Disclaimer

Breathing Games is an open network of medical doctors, scientists, engineers and hackers, working together to make respiratory therapy fun. Sensorica is an open network of scientists, engineers and hackers, collaborating on open source hardware development, design and prototyping. The Sensorica open network interfaces with the Breathing Games open network at the design and prototyping level, the Sensorica lab (sort of makerspace) being used for prototyping and fabrication of small batches of Breathing Games sensors. Making and using this device is entirely your responsibility. Everything about this device and the associated software has been extensively documented and we strongly encourage you to read these documents. We also strongly encourage you to talk to a medical doctor before using this device and the associated software.

Also note that this device is designed to be used in conjunction with the traditional devices, not alone, as shown in the user manual.

Notes on reliability

Some of the following designs and prototypes have been thoroughly tested, some not. Please refer to the corresponding DIY tutorials for more (DIY Instructible 2023 doc and DIY instructables - BG pressure device 2024). Since this hardware has been designed and built by Sensoricans, please contact the community if you have questions.

Licenses

Content on this document is Creative Commons (BY SA).

The hardware developed here is released under CERN-OHL-S.

Document	BG-5-1 Pressure device
Doc purpose	Describe various possibilities for a device to connect the PEP apparatus.
Governance	Contributors must agree with the project vision and BG-1 Governance.
	To have your work surfaced, log contributions prototyping the device, here.
Licence	Content: Creative Commons Attribution-ShareAlike 4.0 Software: GNU AGPL Hardware: CERN-OHL-S All updated on Gitlab
Contributors	Tiberius Brastaviceanu, Jim Anastassiou, Fabio Balli and others
Related links	- Vision for Breathing Games - BG-4-1 Core document games - BG-8-1 Core document healthcare - See BG-5-3 Flowmeter theory - JOH - hardware metapaper

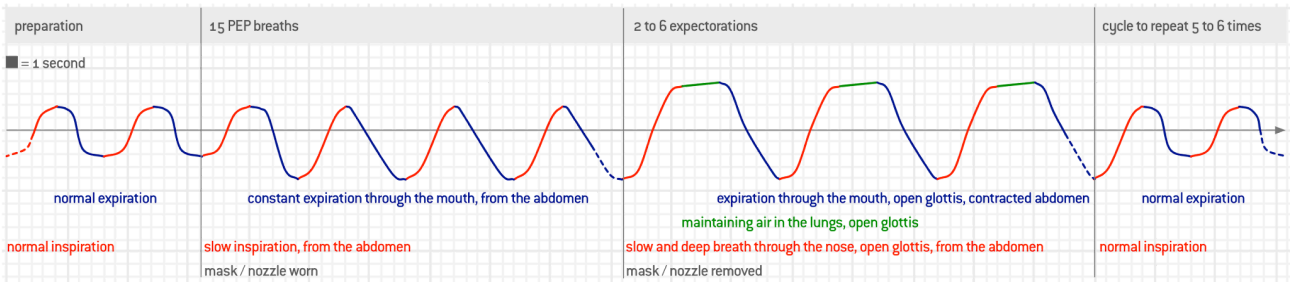
Deliverables

For pictures of the medical apparatus, go to BG-8-1 Core document healthcare.

Our first prototype (BG)	Our second prototype (BG)	Our current prototype (Sensorica)
		

See doc presenting other devices that Beeathing Games has developed not in collaboration with Sensorica

Example of therapy pattern at Sainte Justine



Log Contributions

Table of contents

Goals
Microcontroller boards
- Adafruit Feather 32u4 Bluefruit LE
- Arduino Nano

Pressure sensor configurations
- Adafruit Feather 32u4 Bluefruit with MP3V5010DP sensor
- Adafruit Feather 32u4 Bluefruit with the MPXV5010DP sensor
- Arduino Nano with MPX10DP sensor

Electronics Components
- Pressure sensors
  - MPXV5010DP Pressure sensor differential
  - MP3V5010DP Pressure sensor differential
  - MPX10DP Pressure sensor differential
  - BMP085 Barometric Pressure/Temp./Altitude Sensor- 5V
  - Other good candidates
- Communication
  - Adafruit Feather 32u4 Bluefruit LE - USB cable
  - Adafruit Feather 32u4 Bluefruit LE with bluetooth BLE - wireless
  - Bluetooth LE 4.0 with Arduino - wireless
- Data storage
- Data processing
  - Adafruit Feather 32u4 Bluefruit LE data processing
  - Arduino nano data processing

Power & power management
- Battery for Adafruit Feather 32u4 Bluefruit LE
  - Power management
  - Programmatically turn off the 3V regulator
  - Solid state relay (SSR)
  - Richard's solution
- USB cables and adapters

Others
- Visual feedback

Mechanical
- Organic Box
- Mouthpiece
- Air tube
- Tubing and filters

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

- Air tubes connections
- Others
- Testing this configuration
- Software
  - Firmware for Adafruit Feather 32u4 Bluefruit with USB cable operation
    - Connect to Gitlab
  - Firmware for Adafruit Feather 32u4 Bluefruit with BLE wireless connection mobile
  - Data management
- Identification of device
- Regulatory compliance
  - Validation of critical components

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

Goals

Create hardware interfaces to serious computer games that are used in healthcare.

This particular R&D effort is to create a PEP (Positive expiratory pressure) interface, which can be used as a controller for a computer game, used in cystic fibrosis therapy. In essence, the goal is to design and prototype a hardware device that can sense air pressure and communicate that to a computer or a mobile device that hosts a computer game.



Credit: A new economy documentary film

In respect to the initiative's mindset, the device must

- ☑ be as **accessible** as possible
  - ☑ released under public license with due respect to open standards
  - ☑ minimal cost, possibility of do-it-yourself
  - ☑ compatible with all standard computers (laptops, smartphones, tablets)
- ☐ be **usable autonomously**, to collect all home therapy data even without using the game
  - ☐ device has an LED strip replacing the traditional manometer
  - ☐ device memorizes data even without Internet and electricity (30 min daily therapy)
  - ☐ device sends data once connected (via USB or directly ?)
  - ☑ device reloads battery once plugged
- ☑ be usable with the **main devices** used in breathing therapies
  - ☑ standard apparatus for PEP / nebulizing therapy (Pari, ...)
  - ☑ vibratory apparatus for PEP therapy (Flutter, Acapella, ...)

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

- ☑ standard apparatus for PEP therapy (RC Cornet, ...)

- ☑ provide **accurate measurements**
  - ☑ capture of data every 0.2 seconds
  - ☐ measure of flow
  - ☐ measure of expiratory pressure (0 to 20 cm / H_20)

- ☑ respect the safety and hygiene **standards**
  - ☑ individual use with regular cleaning (boiling)
  - ☑ possibly shared use (filter) ?

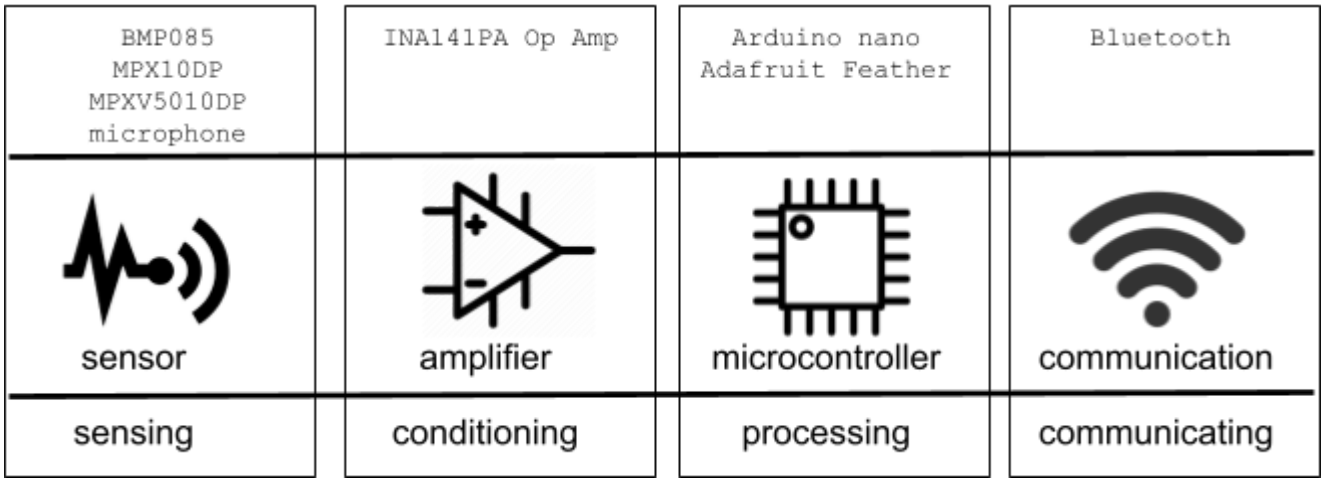
- ☑ be attractive in terms of **ergonomics**
  - ☑ offer an interesting user experience

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

Microcontroller boards

Microcontrollers are used to acquire data from a pressure sensor, digitize that data, process it and communicate it to other electronic devices. See more details in the Electronic components section.



We've only worked with two microcontroller boards for this project. See the two following sections.

Adafruit Feather 32u4 Bluefruit LE

During 2024, the stakeholders of this project, i.e. Breathing Games (which is the network organization that leads the project), Sensorica (hardware design, trust economy and regulatory approval lead organization) and Sainte Justine Hospital (sponsor and early adopter) have adopted the Bluetooth BLE mode of communication and have decided to expand the computer games that also work on mobile devices games to mobile platforms. See the BLE communication section for bluetooth communication development. The USB wire connection is also maintained to support PC-based computer games. See DIY instructables - BG pressure device 2024.

Message written in 2024.

This Adafruit Feather is an arduino-compatible + Bluetooth Low Energy with built in USB and battery charging. At the Feather 32u4's heart is ATmega32u4 (same as the Arduino Uno) clocked at 8 MHz and at 3.3V logic. To program the Feather you can use the Arduino ADI. Following the help page on Adafruit attentively will save you a lot of time to connect to the Feather.

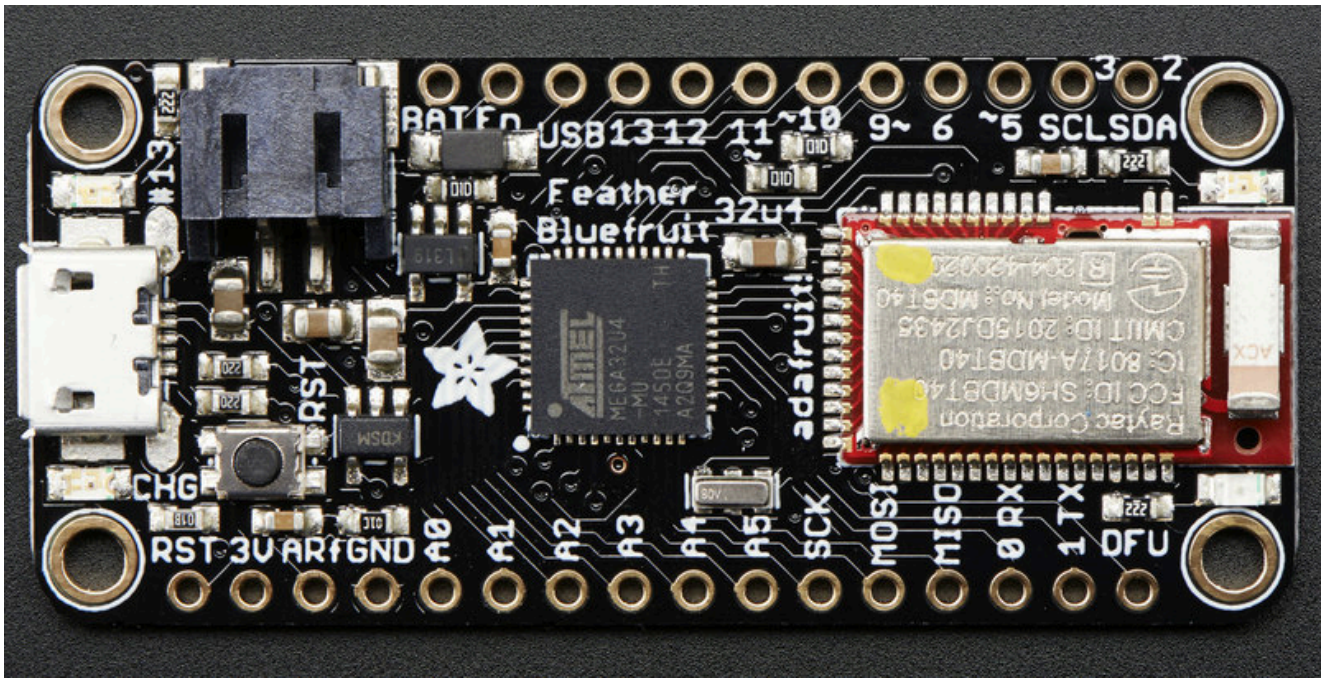
Handy specs:

- Measures 2.0" x 0.9" x 0.28" (51mm x 23mm x 8mm) without headers soldered in
- Light as a (large?) feather - 5.7 grams
- ATmega32u4 @ 8MHz with 3.3V logic/power

Content on this document is ***Creative Commons (BY SA)***.

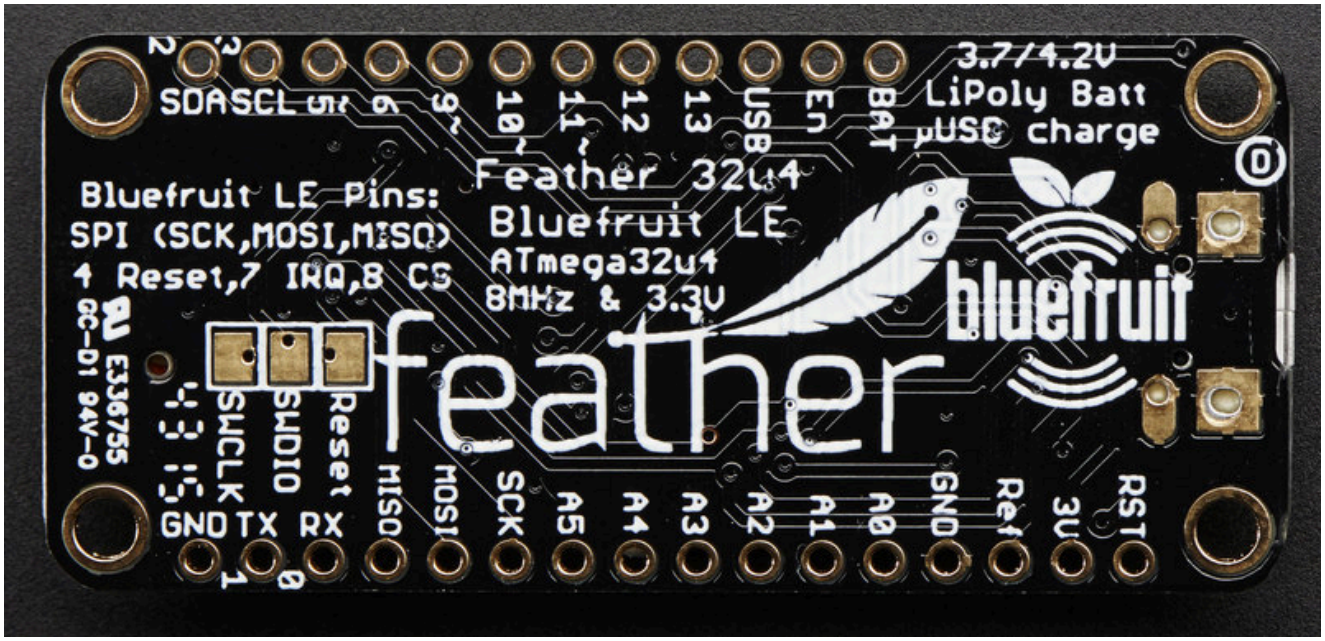
Log Contributions

- 3.3V regulator with 500mA peak current output
- USB native support, comes with USB bootloader and serial port debugging
- You also get tons of pins - 20 GPIO pins
- Hardware Serial, hardware I2C, hardware SPI support
- 7 x PWM pins
- 10 x analog inputs
- Built in 100mA lipoly charger with charging status indicator LED
- Pin #13 red LED for general purpose blinking
- Power/enable pin
- 4 mounting holes
- Reset button

