GPIO Sensing: Using 1-Wire temperature sensors - Part 2

Jacob Marsh

ModMyPi

SKILL LEVEL : BEGINNER

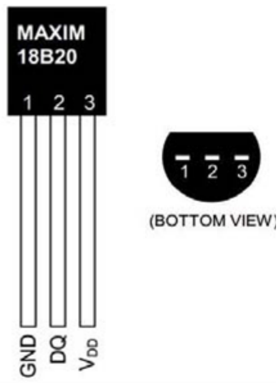
1-Wire sensors

In previous tutorials we've outlined the integration of simple sensors and switches with the Raspberry Pi. These components have had a simple on/off or high/low output, which is sensed by the Raspberry Pi. Our PIR movement sensor tutorial in Issue 21, for example, simply says "Yes, I've detected movement".

So, what happens when we connect a more advanced sensor and want to read more complex data? In this tutorial we will connect a 1-Wire digital thermometer sensor and programme our Raspberry Pi to read the output of the temperature it senses!

In 1-Wire sensors all data is sent down one wire, which makes it great for microcontrollers and computers, such as the Raspberry Pi, as it only requires one GPIO pin for sensing. In addition to this, most 1-Wire sensors will come with a unique serial code (more on this later) which means you can connect multiple units without them interfering with each other.

The sensor we're going to use in this tutorial is the Maxim DS18B20+ Programmable Resolution 1-Wire Digital Thermometer. The DS18B20+ has a similar layout to transistors, called the TO-92 package, with three pins: GND, Data (DQ) and 3.3V power line (V_{DD}). You also need some jumper wires, a breadboard and a 4.7kΩ (or 10kΩ) resistor.



The resistor in this setup is used as a 'pull-up' for the data-line, and should be connected between the DQ and V_{DD} line. It ensures that the 1-Wire data line is at a defined logic level and limits interference from electrical noise if our pin was left floating. We are also going to use GPIO 4 [Pin 7] as the driver pin for sensing the thermometer output. This is the dedicated pin for 1-Wire GPIO sensing.

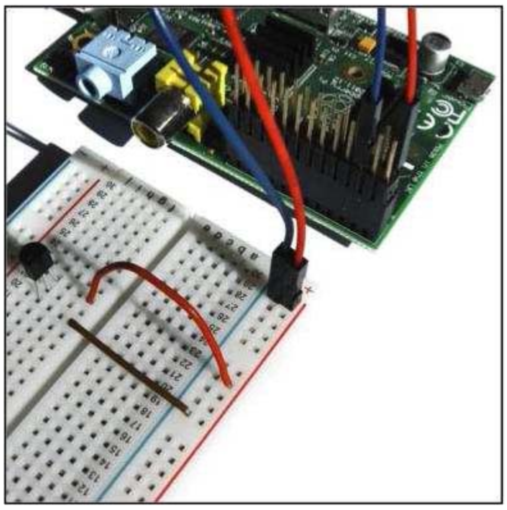
Hooking it up

1. Connect GPIO GND [Pin 6] on the Raspberry Pi to the negative rail on the breadboard.
2. Connect GPIO 3.3V [Pin 1] on the Raspberry Pi to the positive rail on the breadboard.
3. Plug the DS18B20+ into your breadboard,

ensuring that all three pins are in different rows. Familiarise yourself with the pin layout, as it is quite easy to hook it up backwards!

4. Connect DS18B20+ GND [Pin 1] to the negative rail of the breadboard.

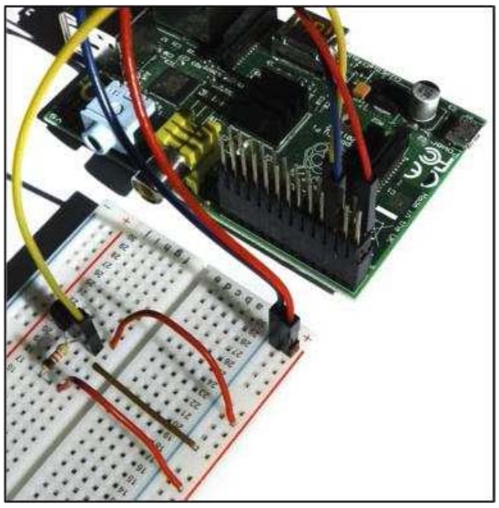
5. Connect DS18B20+ V_{DD} [Pin 3] to the positive rail of the breadboard.



6. Place your 4.7kΩ resistor between DS18B20+ DQ [Pin 2] and a free row on your breadboard.

7. Connect that free end of the 4.7kΩ resistor to the positive rail of the breadboard.

8. Finally, connect DS18B20+ DQ [Pin 2] to GPIO 4 [Pin 7] with a jumper wire.



That's it! We are now ready for some programming!

Programming

With a little set up, the DS18B20+ can be read directly from the command line without the need for any Python programming. However, this requires us to input a command every time we want to know the temperature reading. In order to introduce some concepts for 1-Wire interfacing, we will access it via the command line first and then we will write a Python program which will read the temperature automatically at set time intervals.

