Theremino **System**

Theremino IotHAL
Read the I2C sensors

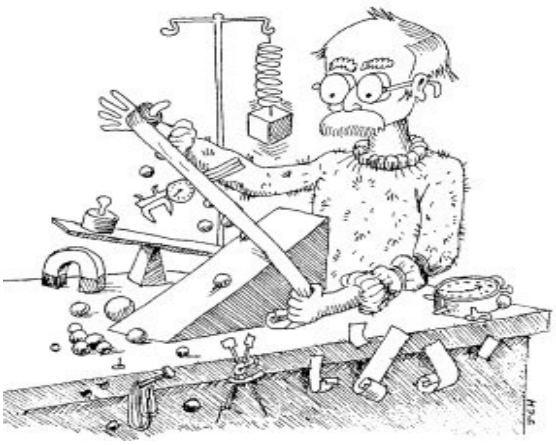
Reading I2C sensors

To use the ESP32 modules with our system, it is not necessary to be a programmer.

All the simple sensors, like buttons, capacitive buttons, servo-motors, leds,
and analogic sensors of any type, can be connected directly to the ESP32,
without any special firmware

All the InOut types are already prepared, and you just choose them in the IotHal application.

What we will explain below only serves
to connect digital sensors and to perform special processing.



In this document we will explain how to read sensors connected with the I2C protocol, and how to make special processings in the module itself.

These examples illustrate the basic techniques for communicating with the special sensors, and for sending their data to the theremino system. These techniques can also serve as a trace to connect other types of sensors, such as SPI sensors.

- - - - - -

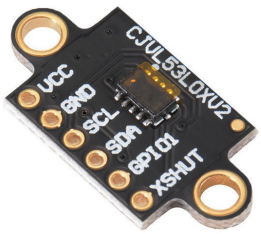
In ***this page*** you download the IotHAL application, and the "IotModule_I2C_Sensors.zip" filee which contains the complete Arduino projects, for all the sensors described in this document.

To program the ESP32, remember to set the name and the password for your network, and to prepare the Arduino IDE, as explained in the IotHAL application documentation.

- - - - - -

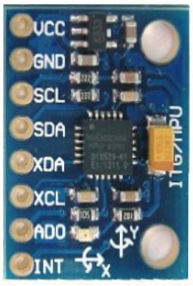
In all the examples, the data are sent from the ESP32 module, to the ***IotHAL application*** via "Generic" Pin types, and then sent to the Slots, through which any other application of Theremino system can use them. For example, the ***ECG application*** for the detection of arrhythmia.

Examples summary

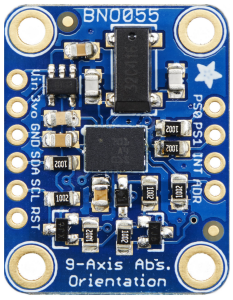


EXAMPLE 1 - A laser sensor that measures the distance up to two meters, with a resolution of one millimeter and accuracy of better than five millimeters.

This example is very simple, the firmware we have prepared for it is simple to understand and to use, without any trimmings.

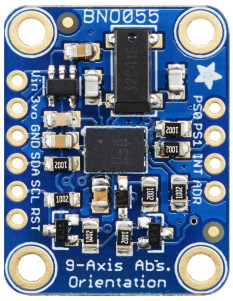


EXAMPLE 2 - A three-axis accelerometer (x, y, z) that can be used for earthquake detection. This sensor is inexpensive (2 or 3 euros) and easy to use, but quite noisy, so you can only detect near and fairly strong earthquakes.



EXAMPLE 3 - A three-axis accelerometer (x, y, z), which can be used for earthquake detection. In this example we use a BNO055 precision sensor (accelerometer, gyroscope and compass) and program it to be an accelerometer only. This sensor costs about ten times, but is less noisy than the previous one.

With this sensor even medium intensity earthquakes can be detected.



EXAMPLE 4 - The BNO055 contains an accelerometer, a gyroscope and a compass, each with three axes (x, y, z). The firmware joins the nine axes and produces an absolute orientation on three axes (which is also called "inertial platform").

This sensor is quite complex, you will have to read a lot to learn how to calibrate it and use it. The instructions are in the first rows of the firmware itself (the .ino file).

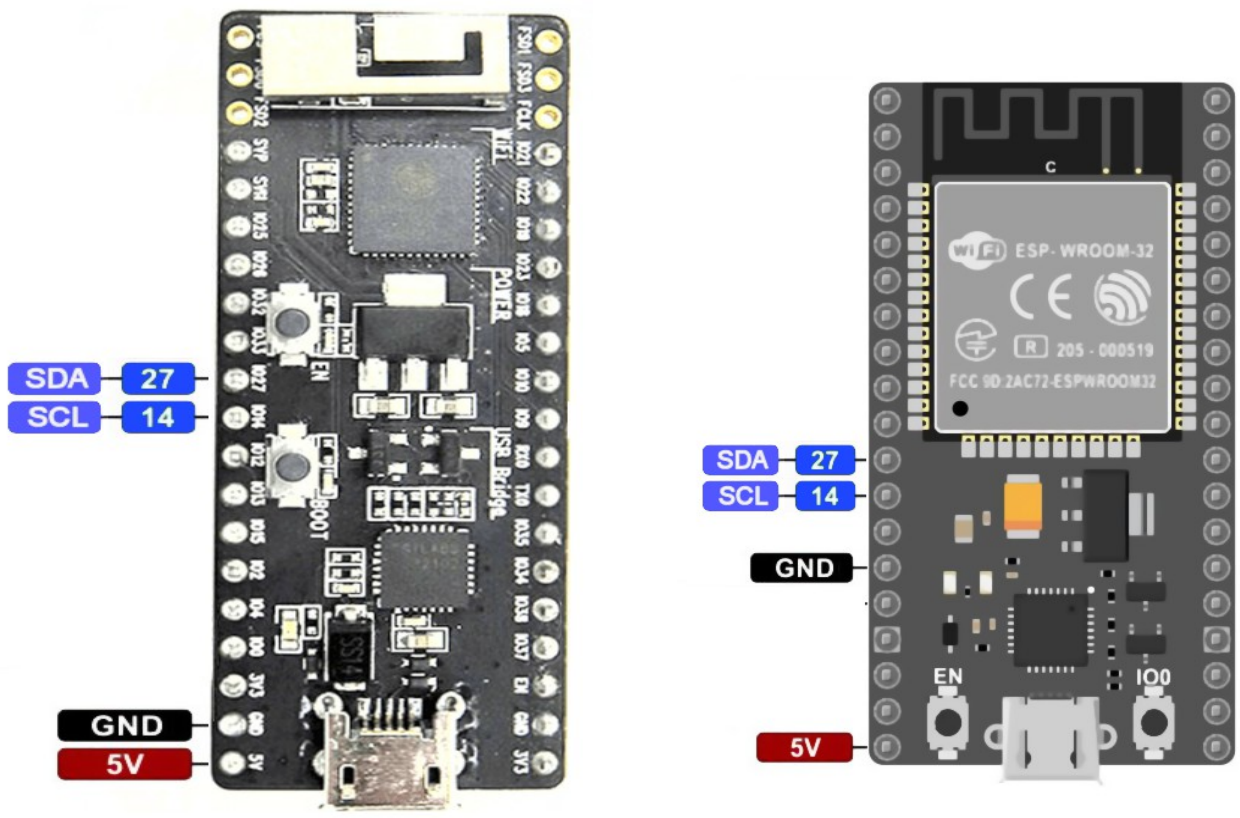
EXAMPLE 5 - A sensor for measuring the heart rate.

This example also demonstrates the firmware techniques, so it could be used to learn how to write similar projects.

Connections to the ESP32 modules

To connect all the sensors of the examples, we use always the same four pins.

Here is their position on the most commonly used cards.



The complete images, with the names and the annotations for all the Pin,
are in the IotHAL application documentation, you download from ***this page***

- - - - -

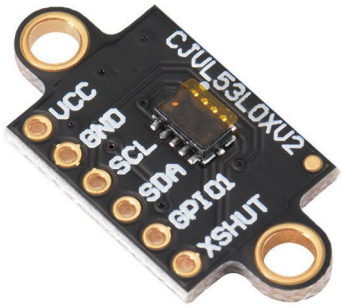
Once the data has been read from the sensors, to send them from the firmware to the IotHAL application, we use Pins 36, 39 and 25 (or 34 that is more convenient an the Wroom module). These "pins" do not correspond to the physical module pins, but are only identifiers that we use for the GenericWrite and GenericRead functions, to communicate between the firmware and the IotHAL application.

Instead of 36, 39, 25 (or 34), we could choose any other unused Pin, or we could program new "Generic" Pins, from 80 onwards, as explained in the last pages of the IotHAL application documentation.

Example 1 - A sensor that measures the distance

This sensor measures the distance by means of a laser beam ***(Note 1)***.

This sensor can accurately recognize the position of a hand and has a fast response. So you can also replace the CapSensors, in the Theremin musical instruments.



Features:

- Measurement distance up to one meter and more.
- Measurement using the "Flight Time" of the Laser light.
- Resolution of a millimeter.
- Accuracy of about 5 mm
- Consumption 20 mA

(Note 1) *Lasers are dangerous because their beam is concentrated. This sensor while using laser light has an opening of 35 degrees, thus very similar to that of a normal LED. And it also has an output power much like that of an LED. To which the emitted light is the same that would be emitted by an infrared LED as those of the television remotes. The only difference is that it is coherent light, ie with a single frequency (or almost). But consistency does not generate danger, so we can consider this device no more dangerous than an infrared LED.*

Avoid, however, to bring an eye within a few centimeters from the exit point, not because it is a Laser but because it is infrared light and therefore not visible. If you point it at a close eye for a long time, even an infrared LED can be harmful. The sun is infinitely more dangerous, but its light is visible and hardly it would look fixed for a long time.

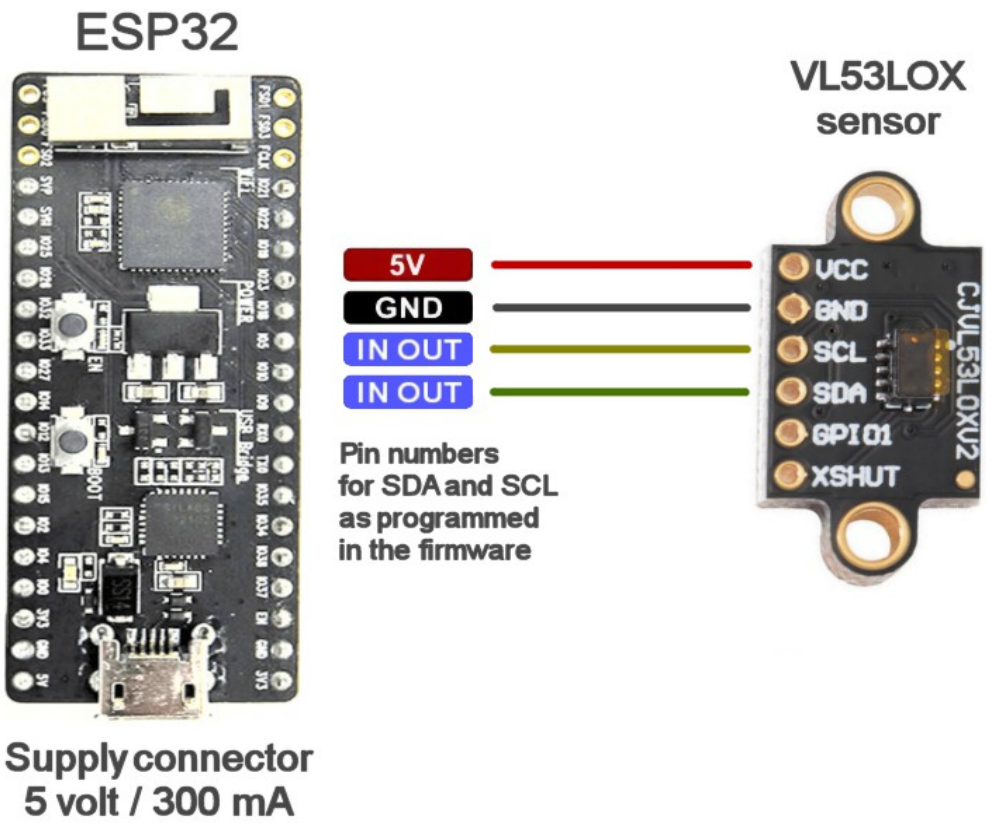
In the datasheet the Laser is defined as "Class 1". It is also specified that it is designed to remain in Class 1, under all conditions, including failures.

