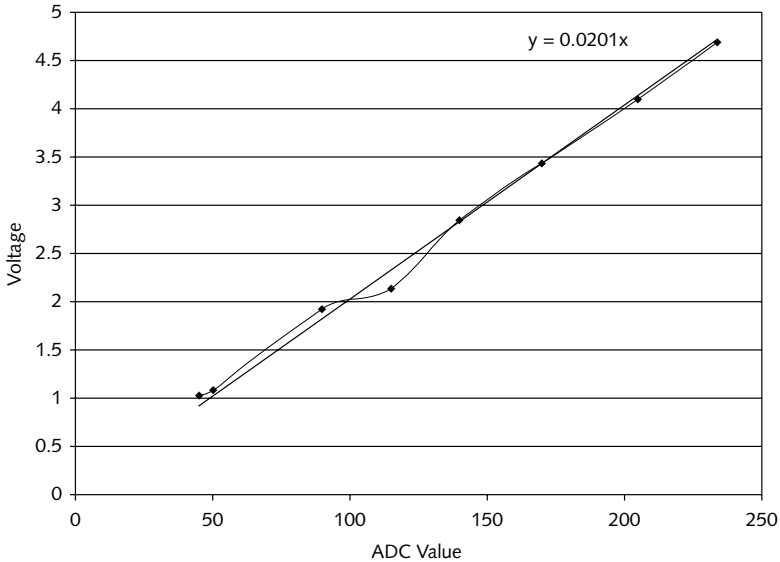

FIGURE 8-13: ADC values versus voltage

The linear relationship between voltage and temperature (over a limited range) shown in Figure 8-5 combines with the relationship in Figure 8-13 to create a linear relationship directly between ADC value and temperature, as shown in Figure 8-14. The actual constant offset (98.151 in the figure) you'll need varies from one sample of the probe to another; you'll probably want to adjust it (and perhaps the variable coefficient too) to match the probe you use. We saw a variation of 5 to 10 degrees among probes.

The essential details you need from Figure 8-14 for software implementation are these:

- The useful range of ADC values runs from 125 to 159.
- The ADC value range corresponds to a temperature range of 211 to 240°F.
- The equation T = 0.8945 * ADC + 98.151 converts ADC values to temperatures.

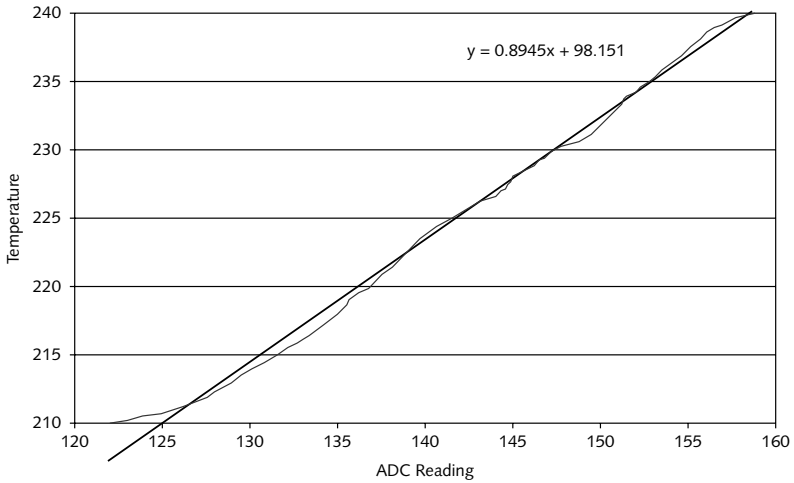

FIGURE 8-14: ADC values versus temperature

The Javelin Stamp only directly implements 16-bit integer arithmetic, however, so the software has to process these equations without floating-point support and without overflowing the 16-bit signed integers. The following code segment converts ADC values to temperatures using the points above:

```
//
//  Constants used to convert ADC readings to other units
//
final static int LOWADC = 125;          // Temp is too non-linear below this
final static int HIGHADC = 159;         // Upper limit of linear readings
final static int TEMPNUM = 178;         // ADC to 100th of deg
final static int TEMPDENOM = 2;         // ADC to (100ths of deg)
final static int TEMPADD = 9815;        // Scaled for 100ths of deg

//------------------------------------------------ ConvertToTemp ----------------
/** ConvertToTemp
 * Converts ADC sample value to temperature in degrees F
 * @nADC - sample value from ADC, range 0 - 255
 * @return - temperature * 100, or +/-1 if out of range
 */
  public int ConvertToTemp( int nADC) {
    int nTemp;

    if (nADC < LOWADC)
      nTemp = -1;
    else if (HIGHADC < nADC)
      nTemp = 1;
    else
```

```
      nTemp = (nADC * TEMPNUM) / TEMPDENOM + TEMPADD;
    return( nTemp );
  }
```

