Connecting a pH Sensor to a Raspberry Pi

by Dominic | May 13, 2016 | 48 comments

Raspberry Pi Ph Sensor

Whether you want to monitor a pool, aquarium or some other body of water, connecting a pH sensor to a Raspberry Pi can be achieved relatively easily. In this tutorial i'll be using the Atlas Scientific pH sensor, it's an industrial grade sensor that works well with the Raspberry Pi, it's fully submersible up to the BNC connector in both fresh and saltwater. The sensor works using either serial communication or via the I2C protocol, for this example I will be configuring the sensor to use the I2C interface on the Raspberry Pi.

To configure the RPi I am assuming that you are running the latest version of Raspian and have the ability to connect to your Pi either through SSH with putty and FTP with filezilla, or directly with a keyboard and monitor, if you haven't set-up your RPi yet then check out my getting started section.

The first thing we need to do is enable the I2C modules on the RPi. This is done by entering the following at the command prompt to start the configuration tool

```
sudo raspi-config
```

select option 9 – Advanced Options
select option A7 – I2C
select "Yes" for all the questions and reboot the RPi

```
sudo reboot
```

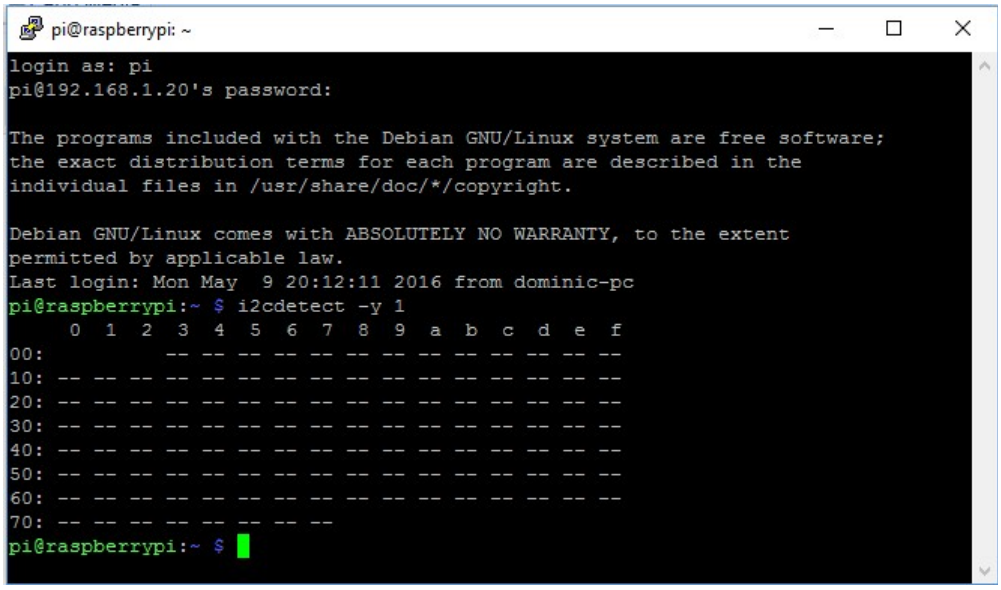
Note: The GPIO pins 2&3 on the RPi have now been configured as the Serial Data Line (SDA) and Serial Clock Line (SCL) for use by the I2C protocol. The TX connection of the sensor circuit will connect to SDA (pin 2) on the Pi and the RX connection will go to the SCL (pin 3).

After the reboot connect to the command prompt and enter

```
sudo apt-get update
```

```
sudo apt-get install i2c-tools
i2cdetect -y 1
```