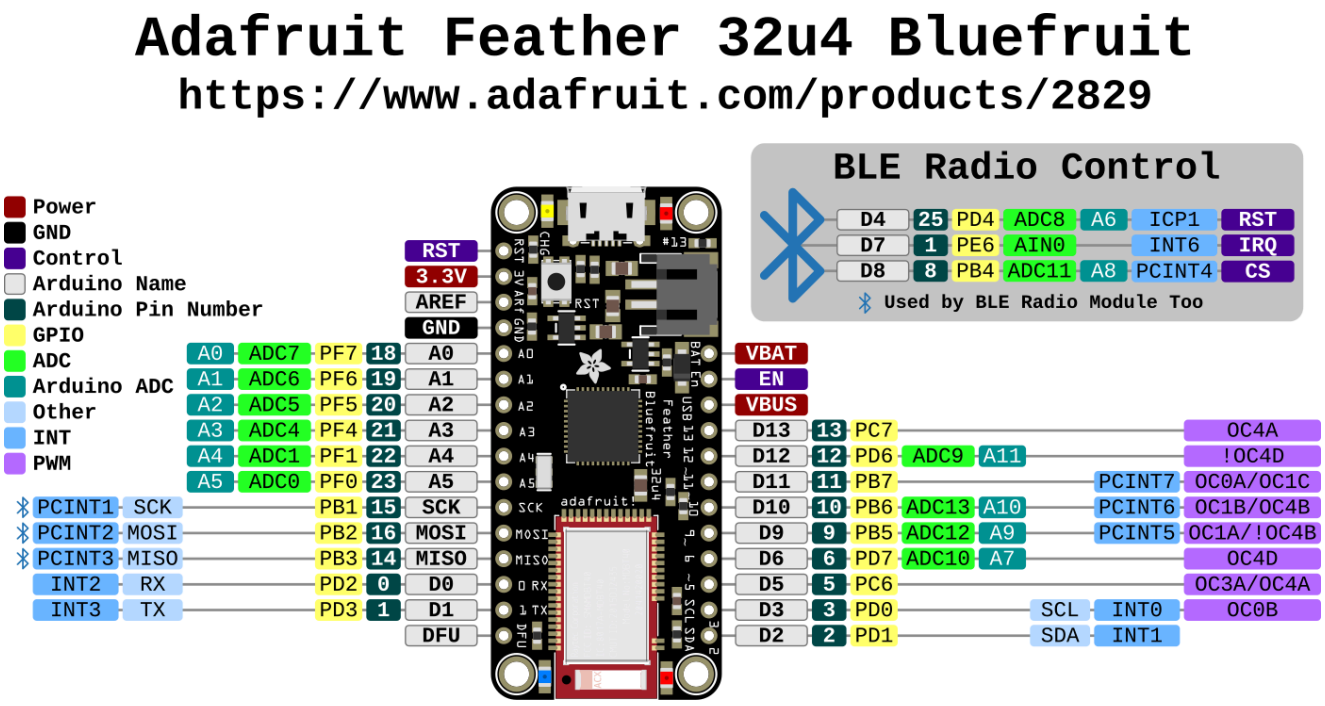


Content on this document is *Creative Commons (BY SA)*.

Log Contributions



The DFU pin is accidentally labeled GND on the bottom, sorry about that! it should be labeled DFU, dont use it as a GND



Content on this document is *Creative Commons (BY SA)*.

Log Contributions

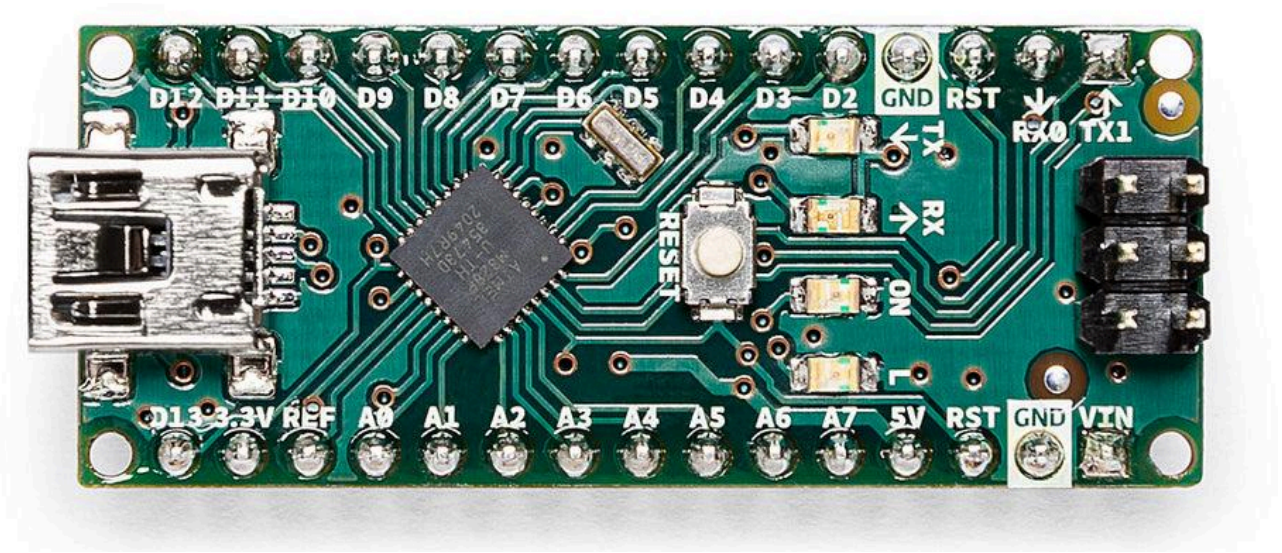
Open learning doc for Adafruit Feather. There is a version on the drive if that URL is deprecated.

Arduino Nano

During 2024, the stakeholders of this project, i.e. Breathing Games (which is the network organization that leads the project), Sensorica (hardware design, trust economy and regulatory approval lead organization) and Sainte Justine Hospital (sponsor and early adopter) have adopted the Adafruit Feather 32u4 Bluefruit with the BLE communication mode and have decided to move games to mobile platforms. **The Arduino Nano-based hardware design and prototyping is on pause for now**.

Message written in 2024.

The Arduino Nano is a small, complete, and breadboard-friendly board based on the ATmega328 (Arduino Nano 3.x). It has more or less the same functionality of the Arduino Duemilanove, but in a different package. It lacks only a DC power jack, and works with a Mini-B USB cable instead of a standard one. More into here.



Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

Pressure sensor configurations

Adafruit Feather 32u4 Bluefruit with MP3V5010DP sensor - V3

During 2024, the stakeholders of this project, i.e. Breathing Games (which is the network organization that leads the project), Sensorica (hardware design, trust economy and regulatory approval lead organization) and Sainte Justine Hospital (sponsor and early adopter) have adopted this MP3V5010DP pressure sensor. See DIY instructables - BG pressure device 2024.

Message written in 2024.

Precisely the device version that is described here has been tested and used in research at the Sainte Justine Montreal Children's Hospital, Department of Pneumology, by Sze Man Tse (MD), in collaboration with Olivia Cerutti Monteventi (MD). You can find official reports about these uses here. If you are a patient and want to use this device please share these reports with your doctor.

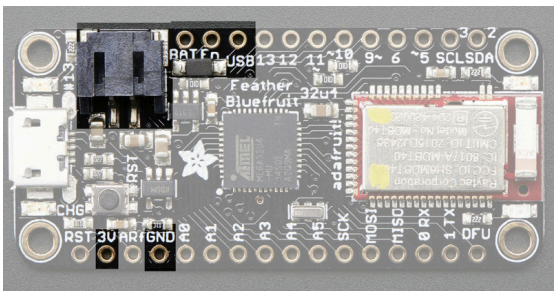
If you want complete instructions to make a device, please open the DIY instructables - BG pressure device 2024 for this particular configuration.

NOTE: This configuration and implementation is very similar to the one using the MPXV5010DP pressure sensor, see above.

Used the Adafruit Feather 32U4 Bluefruit LE microcontroller chip, provided with BLE wireless communication, all in the model b (organic box) (see also the Mechanical section in this doc), built for Saint Justine.

For powering the PM3V5010DP pressure sensor we feed from the 3V pin out pin of the Feather, connected to the 3V regulator. If a 3.3.V battery is used, use the BAT pin (3.3 straight from the battery) with a DC/DC converter.

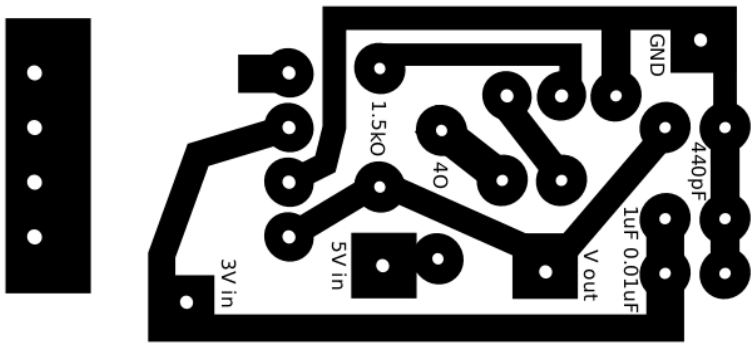
The USB pin provides 5V wherever needed (ex. For the LED visual feedback).



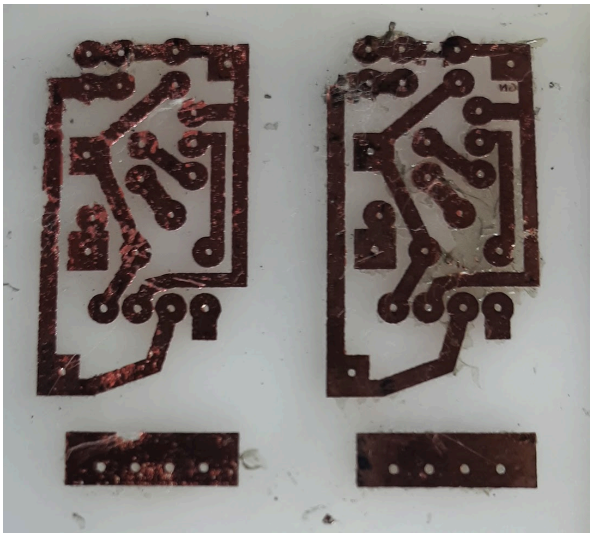
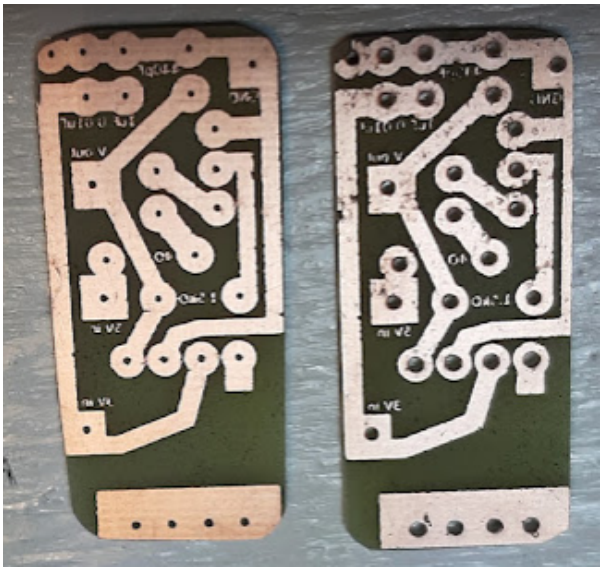
PCB version

Content on this document is *Creative Commons (BY SA)*.

Log Contributions



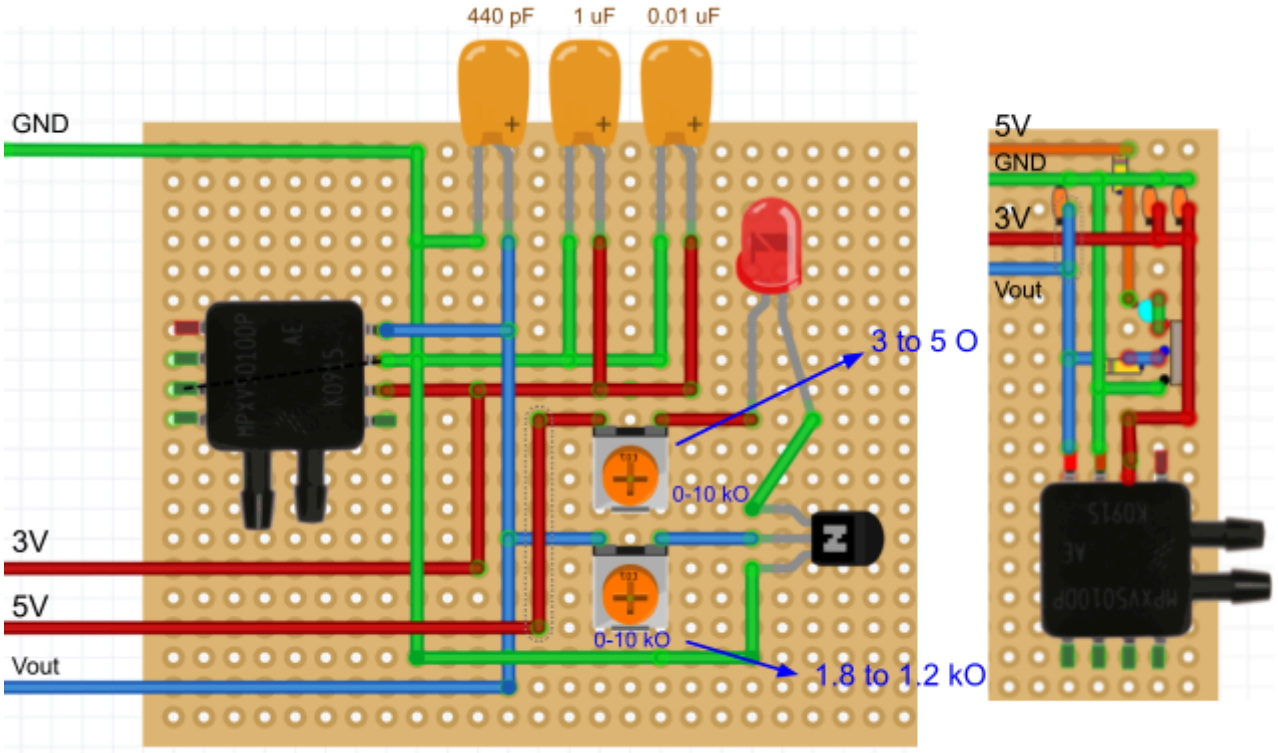
Open the PEP Master PCB board 2024 file (SVG format) - edit in Inkscape (open source image software).

Prototype: made by transferring laser printer toner using heat, to a copper tape that was glued on top of a plastic sheet. This was done to test the electrical traces for potential design errors. 	Prototype: made by transferring laser printer toner using heat, to a virgin, single layer copper PCB board. This was done to fabricate 40 devices to be used at Sainte Justine Hospital. 

Protoboard version

Content on this document is ***Creative Commons (BY SA)***.

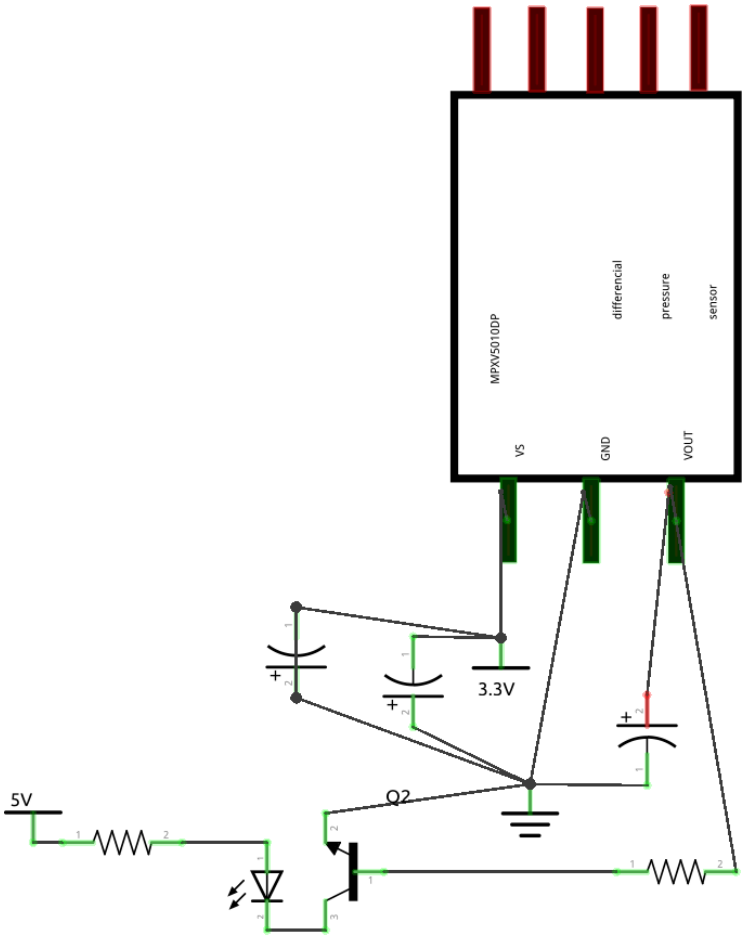
Log Contributions



Open file. 2023 implementation with MP3V5010DP and visual feedback

Content on this document is *Creative Commons (BY SA)*.

