

# Sending output

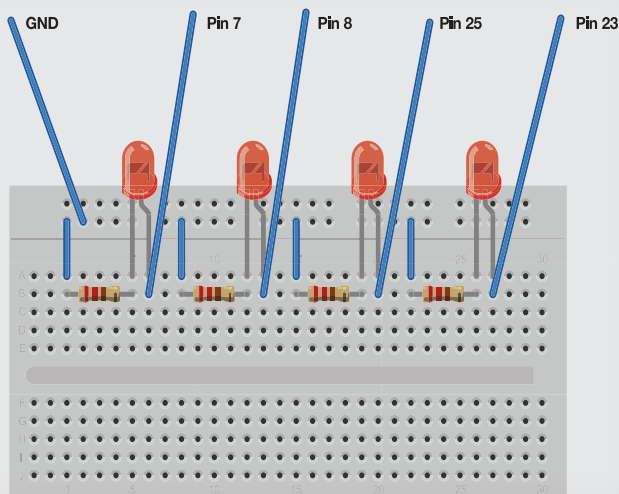
## Use the GPIO pins to light up some LEDs.

The Pi's tiny size makes it ideal for making your own embedded devices. This can be a great way of creating small computing devices to solve specific problems, as we saw with the camera controller.

However, there is the slight problem that it can be hard to know what's going on inside your Pi without a screen. Fortunately, the designers of the Pi thought of this problem and have added the facility to get information on and off a Pi without the bulk of usual PC peripherals. This is done via General Purpose Input and Output (GPIO).

You may have wondered what the spiky pins near the SD card reader are for – well, you're about to find out. This basic circuit

can be used to display information from any source, but here we're going to use it to find the final byte of the IP address. This is useful if you want to remotely access your Pi, but can't configure it with a static IP because, for example, you have to move it between networks. Typically, you can find out the first three bytes from the netmask, but the final one can be elusive unless you have a monitor attached.



■ Figure 1. This shows how half of the LEDs are wired up. The additional ones are added in exactly the same manner.

We're going to use the **gpio** program, which is part of *WiringPi*. You can find out more about this from the website: [bit.ly/RP8UKJ](http://bit.ly/RP8UKJ).

The software comes as source code, so we'll have to unzip it and compile it with:

```
tar xvf wiringPi.tgz
cd wiringPi/wiringPi
make
sudo make install
cd ../gpio
make
sudo make install
```

We'll also use **bc**, so install it with:

```
sudo apt-get install bc
```

Now, that's enough about software – on with the hardware! Just a quick word of warning before we start: it is possible to break your Pi by connecting the wrong wires together, so make sure you double-check before powering up.

The circuit for this is very simple – you just have to connect each output to the positive leg of an LED, then the negative leg of the LED (shorter) to a 1 KOhm resistor, and finally the other leg of the resistor to the common ground. See figures 1, 2 and 3 for details.

Once you have your fully-set-up board connected to your Pi, you can make things happen. To start with, we'll just use the final pin. This is pin 7 (the layout of the pins doesn't follow a numbering pattern). Open up a terminal, and set it to output with:

```
gpio -g mode 7 out
```

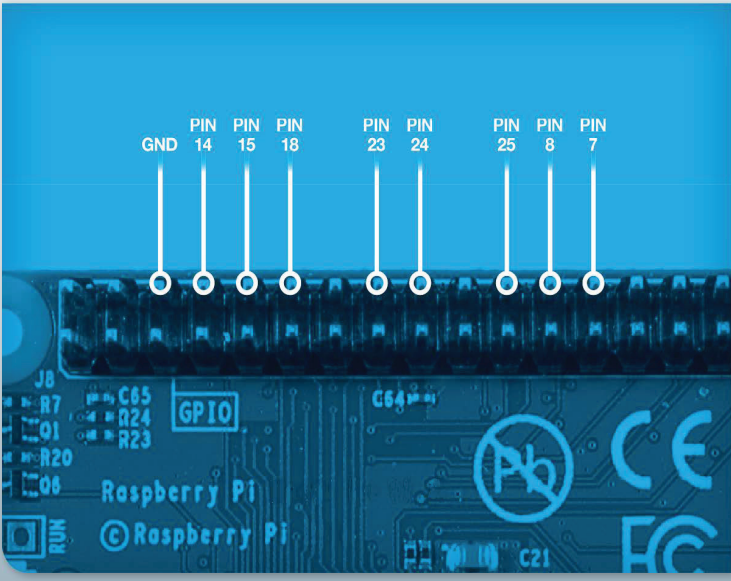
Then you can turn it on with:

```
gpio -g write 7 1
```

and off again with:

```
gpio -g write 7 0
```

If you're like us, you'll do that repeatedly until the novelty of it wears off. Once it has, you're ready to run the script. It contains four parts.



■ Figure 2. Connect the bread board to these pins. We used commercially-available single-pin connectors, but you could also solder connectors on, or use an old IDE cable.