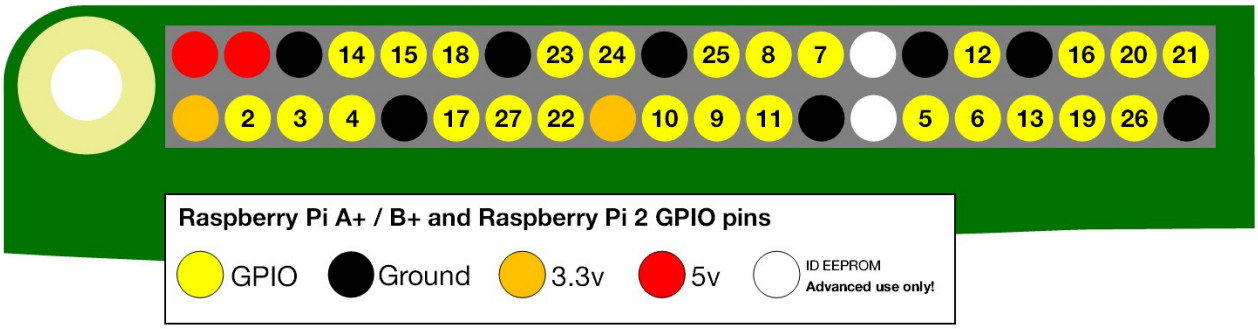
This should produce the following without the sensor attached.



Now that we have our I2C module working correctly we can go ahead and connect our pH sensor. The following materials will be needed to get started:

- Raspberry Pi
- Atlas Scientific pH sensor kit
- Breadboard
- Jumper Wires
- Adafruit T-Cobbler Plus (optional)

When describing the physical pin connections I will be following the GPIO pin numbering convention show below.

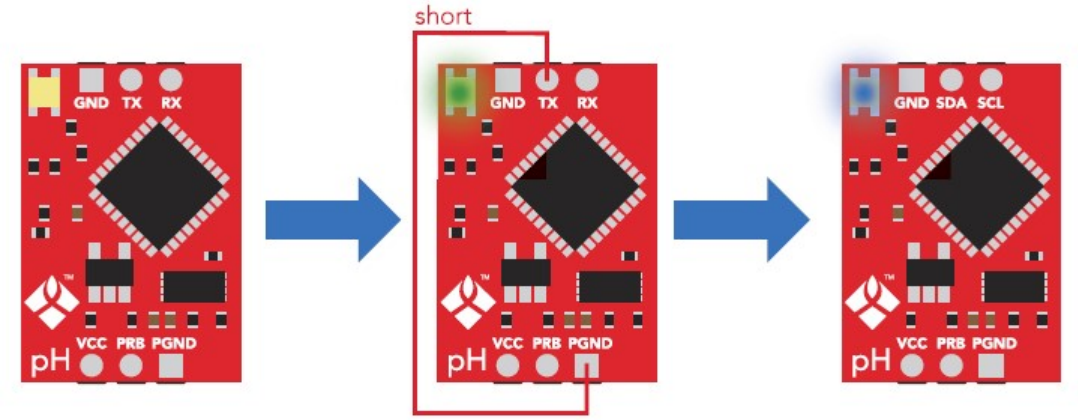


Firstly we need to get the pH circuit into the correct mode, when delivered the pH circuit will be in UART (serial) mode, the pH circuit has to be manually switched from UART mode, to I2C mode. When this is done the pH circuit will have its I2C address set to 99 (0x63).

