CHAPTER 6

PCF8591 ADC

One of the peripherals missing in the Raspberry Pi is an analog-to-digital converter (ADC). One low-cost solution is an 8-bit ADC peripheral based upon the PCF8591 chip. You can purchase the bare chip on eBay for about $1.50, but for about 10 cents more, you can get the assembled YL-40 PCB.

This chapter will explore the PCF8591 ADC and the Linux I2C device driver for it. While the driver source code exists in the kernel source tree, it is not likely compiled into your Raspbian release. Even when installed, there are a few simple things you need to do to get the driver operating. Using the Linux driver support, it will be a snap to take ADC readings using a shell script or a Python program.

The YL-40 PCB

This chapter explores the YL-40 PCB, which is about 1 inch by 1.5 inches in size. You might use a different PCB, using the same PCF8591 chip. The YL-40 comes preassembled with header strips and extras that make it easy to use. In addition to the PCF8591 chip's ability to convert an analog signal into a digital value, it has one digital-to-analog converter (DAC) output channel, which you'll explore later. Figure 6-1 illustrates the topside of the YL-40 PCB.

W. Gay, *Custom Raspberry Pi Interfaces*, DOI 10.1007/978-1-4842-2406-9_6

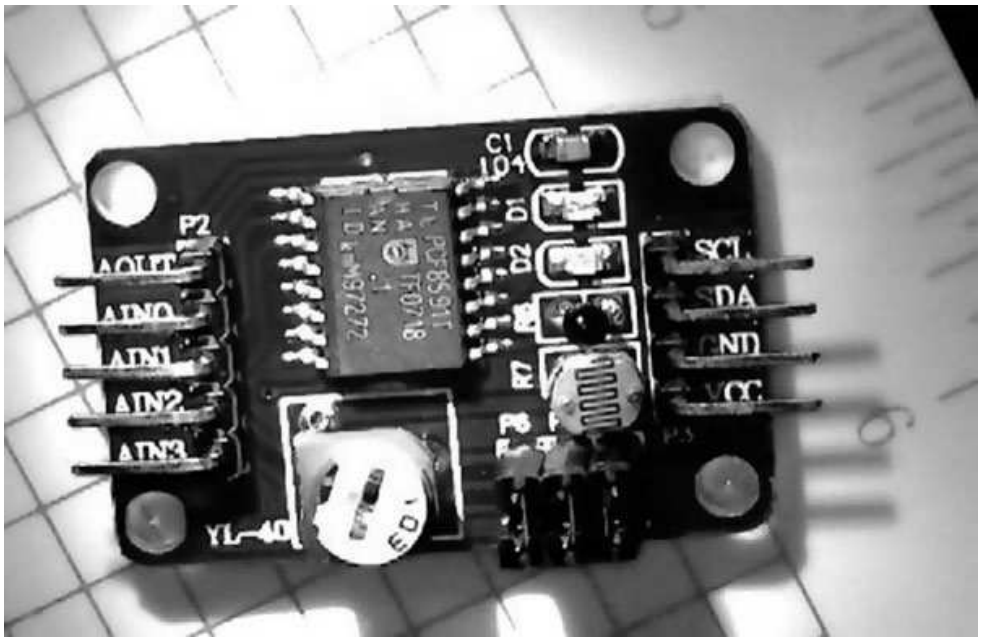

Figure 6-1. *The YL-40 PCF8591 PCB*

Header strip P2 (left) consists of these analog inputs and outputs:

- AOUT (DAC), output
- AIN0 (ADC), input 0
- AIN1 (ADC), input 1
- AIN2 (ADC), input 2
- AIN3 (ADC), input 3

The power is applied on the right through P3 connector pins V_{CC} and GND (shown on the right of Figure 6-1). Additionally, the I2C signals for SDA and SCL are found at P3. One of the advantages of the PCF8591 chip is that it will operate from 3.3 volts, interfacing nicely with the Raspberry Pi. Figure 6-2 is a schematic of the YL-40 PCB.

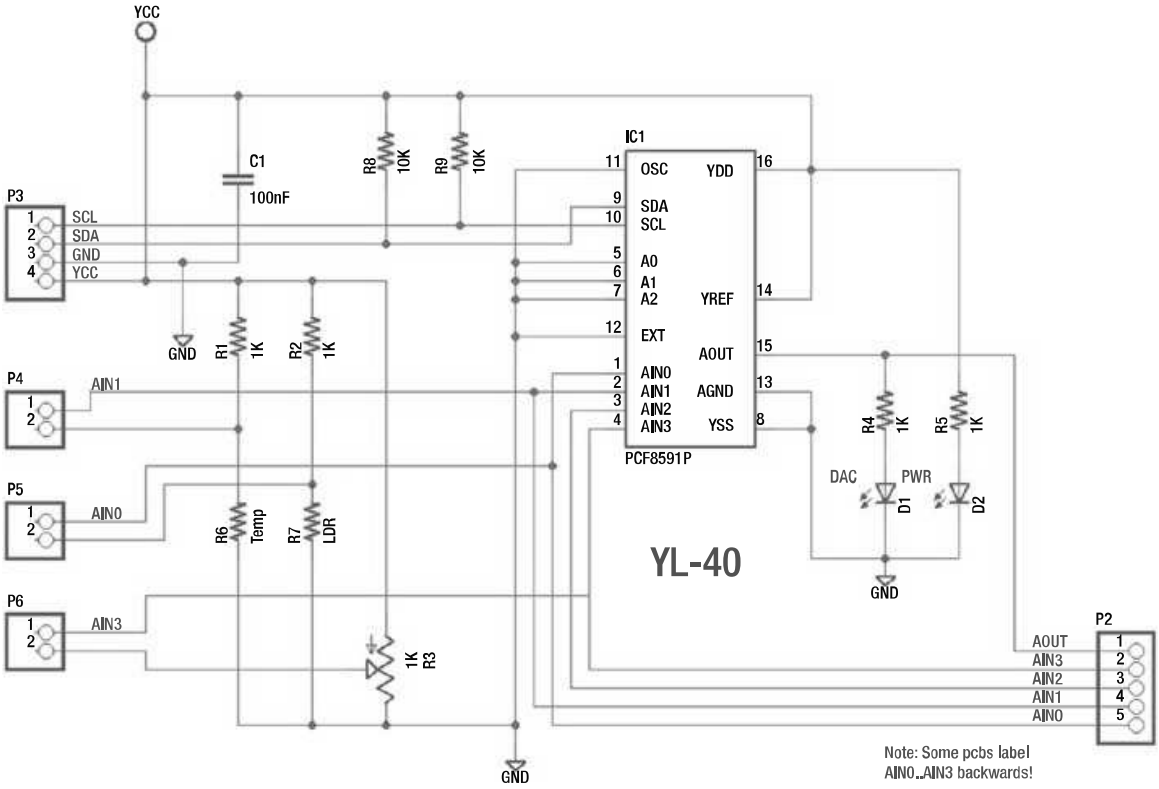

Figure 6-2. *Schematic of the YL-40 PCB*

Three jumpers come installed on the YL-40 that enable or disable extra features.