According to Wikipedia: "A Class 1 laser is safe under all conditions of normal use. This means that the maximum permissible exposure (MPE) can not be exceeded when one looks to the naked eye or with the aid of typical magnification optics (eg. Telescope or microscope)."

Example 1 - Connections

The wires to be connected to the ESP32 are: VCC (5V), GND, SCL and SDA.

So we take four wires (small and flexible) and connect them very carefully. Most of all you must be careful not to reverse VCC and GND.



For all the examples we have chosen the Pin:

- 27 for the SDA signal
- 14 for the SCL signal

To find the Pin positions on the ESP32 modules,
see the images at the beginning of this document (page 4).

Example 1 - Reading data with the IotHAL application

To read the VL53LOX sensor:

- Prepare the Arduino IDE, as explained in the IotHAL application documentation.
- Program the ESP32 with the sketch "IotModule_DistanceMeter.ino".
- Launch the application "IotHal"

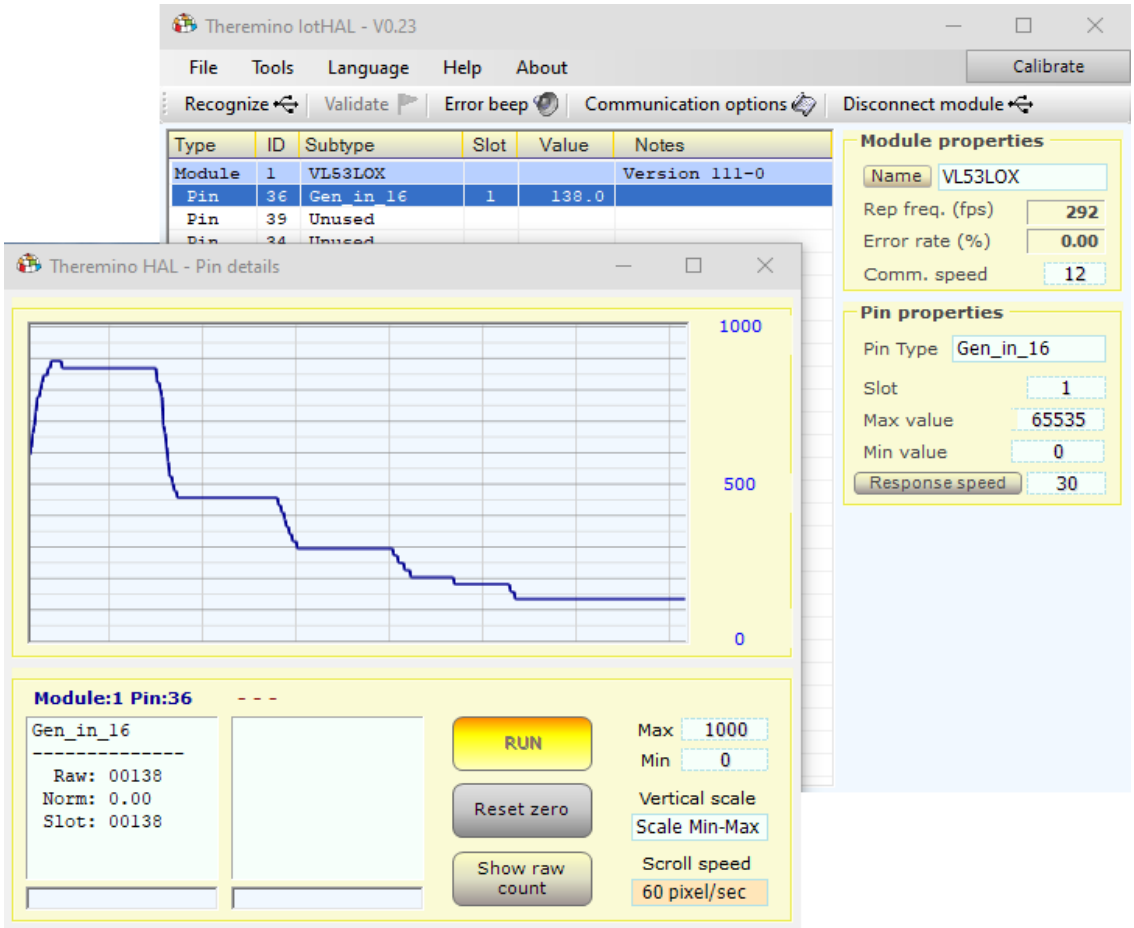
As in the "genericWrite" line we set the Pin "36", also in IotHAL application must configure this pin as a genericInput, so: Pin 36 PinType = Gen_in_16

Also remember to set the "speed response" to 30 on this Pin, in order to filter out the noise and obtain a more stable measurement.

With this sensor you should obtain a value in millimeters, and not from 0 to 1000, that are set by default when configuring the Pin. So we will set Max value = 65535 and we get the number in millimeters.

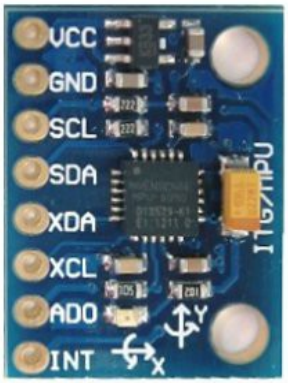
The 65536 comes from the fact that we use a Gen_in_16 input, ie 16 bits, which are 65536. Therefore setting a Min = 0 and Max = 65535 we obtain exactly the raw value that is sent to us by the sensor.

And in this case the raw value of the sensor are just millimeters from 0 to 2000.



Example 2 - A triaxial accelerometer

In this example we use a three-axis accelerometer (x, y, z) that can be used for earthquake detection.



Features:

- This sensor is cheap (2 or 3 euros)
- It's a sensor easy to program, the firmware to read the data is very simple, about ten lines in all.
- Measurement on three axes with full scale of +/- 2G
- The measurement resolution is 16 bit but the data contains a lot of noise, so only near and quite strong earthquakes can be detected.

- - - - - -

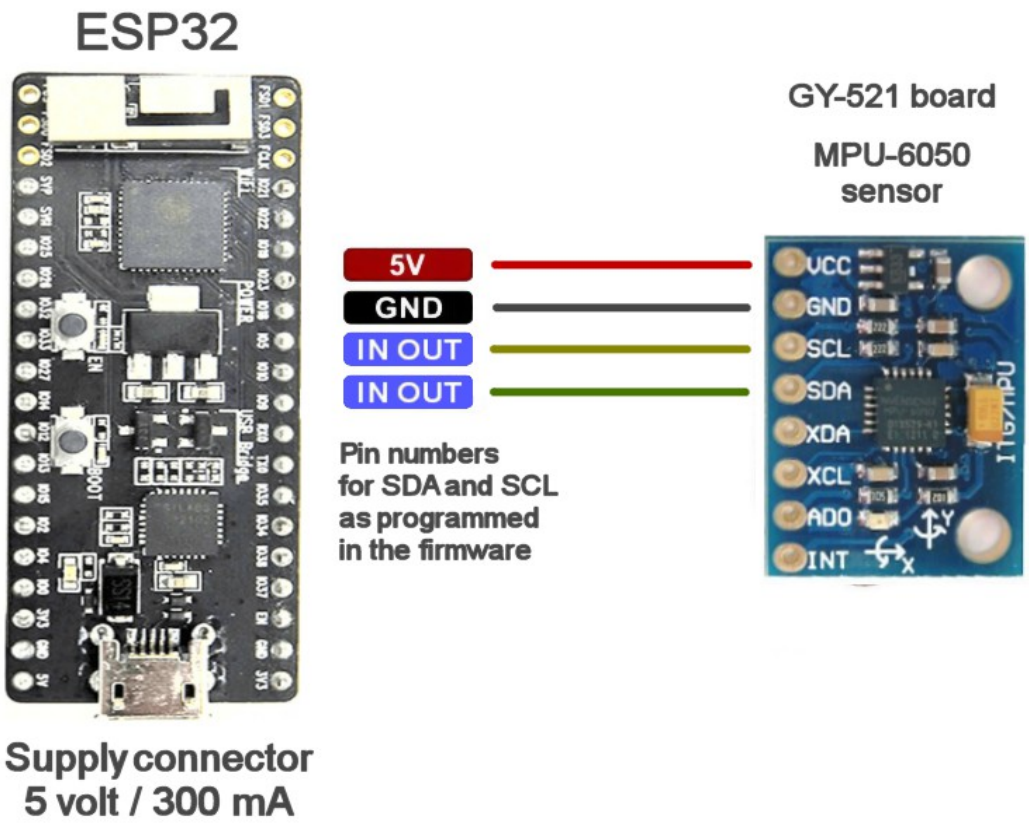
For the earthquakes it is recommended to use the "Example 3" sensor,

which is considerably less noisy.

Example 2 - Connections

The wires to be connected to the ESP32 are: VCC (5V), GND, SCL and SDA.

So we take four wires (small and flexible) and connect them very carefully. Most of all you must be careful not to reverse VCC and GND.



For all the examples we have chosen the Pin:

- 27 for the SDA signal
- 14 for the SCL signal

To find the Pin positions on the ESP32 modules,
see the images at the beginning of this document (page 4).

Example 2 - Reading data with the IotHAL application

To read the MPU-6050 sensor:

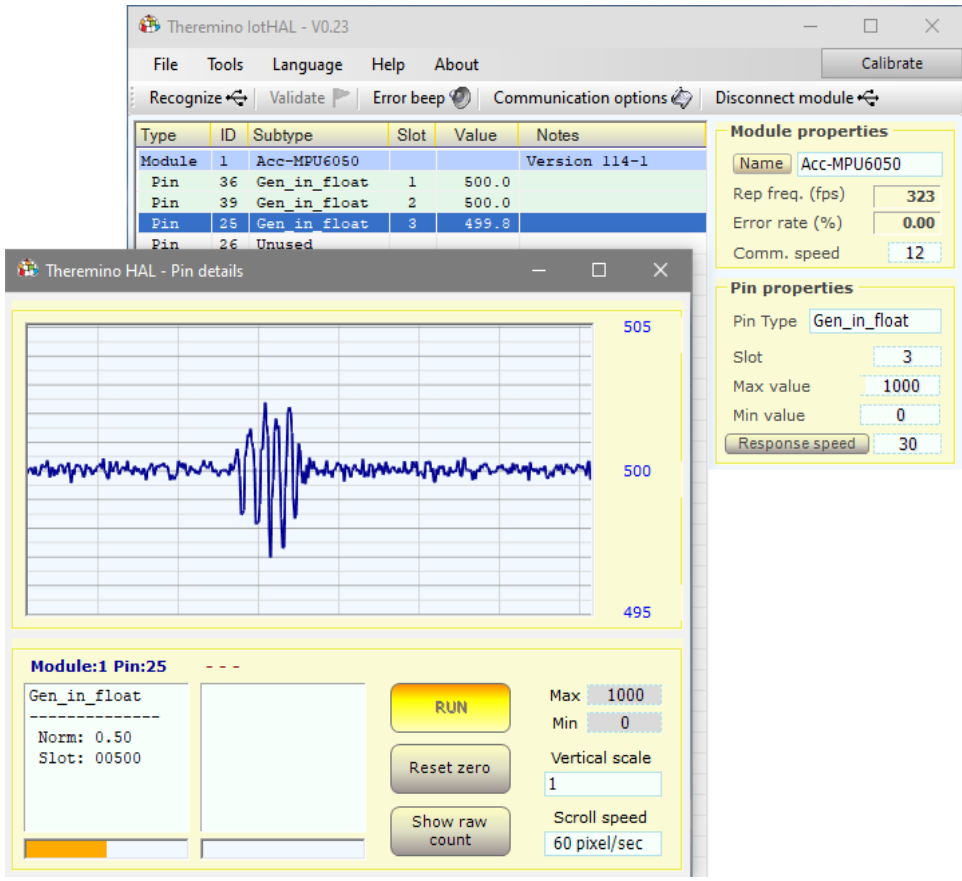
- Prepare the Arduino IDE, as explained in the IotHAL application documentation.
- Program the ESP32 with the sketch "IotModule_AccMPU6050.ino".
- Launch the application "IotHal"

The first three pins are configured as: "Gen_in_float" (using the float numbers, the values are not limited in the range 0 to 1000, and could be less than zero, or more than 1000 with strong motions)

The firmware sends values from 0 to 1000 (centered to 500), but you could change Min e Max in the HAL to values from -1 to +1.