Log Contributions



See also Power Management.

Production

Open **DIY Instructible 2023 doc**.

User Feedback

Problem (reported by Olivia): water vapor accumulation in the tube, could potentially damage the sensor.

> Possible damage mechanisms: mineral deposits or growth of organic matter on the silicon membrane, responsible for measuring the pressure.

Discussed solutions (Olivia and Tibi): use a table salt container inline, between the sensor and the PEP device. Table salt retains water and kills bacteria. Salt can be stored in a small container. Salt can be prefiltered to keep larger particles, to ensure better air flow. Salt may need to be replaced regularly. Aquarium air rock can be used within the salt container as a 3D filter.

Content on this document is ***Creative Commons (BY SA)***.

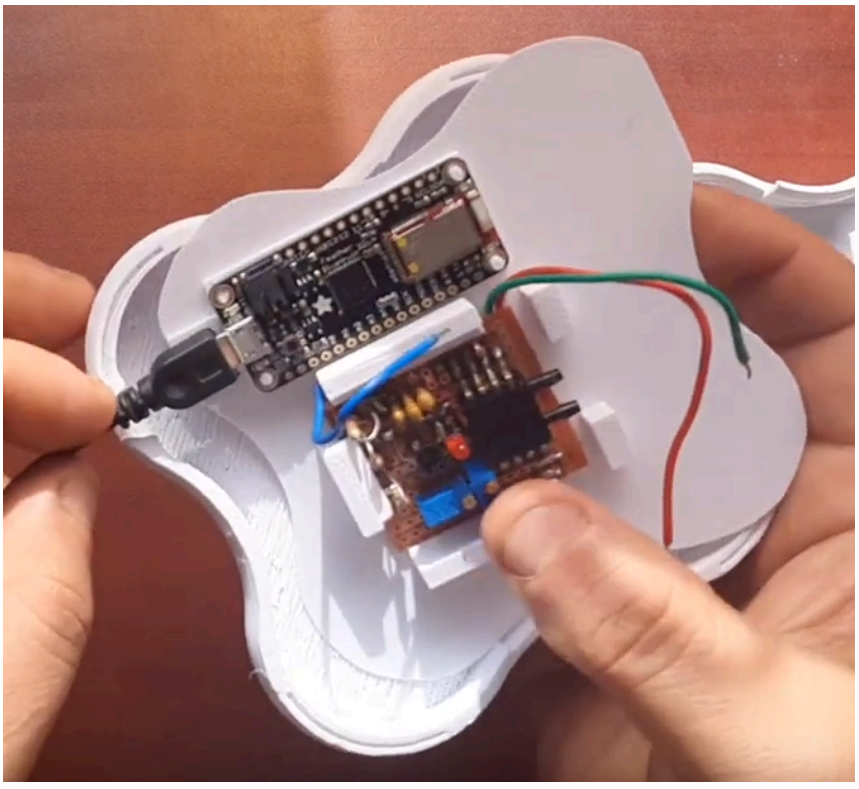
Log Contributions

Adafruit Feather 32u4 Bluefruit with the MPXV5010DP sensor - V2

During 2024, the stakeholders of this project, i.e. Breathing Games (which is the network organization that leads the project), Sensorica (hardware design, trust economy and regulatory approval lead organization) and Sainte Justine Hospital (sponsor and early adopter) have adopted the **PM3V5010DP** pressure sensor.

Message written in 2024.

Precisely the device version that is described here has been tested and used in research at the Sainte Justine Montreal Children's Hospital, Department of Pneumology, by Sze Man Tse (MD), in collaboration with Olivia Cerutti Monteventi (MD). You can find official reports about these uses here. If you are a patient and want to use this device please share these reports with your doctor. Also note that this device is designed to be used in parallel with the traditional PEP, both connected to the same mouthpiece through a flexible silicon tube.



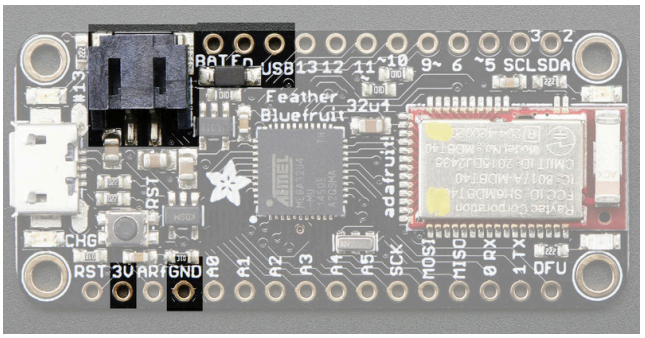
Content on this document is *Creative Commons (BY SA)*.

Log Contributions

NOTE: The MPXV5010DP pressure sensor operates at 5V and the Adafruit Feather 32U4 Bluefruit LE microcontroller chip operates at 3.3V. An equivalent pressure sensor is available, the PM3V5010DP, see configuration below.

This sensor was used in the Breathing Games device model b (organic box) (see also the Mechanical section in this doc), built for Saint Justine, you can see in this video and this video. Also used in this process, the Adafruit Feather 32U4 Bluefruit LE microcontroller chip, provided with BLE wireless communication.

For powering the MPXV5010DP pressure sensor one can think of using a DC/DC upconverter 3V→5V from the 3V pin, after the 3.3V regulator. That can cause instabilities and can heat the regulator (see this discussion in forum). If the Feather is powered via a 5V USB, the best thing is to use the USB pin, which is +5V from the micro USB connector, before the 3.3V regulator. If a 3.3.V battery is used, use the BAT pin (3.3 straight from the battery) with a DC/DC converter.

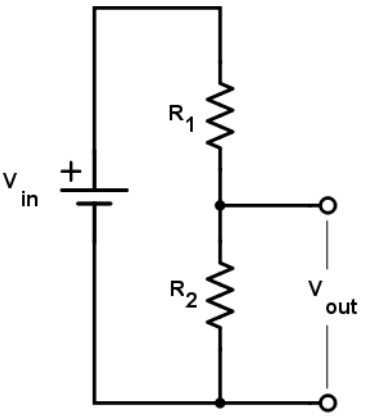


The easiest way is to go with the USB pin. One might think that the power supply is dirty and noisy. Here are some suggestions for a voltage regulator.

- Linear Technology - LTC3534, Datasheet On page 16 it even has a typical application being "USB to 5V at 500mA", so this might do the trick. I was unable to determine the kind of accuracy I might get with this. (see reference Forum discussion)
- Texas Instruments - TPIC74101, Datasheet, It mentions the accuracy is 5V +- 2%.... so that could be a bit better... that's 4.9V to 5.1V that's really not that great if I'm trying to increase accuracy right? (see reference Forum discussion)
- See also this Forum discussion for other suggestions.

Another problem if we go with the 5V USB power supply is the signal voltage from the sensor. The MPXV5010DP *Vo* varies from 0 to 5V. The Feather's ADC (analog to digital converter) operates at 3.3V. We need to scale 0-5V into 0-3.3V. The easy and dirty way to do it is a **voltage divider**. Use an online voltage divider calculator to vet the values of the resistors. In our case, they are : R1 = 47 kO ; R2 = 91 kO

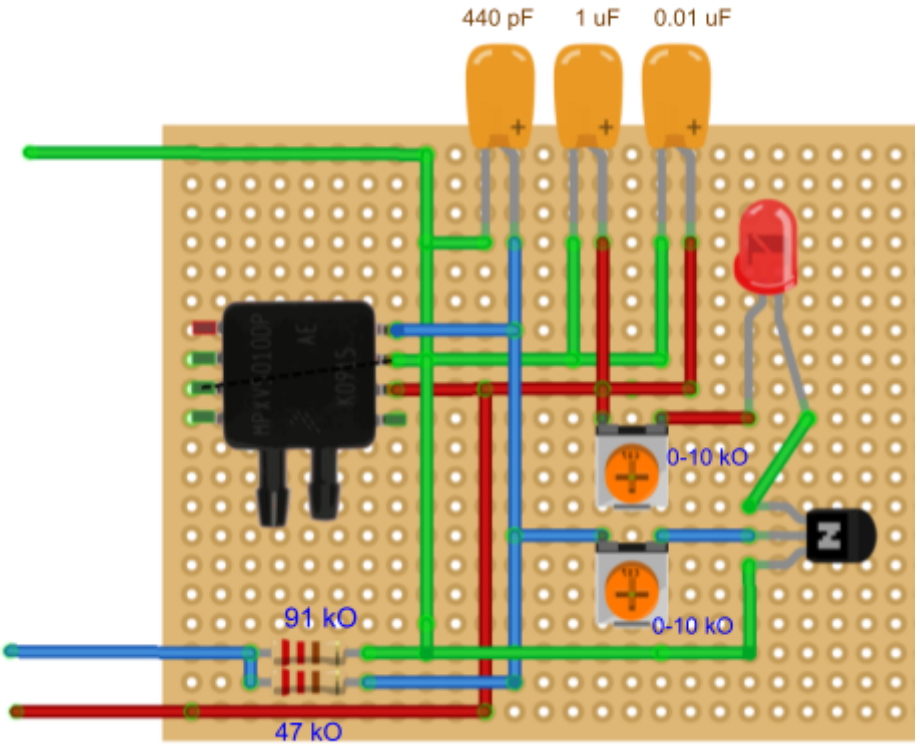
In the adjacent diagram Vin+ is what comes out of the sensor. You pur R1 between that and the analog in of the Feather. You connect R2 from that connection to the common ground.



Content on this document is *Creative Commons (BY SA)*.

Log Contributions

In 2023, the Visual feedback was added to this prototype, an LED that provides approx but direct indications about the pressure level. This LED doesn't depend on the microcontroller, it is fed directly from the pressure sensor through an NPN transistor. So this visual indicator can be used for debugging the hardware as well.



Open file. 2023 implementation with MPXV5010DP and visual feedback

Software

Get the firmware here.

Others

Mouthpiece

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

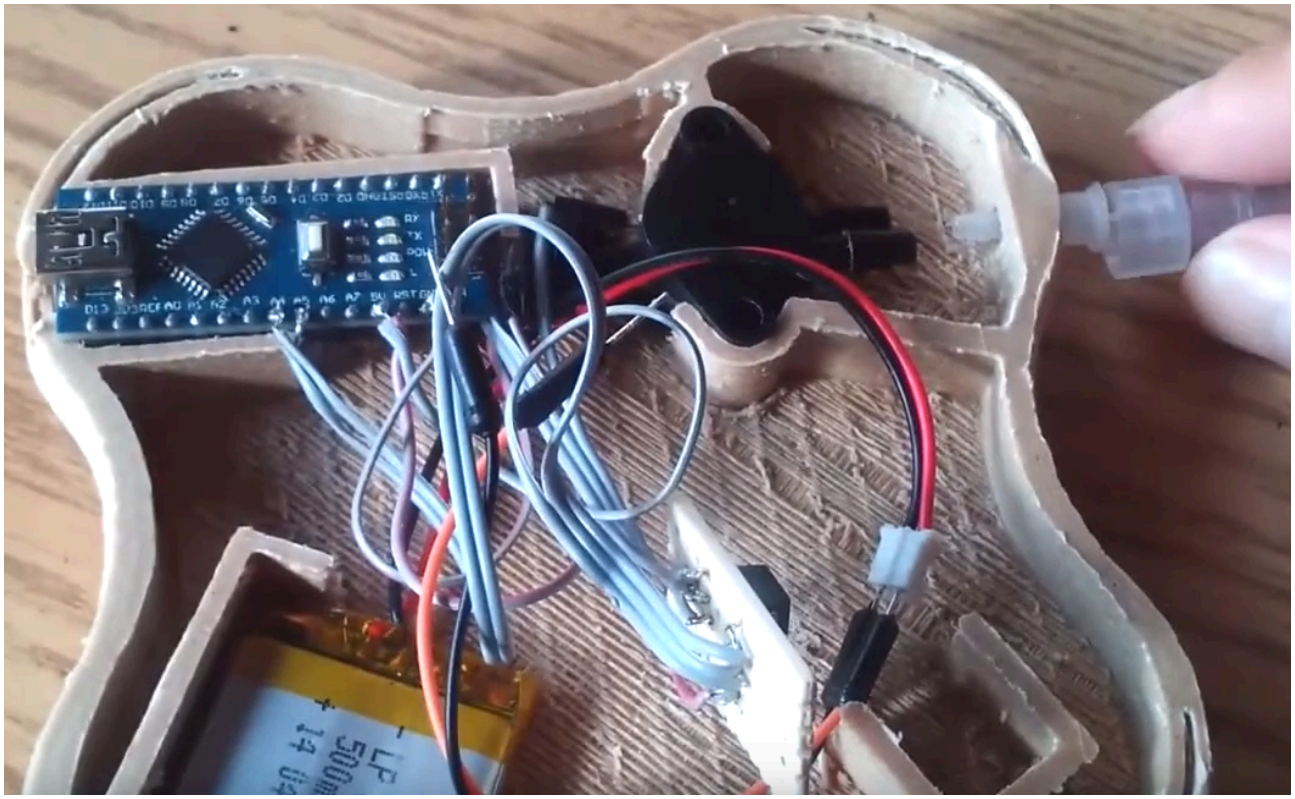
Arduino Nano with MPX10DP sensor - V1

During 2024, the stakeholders of Breathing Games (project lead network organization), i.e. Sensorica (hardware design, trust economy and regulatory approval lead organization) and Sainte Justine Hospital (sponsor and early adopter) have adopted the Adafruit Feather 32u4 Bluefruit with the BLE communication mode and the MP3V5010DP pressure sensor. They have decided to move games to mobile platforms. The Arduino Nano hardware design has been abandoned.

Message written in 2024.

Precisely the device version that is described here has only been used at the beginning of the Breathing Games venture and has not been extensively tested.

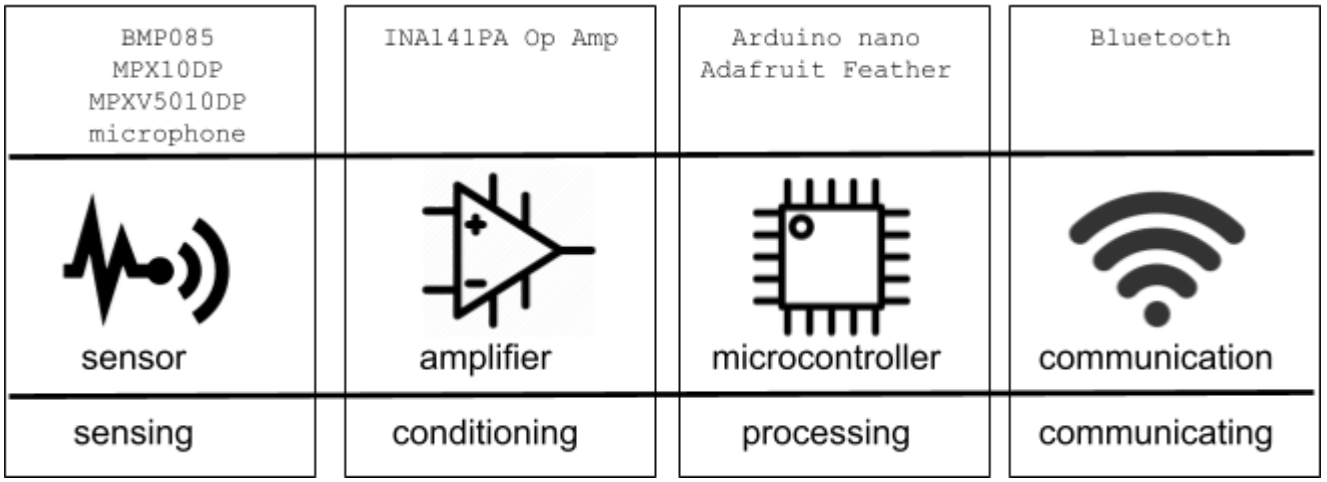
This sensor was first tested by Jim in 2016 without amplification in this video and was integrated with the INA141P Instr. Amp in the first Breathing Games prototype as seen on this video.



Content on this document is *Creative Commons (BY SA)*.

Log Contributions

Electronics Components



The main components are the microcontroller board and the pressure sensor. In the sections below we'll talk about different pressure sensors we have worked with in the past.

Pressure sensors

MPXV5010DP Pressure sensor differential

During 2024, the stakeholders of Breathing Games (project lead network organization), i.e. Sensorica (hardware design, trust economy and regulatory approval lead organization) and Sainte Justine Hospital (sponsor and early adopter) have adopted the Adafruit Feather 32u4 Bluefruit with the MP3V5010DP pressure sensor.