- Jumper P4, when installed, enables a temperature-sensitive resistance sensor (R_6). On my PCB, this didn't seem to work (on two PCBs that I purchased). Later in this chapter, I'll show you how to fix that.
- Jumper P5 enables the light-dependent resistor (LDR) and connects it to AIN0. This jumper can be removed when you want to provide your own analog input to AIN0.
- Jumper P6 connects the wiper arm of potentiometer R_3 to AIN3. Using a small screwdriver, you can turn R_3 clockwise or counterclockwise to cause an analog reading to change. This is extremely useful during initial testing. You can, of course, remove the jumper on P_6 and use AIN3 for any other purpose. Turning R_3 fully clockwise grounds AIN3, while counterclockwise brings AIN3 to V_{CC} potential.

AIN2 has no jumper and is available at header strip P2. No other component on the PCB is attached to it.

Voltage Range

Because the PCF8591 (YL-40 PCB) is powered from the Raspberry Pi's 3.3V supply, the analog input voltages must be kept between 0 volts and 3.3 volts. The V_{REF} is taken from the power supply on the YL-40 PCB. If you're wiring this up yourself, you have the option of using a lower V_{REF} voltage, but the maximum voltage should not exceed V_{DD} (on the YL-40 PCB, this is also known as V_{CC}). The datasheet lists any input voltage V_I as V_{DD} +0.5V (3.3 + 0.5 volts) as the absolute maximum. The lowest voltage for any input is listed as -0.5 volts.

You can find further technical details by Googling *PCF8591 datasheet PDF*.

I2C Bus

The PCF8591 uses the I2C bus to communicate. As an I2C slave peripheral, the YL-40 PCB provides two weak pull-up resistors, R_8 and R_9, which are 10kΩ each. The Raspberry Pi will already have its own pull-up resistors on these lines (1.8K each). If you are rolling your own circuit, you may want to install these 10kΩ pullup resistors also, though they are not strictly necessary.

I2C Addresses

The PCF8591 chip possesses three address pins, A0 to A2, allowing you to choose one of eight possible peripheral addresses from 0x48 to 0x4F. The YL-40 PCB, however, connects all three of these pins to ground, restricting the address to 0x48 (jumpers would have been nice). Unmodified, only one YL-40 PCB can be attached to a given I2C bus. Later in the chapter, you'll see how the PCB can be modified to overcome this limitation.

DAC (AOUT)

With the Linux device driver operating, you can write a value from 0 to 255 to have the DAC output (AOUT) establish an output voltage. The driver input value, however, must be multiplied by ten. Assuming that you have exactly 3.3 volts powering the PCF8591 (and thus V_{REF} = 3.3 volts), a value of 255 should establish a voltage very near that. A value of zero establishes a ground value instead. A midpoint value of 128 should generate a value near half of 3.3 volts (1.65). When multiplied by 10 for the driver, the values used are 2550, 0, and 1280, respectively.

The YL-40 PCB, however, has a yellow LED D_1 through a R_4=1 kΩ resistor is connected in series to AOUT (shown up close in Figure 6-3). With the DAC set to produce maximum output, the yellow D1 should light up. But this turns out to be less brilliant than the red power LED D_2 beside it. When I measured the voltage, AOUT was only 2.89 volts, which is well short of the expected 3.3 volts. After I removed the LED (D_1), the voltage came up to a respectable 3.29 volts and begs the question: just how much current can the DAC drive?

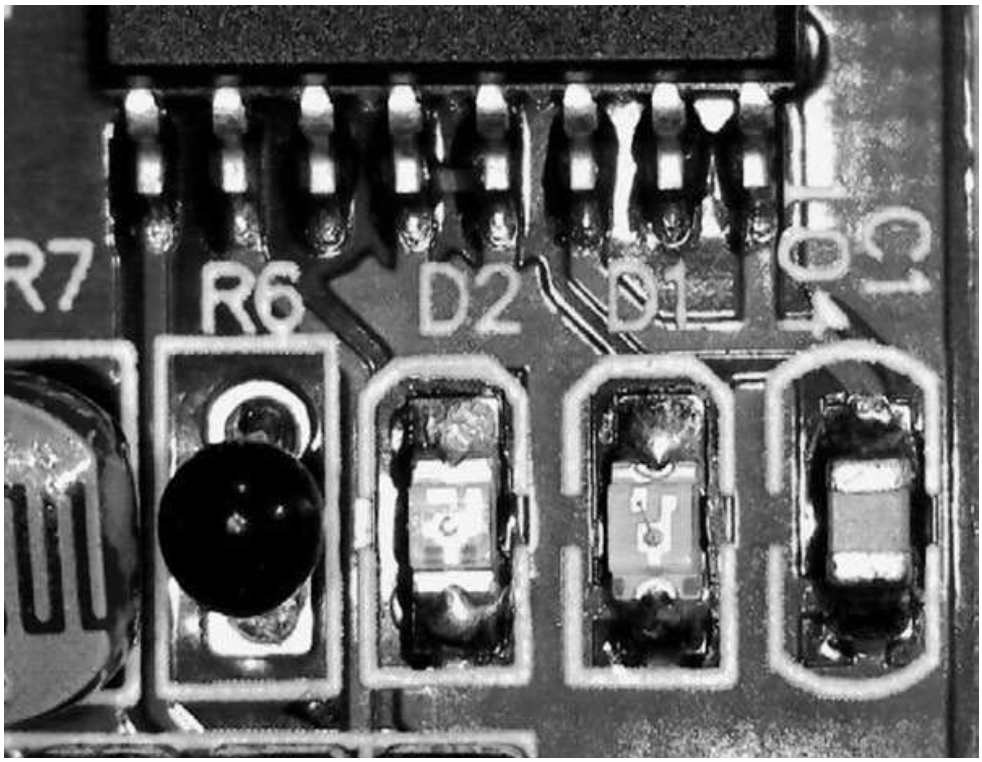

Figure 6-3. *DAC LED D_1 between D_2 and C_1*

A careful look at the NXP datasheet under the heading "14.2 D/A characteristics" lists these parameters:

Symbol	Parameter	Conditions	Min	Typical	Max	Unit
V_{oa}	Analog output voltage	No resistive load	V_{ss}		V_{DD}	V
		R_L = 10kΩ	V_{ss}		0.9 × V_{DD}	V

The upper line where the conditions state "no resistive load" confirms that the maximum output reading should be V_{DD} (a supply voltage of 3.3 volts). What is interesting is that when a 10 kΩ load is attached, the voltage drops to 0.9 × V_{DD}, or 2.97 volts (0.9 × 3.3 volts).

