# *Digital Design and Computer Architecture*

**Lab 7: Temperature Control**

## Introduction

In this lab you will use a C program on the Raspberry Pi to control the temperature of an ohmic heater. Your program should keep the temperature of the heater as close as possible to 60 degrees Celsius. The circuit used in this lab is based on that of the PID temperature control lab in E102. The temperature sensor outputs a voltage that varies linearly with its temperature. After an op-amp gain stage, an external analog-to-digital converter (ADC) samples the voltage to determine the temperature of the heater and communicates the voltage value to the Raspberry Pi. Power to the heater is switched on or off with a power transistor, activated by an output pin on the Pi.

## Background: Feedback Control Systems

As computer technology has become increasingly powerful and more miniaturized, inexpensive microcontrollers have seen increasing use in all sorts of control applications. Systems from simple kitchen appliances to car engines now employ digital components to control behavior previously governed by mechanical devices and analog control circuits.

Most generally, feedback control is when a system uses its current or past state (or output) to control its future state. As students who have taken E102 will know, there are many types of feedback control systems that can be implemented in either the analog or the digital domain. You will be using one of the simplest techniques, called "bang-bang" control (or on-off control).

The three crucial elements of a feedback control system are the current output value of the system under control (the 'plant'), a means of input to the plant, and a controller to determine the appropriate input to keep the plant output within a set of parameters. In this lab, the heater block is the plant system, with the 'output' being its current temperature. The temperature sensor attached to it provides a means for the controller, in this case the Raspberry Pi, to know the state of the system. The power transistor, controlled by an I/O pin on the Pi, determines the input to the plant.

Bang-bang control works by comparing the current state of the system to a desired value, the set point. In our case, the Pi is programmed with a fixed temperature as the set point. If the temperature sensor indicates that the heater is below the set point, it activates the power transistor to increase the temperature. If the temperature is above the set point, the Pi switches the power transistor off.

## 1. Circuit Implementation

You will be using the same resistive heater in this class as in E102. If you have already taken E102, you may reuse your breadboarded circuit. If you haven't, go to the stockroom and obtain a lab kit and small breadboard for this lab. The kit for this lab should contain seven parts: an LM35 temperature sensor, an LM324 op-amp, a TIP31A power transistor, and four resistors: a 10 Ω (the heater), a 270 Ω, a 10 kΩ and a 22 kΩ. Extra parts (including some resistors and TIP31As) are located in the black cabinet in the E85 lab. The kit is yours to keep; hold onto it and plan to reuse it in E102.

You should also get a MCP3002 analog-to-digital converter (ADC). This IC chip will sample the voltage on the temperature sensor and send the data via SPI (a type of serial communication) to the Raspberry Pi.

The circuit diagram is shown below. Check the LM35, LM324 and TIP31A datasheets for pinout information (links to the datasheets are below). Don't forget to connect the power/ground pins for the LM324. Attach the 10 Ω heater resistor to the LM35 temperature sensor with the screw and nut provided so the two are at the same temperature. Build the circuit on your own breadboard.

You will need to connect 7 wires from the Raspberry Pi GPIO header

1. Ground, which should be common for all parts on the board.
2. 3.3V, which is used to power the ADC (Vdd)
3. SCLK to the CLK pin on the ADC
4. MISO (Master Input/Slave Output) to the Dout pin on the ADC
5. MOSI (Master Output/Slave Input) to the Din pin on the ADC
6. CE0 (Channel 0 Enable) to the $\overline{CS}$/SHDN (Chip Select) pin on the ADC
7. GPIO17 to the TIP21A transistor base pin.

You should use the DC Power Supply at your lab station to provide the 5V power. The Raspberry Pi is capable of powering this whole circuit on its own, but using the external supply makes it easy to tell if the heater is on or off (current is about 0.4A or greater if ON, current is negligible if OFF).

You may also find it helpful to add an LED and appropriate resistor, as shown in the box labeled "Optional", to help you tell when the heater is switched on or switched off. You can also lightly touch the "heater" (the 10Ω resistor) to see if it is heating. (It will take a few seconds to start heating up.)

http://www.national.com/ds/LM/LM35.pdf
http://www.national.com/ds/LM/LM124.pdf (also covers LM324; choose one of the 4 op-amps)
http://www.fairchildsemi.com/ds/TI/TIP31A.pdf

Note that the collector of the TIPS31A is connected to both the middle pin (pin 2) and the large flat metal at the top of the device. You will connect the 10Ω resistor directly to the large flat metal (the collector) with a machine screw and nut. For now, connect the transistor base via the

270Ω resistor to ground. Do not connect the LED directly to ground without a resistor, as the total current supplied by any given Pi GPIO pin is limited.