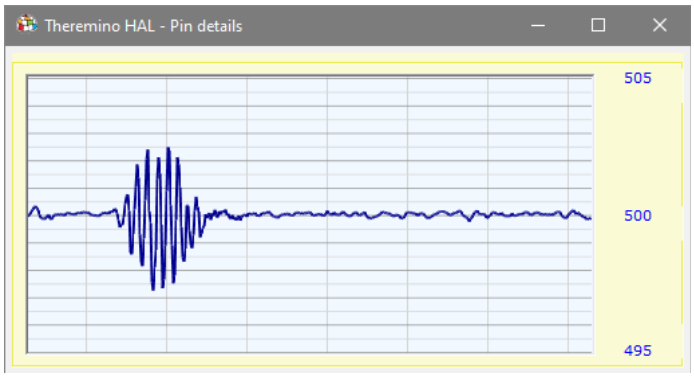
Setting Min=-1 and Max =1 you obtain directly the G values.

```
+2.0 = +2G
+1.0 = +1G
 0.0 =  0G
-1.0 = -1G
-2.0 = -2G
```



The first image is an example of how you could see with a fairly strong and near earthquake.

In the second image you see an oscillation of similar intensity to the previous one, but measured with the sensor of the next page.



As you can see the sensitivity is higher and the noise is less.

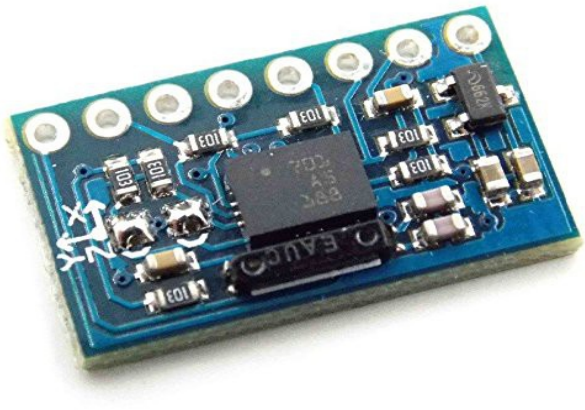
The firmware transforms the data of this sensor from "accelerometer" to "velocimeter", in order to make it similar to a classic 4.5 Hz geophone.
See details on the last page of this document.

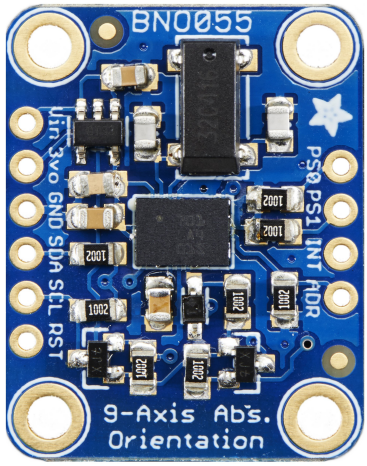
Example 3 - Triaxial accelerometer (better)

In this example we use the BNO055, the same module shown with more details in the following example.

To use this module as an accelerometer we have programmed it in a special way, wasting part of its features, but obtaining a significantly better accelerometer than the previous example.

With this sensor, weaker and more distant earthquakes can be detected. The BNO055 costs a little more than the MPU6050 (from ten to thirty Euros instead of two or three Euros) but the improvement that is obtained is worth the additional expense and also the waiting, in case it were made to arrive from China.





To use the BNO055 sensor as accelerometer, simply program it with the "IotModule_AccBNO055" firmware. The only difference is that the received data are not rotations in space, but are the accelerations on the three axes X/Y/Z (with full scale +/- 2G)

- - - - -

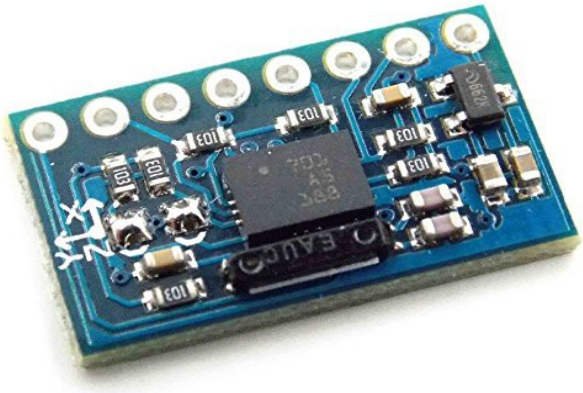
To connect the sensor see the instructions of the following example (Example 4)

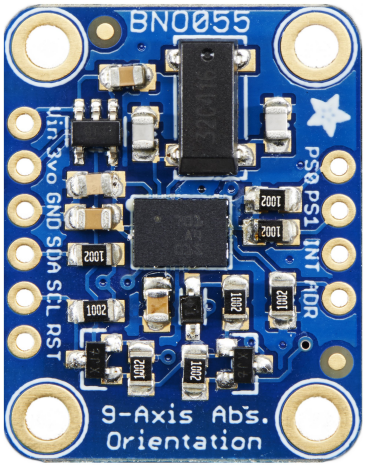
To receive the data with the IotHAL see the instructions of the preceding example (Example 2)

The firmware transforms the data of this sensor from "accelerometer" to "velocimeter", in order to make it similar to a classic 4.5 Hz geophone.
See details on the last page of this document.

Example 4 - Accelerometer, Gyroscope and Compass

In this example we use the BNO055, an intelligent sensor that merges the nine axes directly into the chip. With other sensors, transforming the data coming from the accelerometer, gyroscope and magnetometer, into a real "3D spatial orientation", would require to write complex algorithms, difficult to test and to adjust.





There are various models of this module. The one on the left is on eBay, shipped from China, for less than ten euros, including shipping. The one on the right, produced and marketed by SparkFun, costs more than 30 euros.

Features:

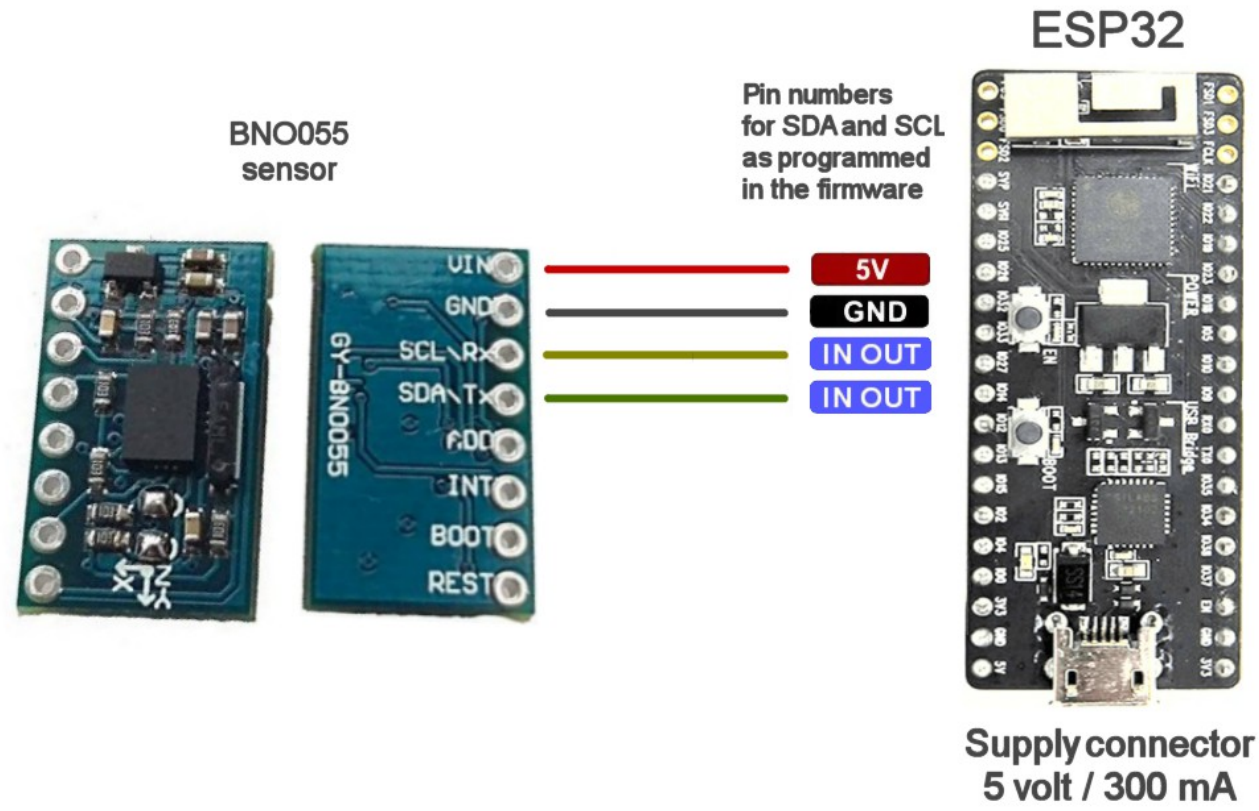
- Supply voltage (BNO055 chip) : 2.4 to 3.6 volt
- Supply voltage (complete module) : 5 volt
- Total supply current : Less than 15 mA
- Resolution : Approximately 12 bit
- Sampling frequency : 100 Hz

Example 4 - Connections (chinese module)

The wires to be connected to the ESP32 are: VCC (5V), GND, SCL, SDA and 3.3V.

So we take four wires (small and flexible), and connect them very carefully.

Most of all you must be careful not to reverse VCC and GND.