If you treat the DAC output like a battery, you can determine what the source resistance R_S is. A practical battery can be *modeled* as an ideal battery with a source resistor in series with it. Knowing the source resistance allows you to predict its output voltage for any given current flow (or load resistance). Figure 6-4 illustrates the idea in schematic form.

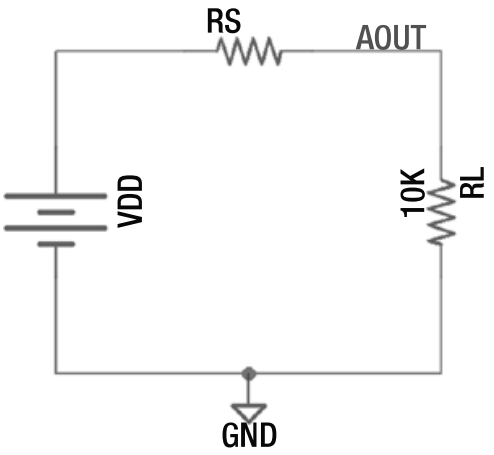

Figure 6-4. *Equivalent circuit for DAC output AOUT*

In Figure 6-4, the ideal battery (V_{DD}) represents the 3.3 volts provided to the PCF8591 chip. The unknown source resistance (R_S) is connected to AOUT feeding the load (R_L). You know the following from the datasheet:

> When R_L = 10kΩ, the voltage $V(R_L)$ is $0.9 \times V_{DD}$, which in this case is 2.97 volts.

Armed with this information, you can apply Ohm's law to determine the resistance of R_S. Because of this information:

- $V_{total} = V(R_S) + V(R_L)$
- $V(R_S) = V_{total} - V(R_L)$
- $V(R_S)$ = 3.3 volts – 2.97 volts, which is 0.33 volts

you also know the following:

- $V(R_L)$ = 2.97 volts
- $I(R_l) = \frac{V(R_L)}{R_L} = \frac{2.97 volts}{10\ k\Omega} = 0.297 mA$

This now allows you to calculate R_S. Because $I(R_S) = I(R_L)$ in the series circuit, you get this:

- $R_S = \frac{V(R_S)}{I(R_S)} = \frac{0.33\ volts}{0.287\ mA} = 1.1\ k\Omega$

From this you can conclude that the series resistance is about 1.1 kΩ. Let's now apply this information.

If you consider the LED (D_1) and its resistor together as R_L (the load), using Ohm's law you can now compute the current flow through the LED (and thus coming out of AOUT).

- $V(R_S) = V_{DD} - V(R_L)$, = 3.3 volts – 2.89 volts = 0.41 volts
- $$I(R_L) = I(R_S) = \frac{V(R_S)}{R_S} = \frac{0.41\ volts}{1.1\ k\Omega} = 0.372\ mA$$

With an LED current flow of 0.372mA, it is no wonder that the LED is weakly lit! LEDs often require 3mA or more for proper illumination.

All of this highlights the fact that the DAC output at AOUT is not suitable for driving loads *requiring current*. It does provide a controllable output voltage, however, but it is not capable of any substantial current drive. Use a voltage-to-current driver, when required. This further supports my suggestion for removing or disabling the output LED on the YL-40 PCB.

Removing YL-40 LED D1

Because the YL-40 PCB connects a resistor (R_4=1kΩ) and yellow LED to the AOUT circuit, the output accuracy of the DAC is compromised. For this reason, remove D_1 if you care about output accuracy. Figure 6-3 illustrates the location of D1 on the YL-40 PCB between the power LED (D2) and capacitor C1. Carefully apply a soldering iron to both ends of D1 to remove it. Make certain that no solder blobs remain to short things out.

Figure 6-5 illustrates the PCB area with D_1 removed. On mine, there was a little white triangle underneath showing the direction for LED current flow. The bottom pad (D_1 cathode) connects to ground, while the upper pad (D_1 anode) connects to R_4, which is then connected to AOUT. Measuring resistance from the upper pad (anode) to ground should show a near-infinite resistance after LED removal. Measuring from the same upper pad to AOUT should read about 1 kΩ (the value of R_4). If you read something else, check for a solder short.

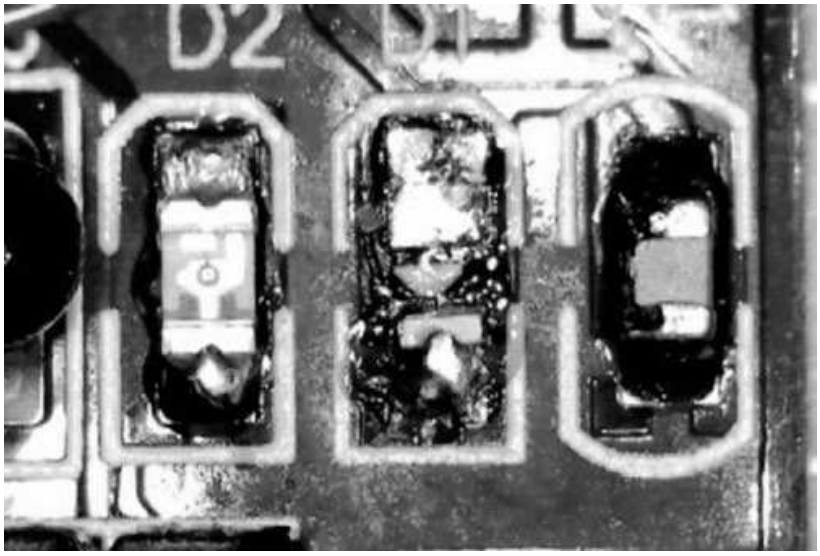

Figure 6-5. *LED D_1 removed*

Hacking YL-40 I2C Address

If you want to use more than one YL-40 module, you'll need to hack the I2C address for each of the additional modules. Figure 6-6 illustrates the lifting of pin 6 to solder a small wire to it. The other end of the wire is soldered to V_{DD} (pin 16) where the +3.3V supply is connected. In this way, address bit A1 becomes a 1 bit instead of a 0 bit. This change results in the I2C address of 0x4A. The wire I used was solid 30-gauge wire-wrap wire.

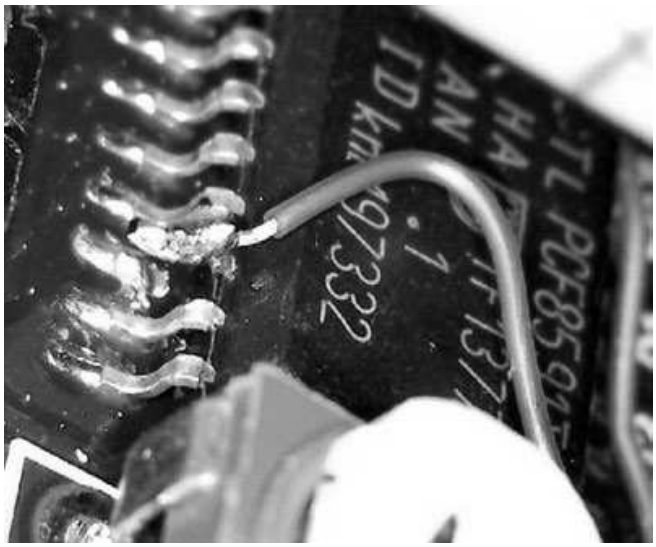

Figure 6-6. *Changing I2C address by lifting pin 6*

These pins are small and difficult to work with. With patience and a small prying tool like an eyeglass screwdriver, you may be able to pry up a leg without using a soldering iron. Don't overdo it or the leg may break off. Once separated from the PCB pad, use a small, tipped soldering iron to apply just enough solder to attach the wire to the pried-up leg.