Message written in 2024.

This is an amplified differential pressure sensor, same format as the PM3V5010DP, but operating at 5V, which can directly connect with current open source microcontrollers such as Arduino and Adafruit.

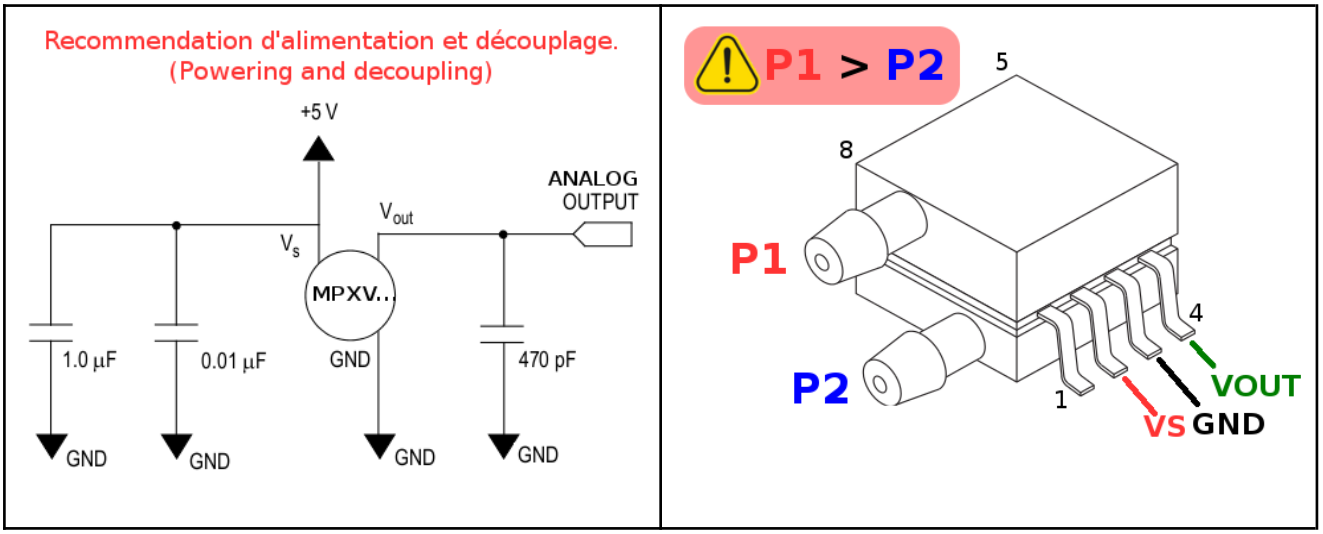
Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

Operating Pressure: 0 kPa to 10 kPa
Pressure Type: Differential
Accuracy: 5 %
Operating Supply Voltage :5 V
Output Voltage: 4.7 V
Operating Supply Current :5 mA
Datasheet



You only need to wire the sensor output (VOut) to an analog input of the microcontroller. To use this sensor, you only need to power it up and measure the output voltage. Not that decoupling capacitors are required for a better precision and stability of the output signal.



Be aware that Pressure P1 > P2.

You can recognize Pin1 by the small notch.

The formula:

$$Vout = Vs \times (0.09 \times P + 0.04)$$

Vs : supply voltage (5V)

P : the pressure difference in kPa

So, it is possible to measure the difference of pression P (in kPa) from the output voltage of the sensor (Vout)

$$P_{\text{in kPa}} = ((Vout / Vs) - 0.04) / 0.09$$

At the end, it is possible to measure the height of water (in mm) from the kPa pressure.

The 1000 coefficient is used to transform the meter unit to millimetres.

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

$$H2O\ mm = P * 0.1019 * 1000$$

Some examples: this video.

Testing this sensor with Arduino Uno.

See our video.

Just connect Vo to an analog in, ground to ground and supply 5V to the sensor from the 5V pin. Nothing more just to see how it works.

MP3V5010DP Pressure sensor differential

During 2024, the stakeholders of Breathing Games (project lead network organization), i.e. Sensorica (hardware design, trust economy and regulatory approval lead organization) and Sainte Justine Hospital (sponsor and early adopter) have adopted the Adafruit Feather 32u4 Bluefruit with this MP3V5010DP pressure sensor.

Message written in 2024.

This is an amplified differential pressure sensor, same format as the MPXV5010DP, but operating at 3V, which can directly connect with current open source microcontrollers such as Arduino and Adafruit.

This is better suited for the Adafruit Feather 32U4 BLE board, which operates at 3V. Another advantage for using this sensor is that it allows use with a 3.3V battery (which plugs well into the Feather and the Feather has charging circuitry on board), since in this case there would not be 5V DC from the USB.

Operating Pressure: 0 kPa to 10 kPa
Pressure Type: Differential
Accuracy: 5 %
Operating Supply Voltage: 3 V
Output Voltage: 3 V
Operating Supply Current :7 to 10 mA
Datasheet



You only need to wire the sensor output (VOut) to an analog input of the microcontroller. To use this sensor, you only need to power it up and measure the output voltage. Not that decoupling capacitors are required for a better precision and stability of the output signal.

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

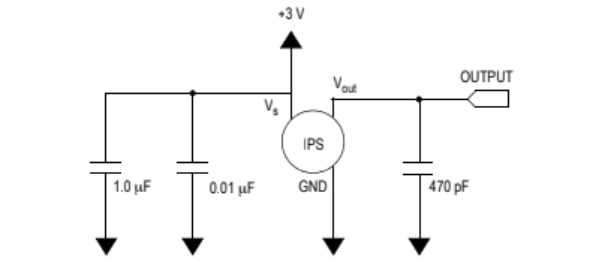
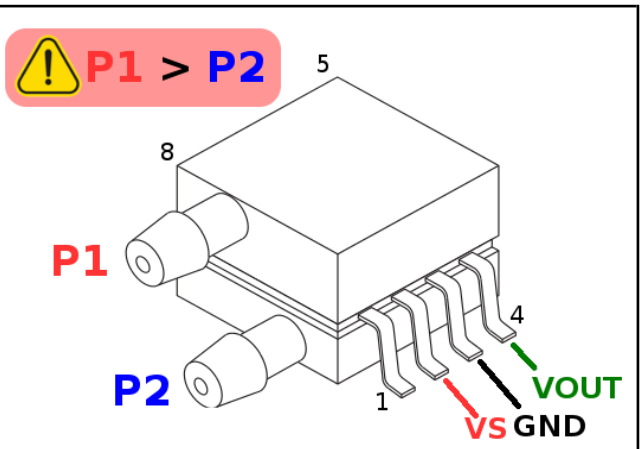
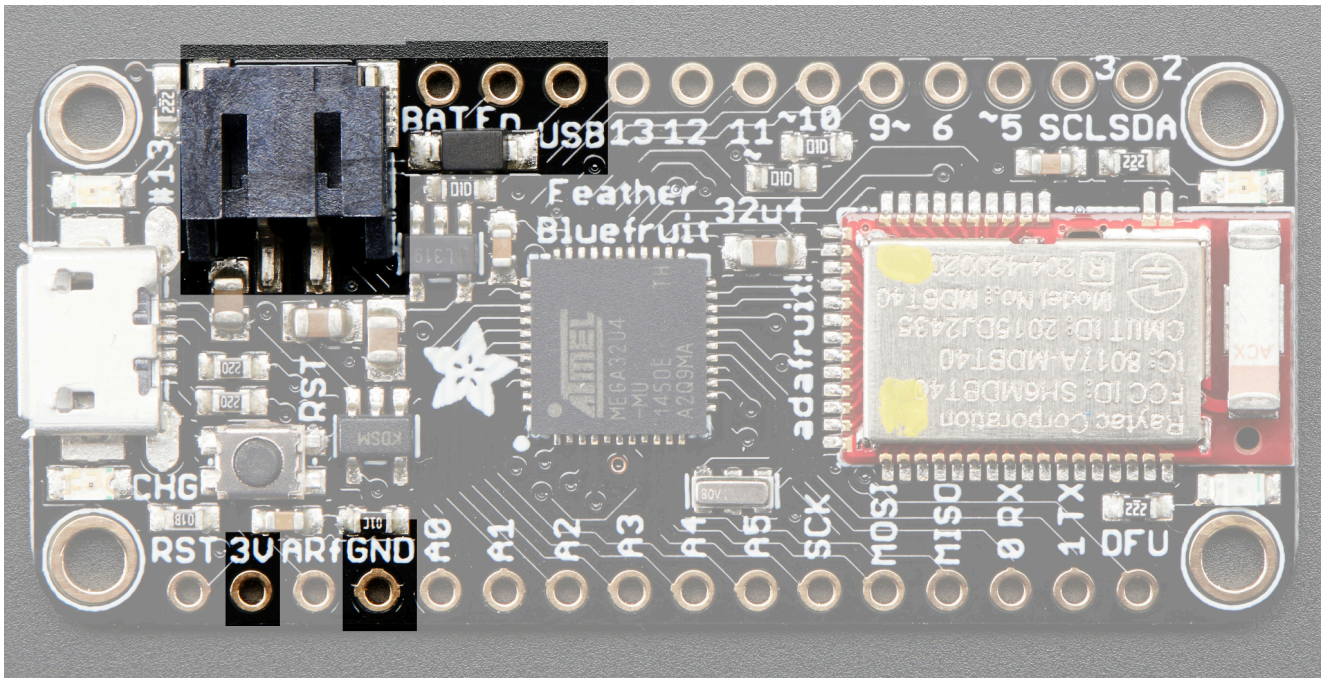


Figure 3. Recommended Power Supply Decoupling and Output Filtering
(For additional output filtering, please refer to Application Note AN1646.)



When used with the Adafruit Feather 32U4 BLE board the sensor is powered from the 3V Out pin, which is connected to the 3V regulator. See Feather power options. This pin supplies a 500mA peak regulator. While you can get 500mA from it, you can't do it continuously from 5V as it will overheat the regulator. But this is OK for the sensor, which only draws 7 to 10 mA.



See more about electronic connections on the Adafruit Feather 32u4 Bluefruit with MP3V5010DP sensor section.

Be aware that Pressure P1 > P2.

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

You can recognize Pin1 by the small notch.

The formula:

$$Vout = Vs \times (0.09 \times P + 0.08)$$

Vs : supply voltage 3V ± 0.3 V DC

P : the pressure difference in kPa

So, it is possible to measure the difference of pression P (in kPa) from the output voltage of the sensor (Vout)

$$P_{\text{in kPa}} = ((Vout / Vs) - 0.08) / 0.09$$

At the end, it is possible to measure the height of water (in mm) from the kPa pressure.

The 1000 coefficient is used to transform the meter unit to millimeters.

$$H2O\ mm = P * 0.1019 * 1000$$

Some examples: this video of water pressure measurement.

Implementation

See more in Configurations.

MPX10DP Pressure sensor differential

During 2024, the stakeholders of Breathing Games (project lead network organization), i.e. Sensorica (hardware design, trust economy and regulatory approval lead organization) and Sainte Justine Hospital (sponsor and early adopter) have adopted the Adafruit Feather 32u4 Bluefruit with the MP3V5010DP pressure sensor.

Message written in 2024.

Used by Jimmy at Sensorica lab in 2015, for the second version of the Breathing Games pressure sensor. This sensor was integrated in the first version of the so-called ***b-organic-3d-box*** pressure device version (see Gitlab repo), which was tested with games in 2015, as seen in this video. See this video documenting its fabrication.

This device was developed and enriched during four co-creation events (between Aug 2015 and Nov 2018), and presented at the Gathering Open Science Hardware in Chile, and at the European Design conference in Rome (2017). It was also featured in the documentary film A New Economy.

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions





This is a simple differential sensor designed as a "bridge" circuit (very common with pressure sensors), it is a raw sensor that has an output range of 0-50mV, it doesn't have the onboard amplification to properly drive the analog Input of a prototyping microcontroller board like the Arduino Nano with a reference of 5 volts.

ATTENTION !!
All microcontroller boards have a voltage reference pin (AREF) where you can change the reference voltage used by the analog inputs. Unfortunately the minimum is usually 1.1v
Don't use anything less than 0V or more than 5V for external reference voltage on the AREF pin! If you're using an external reference on the AREF pin, you must set the analog reference to EXTERNAL in the sketch before calling analogRead(). Otherwise, you will short together the active reference voltage (internally generated) and the AREF pin, possibly damaging your microcontroller board!

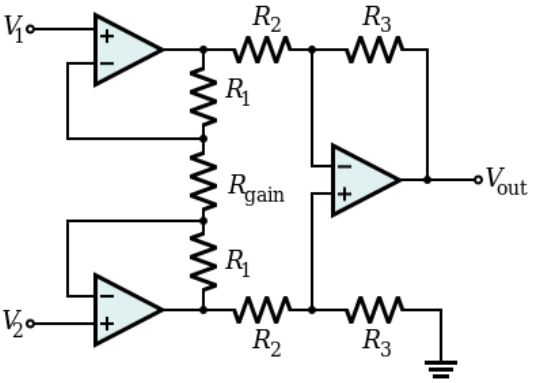
Since this sensor is not amplified, we have a few alternatives. There are many quick and dirty ways to multiply the voltage differential from the sensor pins but these introduce noise, drift and have a lack of proper gain adjustments for long term use. You can also lower the reference voltage on the nano's ADC as mentioned above but this will not increase the resolution of 10-bits (0-1024), just decrease the size of each voltage step.

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

The proper way is through an op-amp configuration known as an instrumentation amplifier.

This can be achieved by using standard op-amps arranged in this configuration. The two amplifiers on the left are the buffers and the one on the right is your standard differential output amplifier. Note that precision resistors must be used if going this route, as value mismatches cause common-mode gain as explained in the wiki. Here is a cool video from hackaday explaining them (well animated too!).



Fortunately there are IC's designed for this specific purpose. An IC instrumentation amplifier typically contains closely matched laser-trimmed resistors, and therefore offers excellent common-mode rejection. Examples include AD8221, MAX4194, LT1167 and INA128

The real issue you need to deal with that I see using this device is that it is a 3V device. Which means you need to have a 3V power source (not a big deal) and then scale it to send 0-5V to an Arduino board for example. This means that you will need to do some calculations (see the INA22 datasheet) to figure out the right gain resistor to get the full scale readings you want.

The two signals coming from the pressure transducer are usually attached to the inputs of an Instrumentation Amplifier (such as an INA122, a single supply device). This device would then connect to an analog input of the Arduino.

Forums showing how to connect the sensor to the Arduino

Connect MPX10DP and LM358 module

Papers

Control system for continuous positive airway pressure (electronics)

Control system design for a continuous positive airway pressure ventilator (mathematics)

The paper above proposes the following circuit.

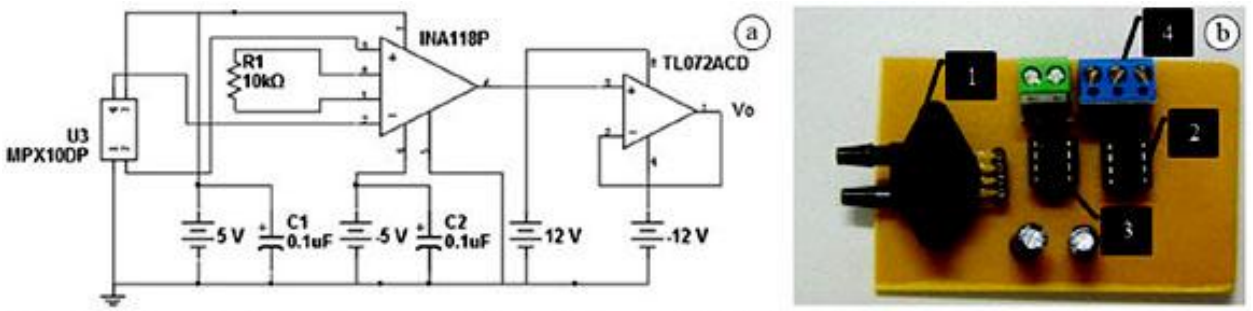
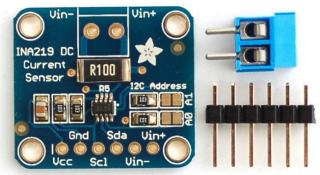


Figure 2. (a) Electronic circuit used to capture the signal with the sensor MPX10DP. (b) Circuit board (PCB): (1) - Sensor MPX10DP. (2) - Operational Amplifier TL072CN. (3) - Instrumentation Amplifier INA118P. (4) - Connectors to power supply.

Content on this document is *Creative Commons (BY SA)*.