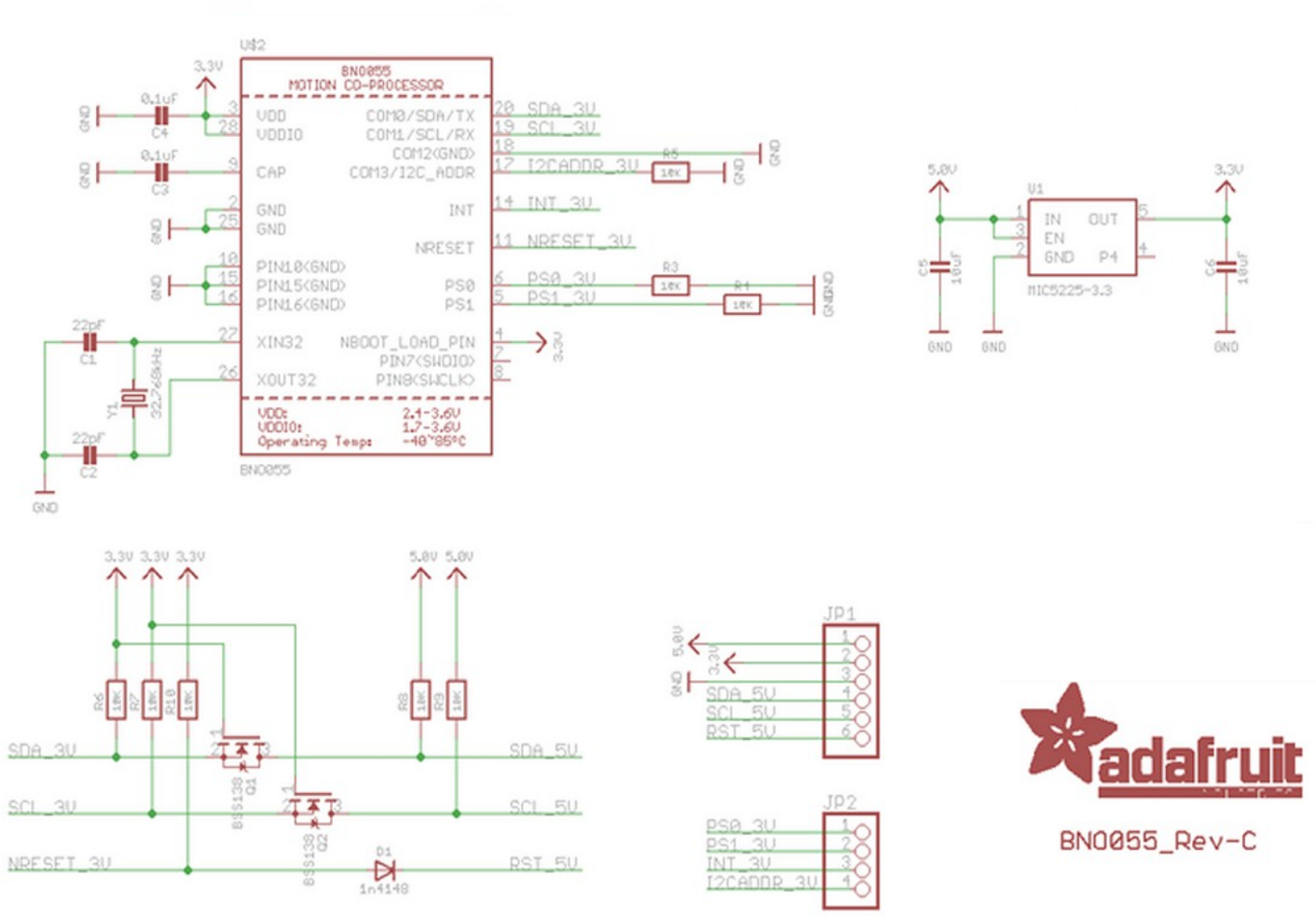


For all the examples we have chosen the Pin:

- 27 for the SDA signal
- 14 for the SCL signal

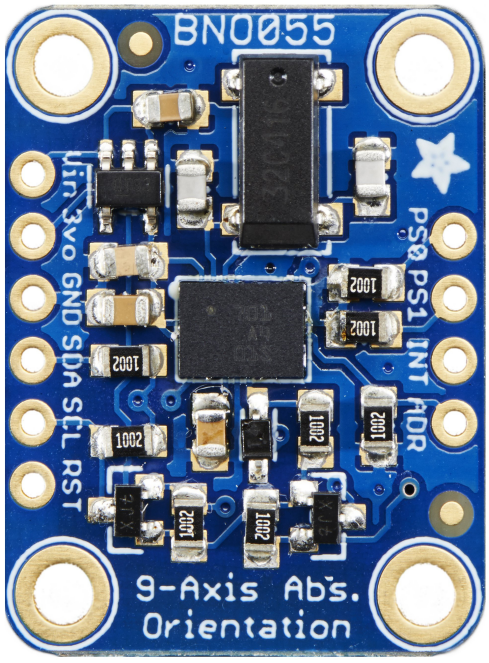
To find the Pin positions on the ESP32 modules,
see the images at the beginning of this document (page 4).

Example 4 - Connections (Adafruit module)



This module, produced by Adafruit, is more expensive.

This module contains a regulator that produces the 3.3 volt, and the level shifters for the SDA and SCL signals.



So we will have to connect:

- GND with the GND of the ESP32
- Vin with the 5 volt of the ESP32
- SDA with the Pin 27
- SCL with the Pin 14

Example 4 - Reading data with the IotHAL application

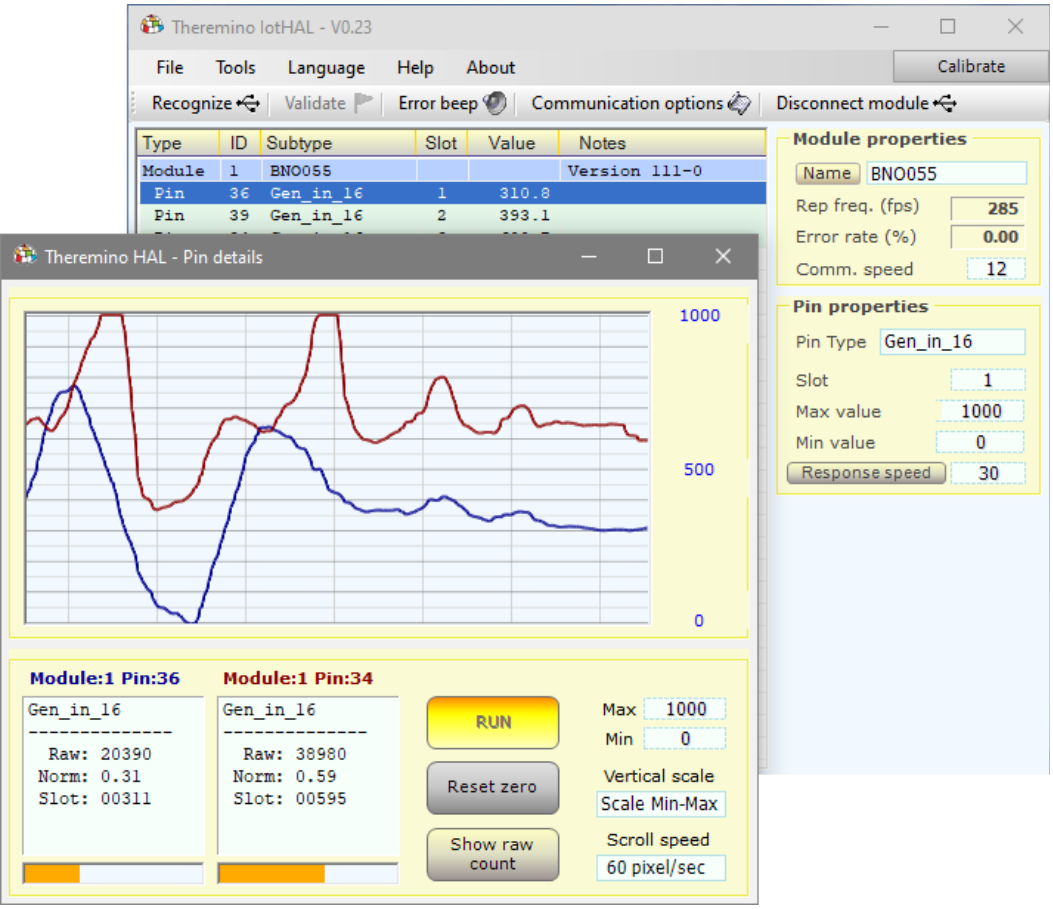
To read the "absolute orientation" of the BNO055 sensor :

- Prepare the Arduino IDE, as explained in the IotHAL application documentation.
- Program the ESP32 with the sketch "IotModule_AbsoluteOrientation.ino".
- Launch the application "IotHal"

As in the "genericWrite" lines of the firmware, we set the Pins 36/39/25, also in IotHAL application must configure this pin as a Generic Input, so:

- Pin 36 PinType = Gen_in_16
- Pin 39 PinType = Gen_in_16
- Pin 25 PinType = Gen_in_16

If you use the WROOM module, to get three consecutive pins, you could use Pin 34 instead of 25. In this case you should change the number from 25 to 34, both in the HAL and in the firmware and then reprogram the module.



Also remember to set the "speed response" to 30 on these Pin, in order to filter out the noise and obtain more stable values.

With this sensor you should obtain values from 0 to 1000, that are set by default from the HAL application.

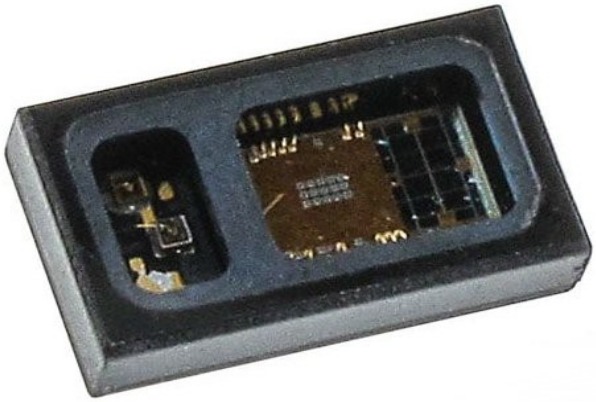
To get measurements in degree, we should change the Min and Max values to -90 and +90

Example 5 - A sensor for the heart rate

This example, in addition to providing the firmware ready, as the previous ones, also explains the details of the functions we have written.

You can use either the MAX30102 and MAX30105 models that are almost identical to each other. Avoid previous MAX30100 which has lower performance, and that would not work because its internal registers are different.

These chips have the links below. So it is very difficult to weld, and usually buys them already soldered on small plates called "breakout board".



Here's an example of a "breakout board".

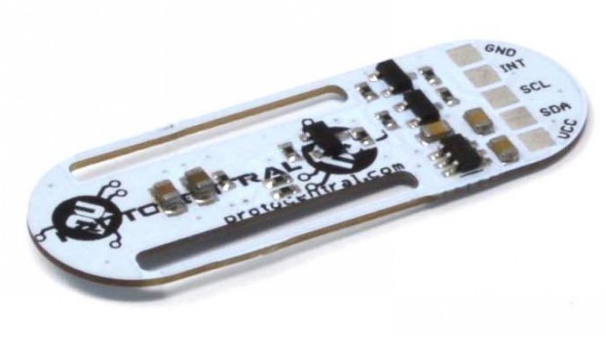
The MAX30102 communicates through an I2C interface so it would not be possible to read it with a Master module.

So this is a good example where you should use an Arduino or an ESP32 instead of a Master. But careful not to take taste, in most cases using an Arduino is more uncomfortable and performances are minor.