The PCF8591 address pins are as follows:

Pin No.	Label	Description
5	A0	Least significant address bit
6	A1	Middle address bit
7	A2	Most significant address bit

The YL-40 PCB has all three pins soldered to ground, resulting in an I2C address of 0x48. By lifting address pins and connecting them to +3.3 volts, additional addresses are possible. Table 6-1 summarizes the extra I2C addresses.

Table 6-1. *Addresses: Lifted Pins Are 1 Bits, While Soldered Pins Are 0 Bits*

A2	A1	A0	I2C Address
0	0	1	0x49
0	1	0	0x4A
0	1	1	0x4B
1	0	0	0x4C
1	0	1	0x4D
1	1	0	0x4E
1	1	1	0x4F

It is important that the lifted pins be connected to +3.3 volts. Otherwise, static electricity will collect on the pins causing the address to change sporadically.

I2C Bus Setup

If you haven't already done so, install `i2c-tools`, as shown here:

```
$ sudo apt-get install i2c-tools
```

Once installed, then list your I2C buses, as shown here:

```
$ i2cdetect -l
```

If nothing is displayed, then you need to change the configuration so that the I2C drivers get loaded at boot time. The new kernels use the `/boot/config.txt` file to enable I2C support. Edit the file so that the `i2c_arm=on` line is uncommented, as shown here:

```
# Uncomment some or all of these to enable the optional hardware interfaces
dtparam=i2c_arm=on
#dtparam=i2s=on
#dtparam=spi=on
```

Save your changes and reboot.

```
sudo /sbin/shutdown -r now
```

After the reboot, try listing the I2C buses again, as shown here:

```
$ i2cdetect -l
i2c-1 i2c        3f804000.i2c                    I2C adapter
```

From this session output, you now see `i2c-1` is available as expected.

Reading from PCF8591

If you are using the YL-40 PCB, you can read the built-in potentiometer using the `in3_input` (to change the potentiometer, turn the white control with a small screwdriver). If you built your own PCF8591 circuit, then attach a potentiometer (R_3), as shown in the schematic (Figure 6-2). Turning R_3 full-clockwise should cause a reading of AIN3 of near 0. Turning R_3 full counterclockwise should cause it to read 255.

In subdirectory `pcf8591` are a few programs, including `readadc` (run `make` if that has not already been done). Invoke the program with option `-h` to display some usage information, as shown here:

```
$ ./readadc -h
./readadc [-a address] [-i input] [-h]
where:
      -a address   Specify I2C address
      -i input     Specify AINx (AIN3 is default)
      -d           Enable and leave DAC enabled
      -h           Help
```

By default, the `readadc` program assumes I2C address 0x48. To specify a different address, use the `-a` option and the new address. If the address is prefixed with `0x`, the address will be interpreted as hexadecimal. For example, the following will use the I2C address of 4A in hex:

```
$ ./readadc -a 0x4A
```

The `-i` option for `readadc` defaults to 3 for accessing the chip input channel AIN3. Specifying a different number allows you to read other channels. For example, the following reads from AIN1 with the I2C address of 4A in hex:

```
$ ./readadc -a 0x4A -i1
```

Finally, the `-d` option just enables the DAC output (I'll discuss this more later).

Experiment

Using the YL-40 PCB, adjust the potentiometer fully clockwise. After doing so, let's take an ADC reading, as shown here:

```
$ ./readadc
0
```

Now turn that potentiometer fully counterclockwise and repeat, as shown here:

```
$ ./readadc
255
```

Don't worry if the value doesn't read exactly 255. In some cases, you might read 254. This is a good sign that the ADC is working. Now turn the potentiometer to exactly midway between the two extremes.

```
$ ./readadc
137
```

If you did better than I did, you might get 127 or 128. With an 8-bit ADC, your range of values are from 0 to 255.

Writing to the DAC

In the same `pcf8591` subdirectory is a program named `writedac`. It accepts nearly the same command-line options, while the trailing arguments are expected to be values to be written to the DAC.

```
$ ./writedac -h
./writedac [-a address] [-h] values to write...
where:
      -a address   Specify I2C address
      -h           Help
```

To write the value 128 to the DAC, use the following command:

```
$ ./writedac 128
```

or the following:

```
$ ./writedac 0x80
```

If multiple values are listed, they are all written to the DAC in sequence before the command exits.

Experiment

You need a voltmeter or DMM for this experiment. Attach the negative lead of the DMM to the ground of the Pi (or PCF8591 module). Set the meter on volts and attach the red lead to the DAC output. This is labeled AOUT on the YL-40 PCB.

With the meter leads attached, you should read about half of 3.3 volts after performing this command:

```
$ ./writedac 128
```

This assumes that you have removed the yellow LED, as shown earlier. If you left the LED in, you may see a lower voltage, but it should be well above 0. Now let's set the DAC to 0:

```
$ ./writedac 0
```

Immediately, your meter should read near 0 volts. Finally, set the DAC to its maximum value.

```
$ ./writedac 255
```

Your meter should now report nearly 3.3 volts (mine read 3.25 volts).
Try the DAC at 25 percent and 75 percent of its maximum value and verify the voltages.

Experiment

If you lack a means of measuring your DAC voltage, you can use the ADC input to measure the DAC. You want to use the AIN2 channel since it is not connected to any extra components. So, attach a Dupont wire from AIN2 to the AOUT terminal.

There are two things tricky about this experiment:

- You must specify the `-i2` option on the `readadc` command to read AIN2.
- You must specify the `-d` option to keep the DAC enabled.

The following session first establishes the output voltage of the DAC at 255 and then reads AIN2 twice:

```
$ ./writedac 255
$ ./readadc -i2 -d
255
$ ./readadc -i2 -d
254
```

From this you have confirmed the DAC output setting. Successive reads can sometimes wander somewhat, so expect that. The `-d` option of the `readadc` command is required to keep the DAC enabled because by default the command turns it off. If you can't repeat the readings of this experiment, this is the first thing to check.

Now change the DAC to 128 and read it back.

```
$ ./writedac 128
$ ./readadc -i2 -d
177
$ ./readadc -i2 -d
127
$ ./readadc -i2 -d
127
```

On my unit, the change in DAC output seemed to take a little extra time. Notice that the first read returned 177 and then afterward 127.