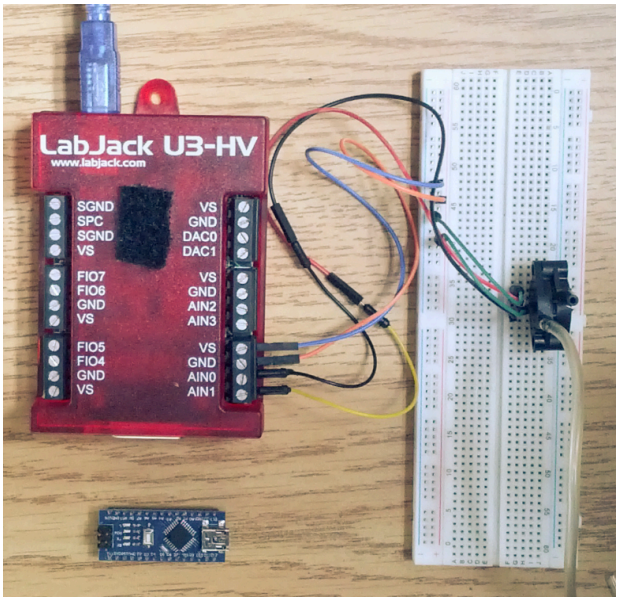
Log Contributions

Lackaly, life seems to be easier… Adafruit offers an open source circuit board that was designed for these types of sensors.

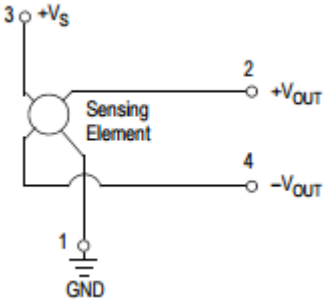


Testing the sensor with a DAQ

Tibi used a DAQ card, a LabJack and the sensor to verify if the sensor works properly, and to understand how it operates.



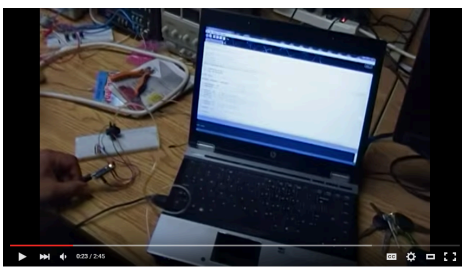
Connections to the LabJack:
pin 1 = gND ; pin2 = +V1 ; pin3 = 5V ; pin4 = -V2



Testing the sensor with an Arduino Nano

We tried a direct connection between the sensor and an Arduino nano board. Since the output of the sensor is very low, max 50mV, we could only get up to 5 points over its entire range. Jim connected the Arduino to a game and drove it by blowing in a tube connected to the sensor.

Link to the Arduino sketch



click on the image to see the video

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

Implementation

See configurations

Amplification: using INA141P Instr. Amp

This was used to amplify the Pressure sensor differential MPX10DP with the Arduino Nano board.



Open Datasheet

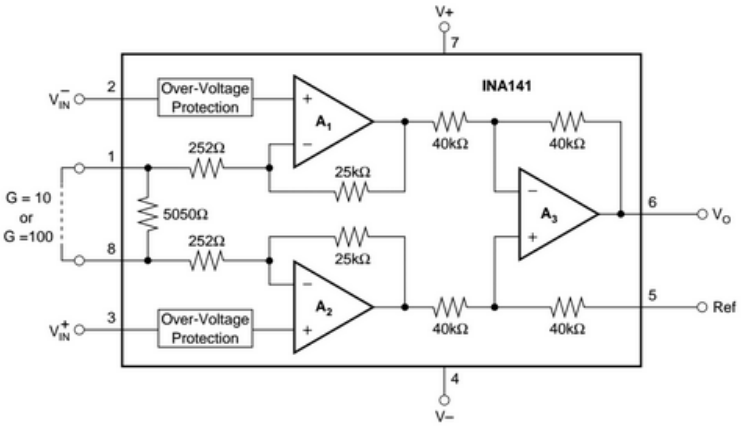
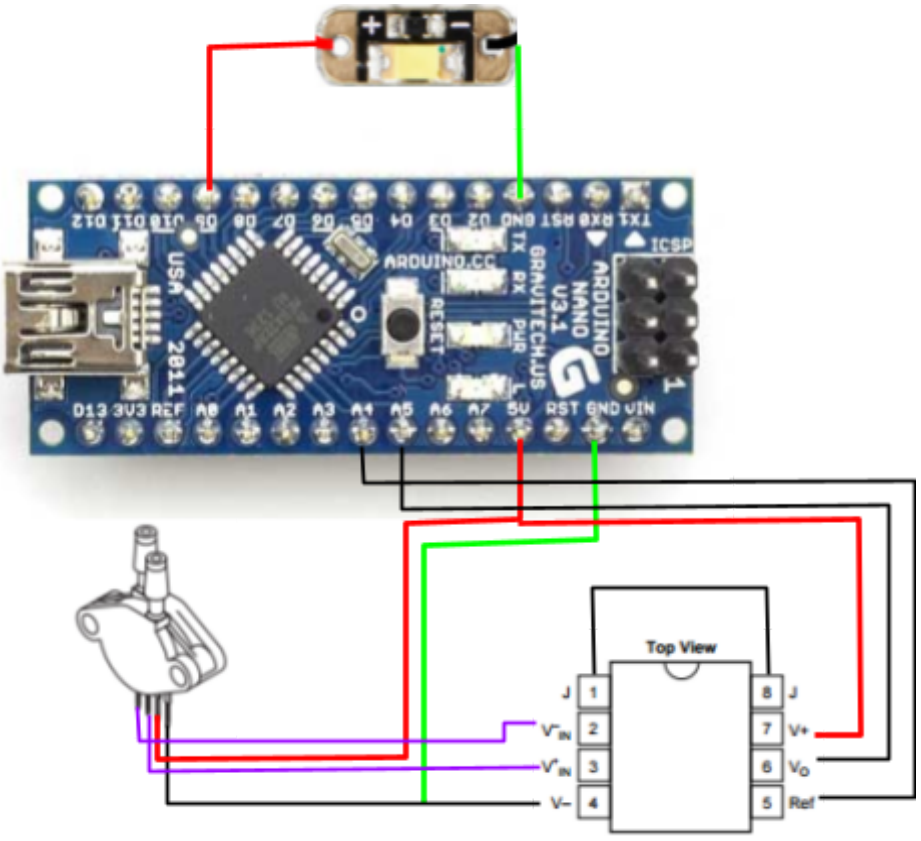


Diagram of connections with Arduino Nano



Content on this document is *Creative Commons (BY SA)*.

Log Contributions

For the LED, we need to insert the LED control code in the rest of the code. See Arduino tutorial page for dimming LED for more.

Implementation

This amplifier was used with the MPX10DP differential pressure sensor and an Arduino Nano in the first Breathing Games prototype as seen on this video. See more in Configurations.

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

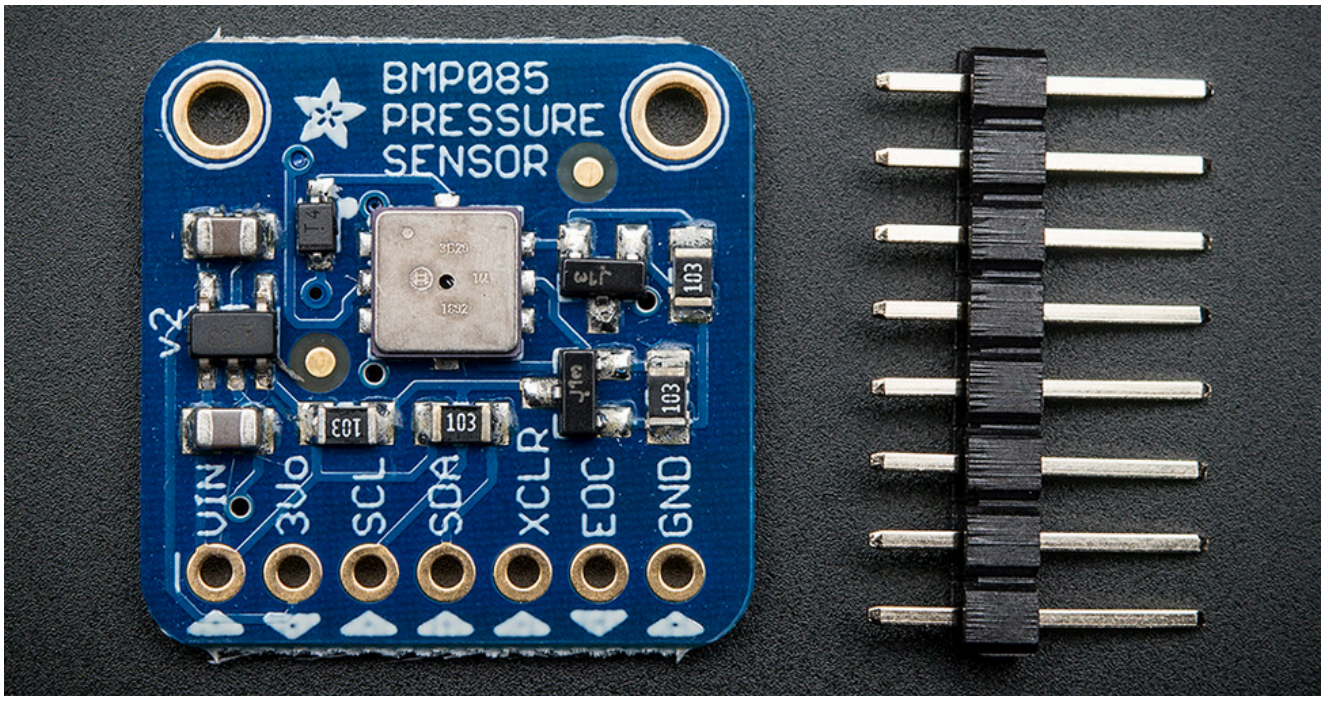
BMP085 Barometric Pressure/Temp./Altitude Sensor- 5V

During 2024, the stakeholders of Breathing Games (project lead network organization), i.e. Sensorica (hardware design, trust economy and regulatory approval lead organization) and Sainte Justine Hospital (sponsor and early adopter) have adopted the Adafruit Feather 32u4 Bluefruit with the MP3V5010DP pressure sensor.

Message written in 2024.

This sensor was used in the first device built by Breathing Games, before the partnership with Sensorica in 2015, when the development / improvement of the PEP pressure sensor was taken by the Sensorica network.

The BMP085 sensor from Adafruit - open Adafruit page.



We do not have the electronic schematics of this device… Here's a picture of how it looked when it was demonstrated at Sainte Justine Hospital.

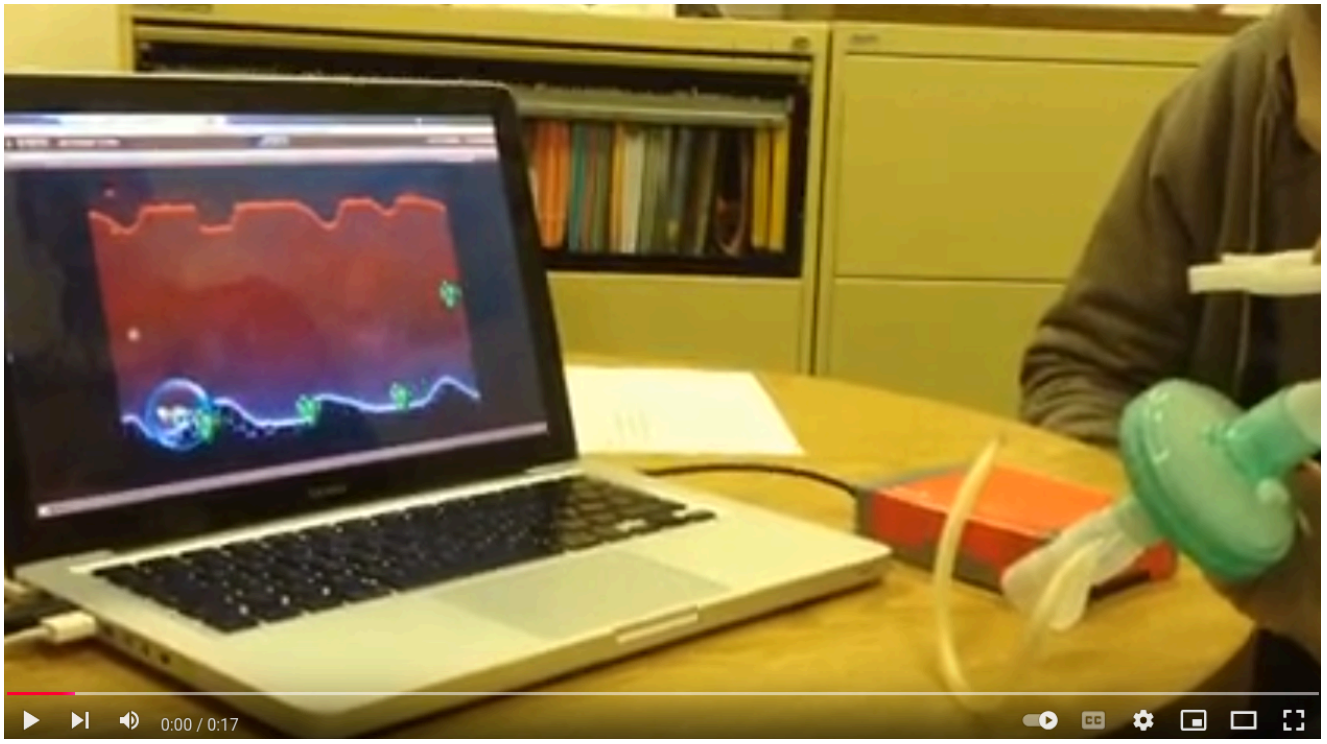
Content on this document is *Creative Commons (BY SA)*.

Log Contributions



See Gitlab repository for the arduino code.

See video published by Breathing Games during demonstration at Sainte Justine Hospital in 2014.



Other good candidates

https://allsensors.com/en/product-listing?family=dlc

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

Communication

Adafruit Feather 32u4 Bluefruit LE - USB cable

During 2024, the stakeholders of Breathing Games (project lead network organization), i.e. Sensorica (hardware design, trust economy and regulatory approval lead organization) and Sainte Justine Hospital (sponsor and early adopter) have adopted the Bluetooth BLE mode of communication and have decided to move games to mobile platforms. See the BLE communication section. The USB connection is not abandoned, it will be kept as an option.

Message written in 2024.

Using with Arduino IDE

See source of the info below.

Since the Feather 32u4 uses an ATmega32u4 chip running at 8 MHz, you can pretty easily get it working with the **Arduino IDE**. Many libraries (including the popular ones like NeoPixels and display) work great with the '32u4 and 8 MHz clock speed.

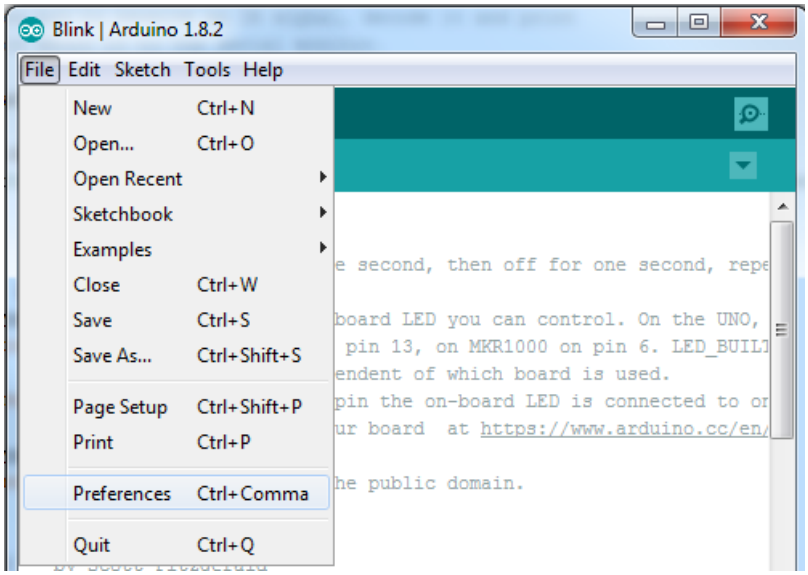
The first thing you will need to do is to download the latest release of the Arduino IDE. You will need to be using **version 1.8** or higher. To install Arduino IDE see Help for Arduino, for Linux (or navigate to other pages for other operating systems). The version Arduino IDE 1.8.19 seems to work better for our application with the Adafruit Feather boards that we are using.

After you have downloaded and installed **the latest version of Arduino IDE**, you will need to start the IDE and navigate to the **Preferences** menu. You can access it from the **File** menu.