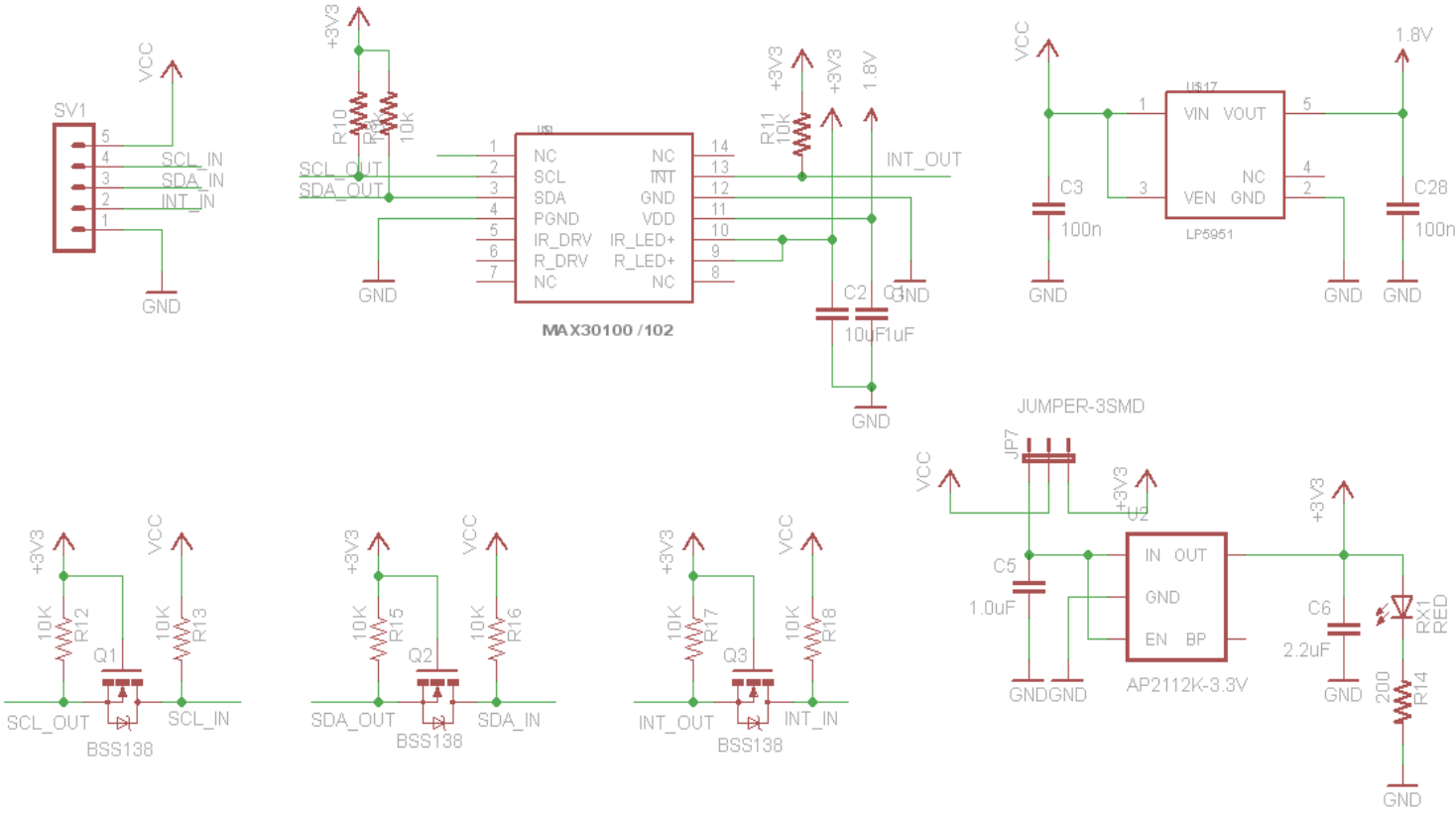
In this particular case a 50 Hz bandwidth is more than sufficient, whereby the Arduino communication slowness does not create problems.

If you use an ESP32 you get good communication speed and also the WiFi connection between the module and the PC that hosts the HAL application and the other theremino system applications.

Example 5 - Sensor models - MAX30102 Protocentral



This model produced by Protocentral, an Indian firm in Bangalore, is among the best on the market. It has a well-studied form and two comfortable cuts where to pass an elastic fabric to hold the finger. Its features are ***well specified*** and there is also the wiring diagram. His only fault is to cost 25 Euro.



This scheme can be used as a reference. Other simpler schemes do not have the Jumper selection and some do not even three MOSFET. The MOSFET translate the level of I2C signals between the voltage of the MAX3010x chip (1.8 volts) and the processor voltage to which it is connected (3.3 volts or 5 volts).

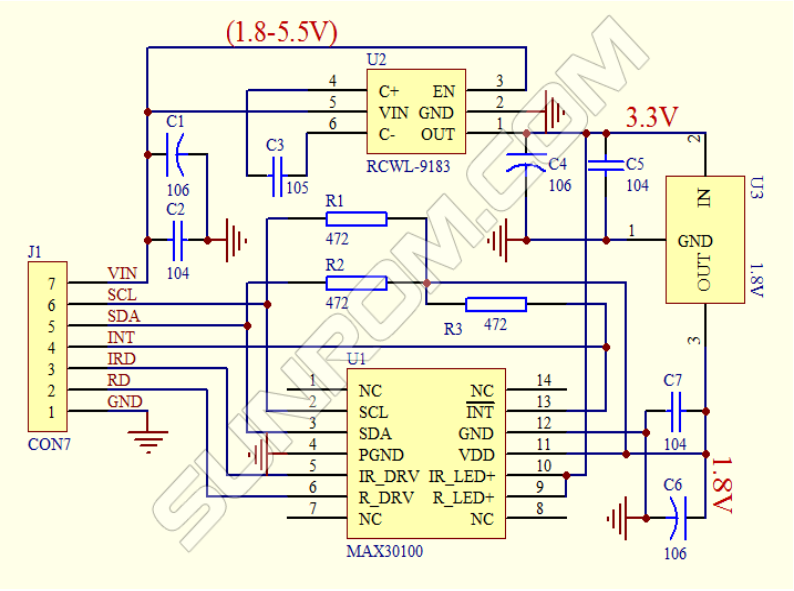
Probably it is possible to communicate even without these transistors, but we did not try all kinds of forms, so we can not secure it. It is important, however, and we recommend to control it, that the I2C signals pullup resistors are connected to the processor voltage (3.3 volts or 5 volts).

Example 5 - Sensor models - MAX30102 Chinese green

This model is to be found on eBay for a few euros. Usually only by the Chinese, so it takes a month to get it.

Apparently the pullup resistors are connected to the 1.8 volts, but we have not tried it, so we can not ensure that it works.

Those who produce the "Pulse - Protocentral" on the previous page, say it can not work. But maybe they have not even tried them and they say to sell them.



Here's his schematics. The marked chip is 30100 instead of 30102, but the printed circuit board and the components are the same for the two chips.

The pullup resistors R1, R2 and R3 are actually connected to the 1.8 volt voltage for which it could not work.

Probably you could cut the track that goes to the 1.8 volt voltage and connect it to VIN.

There is also the possibility that this is one old scheme and that the modules for sale on eBay have been corrected.

Or the I2C can work well too. We should try one.

Sensor models - MAX30102 Chinese magenta

This model is also on eBay. It costs few Euros, but it is only by the Chinese, so it takes a month to get it.

You can not find the wiring diagram to understand if they are connected by level shifters, but we tried and the number "21" that identifies the chip as MAX30102 comes properly.

So we are confident that the I2C communication works.

Unfortunately we have purchased a single copy, and the LEDs do not light. Now we have ordered three more and when they come we will post further news.

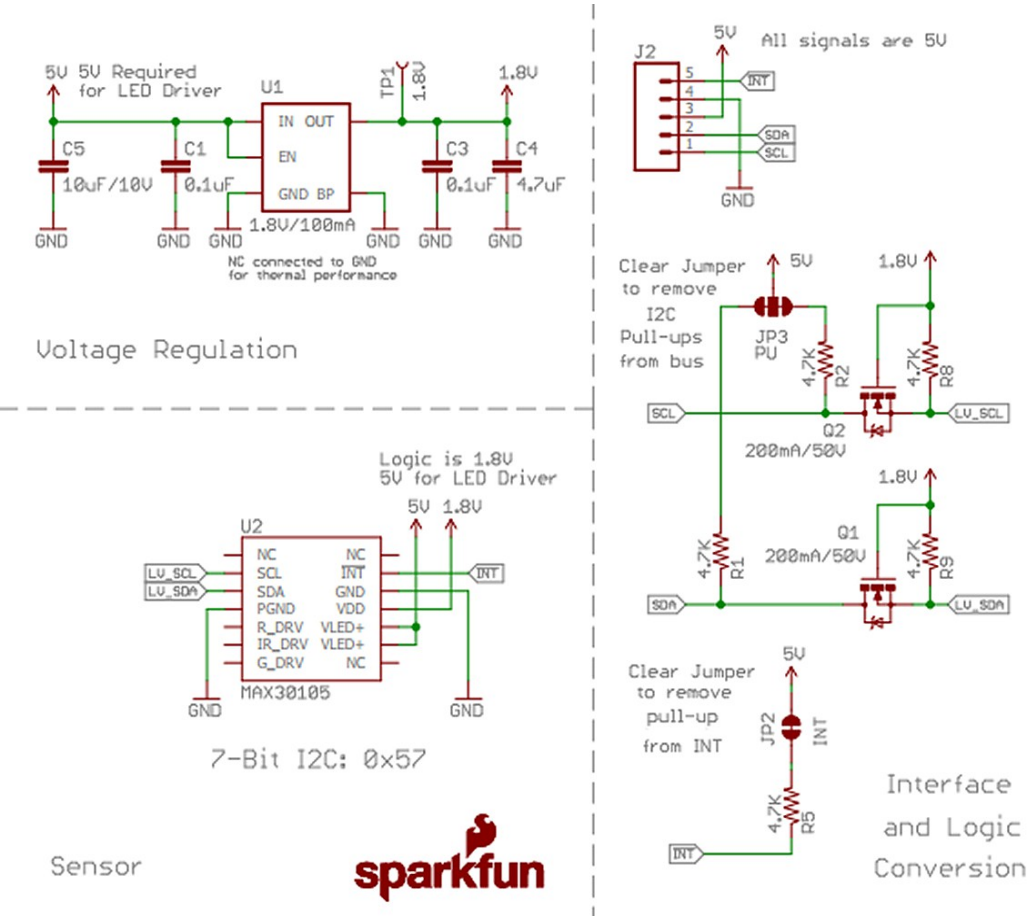
Example 5 - Sensor models - MAX30105 Sparkfun

This MAX30105 is better than the previous, it costs more than Chinese ones (about 19 euro shipping included), but has three LEDs (IR, RED and GREEN) instead of two (RED and IR).

To which can be done by cardio-frequency and pulse-oximeter as the MAX30102, but also as a sensor for dust and fumes.

In addition the informations from Sparkfun are complete and accurate, and the PCB is well studied.

The power supply voltage of 5 volts is well specified on the PCB text. While in the previous page models it is not specified, and this could easily cause doubts and errors. The other models only write Vin, to be able to use both the MAX30100 MAX30102 ranging respectively 3.3 and 5 volts.



The scheme leaves no doubt, the level converters are present and the pullup resistors are properly connected to 5 volts.

There are also jumper to eliminate pullups in case the resistors were already present on the outside. In our case the pullup serve and then you will not change anything.

Another advantage is that there are distributors all over the world and then you can have in two days. For example, the Italians may find it from ***Robot Italy***.

Example 5 - What model to buy ?

For those on a limited budget we recommend using this sensor:

- It's cheap, but it is recommended to buy two, because not all work.
- The latest versions offer the MAX30102 which is better than 30100. The PCB is the same but the builders make a sign with black pen on the last digit. However, beware that in the sales page is well specified 102.

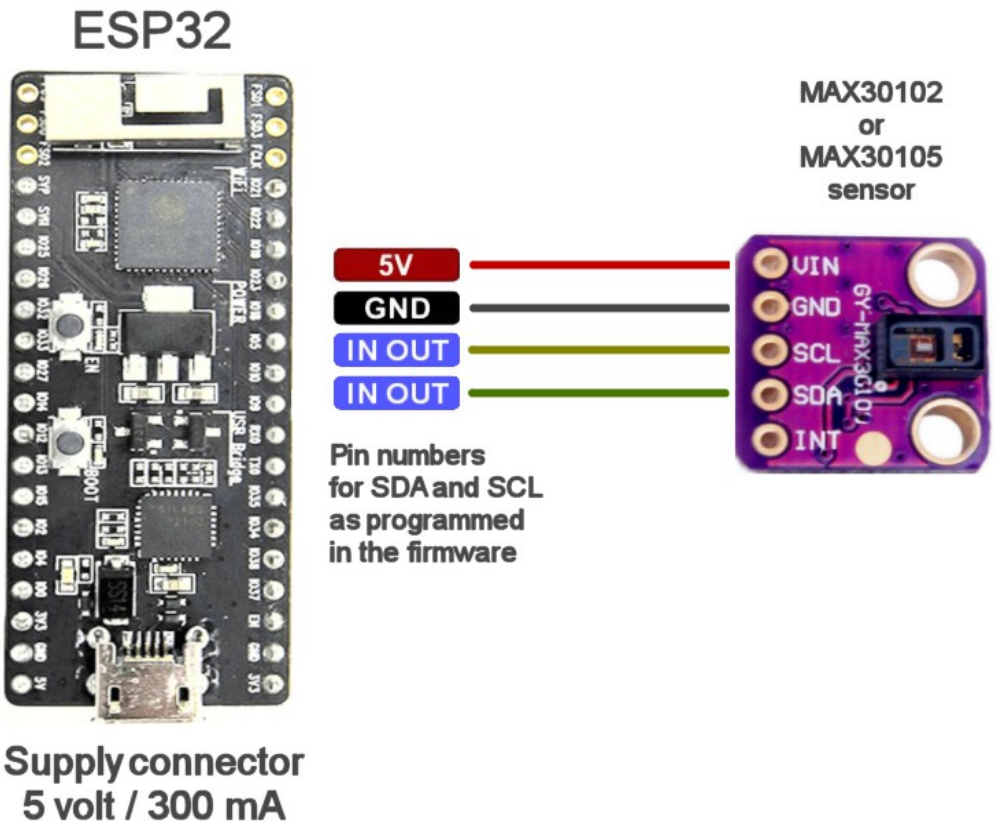
Spending a little more, you buy the Sparkfun:

- The build quality is better and has good documentation.
- Mount the MAX30105 for which can also measure the dust and fumes.
- It comes in two or three days.