Limitations

The PCF8591 chip has some limitations.

- Its maximum sampling rate is f_s=11.1kHz, making it unsuitable for most audio.
- ADC returns the prior result and the current value (further reducing the effective sample rate).
- The ADC value is 8-bit resolution.

Despite the limitations, the PCF8591 has some advantages.

- 3.3V-compatible with the Raspberry Pi
- I2C enabled
- Economically priced
- Built-in Linux driver support

Extending Voltage Range

You've already seen the device do direct voltage measurements. But how do you measure voltages greater than 3.3 volts? Use a voltage divider, as shown in Figure 6-7.

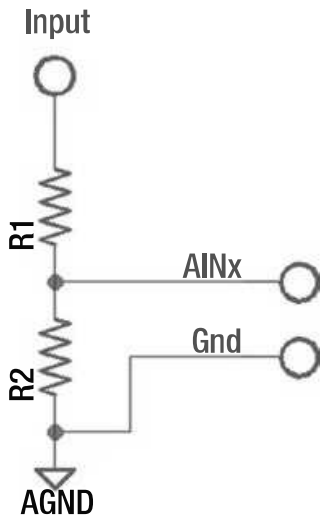

Figure 6-7. *Using a voltage divider*

The object of the voltage divider is to put the input voltage into a reduced range for the ADC input. Since the ADC is powered from the Raspberry Pi 3.3V supply, the ADC input range is limited to a maximum of 3.3 volts. To extend the range to, say, 16 volts for automotive measurement, you can compute the values of R_1 and R_2 in Figure 6-7.

The simplified design procedure is as follows:

1. Arriving at a current flow through R_2 of about 1mA, producing 3.3 volts requires that R_2 be about 3300Ω:

$$\frac{3.3\ volts}{0.0001\ Amps} = 3300\ \Omega$$

2. 16 volts minus 3.3 volts means you want a minimum voltage drop across R_1 of about 12.7 volts.

3. To compute R_2, use this: $R_2 = \frac{12.7\ volts}{0.001\ Amps} = 12.7\ k\Omega$

4. The next 10 percent tolerance resistor value greater than or equal to 12.7kΩ is 15kΩ.

With R_1 = 3.1kΩ and R_2=15kΩ, let's check the actual input voltage range achieved.

To get a full reading of 3.3 volts across R_2, you require 1mA of current. R_1 will have the same current flow as R_2 since it is in series. So if 1mA is flowing through R_1, you know that it would have a voltage drop of I × R = 1mA × 15000 Ω = 15 volts. Therefore, the total voltage that the divider pair can handle is as follows:

3.3 volts + 15 volts = 18.3 volts

What is the resolution of this range using the 8-bit ADC? An 8-bit resolution means you have a total of 256 steps (2^8 = 256). Consequently, the resolution is as follows:

$$\frac{18.3 volts}{256 steps} = \frac{0.071 volts}{256 steps} = \frac{0.071 volts}{step}$$

The conclusion is that the 8-bit ADC can measure 0 to 18.3 volts in 0.071V increments.

Repairing the Temp Sensor

With the jumper installed in P4, you can read the current temperature sensed by thermistor R_6, which changes in resistance with temperature. The thermistor is in series with the R_1=1kΩ resistor, and thus it divides the 3.3V supply. The AIN1 input measures the midpoint voltage of the divide, between the top of R_6 and ground.

But there is a problem—the YL-40 module's PCB didn't ground the lower leg of R_6. It is in fact unconnected. This results in reading the highest value, 255, because the AIN1 input is effectively attached only to the +3.3V supply. With a bit of Googling, you'll find that others have also experienced this problem.

Fortunately, the solution is not difficult to correct if you have a soldering iron. Figure 6-8 shows how to locate the supposed-to-be-grounded end of R_6. Use your DMM to check to see whether that leg and ground shows 0Ω. If it does not, a fix is needed (if your ADC reading is 255, this is already a strong indication that the repair is needed).

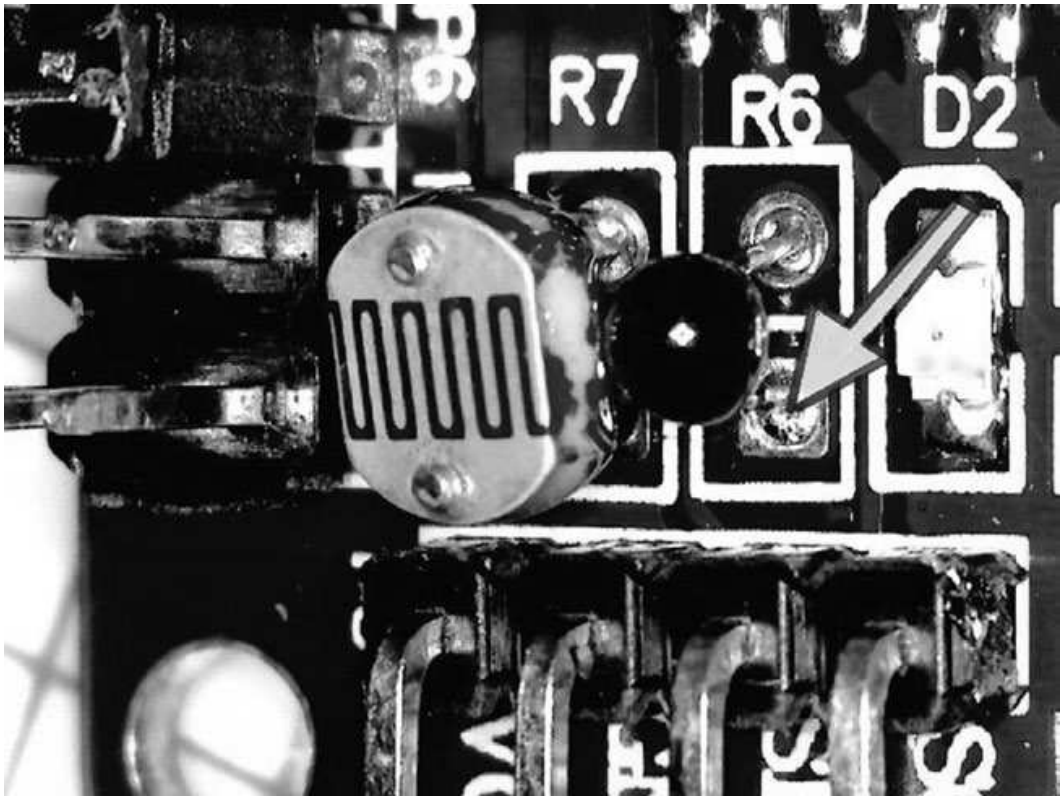

Figure 6-8. *Location of ground end of thermistor R_6*

Figure 6-9 illustrates the underneath side of R_6. Since there are no inner PCB layers, this connection is quite visibly *unconnected*.