The `ConvertToTemp` routine both thresholds the ADC values, returning ±1 for out-of-range values, and computes the temperature for values in range. The computation multiplies by the numerator of the fraction 179/2 first, before the divide, to ensure there's no loss of precision due to integer truncation after the divide. We derived 179/2 in this sequence:

1. The coefficient for the ADC samples is 0.89, derived from the equation T = 0.89*ADC + 98.
2. We converted 0.89 to 89/100 to make the calculation a sequence of integer operations.
3. Multiplying both factors by 2 keeps the ratio constant, but retains the most accuracy in the calculation by using all the bits available for the multiply. You can check this by noting that the maximum ADC sample is 159, so the largest possible multiplier without overflow is 32767/159 = 206.081, which rounds down to 206. The corresponding denominator would be 206/0.89, or 231.46, rounded to 231.
4. We wanted to maintain the integer temperature in hundredths of a degree (that is, multiplied by 100), and chose to incorporate the factor of 100 in the denominator of the factors. Dividing 231 by 100 with integer truncation creates an error of over 13 percent, so instead we chose the denominator to be 200/100 = 2. The corresponding numerator becomes 200 * 0.89 = 178.
5. Finally, the additive constant gets scaled by 100 to match the units we're converting to, becoming 98.151 * 100 = 9815.

The ADC values themselves are sampled by the Javelin Stamp ADC virtual peripheral; the code to set up and sample it in the `TemperatureProbe` class is as follows (the lines above the `sample` method are globals within the class definition):

```
//
// ADC Connections
//
final static int ADC_IN_PIN = CPU.pin9;
final static int ADC_OUT_PIN = CPU.pin8;

//
// ADC device
//
static int ADCValue;
static ADC voltMeter = new ADC( ADC_IN_PIN, ADC_OUT_PIN );

//------------------------------------------- sample ----------------------
/** sample
 * Read the voltmeter and return the sample value.
 * @returns in int with the value, range 0-255
 */
public int sample() {
  return( ADCValue = voltMeter.value() );
}
```

The ADC updates its measurement every 2.1 milliseconds, far faster than we sample in the controller application.

Actuator

Using PWM fan speed control makes the software to control the fan simple, requiring only that we connect one pin on the Javelin Stamp to the gate on the power transistor, and then drive that pin with a `PWM` virtual peripheral in the code. Here's the software to run the fan (the globals and method are within the `FAN` class):

```
//
// Globals
//
final static int PWMPIN = CPU.pin1;
static PWM pwm = new PWM( PWMPIN, 0, 255 );

//------------------------------------------- update ----------------------
/** update
 * Update the running PWM parameters
 * @nPWM - the new duty cycle setting, 0-255
 */
public void update( int nPWM ) {
  if (0 <= nPWM && nPWM <= 255) {
    pwm.update( nPWM, 255 - nPWM );
  }
}
```

The `PWM` object runs in the background, delivering a constant pulse train to the fan at the duty cycle specified to the update routine.

There's a minimum pulse train duty cycle that will actually rotate the fan; the fan we used didn't rotate for PWM values below 30 or 35. For that reason, we incorporated this logic into the main loop of the controller to keep the motor either shut off or rotating slowly:

```
// Adjust fan minimum -- it doesn't run at less than around 30 or 35
if (nPWM <= FANZERO)
  nPWM = 0;
else if (nPWM < FANMIN)
  nPWM = FANMIN;
fan.update( nPWM );
```

We set `FANZERO` to 5, and `FANMIN` to 35.

Display

We programmed the LCD display (see Figure 8-12) to output the information we found useful debugging and operating the controller. We chose to output the ADC sample values, set point temperature, PWM parameter, and temperature computed from the ADC samples, but it's relatively simple to change what's displayed. All the display code is in a method in the main `BBQController` class:

```
//-------------------------------------------- OutputToLCD ------------------
// Show the readings on the LCD as ADC, volts, and degrees
// Line 1: ADC= xx Set= xxx
// Line 2: PWM= xxx T= Low/High/xxx
  final static int BUFLEN = 16;                          // LCD width
  static StringBuffer buf1 = new StringBuffer( BUFLEN );
  static StringBuffer buf2 = new StringBuffer( BUFLEN );

  public static void OutputToLCD( int nADC, int nTemp100, int nSet, int nPWM ) {
    // Line 1
    buf1.clear();
    buf1.append( "ADC= " );
    buf1.append( nADC );
    buf1.append( " SP= " );
    buf1.append( nSet / 100 );
    buf1.append( "    " );

    // Line 2
    buf2.clear();
    buf2.append( "PWM= " );
    buf2.append( nPWM );

    buf2.append( " T= " );
    switch (nTemp100) {
      case -1:
        buf2.append( "Low    " );
        break;

      case 1:
        buf2.append( "High   " );
        break;

      default:
        buf2.append( nTemp100 / 100 );
        buf2.append( "       " );
    }

    // Output
    LCD.WriteLCD( 1, buf1 );
    LCD.WriteLCD( 2, buf2 );
  }
```