Using your breadboard perform the following actions

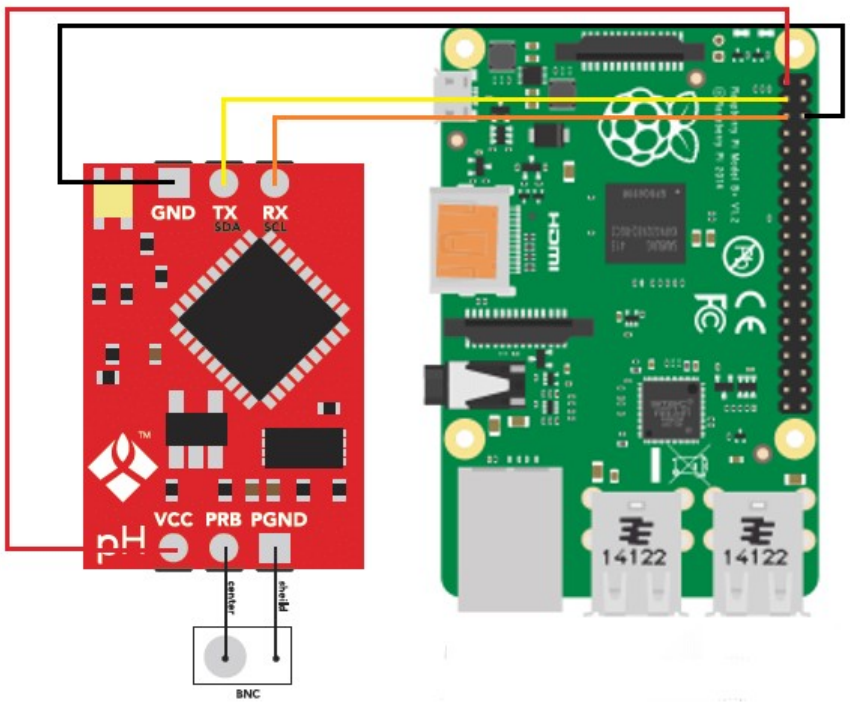
1. Cut the power to the device

2. Disconnect any jumper wires going from TX and RX to the RPi
3. Short the PGND pin to the TX pin
4. Power the device
5. Wait for LED to change from Green to Blue
6. Remove the short from the probe pin to the TX pin
7. Power cycle the device



The device is now I2C mode.

The RPi and pH circuit are now configured so we can go ahead and connect it all together

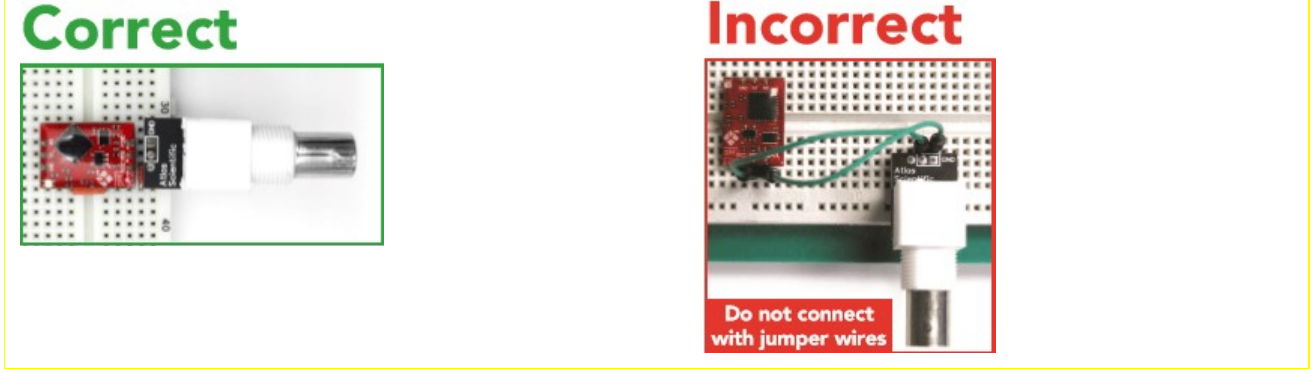


Assuming that all of the parts are now mounted on your breadboard

1. Connect the GND pin of the pH circuit to the ground pin of your RPi.
2. Connect the TX(SDA) pin to GPIO pin 2.
3. Connect the RX(SCL) pin to GPIO pin 3.

Note: Do Not Use jumper wires for these connections or your readings will not be accurate.