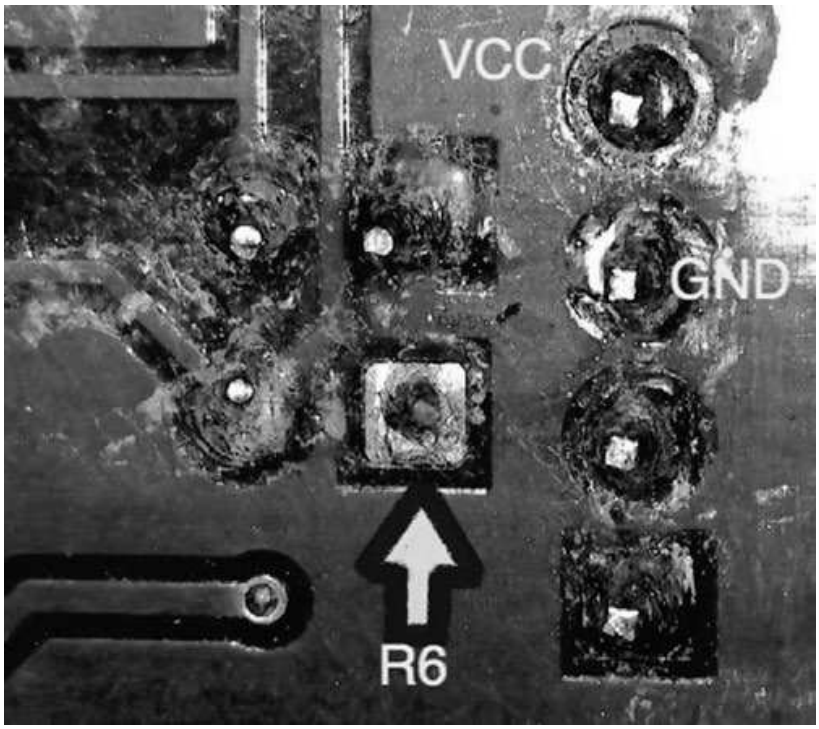

Figure 6-9. *Bottom view of R_6 ungrounded leg*

Based on Figure 6-9, you could run a small wire from R_6 to the ground post just above and to the right. I believed that to be too risky for short circuits and chose to solder a wire to a ground point shown in the bottom left of Figure 6-10 instead.

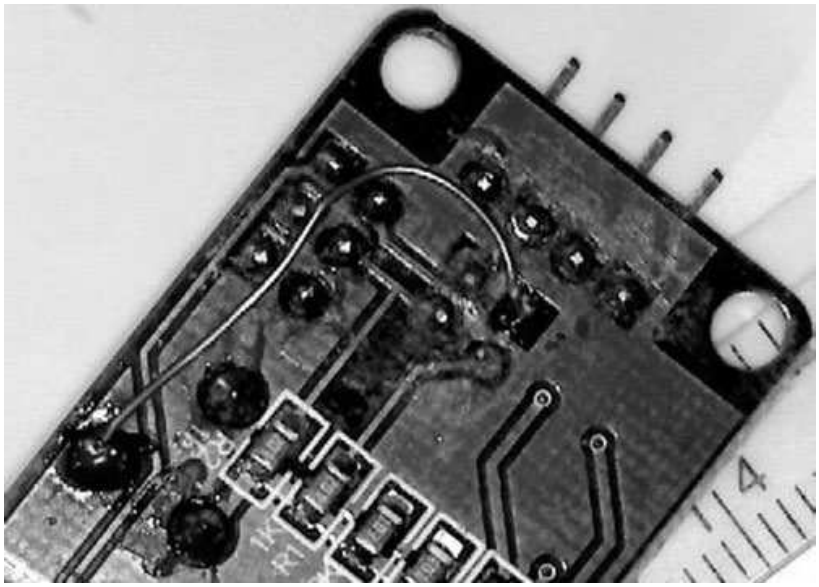

Figure 6-10. *R6 grounded to point, bottom left*

Check with the DMM to see that the R_6 leg is indeed grounded after soldering. After the repair, you should get a reading from AIN1 near 231 when at room temperature. The reading should definitely be less than the value 255 that was obtained before the repair.

Conversion to Celsius

Reading the temperature of R_6 requires some calculation after taking a reading from AIN1. You'll be using the simplified calculation [1] because you don't know all of the parameters for the thermistor part. The calculation requires these four steps:

1. Read AIN1 to get an ADC reading of the voltage at R_6 (you'll assume 234 for this example).
2. Compute the resistance of R_6 based upon the ADC reading.
3. Compute the temperature in degrees K.
4. Convert degrees K to degrees Celsius.

To calculate the current resistance of R_6, the ADC reading "a" is plugged into the following formula:

$$R_6 = \frac{R_1 \times a}{256 - a}$$

The 256 in the previous formula comes from the fact that the ADC has a resolution of 8 bits (256 steps). Assuming that the reading from the ADC is 234, you plug in the following for variable "a":

$$R_6 = \frac{1000 \times 234}{256 - 234} = 10{,}636.36$$

Now that you know the resistance of R_6, you can compute the temperature in degrees Kelvin.

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B} \times ln\left(\frac{R_6}{R_0}\right)$$

In that formula, the values are as follows:

- T is the computed temperature in degrees Kelvin.
- T_0 is the room temperature in degrees Kelvin (25°C is 298.15°K).
- B is the coefficient of the thermistor (`adafruit.com` claimed 3950).
- R_0 is the resistance at room temperature (`adafruit.com` claimed 10kΩ).

Plugging everything in, you get the following:

$$\frac{1}{T}=\frac{1}{298.15}+\frac{1}{3950}\times ln\left(\frac{10636.36}{10000}\right)=0.003369$$

$$\frac{1}{0.003369}=296.8°\text{K}$$

To convert to degrees Celsius, you use this:

$$296.8-273.15=23.65°\text{C}$$

Compare your reading with another thermometer in the room. You may want to add or subtract a small calibration offset for your project.

Reading Temperature

To read the temperature-sensitive device (R_6) attached to input channel AIN1, you can apply the program `readtemp` in the `pcf8591` subdirectory to read it. This program is almost the same as `readadc`, except that it defaults to `-i1` and performs the somewhat nasty Celsius calculation for you.

```
$ ./readtemp -h
./readtemp [-a address] [-h]
```

where:

```
-a address      Specify I2C address
-i input        Specify AINx (AIN1 is default)
-h              Help
```

The default options for `readtemp` are `-a0x48 -i1`.

Experiment

To read the temperature, simply invoke the following command:

```
$ ./readtemp
ADC=228, R6 = 8142.9 ohms, T=302.846 deg K, 29.696 deg C
```

The health monitor in my room reported 28°C, confirming that this is close. The value ADC=228 shows what the ADC peripheral read. R_6=8142.9 is the computed resistance of R_6. The value T=302.846 is the computed temperature in degrees Kelvin. Finally, the calculated Celsius temperature is reported.

