CONNECTING AN ARDUINO TO A RASPBERRY PI USING I2C

Some time ago I created a weather station using a Raspberry PI and an off the shelf weather station, connecting the two via USB.

However, for some time not I've been meaning to create a weather station from scratch – i.e. one or more Raspberry PI's which connect to the network (via Ethernet or WiFi) and directly monitor the sensors directly.

Now the problem here is that some sensors are analog – for example the leaf, soil and UV sensors I have generate an analog signal so we need an ADC (Analogue to Digital Converter) which the Raspberry PI doesn't have.

So we have two possible solutions:

1. Add a Raspberry PI compatible ADC
2. Use an Arduino

With the parts I have available, the Arduino won, not just on available ADC channels but also with the additional digital ports available.

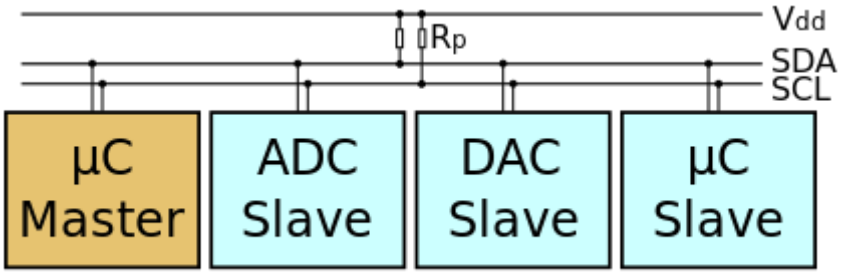
Now how to connect it to the PI? Well the easiest way is to use USB, however the PI only has two USB ports (one for the Model A) and as I'm intending to use Model A's for the final station I need that for WiFi (there won't be room or power for hubs) so USB is out.

There's RS232 which both support, however the PI runs on 3v3 whilst the Arduino (UNO) is 5v so I need to add a level converter between the two. It also limits me to just one arduino and I might need to use more than one so another solution is needed.

Enter I2C

Both the PI and Arduino support two additional types of communication for talking to peripheral devices. There's SPI which is a high speed serial protocol and I2C. Like RS232, SPI needs level shifters, but not exactly so for I2C.

I2C is a 2 wire protocol allowing for 127 devices to be connected to a single bus. One device is the master (The PI in our case) and then the peripherals.



An example I2C network (From Wikipedia)

In the above diagram you can see that there's two connections between devices (other than ground), SDA (Serial Data Line) which is where the data is carried, and SCL (Serial Clock Line). There's also a pair of resistors which pull up the signals to Vdd.

Now the trick, Vdd is only there to pull those signals up and in I2C a 1 is when the signal is pulled down to 0V. It's not there to power the devices so, as long as we keep Vdd at 3v3 and no device has a pull up resistor on them (i.e. to 5V) then we are save to connect it to the PI. There's only a problem if any device on the I2C bus also has a pull up resistor.

Now do the Arduino's have pullup resisitors? Well they actually don't, they actually cannot as the I2C interface is shared by two of the analogue inputs (4 & 5 to be precise) so there cannot be a resistor there else it would affect those pins when not being used for I2C.

So, we have a solution as long as the Raspberry PI is the I2C Master which is what we want. Also, of the available GPIO pins, only SDA and SCL have pull up resistors, so we are set.

First the obligitory warning

If you are uncertain of anything, like blowing up your PI etc then don't follow this any further. You do this at your own risk.

Configuring the PI for I2C

First we need to enable the I2C module on the PI.

Remove I2C from the module blacklist

As root edit /etc/modprobe.d/raspi-blacklist.conf and comment out the line blacklisting i2c-bcm2708

```
$ cat /etc/modprobe.d/raspi-blacklist.conf
# blacklist spi and i2c by default (many users don't need them)
blacklist spi-bcm2708
#blacklist i2c-bcm2708
```

Next add i2c-dev to the /etc/modules file so it's loaded on boot:

```
# /etc/modules: kernel modules to load at boot time.
#
# This file contains the names of kernel modules that should be loade
# at boot time, one per line. Lines beginning with "#" are ignored.
# Parameters can be specified after the module name.

snd-bcm2835
ipv6
i2c-dev
```

Finally install i2c-tools:

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

Now reboot the PI.

Configuring the Arduino

The following sketch implements a simple I2C slave with two commands:

Command 1 will toggle the onboard led on the Arduino.