Example 5 - Connect the sensor to the ESP32

The wires to be connected to the ESP32 are: VCC (5V), GND, SCL and SDA.

So we take four wires (small and flexible) and connect them very carefully. Most of all you must be careful not to reverse VCC and GND.



For all the examples we have chosen the Pin:

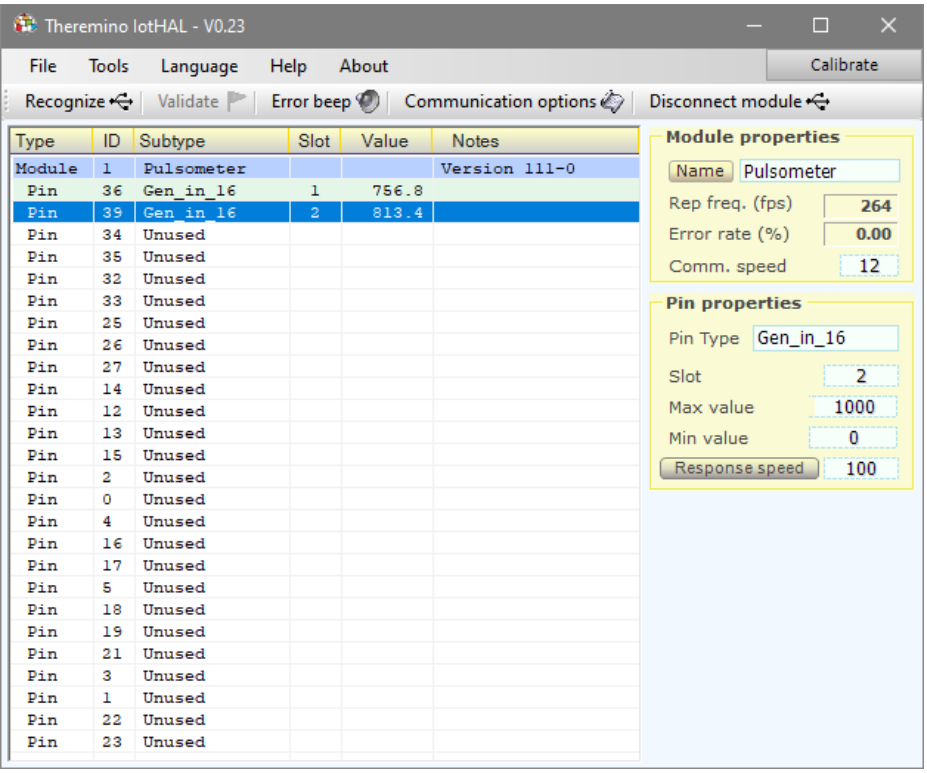
- 27 for the SDA signal
- 14 for the SCL signal

To find the Pin positions on the ESP32 modules,
see the images at the beginning of this document (page 4).

Example 5 - Reading data with the IotHAL application

To read the earth rate sensor:

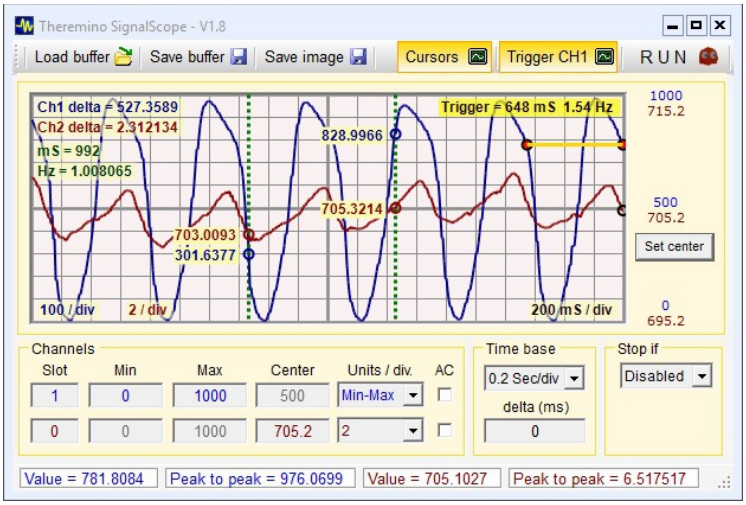
- Prepare the Arduino IDE, as explained in the IotHAL application documentation.
- Program the ESP32 with the sketch "IotModule_Pulsometer.ino".
- Launch the application "IotHal"

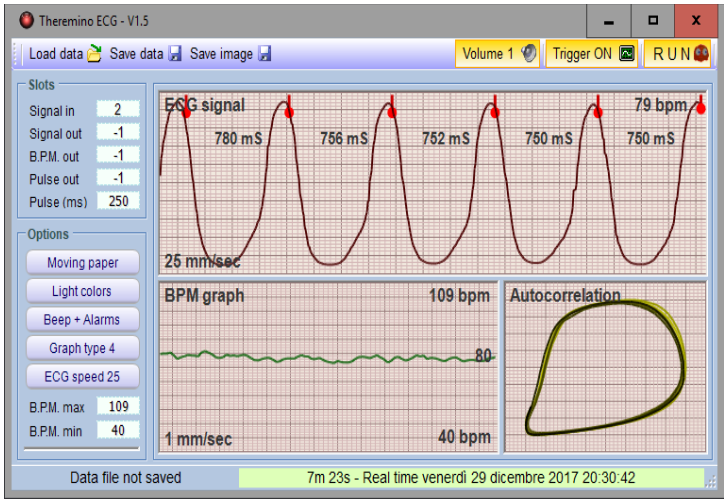


As in the firmware, in the two lines "genericWrite_16" we set the Pin "36" and "39". also in IotHAL application we must configure the "Pin Type" for these two Pins as "Gen_in_16".

Remember also to set "Response speed" to 30 on both the Pins, to filter the data further.

The numeric data received on Pin 39 is sent from the IotHAL to Slot 1, from which all of the Theremino system applications can read it, for example ***Theremino_SignalScope*** or ***Theremino_ECG***, which are shown in the following two images.

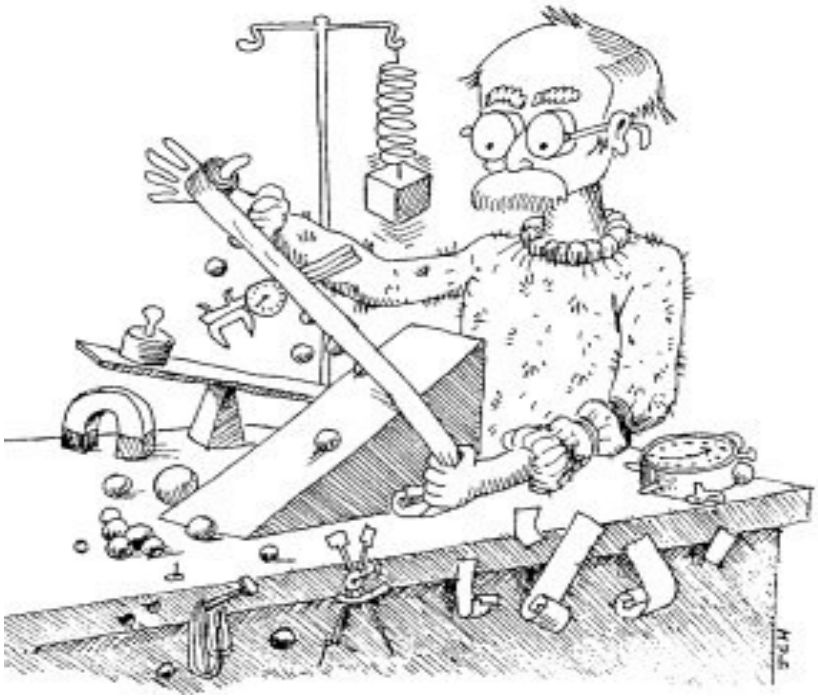




Techniques used in the sensor's firmware

To use the sensors described in this documentation
it is not necessary to continue reading.

Continue only if you want to learn how to write firmware for sensors other than these,
or to study our filter techniques and automatic sensitivity control



On the next page we explain how to start from the basic "IotModule.ino" file
and add the necessary lines to read the cardio tacho sensor.

In the following pages we will explain the techniques we use
to filter the cardio tachometer sensor signal
and to adjust the gain automatically.

Modify the firmware starting from : IotModule.ino

The following steps are only a trace
for those wishing to develop similar projects.

To read the sensors described in this document **do not follow these instructions**,
but return to the precedent pages, and use the firmware already prepared.

Example of reading an I2C sensor (simple version, without the filters)

- You load the file "IotModule.ino" in the ArduinoIDE and save it under a different name, such as "Pulsometer", so as not to alter our original firmware and you can reuse it for other projects.
- You add files "MAX30102.cpp" and "MAX30102.h" specific for this sensor.
- You add the following lines at the beginning of the file "Pulsometer.ino"

```
#include <Wire.h>
#include "MAX30102.h"
MAX30102 sensor;
```

- Also in "Pulsometer.ino" file, in the "void setup ()" function, you add the rows to initialize the Wire library (I2C) and the sensor.

```
Wire.begin(pinSDA, pinSCL);          // Initialize I2C pins
sensor.begin();                      // Initialize the sensor
sensor.setLEDs(60, 60, 0);           // RED, IR and GRN LEDs - Using IR only
sensor.setPulseWidth(pw118);         // PulseWidth = 118 uS (adc 16 bit)
sensor.setSampleRate(sr1000);        // SampleRate = 1000 samples per second
sensor.setWorkingMode(wmHeartRate); // We implemented HeartRate only
sensor.setAdcRange(rge16384);        // adc range = max (16 uA)
```

- Also in the "Pulsometer.ino" file, in the "void loop ()" function, you add the lines to read the sensor data and to send it to the IotHAL

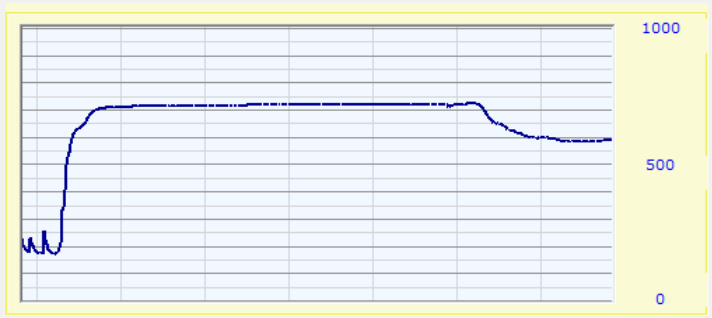
```
// ---------------------------------------- Read the sensor
sensor.readSensor ();
// ---------------------------------------- Send RED and IR values to IotHAL
Theremino.genericWrite16 (36, sensor.IR);
Theremino.genericWrite16 (39, sensor.RED);
```

- You write the firmware on the ESP32, and launches the IotHAL
- In the IotHAL you configure Pins 36 and 39 as Gen_in_16
- If the sensor is connected you should immediately start to receive data.

