A PharoThings Tutorial

Allex Oliveira

January 7, 2019

Layout and typography based on the `sbabook` LaTeX class by Damien Pollet.

Contents

Illustrations

CHAPTER 1

Lesson 8 - I2C Sensors (Temperature, Humidity, Pressure and Acellerometer)

In the previous lessons, we learned how to control LEDs and to use a button to interact with LEDs. Now let's start using some sensors to interact automatically with the real world, taking the temperature, air pressure and humidity This kind of sensor use the I2C protocol to communicate.

1.1 What do we need?

In this lesson, we will use a setup with 3 different I2C sensors.

Components

- 1 Raspberry Pi connected to your network (wired or wireless)
- 1 Breadboard
- 1 BME280 temperature, humidity and pressure sensor
- 1 MCP9808 temperature sensor
- 1 ADXL345 accelerometer sensor
- Jumper wires

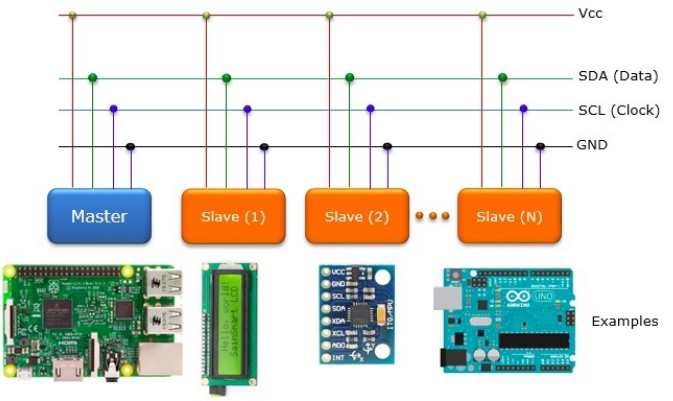

Figure 1-1 Devices connected using I2C bus.

1.2 Experimental theory

Before constructing any circuit, you must know the parameters of the components in the circuit, such as their operating voltage, operating circuit, etc.

The I2C protocol

The I2C communication protocol can be easily implemented in many electronic projects, being a very popular and widely used protocol. It is possible to perform communication between one or more master devices and several slave devices. It is an easy-to-implement protocol because it uses only 2 wires to communicate between up to 112 devices using 7-bit addressing and up to 1008 devices using 10-bit addressing.

The Figure 1-1 shows how you can connect the devices using the same I2C bus.

How I2C works?

How can we communicate with multiple devices using only two wires? For this to happen, each device has set an ID or a unique address. Then the master device can choose which device to communicate with.

The two wires are called Serial Clock (or SCL) and Serial Data (or SDA). The SCL wire is the clock signal that synchronizes the data transfer between devices on the I2C bus and is generated by the master device. The other wire is the SDA line that carries the date.

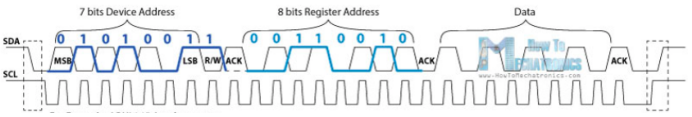

Figure 1-2 Bits sequence.

Protocol

The data is transferred in 8-bit sequences like you can see in Figure 1-2. After a special starting condition occurs, comes the first 8-bit sequence that indicates the address of the slave to which the data is being sent. For example, for the ADXL345 accelerometer device, the default address is 16r53 (0X53) or 0101 0011 (the last bit activated means the device is on reading mode).

After each 8-bit sequence follows a bit called Acknowledge. After the first Acknowledge bit, in most cases another addressing sequence comes, but this time to the internal registers of the slave device.

The internal registers are locations in the slave's memory containing various information or data. For example, the ADX345 accelerometer has a unique device address (16r53) and addition internal record addresses for the X, Y, and Z axes (16r32, 16r33, 16r34, etc.). Therefore, if we want to read the X-axis data, we first need to send the address of the device and then the internal register address specific to the X-axis.

After the addressing sequences, the data streams are as many as they are sent until the data is completely sent and ends with a special stop condition.

1.3 Experimental procedure

Now we will build the circuit. This circuit consists of three sensors (BME280, MCP9808, ADXL345) and a power supply (the Rasp).

- Connect the Ground PIN from Raspberry in the breadboard blue rail (-). In this experiment we will use the PIN6 (Ground);
- Then connect the 3.3V (PIN1) pin in the red rail (+).
- Now let's connect the SCL (PIN5) and SDA (PIN3) wires. Connect them like as shown in the Figure 1-3 ;
- Now push the sensors in the breadboard;
- And insert the jumper wires connecting the sensor in the bus, like as shown in the Figure 1-3;
- The last step is to connect the power 3.3V (+) and ground (-) wires in each sensor.

The Figure 1-3 shows how the electric connection is made.

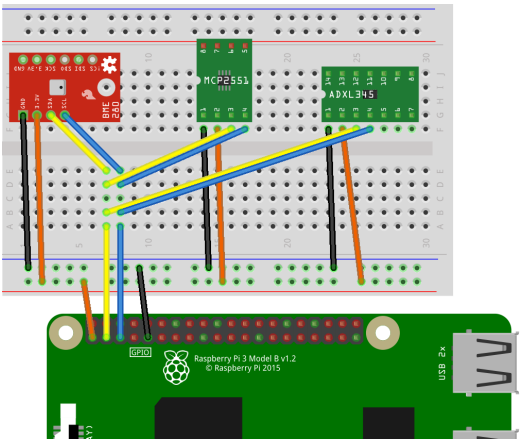

Figure 1-3 Physical sensors connection.