Incorrect

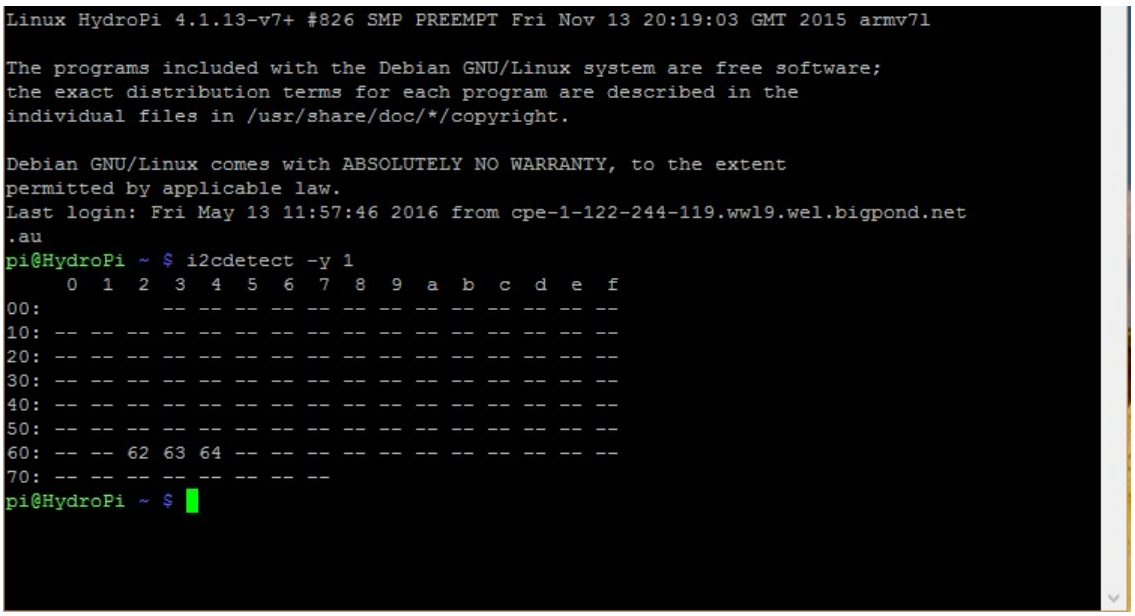


4. The PRB and PGND pins should be connected via your breadboard to the centre and shield pins of your BNC connector.
5. Finally power your pH circuit by connecting the Vcc pin to the +3.3V pin.

You can now run a quick test to prove that we are setup correctly, from the command prompt enter the following:

```
i2cdetect -y 1
```

you should see the following response, if not then check you connections, ensure the light on the pH circuit is blue and reboot your RPi.



In the image above I have 3 sensors connected to my RPi, the pH sensor connection is indicated by Hex value 63. The factory pre-set address for the pH sensor is 99 or 63 in hexadecimal as mentioned above, if you have more than 1 pH circuit connected then you will need to specify a different value. To do this we need to add some python code to our RPi.

Atlas Scientific provide the python code that I will be using here for interfacing with the pH Circuit.

We start by importing the required python modules

```
import io  # used to create file streams
import fcntl  # used to access I2C parameters like addresses
import time  # used for sleep delay and timestamps
import string  # helps parse strings
```

Next we add the class code to interface with the pH circuit (or any other Atlas Scientific circuit for that matter)

```
class atlas_i2c:
    long_timeout = 5  # the timeout needed to query readings and calibrations
    short_timeout = 5  # timeout for regular commands
    default_bus = 1  # the default bus for I2C on the newer Raspberry Pis,
                     #certain older boards use bus 0
    default_address = 99  # the default address for the pH sensor

    def __init__(self, address=default_address, bus=default_bus):
        # open two file streams, one for reading and one for writing
        # the specific I2C channel is selected with bus
        # it is usually 1, except for older revisions where its 0
        # wb and rb indicate binary read and write
        self.file_read = io.open("/dev/i2c-" + str(bus), "rb", buffering=0)
        self.file_write = io.open("/dev/i2c-" + str(bus), "wb", buffering=0)

        # initializes I2C to either a user specified or default address
        self.set_i2c_address(address)

    def set_i2c_address(self, addr):
        # set the I2C communications to the slave specified by the address
        # The commands for I2C dev using the ioctl functions are specified in
        # the i2c-dev.h file from i2c-tools
        I2C_SLAVE = 0x703
        fcntl.ioctl(self.file_read, I2C_SLAVE, addr)
        fcntl.ioctl(self.file_write, I2C_SLAVE, addr)

    def write(self, string):
        # appends the null character and sends the string over I2C
        string += "\00"
        self.file_write.write(string)

    def read(self, num_of_bytes=31):
        # reads a specified number of bytes from I2C,
        #then parses and displays the result
        res = self.file_read.read(num_of_bytes)  # read from the board
        response = filter(lambda x: x != '\x00', res)
        # remove the null characters to get the response
        if(ord(response[0]) == 1):  # if the response isnt an error
            char_list = map(lambda x: chr(ord(x) & ~0x80), list(response[1:]))
            # change MSB to 0 for all received characters except the first
            #and get a list of characters
            # NOTE: having to change the MSB to 0 is a glitch in the raspberry
            #pi, and you shouldn't have to do this!
            return "Command succeeded " + ''.join(char_list)
```

```
            # convert the char list to a string and returns it
        else:
            return "Error " + str(ord(response[0]))

    def query(self, string):
        # write a command to the board, wait the correct timeout,
        #and read the response
        self.write(string)

        # the read and calibration commands require a longer timeout
        if((string.upper().startswith("R")) or
           (string.upper().startswith("CAL"))):
            time.sleep(self.long_timeout)
        elif((string.upper().startswith("SLEEP"))):
            return "sleep mode"
        else:
            time.sleep(self.short_timeout)

        return self.read()

    def close(self):
```