The YL-40 LDR

The YL-40 PCB includes a light-dependent resistor (LDR), which is R_7 on the schematic (shown earlier in Figure 6-2). The nature of the LDR is that it has high resistance in the dark and low resistance in light. The change in resistance is relatively slow, making it unsuitable for audio reception, yet it is fast enough for control applications. The YL-40 connects this device to AIN0 when the jumper P5 is installed.

The LDR can be made of various materials and consequently has varying responses to light and wavelengths. Without knowing the exact part and parameters for the one included on the YL-40, you can make only general assumptions about the device used. The tolerances for the part also vary as much as 30 percent [2].

The resistance of the LDR (R_7) can be calculated in the same way that you did for the thermistor R_6.

$$R_7 = \frac{R_2 \times a}{256 - a}$$

Experiment

In low ambient light conditions, try reading the LDR (R_7) on your YL-40 PCB from input channel AIN0. Here I have turned my desk lamp off in the evening:

```
$ ./readadc -i0
238
```

The reading of 238 is fairly high. Turning the desk lamp on, the reading drops, as shown here:

```
$ ./readadc -i0
160
```

Now if I shine a small LED flashlight directly on the LDR, I get an even lower reading.

```
$ ./readadc -i0
50
```

How low does it get in direct sunlight?

1N914 Experiment

The 1N914 is a small glass-encased signal diode, shown up close in Figure 6-11. The cathode (negative) end of the diode is marked with a black band around it. The other end (anode) is positive for forward conduction, according to the conventional flow of electricity. This experiment will measure the voltage-to-current relationship of this diode, using forward current flow.

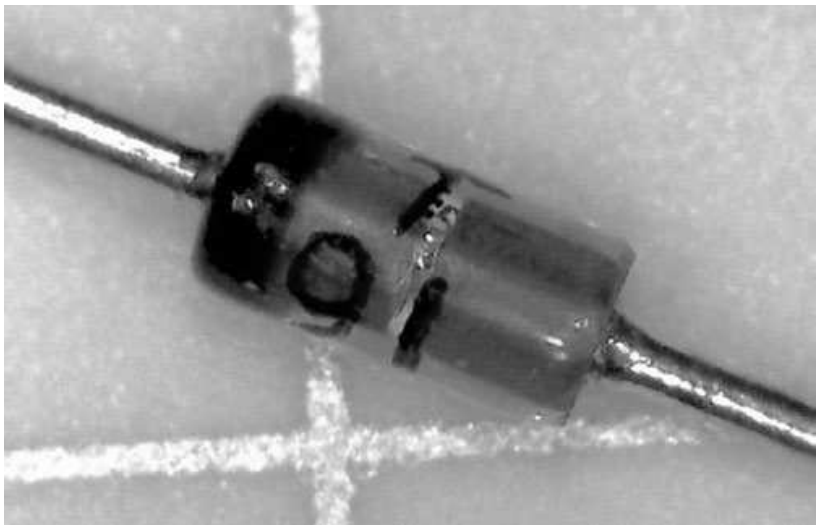

Figure 6-11. *1N914 diode to be tested*

You'll take advantage of the fact that you can establish a small current source from the DAC output AOUT. If the diode being tested were to act as a short circuit, you would want to limit the current to about 1mA for this experiment. You know that full voltage will be near 3.3 volts, so using Ohm's law, you can calculate the series resistance needed.

$$R = \frac{V}{0.001\ A} = \frac{3.3}{0.001} = 3300\ \Omega$$

The breadboard circuit is wired according to Figure 6-12, with R_1=3.3kΩ in series with the 1N914 diode D_1. The voltage will be applied out of the DAC to the series resistor R_1, while you take voltage readings at the top of diode D_1, into AIN0. The DAC will perform a slow sweep from 0 volts to the maximum. By measuring the voltage at the junction of R_1 and D_1, you'll be able to calculate the current in D_1 using Ohm's law.

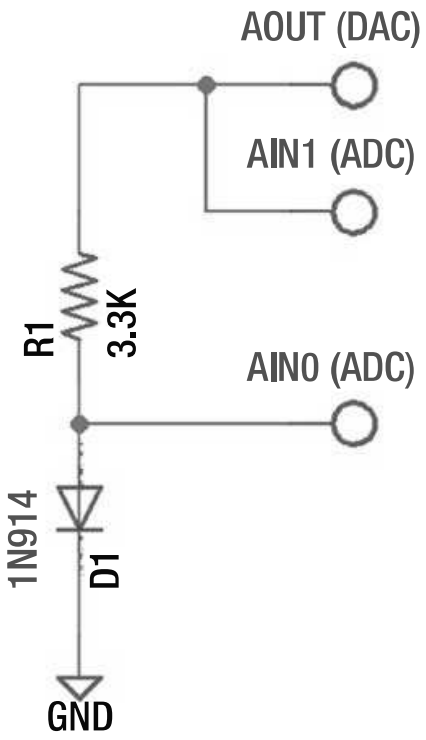

Figure 6-12. *1N914 diode circuit*

You also measure the voltage provided by the DAC itself using ADC input AIN1. This is done because the drive capability of the DAC is limited in current, causing its output voltage to droop. When the DAC output sags in voltage, you can use the AIN1 reading to more accurately assess the current actually flowing.

The program `diode.cpp` sets a test DAC voltage and then reads AIN0 and AIN1 to plot the readings. This experiment will use `gnuplot-x11` to plot the results, so if you don't have it installed yet, do the following:

```
# sudo apt-get install gnuplot-x11
```

If you `ssh`'ed into your Pi, make sure you enable X11 Window tunneling with the `-X` option.

```
$ ssh -X pi@pi
```

Further, if you expect `gnuplot` to open on your remote machine (like your Mac OS X laptop), you need to set up permissions or simply do the following on the display server:

```
$ xhost +
```

If you're using a keyboard and monitor attached to the Pi itself, you can ignore this.