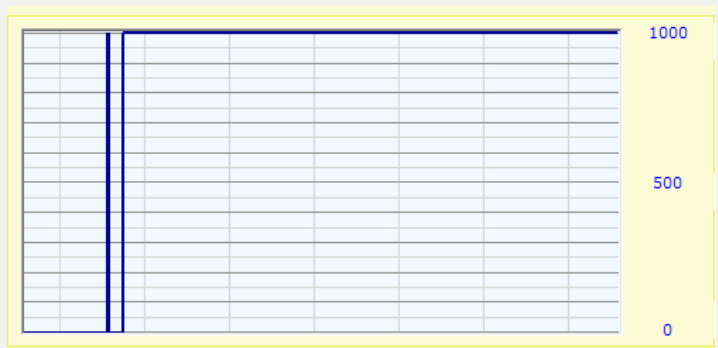
Filtering the data of the cardiac frequency sensor

Incoming data from the sensor are of small amplitude and pulse are almost invisible.

The measurement scale that we see here is "normalized" 0 to 1000 and the pulsations does not reach even one thousandth of this scale. In addition, the pulsations are masked by noise and slow changes due to the movements of the hand. In practice, you see only one line that goes high putting your finger and removing it at the bottom.

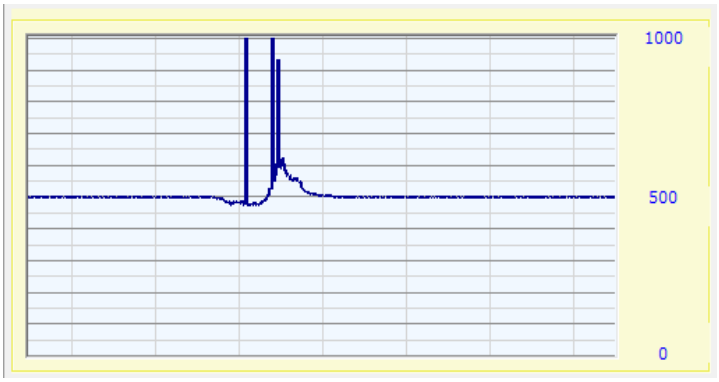


In this image you see the ascent (left) when the finger is inserted, a flat area of about 4 seconds when the finger is inserted, and a drop-down caused by having moved little the finger. As can be seen the pulsations are totally invisible



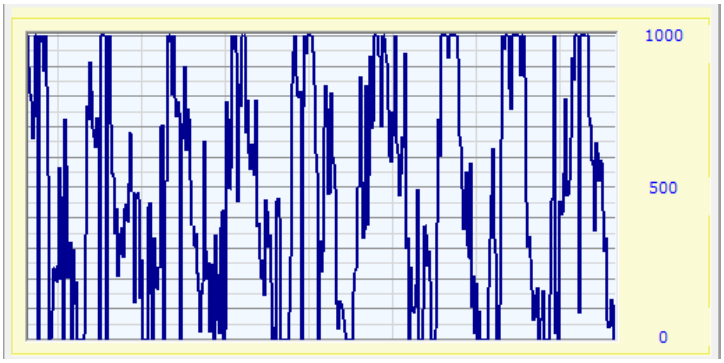
So we need to amplify the signal. But amplifying all that is beyond the middle goes below zero and the parts more than half are out of scale in the upper.

Putting your finger you see something like this. So before to amplify the rocking must be eliminated with a high pass filter.



Here is the high pass filter effect on the non-amplified signal.

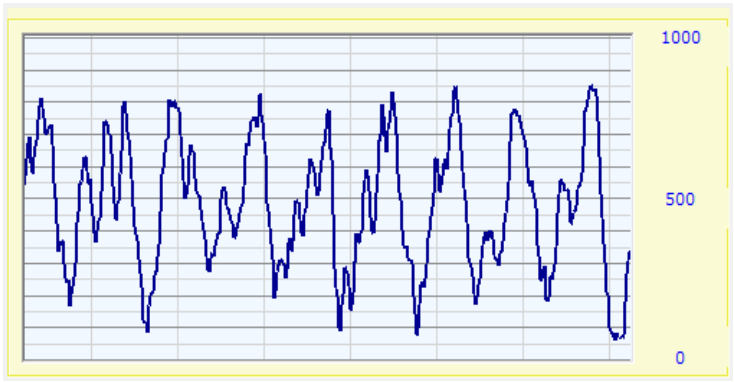
In the left side the finger was not there. In the moment in which it is inserted it shows a strong disturbance and after about half a second the high pass filter re align the signal at half scale.



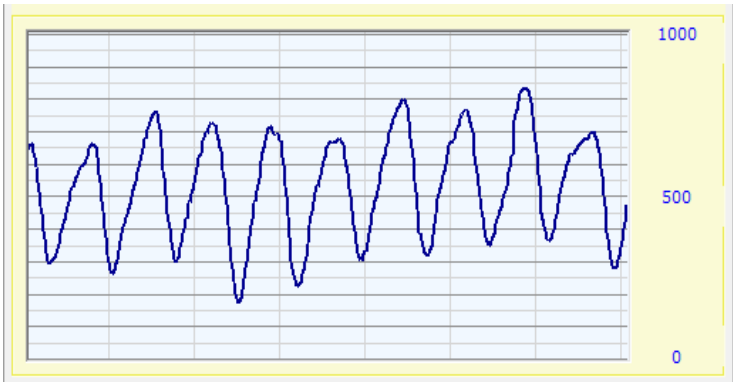
Having a signal centered on the half scale we can amplify a lot and start seeing something.

Here is the effect of an amplification of 2000 times.

We begin to see the pulses but they are masked by a lot of noise.



With a low-pass filter the pulsations become recognizable.

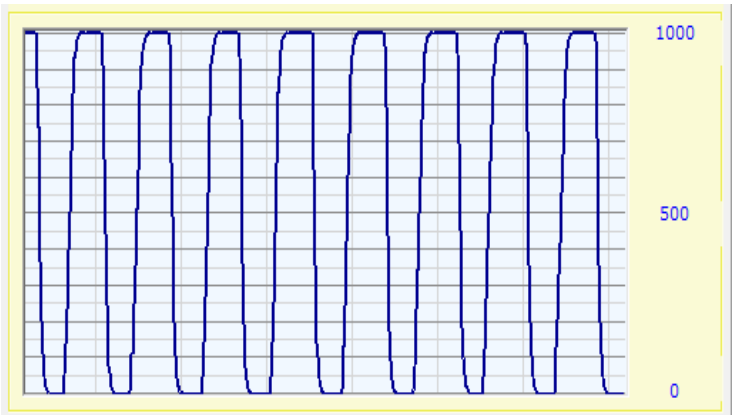


By adding a second stage to the low-pass filter and adjusting to 30 NetHAL the IIR filter in the noise disappears completely.

Stability is not exceptional but is already better the images that are published on the web.

Here is a typical ***example of pulsations*** downloaded from the Sparkfun site that builds the breakboard used in these tests.

At this point it is important to note that all previous tests were done on the worst possible signal. One of us (the writer) almost always has cold hands and a peripheral circulation almost non-existent, so it is a great person to test the sensors.



If you have a strong signal, everything becomes easier. Here the signal of a patient with high blood pressure and a normal peripheral circulation.

In this case it may also amplify less, to not clip the signal top and bottom.

However, to measure the frequency and arrhythmias the shape of the signal does not count. To which a strong signal like this, even if squared, could provide very stable measurements.

Implement the filters in the firmware

A low pass filter can be constructed with a single software line.

```
LowPass + = (InputData - LowPass) * 0:02;
```

To implement a high-pass filter you adde a second line that calculates the difference between the signal and the output of the low pass. So by removing the low frequencies from the signal remain only those high, and you get the high pass.

```
LowPass + = (InputData - LowPass) * 0:02;
 HIPASS = sensor.IR - LowPass;
```

These simple filters are the exact equivalent of the resistor and capacitor hardware filters and are also adjustable. If you increase the coefficient (which is here 0.02) the cutoff frequency rises.

It is necessary to adjust these filters experimentally because the cut-off frequency depends on the repetition time with which these lines are called. And this time depends in turn on how much processing is added to the Arduino Loop.

To avoid this laborious calibration, in the library "ThereminoFilters" we measured the repetition time of the loop and correct the filters at every step.

You can then set a cutoff frequency in Hz (also with decimals), and this will be respected forever (as long as the repetition rate of the loop is at least twice the highest frequency of interest to us). This is not a difficult request to be respected because usually the repetition rate is at least ten times the frequencies of the signal. A high frequency of repetition is called "oversampling" and serves to avoid aliasing phenomena, ie the tilting in the signal band, of the undesired signals (noise) that have frequencies higher of the sampling frequency.

Example of use of the filters

This example shows how to use the filters in the library "ThereminoFilters".

First of all one of the first lines of the ".ino" file you must add the line:

```
#include "Utility/ThereminoFilters.h"
```

Then you must declare all the filters that you use, and their cut-off frequencies, in the area that is located just before the function "void loop ()"

```
// ------------------------------- Filters declarations - HIPASS 0.7 Hz
Filter hipass1 (0.7, true);
Filter hipass2 (0.7, true);
Filter hipass3 (0.7, true);
Filter hipass4 (0.7, true);
// ------------------------------- Filters declarations - LoPass 3 Hz
Filter lopass1 (3, false);
Filter lopass2 (3, false);
Filter lopass3 (3, false);
Filter lopass4 (3, false);
// ------------------------------- Filters declarations - LoPass 2 Hz for Auto Gain
Filter lopass5 (2, false);
```

Finally you use filters one after another. In this case we have used eight filters, in order to obtain exactly the same response of the sensor "Theremino Pulsometer" which is seen in ***This Page***.

Warning: Each filter declared must be used only once. If you repeat twice the row of a filter the effect would be that of a single line, and it would waste computation time.

```
void loop ()
{
    sensor.readSensor();
    float filtered = sensor.IR;

    // ------------------------------------------- Hi Pass - 4 stages
    filtered = hipass1.run(filtered);
    filtered = hipass2.run(filtered);
    filtered = hipass3.run(filtered);
    filtered = hipass4.run(filtered);
    // ------------------------------------------- Low Pass - 4 stages
    filtered = lopass1.run(filtered);
    filtered = lopass2.run(filtered);
    filtered = lopass3.run(filtered);
    filtered = lopass4.run(filtered);

    ....
    ....
```

In the following lines of the loop the filtered signal is amplified and sent to the IotHAL, as we shall see in the next page.