Finally we will add our main program

```
def main():
    device = atlas_i2c()  # creates the I2C port object, specify the address
                          # or bus if necessary
    print ">> Atlas Scientific sample code"
    print ">> Any commands entered are passed to the board via I2C except:"
    print (">> Address,xx changes the I2C address the Raspberry Pi "
    "communicates with.")
    print (">> Poll,xx.x command continuously polls the board every "
    "xx.x seconds")
    print (" where xx.x is longer than the %0.2f second "
    "timeout." % atlas_i2c.long_timeout)
    print " Pressing ctrl-c will stop the polling"

    # main loop
    while True:
        myinput = raw_input("Enter command: ")

        # address command lets you change which address
        # the Raspberry Pi will poll
        if(myinput.upper().startswith("ADDRESS")):
            addr = int(string.split(myinput, ',')[1])
            device.set_i2c_address(addr)
            print ("I2C address set to " + str(addr))

        # contiuous polling command automatically polls the board
        elif(myinput.upper().startswith("POLL")):
            delaytime = float(string.split(myinput, ',')[1])

            # check for polling time being too short,
            # change it to the minimum timeout if too short
            if(delaytime < atlas_i2c.long_timeout):
                print ("Polling time is shorter than timeout, setting "
```

```
            "polling time to %0.2f" % atlas_i2c.long_timeout)
            delaytime = atlas_i2c.long_timeout

        # get the information of the board you're polling
        info = string.split(device.query("I"), ",")[1]
        print ("Polling %s sensor every %0.2f seconds, press ctrl-c "
        "to stop polling" % (info, delaytime))

        try:
            while True:
                print device.query("R")
                time.sleep(delaytime - atlas_i2c.long_timeout)
        except KeyboardInterrupt:  # catches the ctrl-c command,
                                   # which breaks the loop above
            print "Continuous polling stopped"

    # if not a special keyword, pass commands straight to board
    else:
        try:
            print device.query(myinput)
```

All of this python code is available for both 2.x and 3.x on my HydroPi GitHub repository.

We now transfer our code to our chosen folder on the RPi using an FTP client and then run the program.

```
Linux HydroPi 4.1.13-v7+ #826 SMP PREEMPT Fri Nov 13 20:19:03 GMT 2015 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri May 13 12:41:49 2016 from cpe-1-122-244-119.ww19.wel.bigpond.net
.au
pi@HydroPi ~ $ cd /home/pi/Hydropi
pi@HydroPi ~/Hydropi $ ls
read_temperature.py  rpi_i2c_ph_sensor_code.py  WiFi_Check.sh
pi@HydroPi ~/Hydropi $ python rpi_i2c_ph_sensor_code.py
>> Atlas Scientific sample code
>> Any commands entered are passed to the board via I2C except:
>> Address,xx changes the I2C address the Raspberry Pi communicates with.
>> Poll,xx.x command continuously polls the board every xx.x seconds
 where xx.x is longer than the 1.50 second timeout.
 Pressing ctrl-c will stop the polling
Enter command:
```