The `BBQLCD` class wraps around the `BPI216` class provided by Parallax; the most significant added functionality in `BBQLCD` is the `itoas` routine, which takes an integer scaled by a power of ten and decodes it to a `StringBuffer`.

The display routine, as written, pads the output strings with extra spaces at the end to ensure all previous characters on the display are overwritten. We originally wrote the code to simply clear the LCD and then write the text, but noticed some objectionable flicker we were able to eliminate with this approach. The text moves left and right a little depending on the number of characters in the first set of digits (the ADC and PWM values); if you found that objectionable, you could use direct cursor addressing and output the display items as four different sequences (two on each line) instead of two (one per line).

PID Loop

What connects temperature readings to fan speed settings is a *Proportional-Integral-Derivative (PID)* control loop. The equation underlying the calculation is

$$Co_{pid} = \left(Kp \times E\right) + \left(Ki \times \int Et\right) + \left(Kd \times \frac{\Delta E}{\Delta t}\right)$$

where

Co = controller output (the fan drive)

Kp = proportional drive gain

Ki = integral drive gain

Kd = derivative drive gain

E = error, computed as reading – setpoint

t = time

ΔE / Δt = change in E over time

Our code implementing this equation is as follows. We compute the error as a percentage over the measurement range (scaled by 100) to normalize the computations, and limit the resulting drive to 100 percent.

```
//------------------------------------------------ compute ----------------------
/** compute
 * Do the PID loop calculation, returning a 0-255 value for PWM use
 * @nTemp100 - current temperature * 100
 * @nSetPoint - what the correct temperature should be * 100
 * @return - PWM value
 */
public int compute( int nTemp100, int nSetPoint ) {
  int nErr;               // Error signal
  int nPctErr;            // Percentage Error
  int nProp;              // Proportional drive
  int nInt;               // Integral drive
  int nIntDelta;          // Integral total change value
  int nDeriv;             // Derivative drive
  int nDrive;             // Total drive
  int nPWM;               // PWM result

  // Calculate percentage error
  nErr = nTemp100 - nSetPoint;    // 100ths of a degree
  nPctErr = nErr / (RANGE / 100 / 2);

  // Calculate proportional drive
  nProp = (nPctErr * KP + ONEHALF) / 100;

  // Calculate integral drive
  nIntegralError += nPctErr;
  if (++nIntegralCount >= nIntegralLimit) {
```

```
    nIntDelta = (((nIntegralError + ONEHALF) / nIntegralLimit) * KI) / 100;
    nIntegralTotalError += nIntDelta;
    nIntegralTotalError = Math.max( -100, nIntegralTotalError );
    nIntegralTotalError = Math.min( 100, nIntegralTotalError );
    nIntegralError = 0;
    nIntegralCount = 0;
  }
  nInt = nIntegralTotalError;

  // Calculate derivative drive
  if (bFirst) {
    nDeriv = 0;
    bFirst = false;
  }
  else
    nDeriv = ((nPctErr - nLastDerivativeError) * KD + ONEHALF) / 100;
  nLastDerivativeError = nPctErr;

  // Total Drive
  nDrive = (BIAS + nProp + nInt + nDeriv);
  nDrive = Math.max( -100, Math.min( 100, nDrive ) );
  nPWM = 127 * -nDrive / 100 + 127;

  return( nPWM );
}
```

The code is a relatively direct implementation of the equation, integrating the error over a predefined number of periods and controlling the computations to maintain accuracy and avoid overflows.

The first element of the control loop to understand is the proportional drive, which simply computes a drive value as a factor times the error. The more the measured value differs from the set point, the greater the drive. Negative error speeds up the fan, while positive error slows it down.

The problem with using only proportional control is that it produces zero drive for zero error — when the measured temperature is equal to the set point temperature. If the fire can't maintain the set point temperature with zero drive, the system temperature will drift off. You can fix that with a bias drive component (the `BIAS` term in the code above, along with the bias introduced by making 127 equivalent to zero drive in the conversion from drive to PWM), but it's subject to error. Integral control drives that error out.