1.4 Configuring the Raspberry Pi I2C

We need to configure the Raspberry Pi to use the I2C protocol. To do this, access the Raspberry using SSH and go to file `/boot/config.txt` and add the line `dtparam=i2c1=on`.

You can run the follow command to do this:

```
sudo echo dtparam=i2c1=on >> /boot/config.txt
```

Add the 'pi' user to I2C group and restart the Raspberry

```
sudo adduser pi i2c
reboot
```

Now your Raspberry is configured to communicate with the sensor using the I2C protocol.

1.5 Connecting remotely

Through your local Pharo image, let's connect in the Pharo image by running on Raspberry, enable the auto-refresh feature of the inspector, and open the inspector. Run this code in your local playground:

```
remotePharo := TlpRemoteIDE connectTo: (TCPAddress ip: #[193 51 236
    212] port: 40423)
GTInspector enableStepRefresh.
remoteBoard := remotePharo evaluate: [ RpiBoard3B current].
remoteBoard inspect.
```

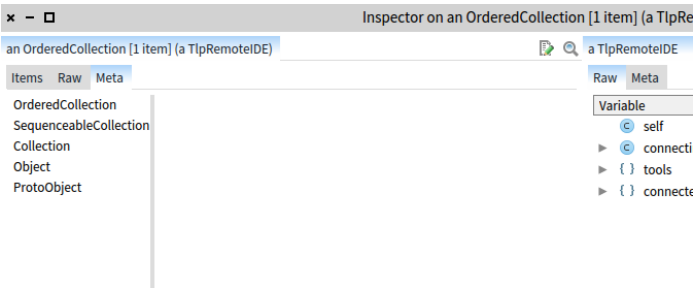

Figure 1-4 Inspecting remote object.

1.6 Exploring the properties of a remote object with the remote inspector

In your inspect window (Inspector on a PotRemoteBoard), let's create the instances of the first sensor.

```
a := board installDevice: PotBME280Device new.
```

With this proceeding, we are creating an object and we can inspect it to see the properties and values. To see the details of this object, like for example what you can do and what you can ask to it (methods), press cmd + I to inspect it and you will see the inspect window.

In the `Meta` tab, you can see the Class and Subclass of this object. To see what you can do with this object (methods), you can start selecting the top Class and will be shown in the right window the methods that you can to use.

Let's see what the method `readParameters` can do. When you select it, you can see the code of method on the bottom of the window. When you call this method, it returns an array with 3 values: temperature, pressure, and humidity. The carrot symbol (little hat ^) is to return something when the method is called.

So let's use the method `readParameters` to ask to the object what are the values of temperature, pressure, and humidity. To get the return of this object, select the following code and press cmd + P. You will see a big number. This number is an array with 3 indexes like the Figure 1-4.

```
a readParameters.
```

To see the details of this object, you can press Cmd + I to inspect it and you will see the 3 indexes separately.

1.7 Getting the temperature with BME280

You can also ask to only one value, selecting the array index using the method `at:`. In this case, we are selecting only the index 1 of an array and will return only the temperature:

```
a readParameters at: 1.
```

If you want to format the number to be more legible, you can use:

```
(a readParameters at: 1) printShowingDecimalPlaces: 2.
```

1.8 Getting the humidity and pressure with BME280

Now you know how to ask to the object the specific value that you want. To get the humidity and pressure is very simple, just choose the index of each one and format them as you want. To get the pressure:

```
(a readParameters at: 2) printShowingDecimalPlaces: 2.
```

and to get the humidity:

```
(a readParameters at: 3) printShowingDecimalPlaces: 2.
```

This is very simple and you can get these values and stored in a variable, to use to different proposes, like send a message to an LCD display, send the values to a cloud server and simply do some action, like for example turn on a LED or turn on an Air Conditional (using relays). In the following example, after you put the temperature value in a temperature variable, you can select the variable and press Cmd + P to see just the temperature.

```
temperature := (a readParameters at: 1) printShowingDecimalPlaces: 2.
temperature.
```

1.9 Getting the temperature with MCP9808

How we see before, is very easy to read the values of the sensors. To read the values of the MCP9808 sensor, just create an object using the MCP9808 sensor with the follow code and read the temperature with the method `readTemperature`:

```
b := board installDevice: PotMCP9808Device new.
b readTemperature.
```

You can format the answer as you want also:

```
(b readTemperature) printShowingDecimalPlaces: 2.
```

1.10 Getting the axis X, Y, and Z with ADXL345

The process is the same before. Let's create an object with the sensor and ask to it what is the value of the X, Y, Z axis:

```
c := board installDevice: PotADXL345Device new.
c readCoordinates.
```

As like the BME280 sensor, the ADXL345 is returning an array with 3 indexes, the 3 axes. To ask to a specific axis, you can select the position that you want. In the following case we are getting the X-axis:

```
c readCoordinates at: 1.
```

1.11 Conclusion

In this tutorial, you learned how to inspect remote objects to understand what this object can do. You learned also how to get the value of temperature, humidity and pressure to store in a variable, as like the X, Y, Z axis.

In the next lesson, let's see a different kind of sensor, the ultrasonic sensor. It uses only 2 wires to work, but don't use the I2C protocol. See you there.