Raspberry Pi Python GPIO Zero Distance Sensor *Additional Notes*

Tutorial by Jonathan Teague - Public Domain
28th Jan 2017 - www.cotswoldjam.org

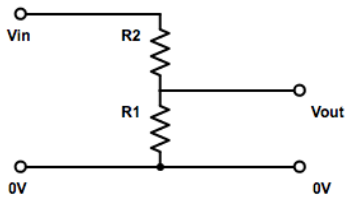
These notes are additonal to the main tutorial and explain the use of two resistors to make a “voltage divider” and give an example program that directly controls the sensor..

The Voltage Divider

The Raspberry Pi is a 3.3V device. The sensor used in this tutorial, an HC-SR04, is a 5V device. This means that the sensor needs 5V to power it and, crucially for the Pi, that the voltage levels that the sensor returns the results to the Pi are also at 5V. Connecting the sensor output directly to the Pi would almost certainly result in damage to the Pi.

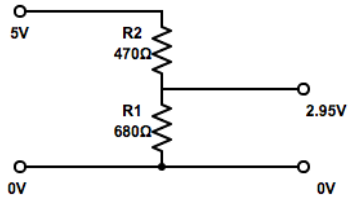
This sort of situation is often found in the real world, you have two devices you wish to connect to each other but they operate at different voltages. There are several ways to solve this but one of the simplest, and the one used here, is to drop the voltage from 5V to just under the safe value of 3.3V by making a “voltage divider” out of two resistors.

The two resistors are placed in series across the source voltage and zero volts and the voltage at the point between the two resistors will be reduced in proporation with the values of the two resistors.



$$Vout = Vin\frac{R1}{R1 + R2}$$

So substituting in the values for 5V as the Vin and the two resistors we have used, R1 = 680Ω and R2 = 470Ω we get:



$$Vout = 5V\frac{680}{470 + 680} = 2.95V$$

This 2.95V is then safe to connect to the Pi.

A program to interface to the HC-SR04 directly

In the tutorial we don't talk directly to the HC-SR04 but leave all that to the DistanceSensor functionality of the GPIO Zero library. This is convenient but sometimes it's useful to know exactly what is going on and to be able to understand how to control devices directly.

In the **`~/python/distance` folder** you will find a file `rpigpio_DistanceSensor.py` which shows how to trigger and read the HC-SR04 sensor directly and which uses the RPi.GPIO library.

Here is the important bit that triggers the HC-SR04 to take a measurement and gets the result back, it's all contaned in a function called `measure()`:

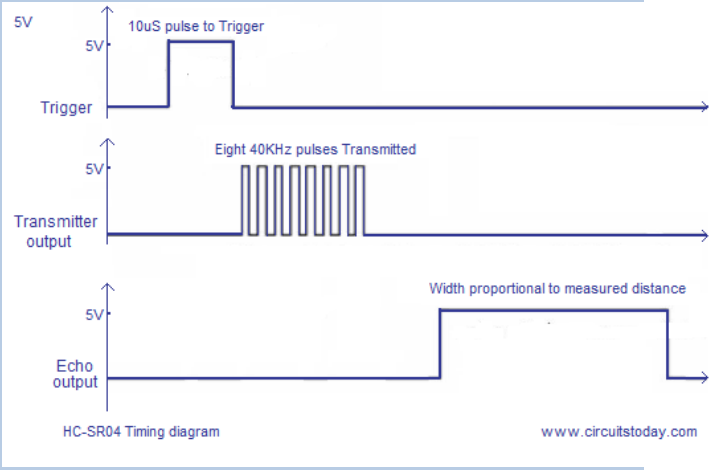
```
def measure():
  # This function measures a distance
  GPIO.output(GPIO_TRIGGER, True)
  time.sleep(0.00001)
  GPIO.output(GPIO_TRIGGER, False)

  start = time.time()
  while GPIO.input(GPIO_ECHO)==0:
    start = time.time()

  while GPIO.input(GPIO_ECHO)==1:
    stop = time.time()

  elapsed = stop-start
  distance = (elapsed * 34300)/2

  return distance
```



What does this do?

Setting the GPIO_TRIGGER to True, sleeping for 10uS then setting the GPIO_TRIGGER to False triggers the sensor to take a reading.

The code then sets a variable called `start` to the current time and goes into a tight loop while GPIO_ECHO is 0 (= low), every time round the loop it resets the variable `start`.

Once GPIO_ECHO is 1 (= high) this first look will stop. At this point `start` contains the time when the Echo pin went high.

Now the code goes into a tight loop while GPIO_ECHO is high, every time round the loop setting the variable `stop`.

Finally when the GPIO_ECHO goes low again the loop exits. At this point `start` contains the time when the Echo went high and `stop` the time when it went low again.

Now it's just some maths to work out how long the Echo pin was high for, multiply this by the speed of sound (the 34300) and divide by 2 (because the sound from the sensor went out and back) and we get the `distance` value in cm.