To amplify the signal and send it to the IotHAL

The following block adjusts the gain (amplification) to obtain an output signal of constant amplitude with all patients.

```
    // ------------------------------------------- Auto gain
    float v = abs(filtered);
    v = lopass5.run(v);
    float gain = 24000 / v;
    if (gain > 5000) gain = 5000;
    filtered *= gain;
```

The filtered signal is "rectified" with the function abs. That is, the negative part of the signal is reversed into positive. Then the adjusted value is passed into a low pass filter and you get an estimate of the amplitude of the signal. Then you calculate the gain that should be applied to the signal. The number 24000 has been found experimentally to obtain the maximum amplitude but leaving a small margin above and down. Then the gain is limited to 5000 to avoid that grow too much when there is no signal. If it grow too much then the noise would be amplified and will look like a useful signal. And finally carries out the amplification with the line "filtered *= gain"

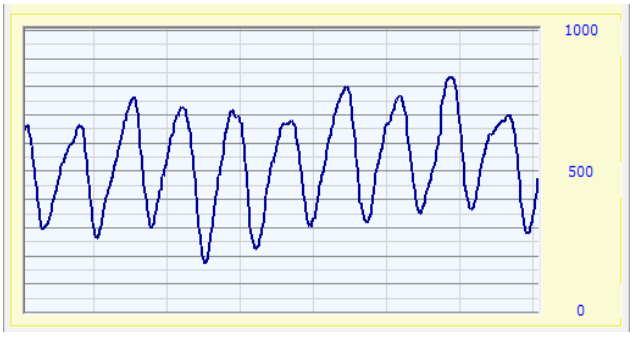
```
    // ---------------------------------------- Limit amplitude to unsigned 16 bits
    filtered += 32768;
    if (filtered > 65535) filtered = 65535;
    if (filtered < 0) filtered = 0;
```

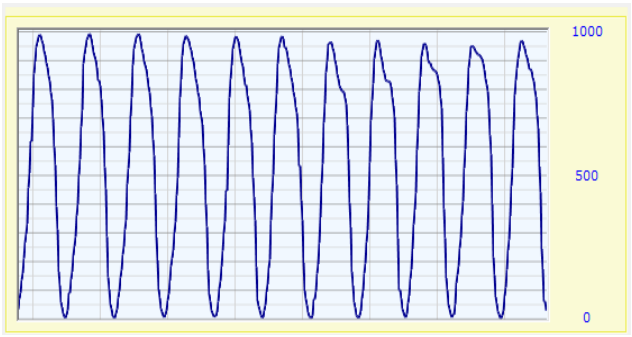
In the first line the signal is translated into high 32768 (half of a 16-bit) so it is no longer a number centered on zero but centered in an unsigned integer of 16 bits. In the two following lines it is limited to a 16-bit number, ie between 0 and 65535.

```
    // ---------------------------------------- Send raw IR and filtered IR to IotHAL
    Theremino.genericWrite16(36, sensor.IR);
    Theremino.genericWrite16(39, filtered);
}
```

Finally, you send the unfiltered and not amplified "sensor.IR" signal to the IotHAL.
And with a second row we send also the filtered value.

In the next two images you see the improvement that is achieved with the automatic gain

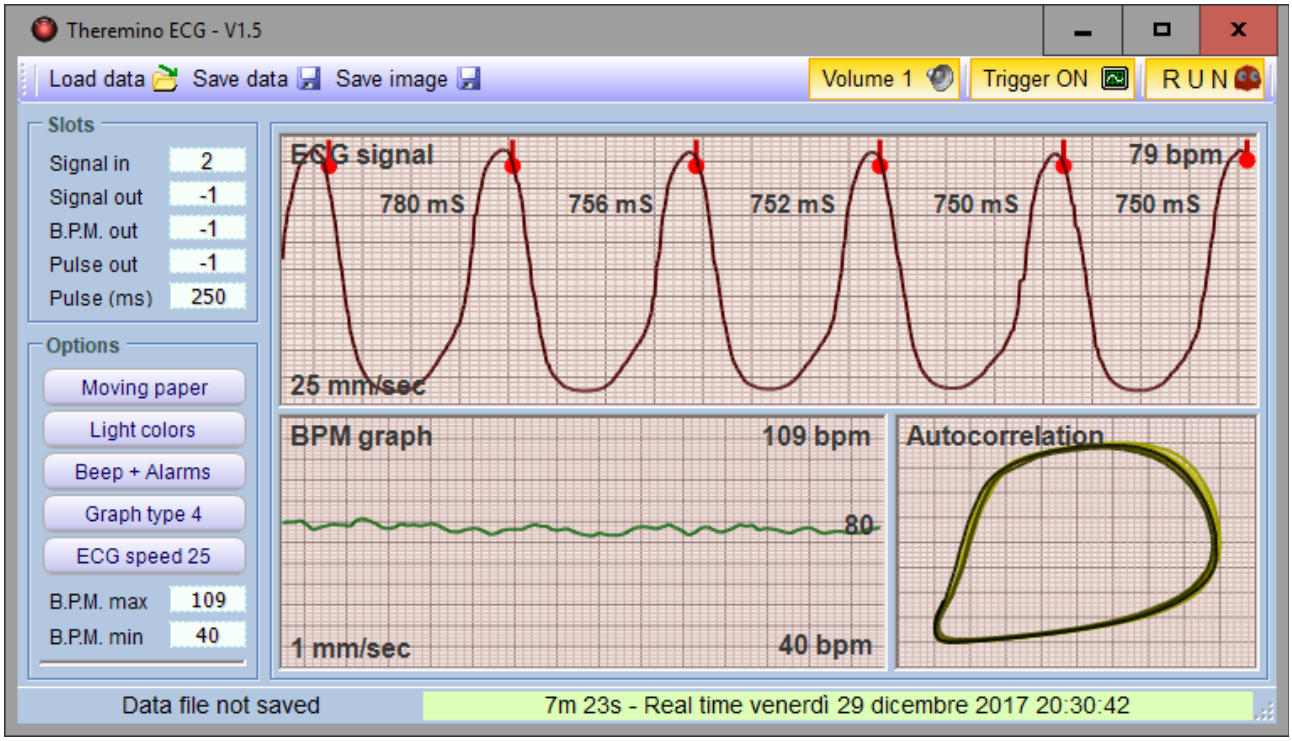




Comparison with the PulsoSensor

Who has the low-pressure and cold hands and always produces a weak signal because it has a poor peripheral circulation. In some circumstances, for example during the digestion, the circulation device is further reduced. In these moments some people could become subjects very difficult to measure.

So, both in ***PulsoSensor*** (connected to the master module), and in this sensor MAX30102 (connected to an ESP32), we have optimized the response curve and the amplification to maximize reliability in ***Frequency Measurement and research of arrhythmias***.



In this image you see the ECG application you download from ***This Page***.

Using four high-pass filters, and four low-pass are obtained almost the same results that we have obtained with the resistors and capacitors filters on the ***PulsoSensor***.

The PulsoSensor still is slightly better, because the light passes through the finger and is not reflected by the first layers of the skin. This is well explained in its ***documentation***.

- - - - - -

All measurements shown here were made on a notoriously difficult subject (the author of these pages) that have low blood pressure and that at certain times of the day has almost no peripheral circulation (frost hands).

With other subjects the signal can be greatly better. In some cases, the signal may be strong enough to saturate and become almost a square wave. This signal distortion is not a problem since for research arrhythmias are interested only the frequency and not the waveform.

Calculate the oxygen saturation

The MAX3010x sensor signal is just sufficient to measure the heart rate. And even under the best conditions the signal is not very stable and you have to stand still during the frequency measurement.

To obtain a minimum of reliability we filter the signal heavily and continuously modify the amplification. But these techniques are incompatible with the measure of saturation because the algorithm that calculates it needs unfiltered "RED" and "IR" signals.

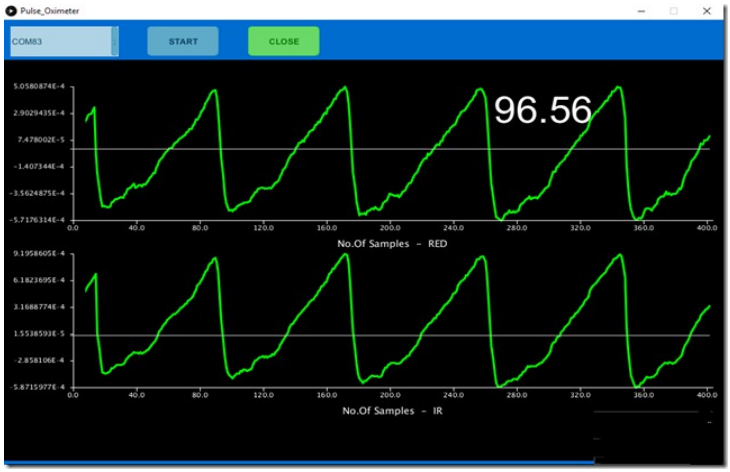
Measuring the oxygen saturation would require a much larger and noise-free signal, and even under the best conditions, the precision of measurement would be poor, ***see this page***.

From our tests, in most real cases you can only get random and useless numbers. Only with certain patients who give a very strong signal, and standing perfectly still, you could get a minimum of precision.

We think such an unreliable device is not much useful, and not worth it to lose the time, so willingly leave to others the fun trying.

- - - - - - -

The Maxim libraries to calculate the saturation is here:
https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library/tree/master/src



Beware that you can not just add the files "C ++" and "h" to our project. All our firmware works continuously, while Maxim algorithms only work on a long buffer of stored samples. You should, therefore, decompose and recompose the Maxim firmware in a different way, and it's a job that requires a lot of experience in programming.

Before undertaking such work we suggest that you try the complete Maxim project, without IotHAL but with the serial port, as they conceived them. You will see that in most cases does not provide any results, that goes wrong or produces totally wrong numbers.

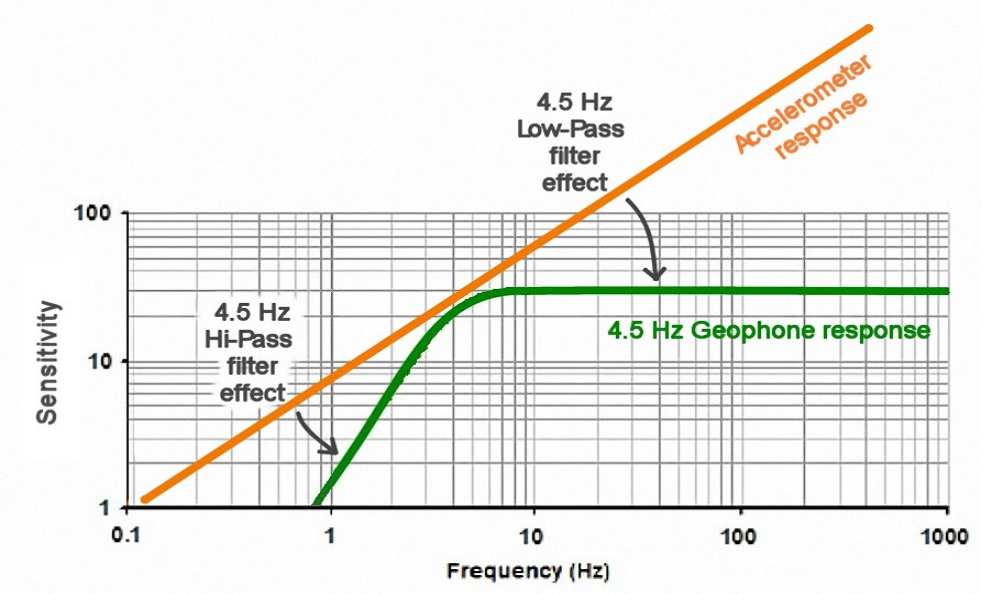
Only managing to make it work reliably you might think to lose some time, and connect it to the IotHAL.

Translate accelerometers to velocimeters

The firmware of the examples 2 and 3 (three axial accelerometers) contains low-pass filters (one per axis) that transform the sensor from "accelerometer" to "velocimeter".

The low pass filter integrates the acceleration data over the time, and the acceleration integral is the speed.

With this filter you get data similar to those you would get from electromechanical geophones, like those in the image on the right.



The most used geophones have a response that starts at 4.5 Hz, so in the firmware we set the cutoff frequency of the filters to 4.5 Hz.

High pass filters correct the slope for frequencies from 4.5 Hz down, while high pass filters correct it from 4.5 Hz up.

High pass filters, in addition to obtaining a frequency response similar to that of geophones, also eliminate the continuous component of the signals. So the average value of the signal is always exactly halfway between the minimum and the maximum (the value 500 in our system).

In the firmware you will find the line:

```
#define USE_FILTERS
```

Commenting this line you eliminate all the filters. Testing without filters could be useful to calibrate the sensitivity with the gravity acceleration value, which is about one.

And then two lines that determine the filter frequencies:

```
const float HiPassFreq = 4.5;
const float LowPassFreq = 4.5;
```

By changing the cutoff frequency, different geophones could be simulated. For example, the 10 Hz geophones, or those very expensive at 2 Hz.

Normally you should keep the two frequencies (high pass and low pass), equal to each other, but we left the possibility to do experiments and to adjust them independently.