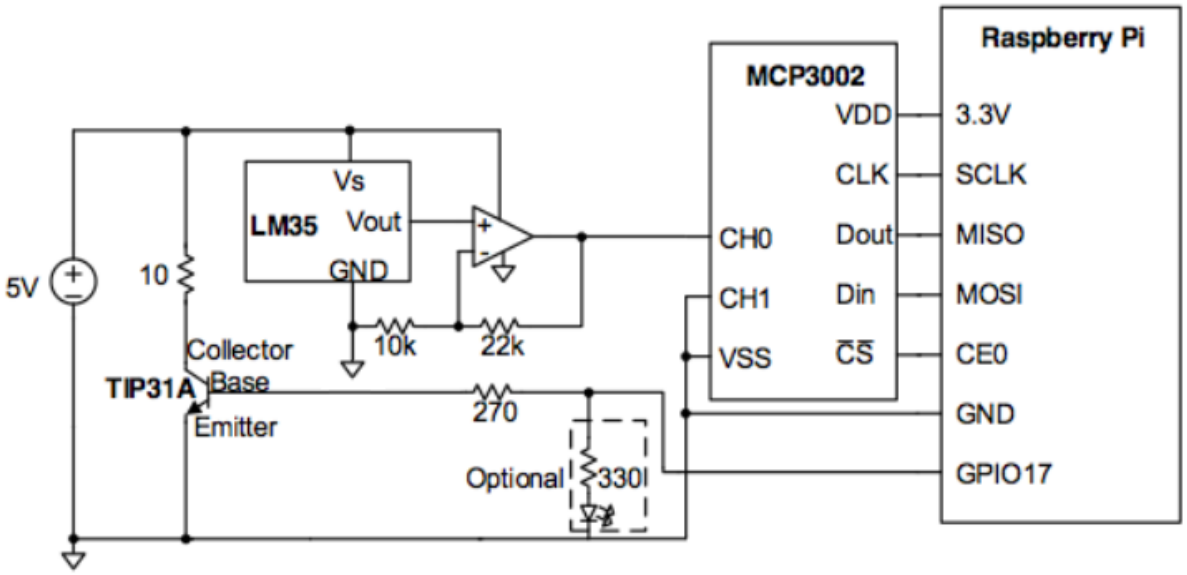


**Figure 1:** Temperature control circuit diagram. LED to indicate heater status is optional.

The LM35 produces a voltage output $V_{out}$ = 0.01 * T, where T is the temperature in degrees Celsius. For our desired set point of 60 C, the output is just 0.6V. The Analog-to-Digital Converter (ADC) you put in your circuit (the MCP3002) has a range from 0V to 3.3V and provides 10 bits of resolution. Thus, a least significant bit corresponds to $3.3 / 2^{10} = 3.2$ mV, or 0.32 degrees. To increase the accuracy of your control, the LM324 op-amp is set up as a non-inverting amplifier with DC gain of 3.2. This produces a voltage of 0.6V × 3.2 = 1.92V at the set point, making more use of the ADC's range. Each least significant bit of the ADC output will now correspond to 0.1 degree.

Before you turn on the 5V supply, open up a terminal to your Raspberry Pi and type

pi@raspberrypi ~ $ `gpio write 0 0`

This will make sure that GPIO17 (which maps to the wiringPi pin 0) is at ~0V.
Turn on the 5V power supply. Since GPIO17 is low, the power usage should be low. If it is not (about 0.4A or greater), the heater is probably on. Go back and check your circuit. Check that you connected the appropriate signals to the appropriate pins on each device. If the problem persists, ask a tutor or professor.

Using a multimeter, measure the voltage from the LM35, and makes sure it matches your expectations given the temperature of the room. Next, verify that the op-amp output voltage is 3.2 times the LM35 voltage.

Now disconnect the 270 Ω resistor from GPIO17 and hook it to 3.3 V instead. Continue to measure the op-amp output voltage. After a few seconds, the voltage should start to steadily rise,

corresponding with the increase in temperature of the resistor. Once you are satisfied that the circuit is working properly, reconnect the 270 Ω resistor to GPIO17 to shut off the heater.

## 2. Control Program

You are now ready to connect the circuit to the Raspberry Pi and write your temperature control program. The *wiringPi* library you used in your previous lab has built-in functions for SPI communication with devices like the MCP3002 serial ADC. To simplify your code, download "e85lab7.h" from the course website. This helper file defines the function `analogReadSPI()`, which will handle the SPI communication and simply returns the value that the ADC reads on its channel 0 input pin. This value will be an integer between 0 and $2^{10}$-1. You must call `initAnalogSPI()` before you can use `analogReadSPI()`. To use functions defined in this file, you must include it at the top of your code;

```
#include "e85lab7.h"
```

Write your program. If you keep your program simple, it will be very short. Don't forget that you must `#include<wiringPi.c>` and run `wiringPiSetup()` in order to use `digitalWrite()` from the *wiringPi* library.