Command 2 will return the arduino's temperature in Celsius.

```
#include <Wire.h>

#define SLAVE_ADDRESS 0x04
int number = 0;
int state = 0;

double temp;
```

```
 8
 9  void setup() {
10   pinMode(13, OUTPUT);
11
12   // initialize i2c as slave
13   Wire.begin(SLAVE_ADDRESS);
14
15   // define callbacks for i2c communication
16   Wire.onReceive(receiveData);
17   Wire.onRequest(sendData);
18  }
19
20  void loop() {
21   delay(100);
22   temp = GetTemp();
23  }
24
25  // callback for received data
26  void receiveData(int byteCount){
27
28   while(Wire.available()) {
29    number = Wire.read();
30
31    if (number == 1){
32     if (state == 0){
33      digitalWrite(13, HIGH); // set the LED on
34      state = 1;
35     } else{
36      digitalWrite(13, LOW); // set the LED off
37      state = 0;
38     }
39    }
40
41    if(number==2) {
42     number = (int)temp;
43    }
44   }
45  }
46
47  // callback for sending data
48  void sendData(){
49   Wire.write(number);
50  }
51
52  // Get the internal temperature of the arduino
53  double GetTemp(void)
54  {
55   unsigned int wADC;
56   double t;
57   ADMUX = (_BV(REFS1) | _BV(REFS0) | _BV(MUX3));
58   ADCSRA |= _BV(ADEN); // enable the ADC
59   delay(20); // wait for voltages to become stable.
60   ADCSRA |= _BV(ADSC); // Start the ADC
61   while (bit_is_set(ADCSRA,ADSC));
62   wADC = ADCW;
63   t = (wADC - 324.31 ) / 1.22;
64   return (t);
65  }
```

The Raspberry PI client

Here's a simple C application which will now talk to the Arduino over I2C:

```
 1  #include <string.h>
 2  #include <unistd.h>
 3  #include <errno.h>
 4  #include <stdio.h>
 5  #include <stdlib.h>
 6  #include <linux/i2c-dev.h>
 7  #include <sys/ioctl.h>
 8  #include <fcntl.h>
 9  #include <unistd.h>
10
```

```
11 // The PiWeather board i2c address
12 #define ADDRESS 0x04
13
14 // The I2C bus: This is for V2 pi's. For V1 Model B you need i2c-0
15 static const char *devName = "/dev/i2c-1";
16
17 int main(int argc, char** argv) {
18
19   if (argc == 1) {
20     printf("Supply one or more commands to send to the Arduino\n");
21     exit(1);
22   }
23
24   printf("I2C: Connecting\n");
25   int file;
26
27   if ((file = open(devName, O_RDWR)) < 0) {
28     fprintf(stderr, "I2C: Failed to access %d\n", devName);
29     exit(1);
30   }
31
32   printf("I2C: acquiring buss to 0x%x\n", ADDRESS);
33
34   if (ioctl(file, I2C_SLAVE, ADDRESS) < 0) {
35     fprintf(stderr, "I2C: Failed to acquire bus access/talk to slave
36     exit(1);
37   }
38
39   int arg;
40
41   for (arg = 1; arg < argc; arg++) {
42     int val;
43     unsigned char cmd[16];
44
45     if (0 == sscanf(argv[arg], "%d", &val)) {
46       fprintf(stderr, "Invalid parameter %d \"%s\"\n", arg, argv[arg
47       exit(1);
48     }
49
50     printf("Sending %d\n", val);
51
52     cmd[0] = val;
53     if (write(file, cmd, 1) == 1) {
54
55       // As we are not talking to direct hardware but a microcontrol
56       // need to wait a short while so that it can respond.
57       //
58       // 1ms seems to be enough but it depends on what workload it h
59       usleep(10000);
60
61       char buf[1];
62       if (read(file, buf, 1) == 1) {
63     int temp = (int) buf[0];
64
65     printf("Received %d\n", temp);
66       }
67     }
68
69     // Now wait else you could crash the arduino by sending requests
70     usleep(10000);
71   }
72
73   close(file);
74   return (EXIT_SUCCESS);
75 }
```

Save that as main.c and compile it:

```
1 pi@mimas ~ $ gcc main.c -o main
```

Now you'll notice there's a couple of usleep() waits in this code, once between sending the command and again after reading the response. I've found that this is necessary for two reasons.

1. The arduino is emulating an I2C device, so it won't respond immediately unlike a dedicated device so you need to wait a short while before reading it otherwise you won't get a response.
2. Without the second delay you can confuse the arduino by requesting another command too quickly, necessitating the arduino to be reset before it can be used again.

I found that 10000 (10ms) is enough here.

Wiring the two together

Now this is simple: First power down both the Arduino and the Raspberry PI – never connect things whilst they are powered up!

Next simply connect the two with 3 wires using the following table:

Raspberry PI		Arduino
GPIO 0 (SDA)	<–>	Pin 4 (SDA)
GPIO 1 (SCL)	<–>	Pin 5 (SCL)
Ground	<–>	Ground

Testing

Power up both the Arduino and Raspberry PI. Once it's up and running log in and run i2cdetect:

```
pi@mimas ~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  04 -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- UU -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

What you are now seeing is a list of all I2C devices connected. The one you are interested in is 04 which happens to be your arduino.

Lets toggle the led:

```
pi@mimas ~ $ ./main 1
I2C: Connecting
I2C: acquiring buss to 0x4
Sending 1
Received 1
```

You should now see the onboard led turn on. Run it again and the led goes out.

How about the temperature?

```
pi@mimas ~ $ ./main 2
I2C: Connecting
I2C: acquiring buss to 0x4
Sending 2
Received 35
```

35 just happens to be the Arduino's internal temperature in Celsius (in this example we've just returned the integer temperature).

That’s about it. The next thing I now need to do is to get those additional sensors working with the arduino and wrap them in an I2C slave.

Oh, one last thing, as I said earlier and in that diagram, you can have many devices on the one I2C bus, so to add another arduino all you need to do is to connect the three wires to that arduino as well and make certain it is using a different address.