The Raspberry Pi comes equipped with a range of drivers for interfacing. However, it's not feasible to load every driver when the system boots, as it increases the boot time significantly and uses a considerable amount of system resources for redundant processes. These drivers are therefore stored as loadable modules and the command `modprobe` is employed to boot them into the Linux kernel when they're required.

The following two commands load the 1-Wire and thermometer drivers on GPIO 4. At the command line enter:

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
```

We then need to change directory to our 1-Wire device folder and list the devices in order to ensure that our thermometer has loaded correctly. Enter:

```
cd /sys/bus/w1/devices
ls
```

In the device list, your sensor should be listed as a series of numbers and letters. In my case, the device is registered as 28-000005e2fdc3. You then need to access the sensor with the `cd` command, replacing the serial number with that from your own sensor. Enter:

```
cd 28-000005e2fdc3
```

The sensor periodically writes to the `w1_slave` file. We can use the `cat` command to read it:

```
cat w1_slave
```

This yields the following two lines of text, with the output `t` showing the temperature in milli-degrees Celsius. Divide this number by 1000 to get the temperature in degrees, e.g. the temperature reading we've received is 23.125 degrees Celsius.

```
72 01 4b 46 7f ff 0e 10 57 : crc=57 YES
72 01 4b 46 7f ff 0e 10 57 t=23125
```

In terms of reading from the sensor, this is all that's required from the command line. Try holding onto the thermometer for a few seconds and then take another reading. Spot the increase? With these commands in mind, we can now write a Python program to output our temperature data automatically.

Python program

Our first step is to import the required modules. The os module allows us to enable our 1-Wire drivers and interface with the sensor. The time module allows the Raspberry Pi to define time, and enables the use of time periods in our code.

```
import os
import time
```

We then need to load our drivers:

```
os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')
```

The next step is to define the sensor's output file (the `w1_slave` file) as defined above. Remember to utilise your own temperature sensor's serial code!

```
temp_sensor = '/sys/bus/w1/devices/28-000005e2
fdc3/w1_slave'
```

We then need to define a variable for our raw temperature value, `temp_raw`; the two lines output by the sensor, as demonstrated by the command line example. We could simply print this statement now, however we are going to process it into something more usable. To do this we open, read, record and then close the `temp_sensor` file. We use the `return` function here, in order to recall this data at a later stage in our code.

```
def temp_raw():
  f = open(temp_sensor, 'r')
  lines = f.readlines()
  f.close()
  return lines
```

First, we check our variable from the previous function for any errors. If you study our original output, as shown in the command line example, we get two lines of output code. The first line was "72 01 4b 46 7f ff 0e 10 57 : crc=57 YES". We strip this line, except for the last three characters, and check for the "YES" signal, which indicates a successful temperature reading from the sensor. In Python, not-equal is defined as "!=", so here we are saying that while the reading does not equal "YES", sleep for 0.2s and repeat.

```
def read_temp():
  lines = temp_raw()
  while lines[0].strip()[-3:] != 'YES':
    time.sleep(0.2)
    lines = temp_raw()
```

Once a YES signal has been received, we proceed to our second line of output code. In our example this was "72 01 4b 46 7f ff 0e 10 57 t=23125". We find our temperature output "t=", check it for errors and strip the output of the "t=" phrase to leave just the temperature data. Finally we run two calculations to give us the temperature in Celsius and Fahrenheit.

```
  temp_output = lines[1].find('t=')
  if temp_output != -1:
    temp_string = lines[1].strip()[temp_output
+2:]
    temp_c = float(temp_string) / 1000.0
    temp_f = temp_c * 9.0 / 5.0 + 32.0
    return temp_c, temp_f
```

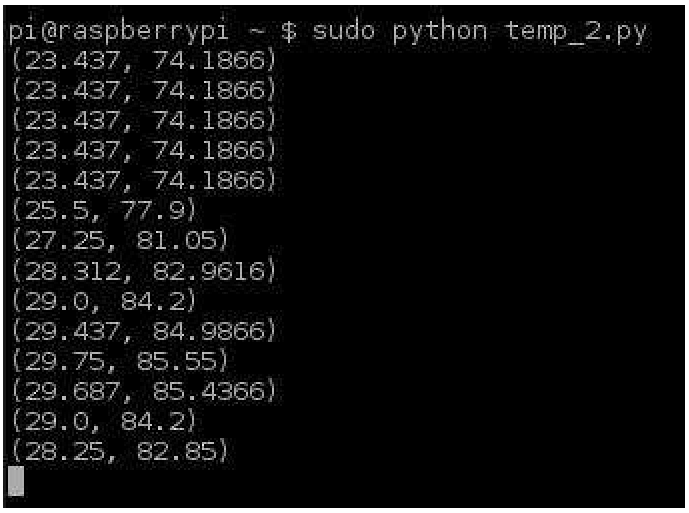
Finally, we loop our process and tell it to output our temperature data every 1 second.