Debugging programs that depend on interaction with I/O devices like the ADC can be challenging, since the program must run in real time for its behavior to be properly observed. You will find it useful to print the current temperature after it is returned from the ADC.

Run your program. Does the temperature rise and stabilize as expected? Measure the rise time of your circuit from room temperature to the first crossing of the set point, and how much the temperature overshoots the desired value. The temperature will likely oscillate around the set point, but should stay within 3 or 4 degrees of the desired value.

You can introduce a disturbance to the system by blowing on the heater. This decreases temperature by a few degrees in just a few seconds. Watch your measured temperature drop, then rise back up. Does your controller deal with the disturbance effectively?

If you have the chance, you may find it interesting to observe the sensor voltage on an oscilloscope in the lab. Figures 2 and 3 show the transient behavior and the oscillation around the set point.

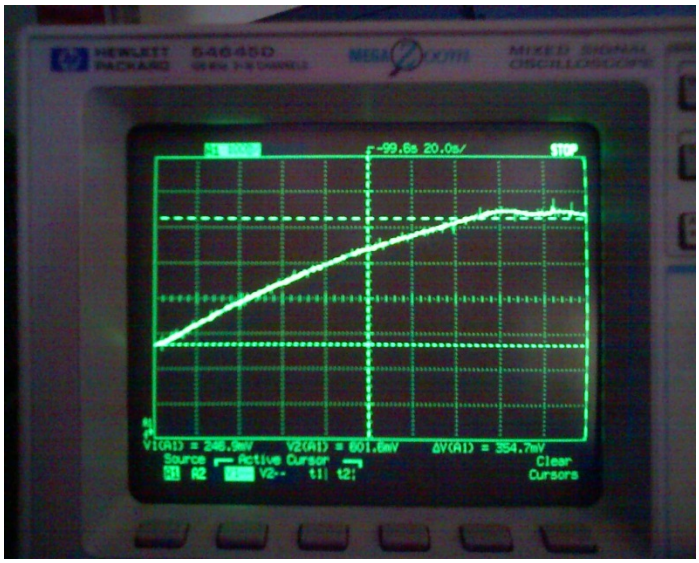


**Figure 2:** Rise and oscillation of the heater temperature under proper control.

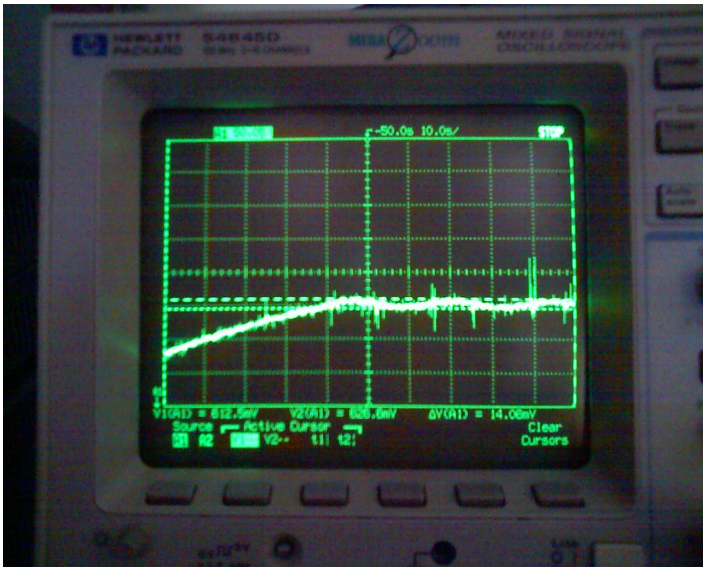


**Figure 3:** Measurement of set point oscillations.

## What to Turn In

Include each of the following items in your submission **in the following order**.

1. How many hours did you spend on this lab? This will not count toward your grade (unless omitted entirely).
2. Your C program. It should be clean and well commented.
3. The rise time and overshoot measurements of your system.
4. Do you have any suggestions for improving this lab prompt or the lab setup?

**Acknowledgements:**
This lab was developed by Leo Altmann based on the E102 lab developed by Prof. Tony Bright. Sarah Lichtman developed the changes from a PIC32 to a Raspberry Pi using an external MCP3002 with bug fixes from Leif Park Jordan.