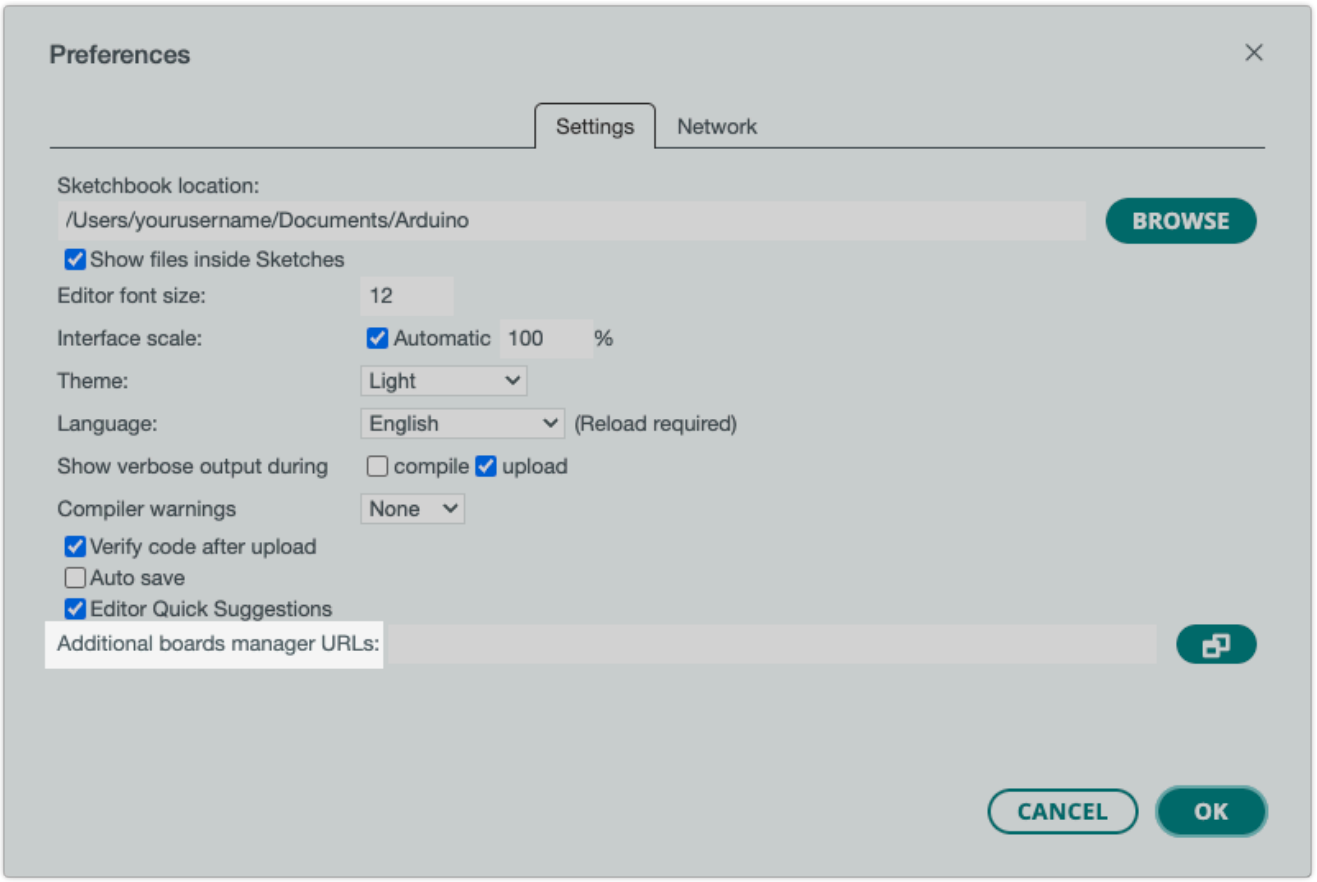
NOTE: The following comes from this source on Adafruit and this source on Arduino.

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions



A dialog will pop up just like the one shown below.



We will be adding a URL to the new **Additional Boards Manager URLs** option. The list of URLs is comma separated, and *you will only have to add each URL once.* New Adafruit boards and updates

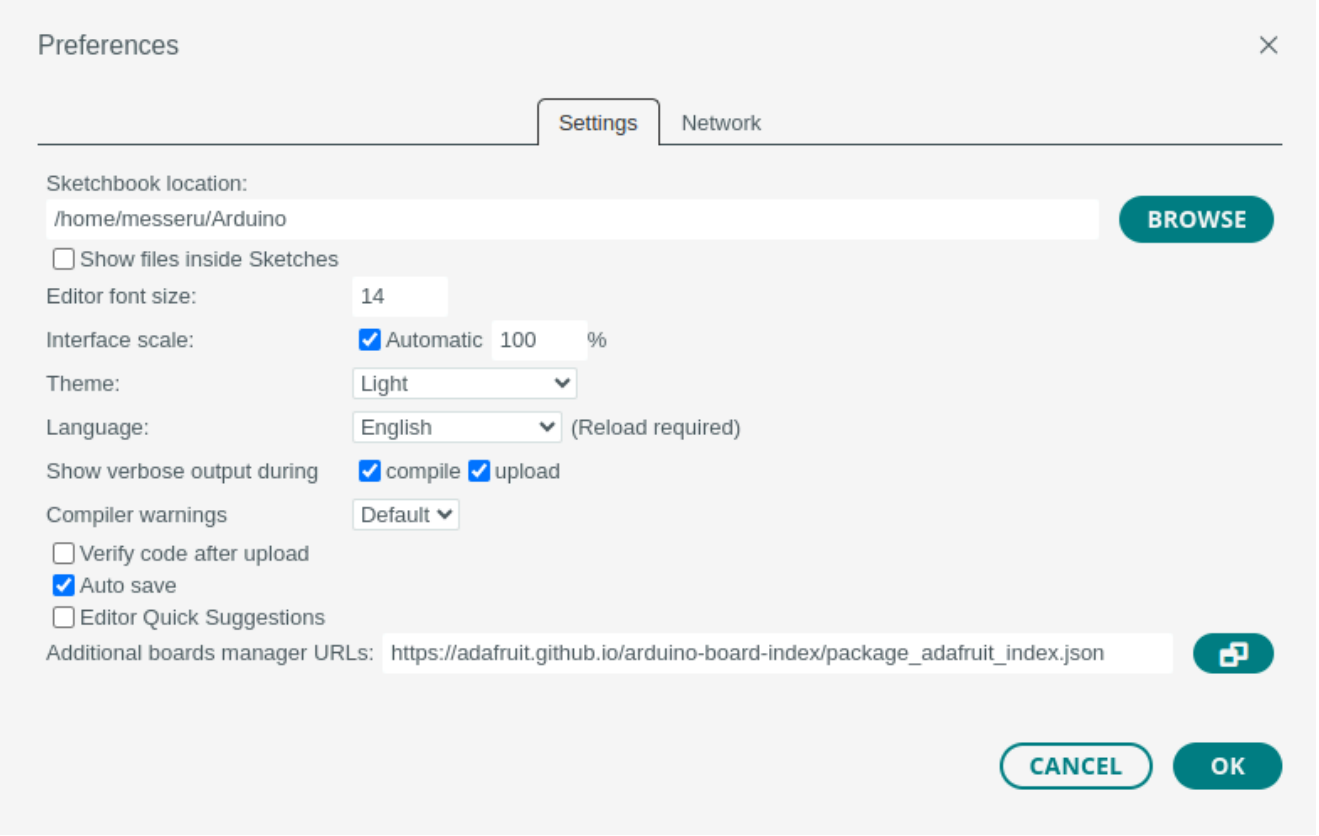
Content on this document is *Creative Commons (BY SA)*.

Log Contributions

to existing boards will automatically be picked up by the Board Manager each time it is opened. The URLs point to index files that the Board Manager uses to build the list of available & installed boards.

To find the most up to date list of URLs you can add, you can visit the list of third party board URLs on the Arduino IDE wiki. We will only need to add one URL to the IDE in this example, but ***you can add multiple URLS by separating them with commas***. Copy and paste the link below into the **Additional Boards Manager URLs** option in the Arduino IDE preferences.

https://adafruit.github.io/arduino-board-index/package_adafruit_index.json



Here's a short description of each of the Adafruit supplied packages that will be available in the Board Manager when you add the URL:

- **Adafruit AVR Boards** - Includes support for Flora, Gemma, Feather 32u4, ItsyBitsy 32u4, Trinket, & Trinket Pro.
- **Adafruit SAMD Boards** - Includes support for Feather M0 and M4, Metro M0 and M4, ItsyBitsy M0 and M4, Circuit Playground Express, Gemma M0 and Trinket M0
- **Arduino Leonardo & Micro MIDI-USB** - This adds MIDI over USB support for the Flora, Feather 32u4, Micro and Leonardo using the arcore project.

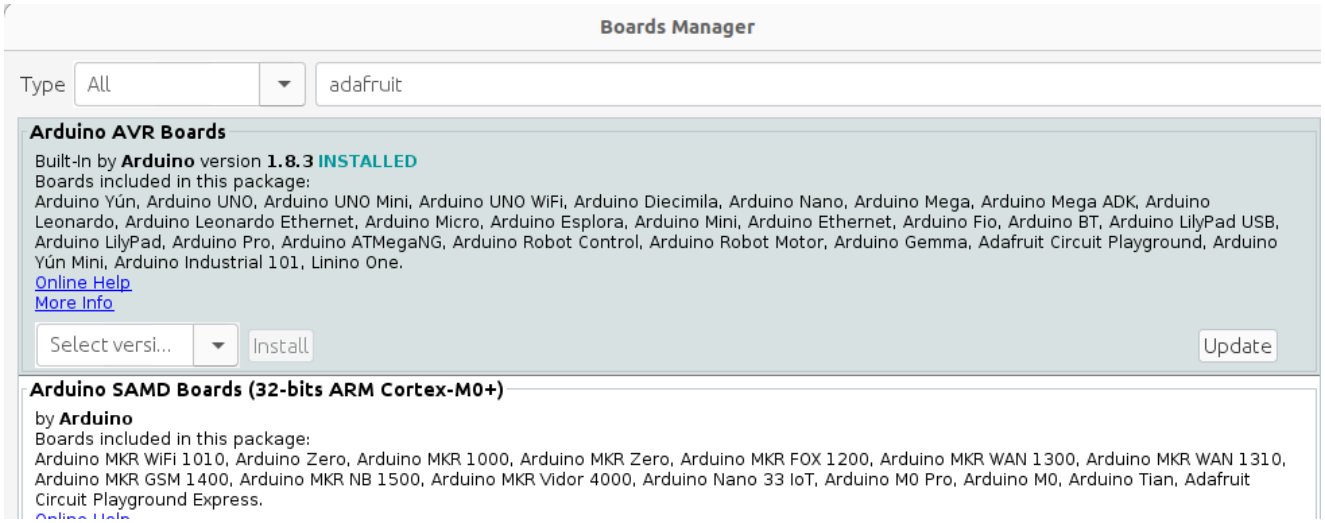
If you have multiple boards you want to support, say ESP8266 and Adafruit, have both URLs in the text box separated by a comma (,)

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

Once done click **OK** to save the new preference settings. Next we will look at installing boards with the Board Manager.

Now that you have added the appropriate URLs to the Arduino IDE preferences, you can open the *Boards Manager* by navigating to the **Tools->Board** / ***Boards Manager*** menu. Note that for Arduino ADE version 2.3.3 the board installed automatically, but this may not be the case for everyone. In that case, once the ***Board Manager*** opens, click on the category drop down menu on the top left hand side of the window and select **Contributed**. You will then be able to select and install the boards supplied by the URLs added to the preferences. In the example below, we are installing support for ***Adafruit AVR Boards***, but the same applies to all boards installed with the Board Manager.

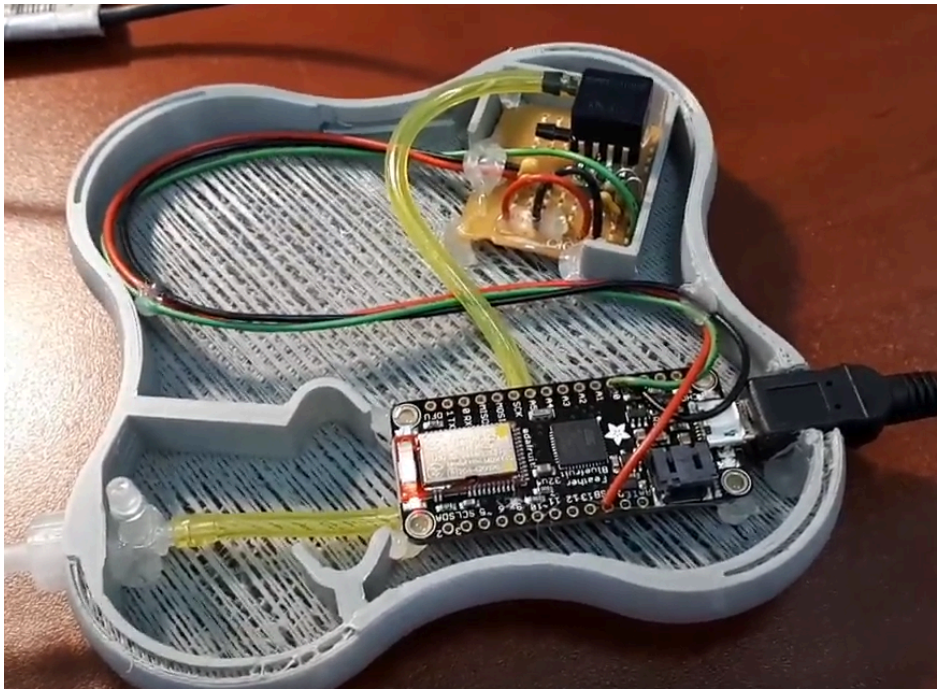


Next, **quit and reopen the Arduino IDE** to ensure that all of the boards are properly installed. You should now be able to select and upload to the new boards listed in the **Tools->Board** menu.

This microcontroller board was used in the Breathing Games device model b (organic box), built for Saint Justine, you can see in this video and this video. Also used, the MPXV5010DP differential pressure sensor, with onboard amplification. See more in Configurations.

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions



You interact with the Feather using Arduino IDE. Connect a USB cable with micro into the Feather board and the other side into a computer USB port. Make sure that you have the proper version of the Arduino IDE that corresponds with the Feather microcontroller boards that we are using, you may have difficulties connecting the computer to the Feather. In the ADI settings **Tools / Board:...** make sure that you choose "***Adafruit Feather 32u4***". If you don't have that option you need to install the Adafruit library, see more on the Feather section. Load the firmware into the device (see Software/Firmware for USB cable operation section).

If while loading a sketch / firmware to the device you have an error message that contains the term avrdude you may need help with that, see **help doc**.

It also happened that the USB connection didn't work on a board, after cleaning with 99% isopropanol the problem was fixed. Sometimes there are conductive particles that get stuck on the board during the fabrication / prototyping, and may even get stuck under the floss while soldering wires.

Adafruit Feather 32u4 Bluefruit LE with bluetooth BLE - wireless

During 2024, the stakeholders of Breathing Games (project lead network organization), i.e. Sensorica (hardware design, trust economy and regulatory approval lead organization) and Sainte Justine Hospital (sponsor and early adopter) have adopted the Bluetooth BLE mode of communication and have decided to move games to mobile platforms.

Message written in 2024.

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

Bluetooth Low Energy is a 2.4GHz spectrum wireless protocol. In particular, it's the only wireless protocol that you can use with iOS without needing special certification and it's supported by all modern smartphones. This makes it excellent for use in portable projects that will make use of an iOS or Android phone or tablet. It also is supported in Mac OS X and Windows 8+

Like in the previous section, you need to have your Arduino IDE installed and configured to talk to the Feather 32u4 Bluefruit LE, but in this case the IDE configuration needs to be different. Install the ***Adafruit nRF51 BLE*** Library (Note: the following info comes from this reference): download the Adafruit BLE library for the nRF51 based modules named ***Adafruit_BluefruitLE_nRF51***. You can check out the code here at github, but it's likely easier to just download via the Arduino library manager.