```
while True:
  print(read_temp())
  time.sleep(1)
```

That's our code! A screenshot of the complete program is shown below. Save your program and run it to display the temperature output, as shown on the right.

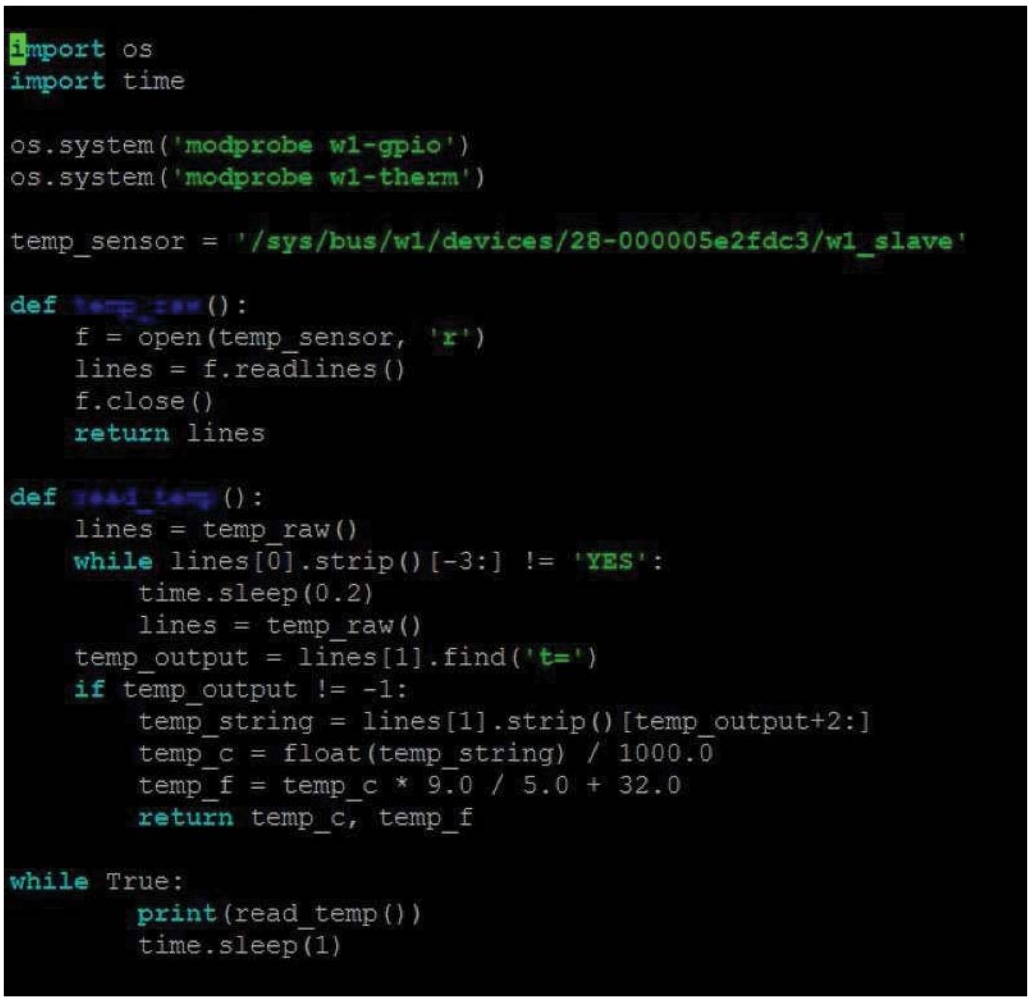
Multiple sensors

DS18B20+ sensors can be connected in parallel and accessed using their unique serial number. Our Python example can be edited to access and read from multiple sensors!



```
pi@raspberrypi ~ $ sudo python temp_2.py
(23.437, 74.1866)
(23.437, 74.1866)
(23.437, 74.1866)
(23.437, 74.1866)
(23.437, 74.1866)
(25.5, 77.9)
(27.25, 81.05)
(28.312, 82.9616)
(29.0, 84.2)
(29.437, 84.9866)
(29.75, 85.55)
(29.687, 85.4366)
(29.0, 84.2)
(28.25, 82.85)
```

As always, the DS18B20+ sensor and all components are available separately or as part of our workshop kit from the ModMyPi website http://www.modmypi.com.



```
import os
import time

os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')

temp_sensor = '/sys/bus/w1/devices/28-000005e2fdc3/w1_slave'

def temp_raw():
    f = open(temp_sensor, 'r')
    lines = f.readlines()
    f.close()
    return lines

def read_temp():
    lines = temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = temp_raw()
    temp_output = lines[1].find('t=')
    if temp_output != -1:
        temp_string = lines[1].strip()[temp_output+2:]
        temp_c = float(temp_string) / 1000.0
        temp_f = temp_c * 9.0 / 5.0 + 32.0
        return temp_c, temp_f

while True:
        print(read_temp())
        time.sleep(1)
```

This article is sponsored by ModMyPi

All breakout boards and accessories used in this tutorial are available for worldwide shipping from the ModMyPi webshop at www.modmypi.com