Finally, neither proportional nor integral control responds rapidly to more dramatic changes (such as opening the smoker lid) without overcontrolling the system. Derivative control solves that problem by adding drive in response to the change from one sample to the next.

There's a good overview of PID control in the Experiment 6 section of the Industrial Control text offered by Parallax at `www.parallax.com/dl/docs/books/edu/ic.pdf`. You can order the text as a book from Parallax at `www.parallax.com/detail.asp?product_id=28156`.

Somewhat harder is tuning the coefficients so the control loop drives the system properly. We've included this code at the bottom of the `compute` method (commented out) to dump the key variables back to a connected PC:

```
System.out.print( "T: " );
System.out.print( nTemp100 );
System.out.print( " S: " );
System.out.print( nSetPoint );
System.out.print( " Err: " );
System.out.print( nErr );
System.out.print( " PctErr: " );
System.out.print( nPctErr );
System.out.print( " P: " );
System.out.print( nProp );
System.out.print( " I: " );
System.out.print( nInt );
System.out.print( " D: " );
System.out.print( nDeriv );
System.out.print( " Drive: " );
System.out.print( nDrive );
System.out.print( " PWM: " );
System.out.println( nPWM );
```

If you plan to use the values with Excel, you might want to revise the code to output in comma separated value (CSV) form.

Extending the Controller

There are many ways you can extend our basic design and software. Here's a few; they're not necessarily ordered in increasing difficulty:

- Build it into a case able to protect it from being dropped or spilled on, and package it with a display window and connectors able to withstand outdoor conditions. RadioShack has a variety of project boxes you could use (`www.radioshack.com/category.asp?catalog%5Fname=CTLG&category%5Fname=CTLG%5F011%5F002%5F012%5F000&Page=1`), and Hobby Engineering has solderable prototyping boards (`www.hobbyengineering.com/SectionCP.html`).
- It's a nuisance to have to have a power cord to run the controller and fan, and you need less than 10 watts to run the entire assembly. If you integrated a Peltier cell (`www.peltier-info.com`) into the firebox (which will generate power when one side is hot and the other cold), you could generate the necessary power with nothing but the fire itself.
- We noted that the readings from different samples of the same model temperature probe can vary. You could calibrate your specific probe against an accurate source, such as an infrared temperature gun (for example, the Raytek MiniTemp at `www.raytek-northamerica.com/cat.html?cat_id=2.3.6&PubSessID=6d7694588d3d0eb588117c988675ab0e&PHPSESSID=publicRaytekNorthAmerica`), modifying the additive constant and, if necessary, the variable coefficient for more accurate readings.

- The 16-bit arithmetic in the Javelin Stamp made some of the code awkward, and creates some restrictions such as the duration over which you can run the integrator in the PID loop. Parallax has both `Int32` and `IEEE754` floating-point math classes on their Web site you could use to revise the code. The Int32 code is at `www.parallax.com/javelin/applications.asp#AN011`, while the floating-point code is at `www.parallax.com/dl/appnt/jav/class/float32.zip`.
- We fixed the set point as a constant in the code to simplify the design. If you added some switches (or a keypad), you could create a user interface that lets you modify the set point (and perhaps the tuning coefficients), storing the new values in the EEPROM.
- The coefficients in our code are specific for our smoker, and aren't yet optimal. You could tune the coefficients to your equipment; one of the best resources for tuning we found on the Internet is *Tuning Criteria or "How do we know when it's tuned"* by John A. Shaw (`www.jashaw.com/pid/tutorial/pid6.html`). His book on PID control is available at `www.jashaw.com/pidbook`. You might also want to read the *sci.engr.** newsgroup FAQ on PID controller tuning (`www.tcnj.edu/~rgraham/PID-tuning.html` or directly in the newsgroups).
- The range of temperatures over which we use the sensor is very limited to avoid problems with thermistor non-linearity. You could extend the range; one approach would be to create a lookup table based on the fundamental equation for the sensor and some calibration measurements. If you do, you'll still need to bound the interval over which the PID algorithm operates, considering the error to be 100 percent outside that range. You might have to adjust the bias for the significantly different set points this enhancement makes possible, such as cold smoking at 150°F.

Summary

Using the barbeque controller is something of an art, because there's a lot of variation in fire size and heat loss — coefficient tuning simply can't do everything. You'll need to tune the inlet and outlet dampers to make sure the combination of fire size and damper settings keeps the system within the range of controllability of the system, or build a servo system to adjust the dampers automatically when the control loop goes beyond its limits. Categorically, too small a fire won't let the smoker stay hot enough; too big a fire tends to be more controllable because you can simply close the dampers more.

It's worth the effort.