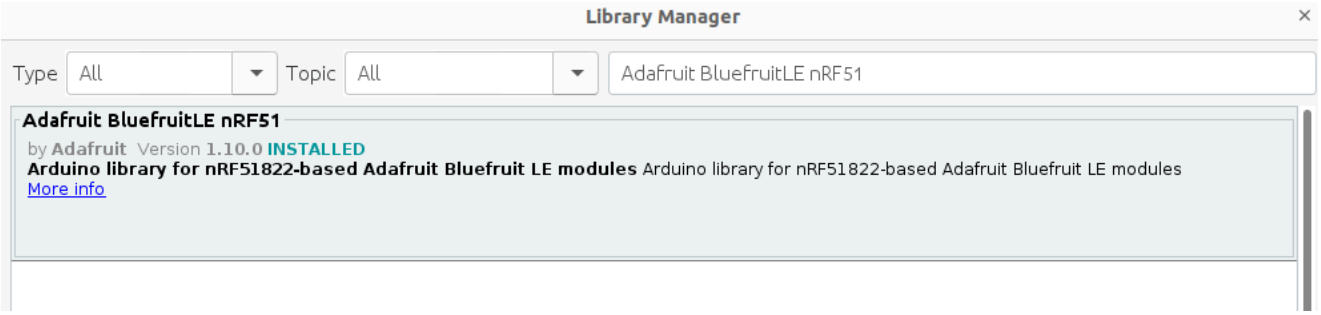
If you don't already have Arduino IDE installed

1. Download and install the Arduino IDE (At least v1.6.12)
2. install Arduino IDE see Help for Arduino, for Linux (or navigate to other pages for other operating systems). The version Arduino IDE 1.8.19 seems to work better for our application with the Adafruit Feather boards that we are using.
3. Start the Arduino IDE
4. Go into Preferences
5. Add https://adafruit.github.io/arduino-board-index/package_adafruit_index.json as an 'Additional Board Manager URL'
6. Restart the Arduino IDE

Then, if IDE is successfully installed

1. Open the Boards Manager from the **Sketch -> Include Libraries / Manage libraries** and install '**Adafruit BluefruitLE nRF51**'

Install the **Adafruit BluefruitLE nRF51** library.



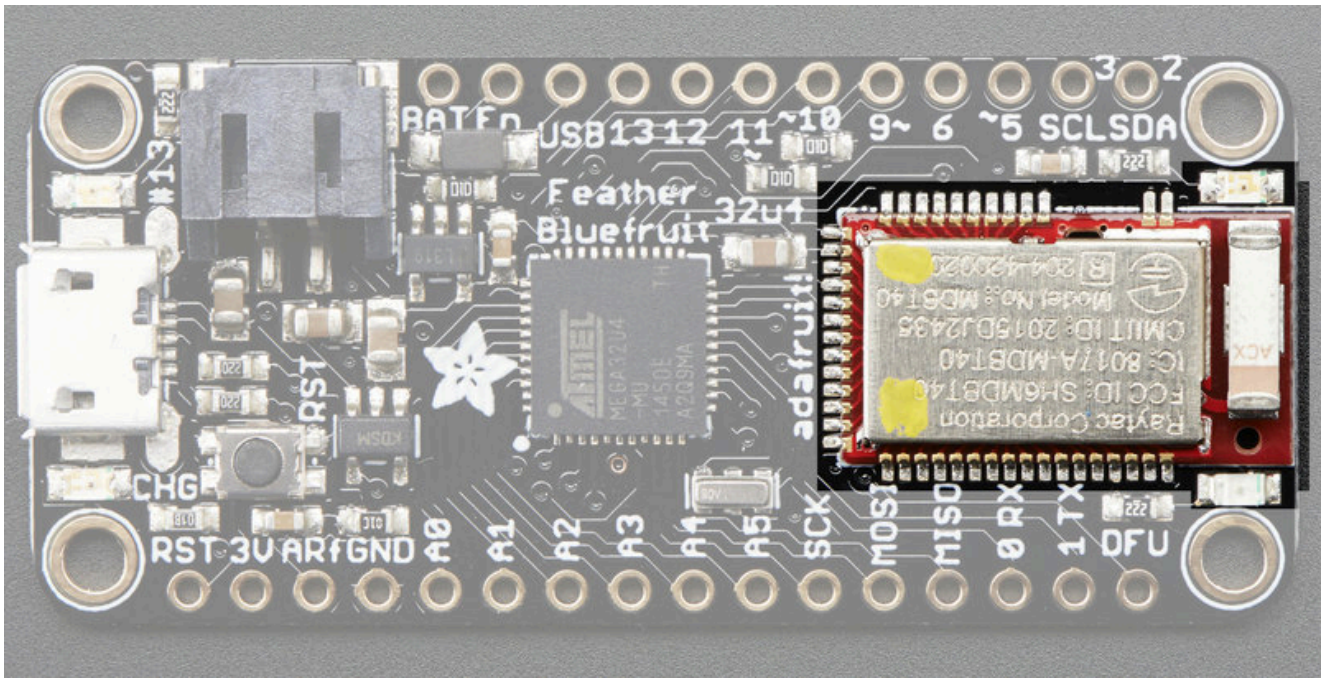
Github repo for Adafruit nRF51. There is also a great tutorial on Arduino library installation.

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

Do not install the Board or Library for nRF52 (instead of nRF51), because they are incompatible. You'll get an error during compilation that is hard to decode: error: *conflicting declaration 'typedef uint32_t u32_t' · Issue #3771 · esp8266/Arduino*

The Bluefruit LE module is an nRF51822 chipset from Nordic, programmed with multi-function code that can do quite a lot! For most people, they'll be very happy to use the standard Nordic UART RX/TX connection profile. In this profile, the Bluefruit acts as a data pipe that can 'transparently' transmit back and forth from your iOS or Android device. You can use our iOS App or Android App, or write your own to communicate with the UART service.



The board is capable of much more than just sending strings over the air! Thanks to an easy to learn AT command set, you have full control over how the device behaves, including the ability to define and manipulate your own GATT Services and Characteristics, or change the way that the device advertises itself for other Bluetooth Low Energy devices to see. You can also use the AT commands to query the die temperature, check the battery voltage, and more, check the connection RSSI or MAC address, and tons more. Really, way too long to list here!

Use the Bluefruit App to get your project started

Using our Bluefruit iOS App or Android App, you can quickly get your project prototyped by using your iOS or Android phone/tablet as a controller. We have a color picker, quaternion/accelerometer/gyro/magnetometer or location (GPS), and an 8-button control game pad. This data can be read over BLE and piped into the ATmega32u4 chip for processing & control.

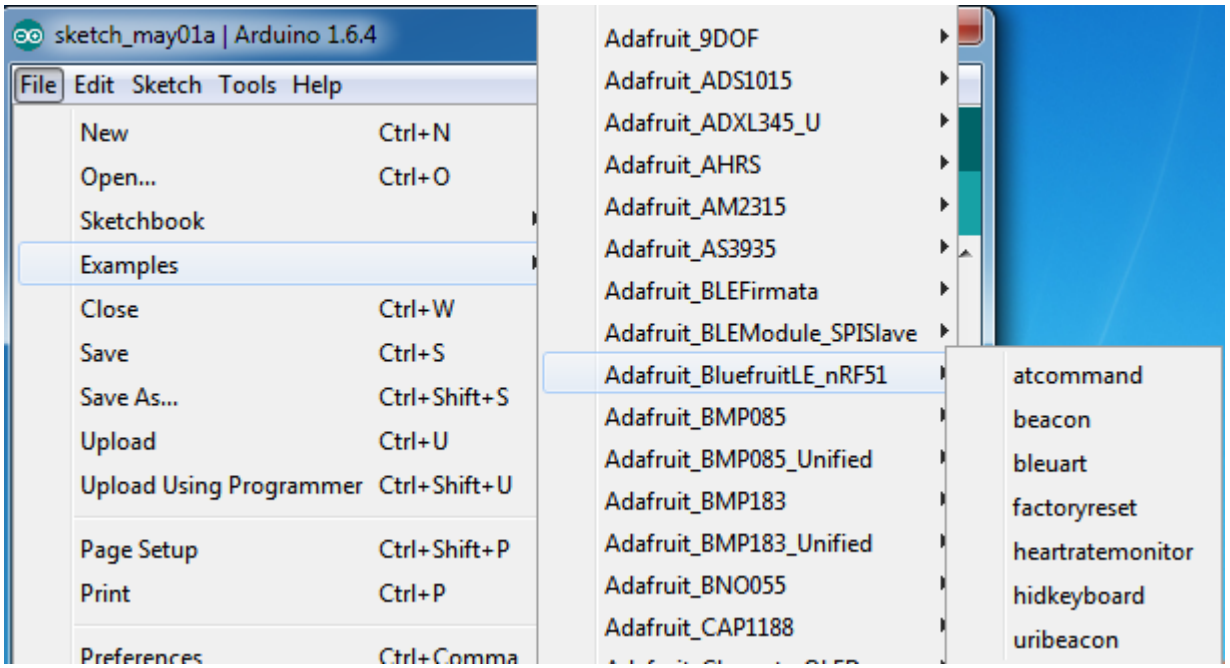
Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

Sending data to android using Command Mode

We are going to use an existing example to send sensor data from the Pressure Device to the Android device, through the Android App, using the UART function.

Check that you see the library folder with examples:



To see if you can communicate with your Feather device, use the BLEUart / bleuart_cmdmode example to send text from the Pressure Device to the Computer (using the Serial Monitor on Arduino ADE) and from the Computer to the Device.

If everything works between the computer and the Pressure Device, now we can test communication with an Android device. We can now run a test to send an analog read from our sensor to the Android App.

Open a new sketch and copy this code below into it. Upload to the Pressure Device using the Arduino IDE. Make sure that the Arduino IDE is properly configured, **Board** (Adafruit Feather 32U4) and **Programmer** (AVRISP mkII) - see other instructions above.

Sketch for sending data to the Android App

```
/*********************************************************************
This is an example for our nRF51822 based Bluefruit LE modules

Pick one up today in the adafruit shop!
```

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

```
Adafruit invests time and resources providing this open source code, please support Adafruit
and open-source hardware by purchasing products from Adafruit!

MIT license, check LICENSE for more information

All text above, and the splash screen below must be included in any redistribution

*****************************************************************/

#include <Arduino.h>

#include <SPI.h>

#include "Adafruit_BLE.h"

#include "Adafruit_BluefruitLE_SPI.h"

#include "Adafruit_BluefruitLE_UART.h"

#include "BluefruitConfig.h"

//Analog Read Pressure

int sensorPin = A1;     // select the input pin for the pressure sensor

int sensorValue = 0;    // variable to store the value coming from the sensor

#if SOFTWARE_SERIAL_AVAILABLE

 #include <SoftwareSerial.h>

#endif

/*=========================================================================

   APPLICATION SETTINGS

    FACTORYRESET_ENABLE       Perform a factory reset when running this sketch

                              Enabling this will put your Bluefruit LE module

                             in a 'known good' state and clear any config

                             data set in previous sketches or projects, so

                              running this at least once is a good idea.

                             When deploying your project, however, you will

                             want to disable factory reset by setting this

                             value to 0.  If you are making changes to your

                              Bluefruit LE device via AT commands, and those

                             changes aren't persisting across resets, this

                             is the reason why.  Factory reset will erase

                             the non-volatile memory where config data is

                             stored, setting it back to factory default

                             values.

                              Some sketches that require you to bond to a

                             central device (HID mouse, keyboard, etc.)

                             won't work at all with this feature enabled

                             since the factory reset will clear all of the

                             bonding data stored on the chip, meaning the
```

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

```
                              central device won't be able to reconnect.
   MINIMUM_FIRMWARE_VERSION   Minimum firmware version to have some new features
   MODE_LED_BEHAVIOUR         LED activity, valid options are
                              "DISABLE" or "MODE" or "BLEUART" or
                              "HWUART"  or "SPI"  or "MANUAL"
   ------------------------------------------------------------------------*/
   #define FACTORYRESET_ENABLE         1
   #define MINIMUM_FIRMWARE_VERSION    "0.6.6"
   #define MODE_LED_BEHAVIOUR          "MODE"
/*=========================================================================*/


// Create the bluefruit object, either software serial...uncomment these lines
/*
SoftwareSerial         bluefruitSS         =         SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN,
BLUEFRUIT_SWUART_RXD_PIN);
Adafruit_BluefruitLE_UART ble(bluefruitSS, BLUEFRUIT_UART_MODE_PIN,
                      BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);
*/
/* ...or hardware serial, which does not need the RTS/CTS pins. Uncomment this line */
// Adafruit_BluefruitLE_UART ble(Serial1, BLUEFRUIT_UART_MODE_PIN);
/* ...hardware SPI, using SCK/MOSI/MISO hardware SPI pins and then user selected CS/IRQ/RST
*/
Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS, BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);
/*  ...software  SPI,  using  SCK/MOSI/MISO  user-defined  SPI  pins  and  then  user  selected
CS/IRQ/RST */
//Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_SCK, BLUEFRUIT_SPI_MISO,
//                             BLUEFRUIT_SPI_MOSI, BLUEFRUIT_SPI_CS,
//                             BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);
// A small helper
void error(const __FlashStringHelper*err) {
 Serial.println(err);
 while (1);
}
/**************************************************************************/
/*!
   @brief  Sets up the HW an the BLE module (this function is called
           automatically on startup)
*/
/**************************************************************************/
```

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

```
void setup(void)
{
  while (!Serial);  // required for Flora & Micro
 delay(500);

 Serial.begin(115200);
 Serial.println(F("Adafruit Bluefruit Command Mode Example"));
 Serial.println(F("---------------------------------------"));
 /* Initialise the module */
 Serial.print(F("Initialising the Bluefruit LE module: "));
 if ( !ble.begin(VERBOSE_MODE) )
 {
   error(F("Couldn't find Bluefruit, make sure it's in CoMmanD mode & check wiring?"));
 }
 Serial.println( F("OK!") );
 if ( FACTORYRESET_ENABLE )
 {
   /* Perform a factory reset to make sure everything is in a known state */
   Serial.println(F("Performing a factory reset: "));
   if ( ! ble.factoryReset() ){
     error(F("Couldn't factory reset"));
   }
 }
 /* Disable command echo from Bluefruit */
 ble.echo(false);
 Serial.println("Requesting Bluefruit info:");
 /* Print Bluefruit information */
 ble.info();
 Serial.println(F("Please use Adafruit Bluefruit LE app to connect in UART mode"));
 Serial.println(F("Then Enter characters to send to Bluefruit"));
 Serial.println();
 ble.verbose(false);  // debug info is a little annoying after this point!
 /* Wait for connection */
 while (! ble.isConnected()) {
     delay(500);
 }
 // LED Activity command is only supported from 0.6.6
 if ( ble.isVersionAtLeast(MINIMUM_FIRMWARE_VERSION) )
```

Content on this document is *Creative Commons (BY SA)*.

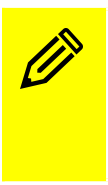
Log Contributions

```
 {
   // Change Mode LED Activity
   Serial.println(F("******************************"));
   Serial.println(F("Change LED activity to " MODE_LED_BEHAVIOUR));
   ble.sendCommandCheckOK("AT+HWModeLED=" MODE_LED_BEHAVIOUR);
   Serial.println(F("******************************"));
 }
}
/**************************************************************************/
/*!
   @brief  Constantly poll for new command or response data
*/
/**************************************************************************/
void loop(void)
{
 // read the value from the sensor:
sensorValue = analogRead(sensorPin);
delay(10);
// Send characters to Bluefruit
   Serial.print("[Send] ");
   Serial.println(sensorValue);
   ble.print("AT+BLEUARTTX=");
   ble.println(sensorValue);
 // Check for user input
 char inputs[BUFSIZE+1];
 if ( getUserInput(inputs, BUFSIZE) )
 {
   // Send characters to Bluefruit
   Serial.print("[Send] ");
   Serial.println(inputs);
   ble.print("AT+BLEUARTTX=");
   ble.println(inputs);
   // check response stastus
   if (! ble.waitForOK() ) {
     Serial.println(F("Failed to send?"));
   }
 }
 // Check for incoming characters from Bluefruit
```

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

```
 ble.println("AT+BLEUARTRX");
 ble.readline();
 if (strcmp(ble.buffer, "OK") == 0) {
   // no data
   return;
 }
 // Some data was found, its in the buffer
 Serial.print(F("[Recv] ")); Serial.println(ble.buffer);
 ble.waitForOK();
}
/**************************************************************************/
/*!
   @brief  Checks for user input (via the Serial Monitor)
*/
/**************************************************************************/
bool getUserInput(char buffer[], uint8_t maxSize)
{
 // timeout in 100 milliseconds
 TimeoutTimer timeout(100);
 memset(buffer, 0, maxSize);
 while( (!Serial.available()) && !timeout.expired() ) { delay(1); }
 if ( timeout.expired() ) return false;
 delay(2);
 uint8_t count=0;
 do
 {
   count += Serial.readBytes(buffer+count, maxSize);
   delay(2);
 } while( (count < maxSize) && (Serial.available()) );
 return true;
}
```



NOTE: Tibi tested the sketch, modified to read the analog sensor value and send it to the UART feature on the Android device via the BLE. It does send data to the android app, but only works if connected to a computer.

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

So when the Pressure Sensor is NOT connected to the computer, and the serial monitor of the Arduino IDE is not on, i.t. When the Pressure Sensor gets the power from a 5V power supply, the Pressure sensor and the Android device cannot establish proper connection.