The screenshot above shows that we are ready to start sending commands to our pH circuit, to confirm that sensor is now fully functioning we will enter the following command

Poll,2.0

This will poll the sensor every 2 seconds and return the result until a ctrl-c command is entered as shown below, to stop the program enter ctrl-c again.

```
Linux HydroPi 4.1.13-v7+ #826 SMP PREEMPT Fri Nov 13 20:19:03 GMT 2015 armv7l

The programs included with the Debian GNU/Linux system are free software;
```

```
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri May 13 17:45:10 2016 from cpe-1-122-244-119.ww19.wel.bigpond.net
.au
pi@HydroPi ~ $ cd /home/pi/Hydropi
pi@HydroPi ~/Hydropi $ ls
read_temperature.py  rpi_i2c_ph_sensor_code.py  WiFi_Check.sh
pi@HydroPi ~/Hydropi $ python rpi_i2c_ph_sensor_code.py
>> Atlas Scientific sample code
>> Any commands entered are passed to the board via I2C except:
>> Address,xx changes the I2C address the Raspberry Pi communicates with.
>> Poll,xx.x command continuously polls the board every xx.x seconds
 where xx.x is longer than the 1.50 second timeout.
 Pressing ctrl-c will stop the polling
Enter command: Poll,2.0
Polling pH sensor every 2.00 seconds, press ctrl-c to stop polling
Command succeeded 7.508
Command succeeded 7.491
Command succeeded 7.403
Command succeeded 7.384
Command succeeded 7.465
Command succeeded 7.525
Command succeeded 7.463
^CContinuous polling stopped
Enter command:
```

Warning: If you are using an Electrical Conductivity (EC) sensor in your project then it is important to electrically isolate other sensors from it, this can be done using an Atlas Scientific Pwr-Iso board on each of the circuits. The EC circuit discharges a small electrical current into the water. This small current creates an interference field that can be detected by devices such the pH probe which may make your readings inaccurate.

With the sensor now working there are also a series of other commands that we now have available to us to configure our probe.

Enable/disable the LED on the pH circuit:

L,1 – LED enable
L,0 – LED disable
L,? – Query the LED

Take a single reading:

R – Returns a single result

Temperature Compensation:

To ensure accurate results the probe needs to know the temperature of the liquid it is measuring in °C, factory default is 25°C

T,22.5 – Set the temperature offset value
T,? – Query the set temperature

Calibration:

The pH circuit allows for calibration of either 1,2 or 3 points, pH 7.00 must be performed first and is known as the mid calibration point. There are 2 other points available known as low and high.

Cal,mid,X.XX – Where X.XX represents the pH midpoint. In most cases this should be 7.00
Cal,low,X.XX – Where X.XX represents a low calibration point (pH 1 to pH 6)
Cal,high,XX.XX – Where XX.XX represents a high calibration point (pH 8 to pH 14)

Cal,clear – Clears all calibration data
Cal,? – Query the calibration

Circuit Address Change:

I2c,n – n is the new decimal address

Changes to the address of the circuit will cause a loss of connectivity until the python script is restarted with the new address.

Info, Status, Low Power and Factory Reset:

I – Device information
STATUS – Reports reason for last reboot and Vcc voltage
FACTORY – Factory reset. This will not change the communications protocol back to UART.
SLEEP – Enter low power sleep state.

> **Note:** Any command sent to the pH circuit will wake it but 4 readings should be taken before considering them to be accurate.

If you would like to see how I have implemented a pH sensor why not check out my Hydropi overview page which has all the articles related to my own personal project.

For more information on configuration of the Atlas Scientific pH circuit read this.

There we have it, you have now configured your RPi to interface with the Atlas Scientific pH Sensor.

If you have any thought's about this article, improvements or errors let me know in the comments below and if you found this helpful, why not share it with others.