The first just sets the pins to the right mode and makes sure they're turned off:

```
pins="7 8 25 24 23 18 15 14"

for x in $pins
do
    gpio -g mode $x out
    gpio -g write $x 0
done
```

The second grabs the IP address from `ifconfig`, converts it to binary, then pads it out with leading zeros, if necessary.

```
ipaddress='ifconfig eth0 | grep
'inete' | awk '{print $2}' | cut
-f4 -d'.'

binary='echo
"ibase=10;obase=2;$ipaddress" | bc`
paddedBinary='printf %08d $binary`
```

The next part uses `cut` to extract the part we want from this binary string and outputs it to the appropriate pin.

```
bit=1
for x in $pins
do
    out='echo $paddedBinary | cut
-b$bit`
    gpio -g write $x $out
    bit=$((bit+1))
done
```

And, finally, we tell the script to sleep for five minutes, then turn the LEDs off.

```
sleep 5m
for x in $pins
do
    gpio -g write $x 0
done
```

That's it! Save this into a file called `showIP.sh`, make it executable with:

```
chmod a+x showIP.sh
```

and type `sudo ./showIP.sh` to display your IP. To get this run automatically on boot, you just need to add the line:

```
/home/pi/showIP.sh &
```

to `rc.local`. See the 'Camera controller' section for details on how to do this.

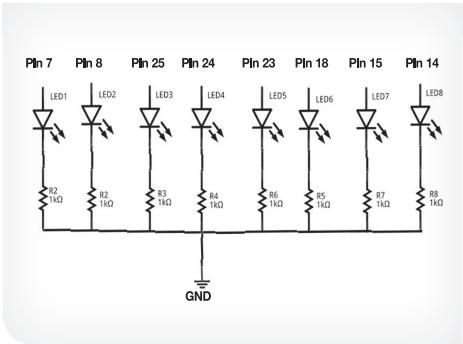
## Gertboards and Arduinos

Connecting directly to your Pi's GPIO pins can provide you with basic input and output control, but there are limitations. There are two additional items that you can obtain to help you interact more precisely with the world around you.

The Gertboard is a fairly complete expansion pack for connecting between your Pi and the real world, including a micro controller, and a range of input and output options.

Meanwhile, the Arduino is a micro controller that can connect to your Pi (or any other computer) via the USB port. Typically, it comes assembled, but kit forms are also available. In its raw form, it has fewer features than the Gertboard (which includes an Arduino microcontroller), but it can be expanded with a huge range of shields.

■ Figure 3. The simple circuit in all its glory.



We've shown you how to send output via the GPIO, but as the name suggests, they can also receive input.

---

With this, it's even more important to ensure that you don't send too much power into the pins. To get input, just set the mode to input with **gpio -g mode <pin number> in** and then read the value with **gpio -g read <pin number>**.

This hardware can display any eight bits of information, so you don't have to limit it to displaying just IP addresses. For example, you could make a modified version of the camera controller script to use the LEDs to indicate its progress.

You can find details on the full selection of GPIO pins at [bit.ly/JTIFE3](http://bit.ly/JTIFE3). If you design your

own circuits, or use ones off the web, make sure you use the right pins for your board.

You don't have to limit yourself to just switching pins on and off. The Pi supports a few methods of passing larger amounts of data through the GPIO. The two most common of these are Serial Peripheral Interface bus (SPI) and Inter-Integrated Circuit (I<sup>2</sup>C).

There are a number of devices available that use these, and plenty of information online to help get you started. So what's stopping you? Get out your soldering iron and build a robot army.

## Ohm's Law

---

There are two key ways of measuring electricity: voltage and current. Voltage (measured in volts) is the amount of energy a given quantity of electrons has, while current (measured in amps) is the amount of electrons flowing past a point.

The two are intimately connected by Ohm's law which states: Voltage = Current x Resistance, or  $V=IR$ . You can use this connection to make sure you don't accidentally toast your Raspberry Pi by pushing too much current into it. The exact setup of the Pi is a little complex. If you wish to delve into it, Gert van Loo (one of the designers) has put together an explanation, which can be found at: [bit.ly/Qp4PMI](http://bit.ly/Qp4PMI)

As a rough rule of thumb, you can expect to draw voltage out of a GPIO pin at 3.3V, and you shouldn't draw more than 16mA, or push more than this into an input pin.

This is the maximum current; you should aim to use less. So, with Ohm's law we know  $V=IR$ , so  $R=V/I$ . If we put in the data from the Pi, and want to ensure we don't damage it, we know that R must be greater than  $3.3/0.016$ , which is 206.25 Ohms.

Remember, this is the smallest amount of resistance it's safe to use with a GPIO output. You should aim for a margin of safety several times above this unless absolutely necessary. In our circuits, we've used 1,000 Ohms, which gives us a safety factor of almost 5.