Running the diode program requires no command-line options, unless you need to specify the I2C address.

```
$ ./diode -h
./diode [-a address] [-h]
where:
      -a address   Specify I2C address
      -h           Help
```

Once your breadboard circuit is ready, run the diode program.

```
$ ./diode
DAC 0, AIN0 1, AIN1 1, VD1=0.0, VR=0.0, I=0.0000
DAC 1, AIN0 1, AIN1 1, VD1=0.0, VR=0.0, I=0.0000
DAC 2, AIN0 2, AIN1 2, VD1=0.0, VR=0.0, I=0.0000
DAC 3, AIN0 3, AIN1 3, VD1=0.0, VR=0.0, I=0.0000
DAC 4, AIN0 4, AIN1 4, VD1=0.1, VR=0.0, I=0.0000
DAC 5, AIN0 5, AIN1 5, VD1=0.1, VR=0.0, I=0.0000
DAC 6, AIN0 6, AIN1 6, VD1=0.1, VR=0.0, I=0.0000
DAC 7, AIN0 6, AIN1 7, VD1=0.1, VR=0.0, I=0.0000
DAC 8, AIN0 7, AIN1 7, VD1=0.1, VR=0.0, I=0.0000
DAC 9, AIN0 8, AIN1 8, VD1=0.1, VR=0.0, I=0.0000
DAC 10, AIN0 9, AIN1 9, VD1=0.1, VR=0.0, I=0.0000
DAC 11, AIN0 10, AIN1 10, VD1=0.1, VR=0.0, I=0.0000
...
```

The program reports several lines to the terminal, which can be helpful in debugging your experiment. The DAC, AIN0, and AIN1 values are reported on each line. Additionally, the calculated V_{D1}, V_R, and I are displayed. V_{D1} and I are written to the file diode.dat. The diode program also writes out the file gnuplot.cmd that you can use to plot the results.

```
# gnuplot -p gnuplot.cmd
```

If all went well with gnuplot, you should have a plot displaying the results, as shown in Figure 6-13. If you chose not to remove the LED (D1) from the DAC circuit, you might not reach as high a current as shown, but you should get similar results. If gnuplot is not cooperating, check that your environment variable DISPLAY is set and exported.

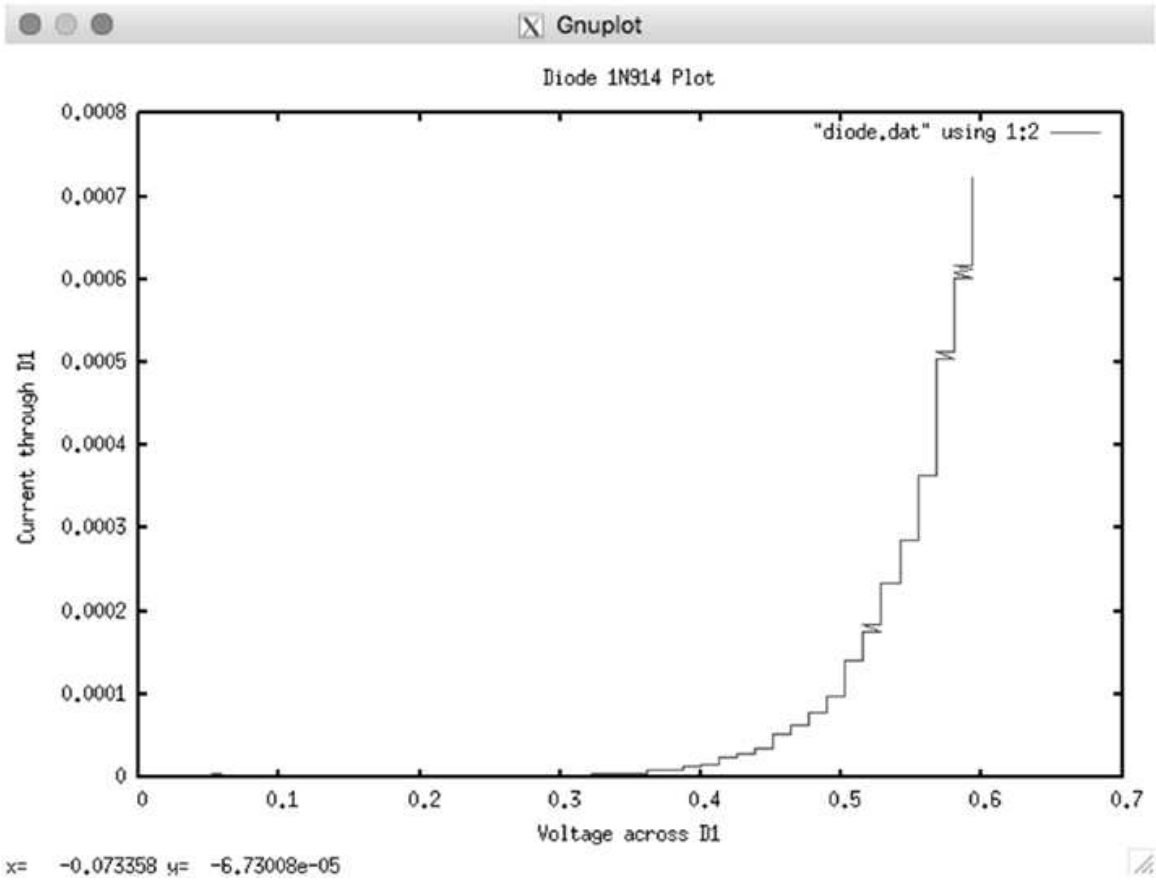

Figure 6-13. *gnuplot of diode test*

The plotted results show you that your diode reached a maximum of about 0.0007A in current (0.7mA) when the voltage reached or exceeded 0.6 volts. The curve demonstrates that the diode's response is exponential in nature, becoming almost linear after the 0.6V level was reached. This is the general characteristic for silicon diodes.

What happens if you try an LED instead? Does color make a difference? Also, try a germanium diode (1N34A).

Software

In this chapter you used the I2C bus driver in Linux to communicate to the PCF8591 device using the provided C++ programs. These are found in the `pcf8591` subdirectory and can be cloned and modified for your own purposes. The basics of I2C programming will be discussed in Chapter 11.

Potential Experiments

The following are potential experiment ideas that you can perform using the PCF8591 with any Raspberry Pi:

1. Measure the voltage of a 1.5V dry cell (of your choice) wired to a flashlight bulb. Measure the voltage over time, measuring the lifetime of the cell. Repeat with different brands, noting the cost of each cell. Construct a chart of the most economical batteries based upon brand, cost, and lifetime.
2. Repeat experiment 1 for rechargeable cells. How do they compare?
3. Using the YL-40 LDR, record the moonlight level for a week or more and use `gnuplot` to plot it.
4. Place your Pi in the refrigerator and track the light-on time to measure door open times.
5. Use the DAC to produce a low-frequency sine wave. Produce a triangle, saw, and square waves.

Summary

This chapter showed the versatility of the economical PCF8591 ADC/DAC chip. You also saw the variety of experiments that the YL-40 PCB offers after certain adjustments are made. Despite the limitations of 8-bit resolution, the response of a signal diode was measured and plotted. At the low price of a PCF8591, how can you argue with the geek fun that it provides?

Bibliography

[1] "Thermistor." *Using a Thermistor*. Adafruit.com, n.d. Web. 30 Jan. 2016. <https://learn.adafruit.com/thermistor/using-a-thermistor>.

[2] "Measuring Nocturnal Light." *Measuring Nocturnal Light*. N.p., n.d. Web. 30 Jan. 2016. <http://home.earthlink.net/~nevadabat/Moonlight/MoonLight.html>.