See the Plotter example below, which doesn't exhibit this problem. Compare both sketches to see what causes this misbehavior.

Some learning

Another big difference between the two versions of Bluetooth is the way it is used to transfer data. Bluetooth Classic uses something similar to Serial Communication, while Bluetooth Low Energy uses a client-server model, where it employs the GATT (Generic Attribute Profile) to structure data.

Data can be sent in several ways:

- UART mode: Allows transparent UART data transfer between the Bluefruit device and the app. Data sent to the RXI pin of the Bluefruit module is transmitted to the app, and data from the app is sent out the TXO pin.
- MQTT mode: Allows sending and receiving data using the MQTT protocol. This can be used to send sensor data from a Bluefruit device to the app and then to Adafruit IO cloud service.
- Plotter mode: Visualizes data received from the Bluefruit device on a graph in the app.

For MQTT mode, the app can be configured to:

- Publish data to Adafruit IO feeds (e.g. "rx" and "tx" feeds)
- Subscribe to Adafruit IO feeds to receive data (e.g. "output" feed)

The "UART" being referred to is the Nordic UART GATT Service. This is **not a serial port or USB connection**. It is a high level BLE protocol that *simulates* a serial port via the Bluetooth connection. Yes, you can use it *like* a serial port, but the point I am making is there is no GATT profile for serial data transfer. Every module manufacturer has their own way of doing it, so it's down to your application to implement that method (usually using libraries provided by the manufacturer). You can't present BLE as a serial port (COM3, /dev/rfcomm0 etc). There is no serial - there is only serial emulation and it's specific to the device. BLE is intended for sending short packets of data back and forth using minimal power. Many manufacturers have chosen to create a profile that can be used in a similar way to a serial port (in that you send data through a virtual "pipe"), however you require special software to interface with it. And, since there is no standard, and manufacturers just don't talk to each other, the software is specific to the device you are talking to. If you are directly programming a BLE chip itself with your own firmware there is nothing to stop you from implementing your own protocol for serial data transfer and writing an application, driver, daemon, whatever, for communicating with that custom firmware. People use it for what it is intended - connecting sensors

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

and HID devices to mobile phones etc. That's why all the GATT profiles are for such things as heart rate, blood sugar levels, cycle speed, etc.

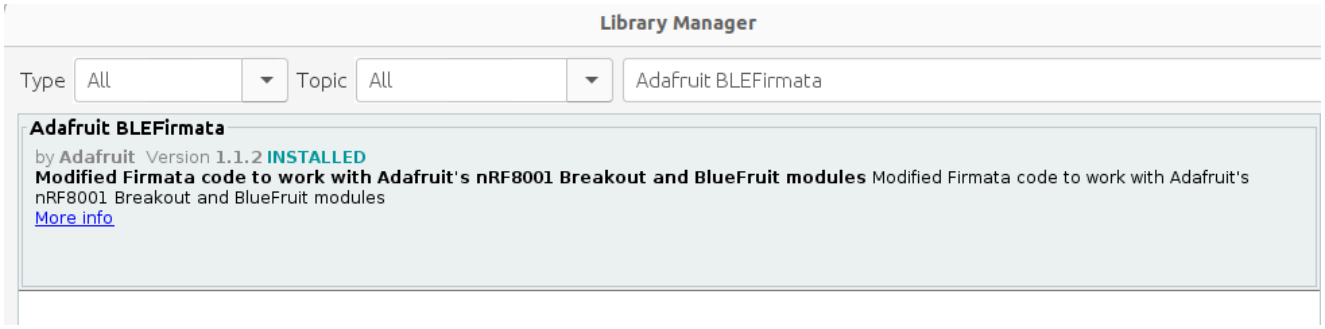
For programming the game, perhaps we need the following adafruit / Adafruit_Android_BLE_UART. This shows how one can build Android apps that can communicate with the Feather via BLE.

Read and Control the Pins configuration of the Feather from Android device

BLE Pin I/O - reference

Firmata is a generic protocol for communicating with microcontrollers from software on a host computer. The protocol can be implemented in firmware on any microcontroller architecture as well as software on any host computer software package. The Arduino repository for Firmata is a library for Arduino and Arduino-compatible devices. This can allow you to send and receive data, controll the board's pins such as setting the GPIO outputs and inputs, PWM output, analog reads, etc.... More here.

Install the ***Adafruit BLEFirmata*** library from **Sketch / Include Libraries / Manage libraries**.



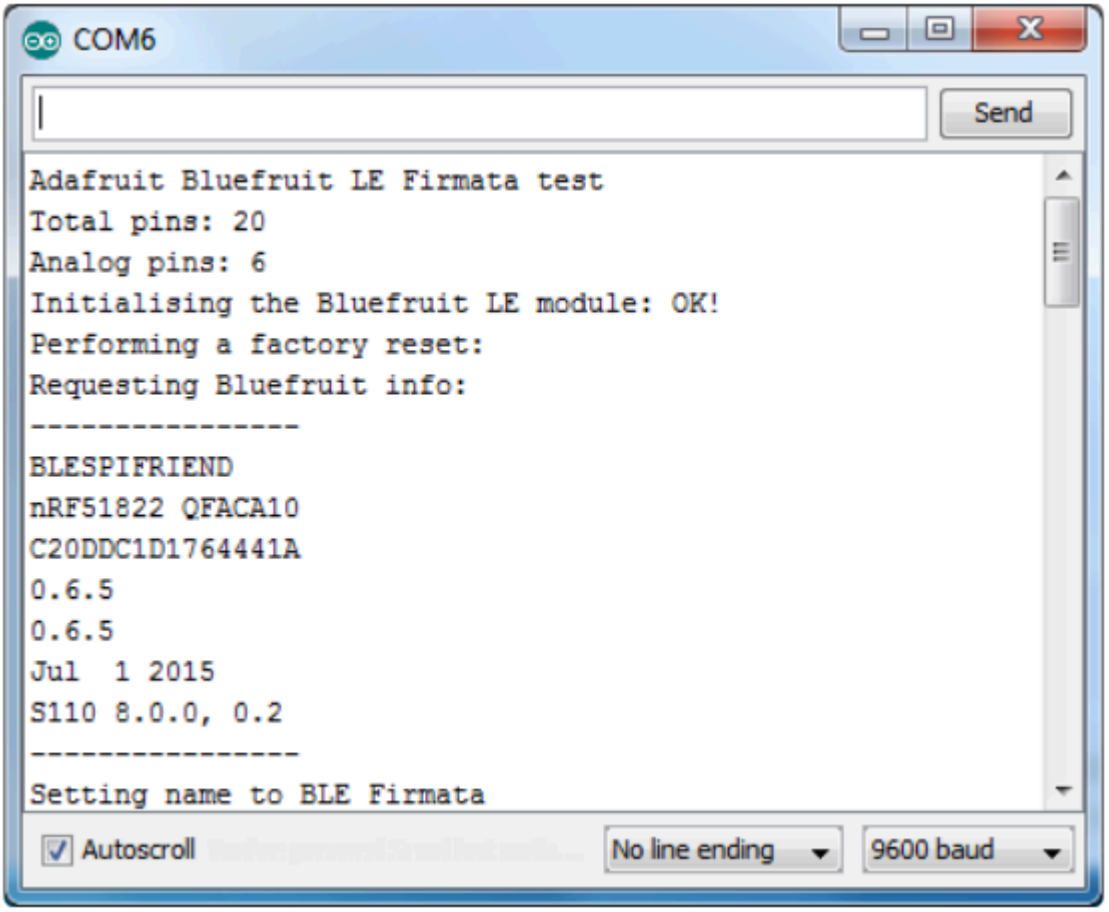
Open Sketch and Configure Device

Restart the IDE and load the ***BluefruitLE_nrf51822*** sketch from **Files / Examples / Adafruit BLEFirmata / *BluefruitLE_nrf51822*.**

Connect the Adafruit Feather - based pressure sensor, compile and upload this sketch, open the **Serial Monitor** from **Tools**

Content on this document is *Creative Commons (BY SA)*.

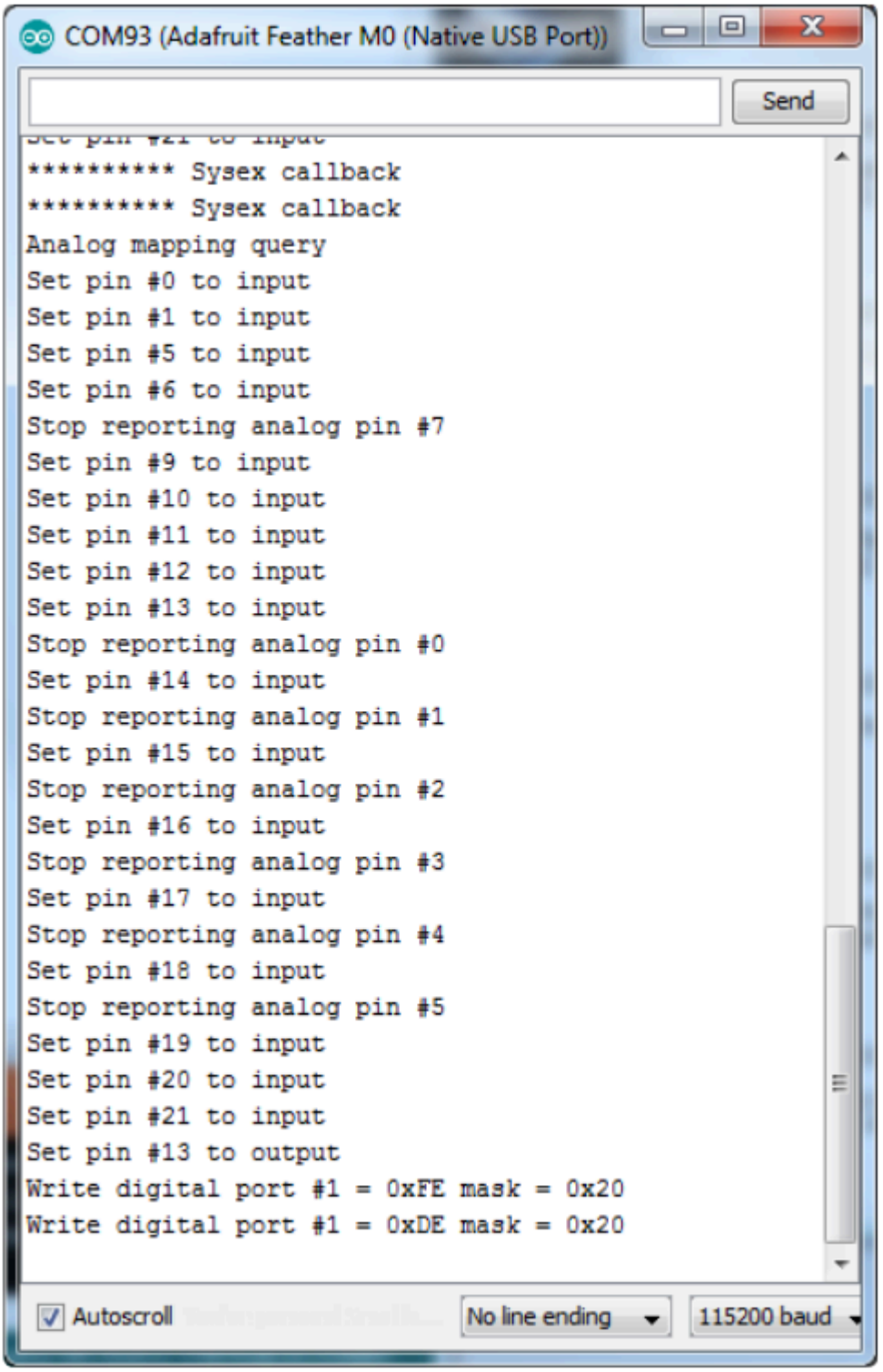
Log Contributions



Then open the Android Adafruit app. Once the app is opened, connect to the device. Wait for the **Serial Monitor** to update the pins

Content on this document is *Creative Commons (BY SA)*.

Log Contributions



Then press Pin I/O on the Android App. The App will provide the list of all the pins and their state.

More here..

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

Serial Peripheral Interface (SPI) is an interface bus commonly used to send data between microcontrollers and small peripherals such as shift registers, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device you wish to talk to.

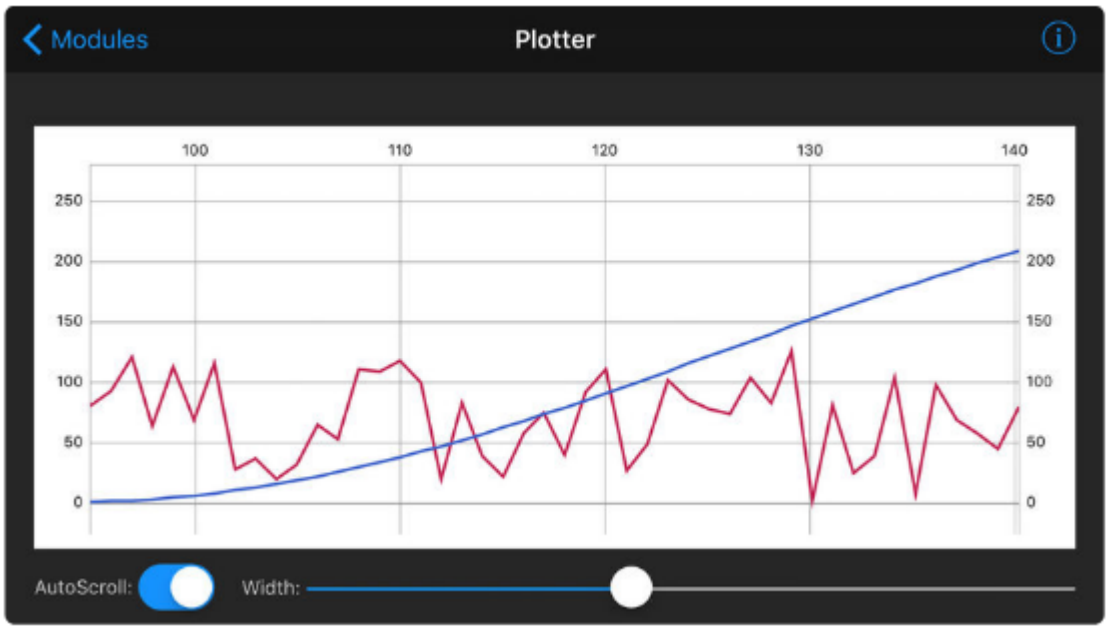
Observation, on the Android app the information about the pins is available only if the following line is activated in the ***BluefruitLE_nrf51822*** sketch (an example, see above).

```
// Create the bluefruit object, either software serial...uncomment these lines

...hardware SPI, using SCK/MOSI/MISO hardware SPI pins and then user selected CS/IRQ/RST */

Adafruit_BluefruitLE_SPI bluefruit(BLUEFRUIT_SPI_CS, BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);
```

I tried the Software serial instead, but it didn't work.

See analog data using the Plotter

The Plotter mode in the Android App allows for users to visualize data received from Bluetooth LE compatible devices. This data is sent from the Bluefruit device to the phone/tablet. Feel free to test the plotting feature using this demo sketch (https://adafru.it/xTc)!



NOTE: Tibi tested the sketch, modified to read the analog sensor value and send it to the Plotter. Works.

Also, this sketch works even when the Pressure Sensor is NOT connected to the computer, so it doesn't matter if the serial monitor of the Arduino IDE is not on. The communication between the

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

Pressure Sensor and the Android device is established when the Pressure Sensor is only plugged into a 5V power supply.

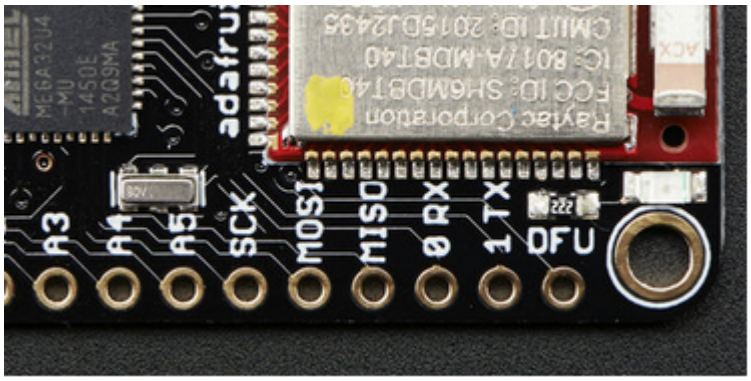
The UART mode / example above, data was sent to the Android device but the connection between the Feather and the Android device could not be established without the Feather being connected to the Computer with the IDE on and Serial Monitor. Compare both examples to isolate the code that causes this.

Potential problems

Update the firmware of the BLE device.

DFU: Device Firmware Update

This is a pin that puts the device in update mode. You find it on the board here.

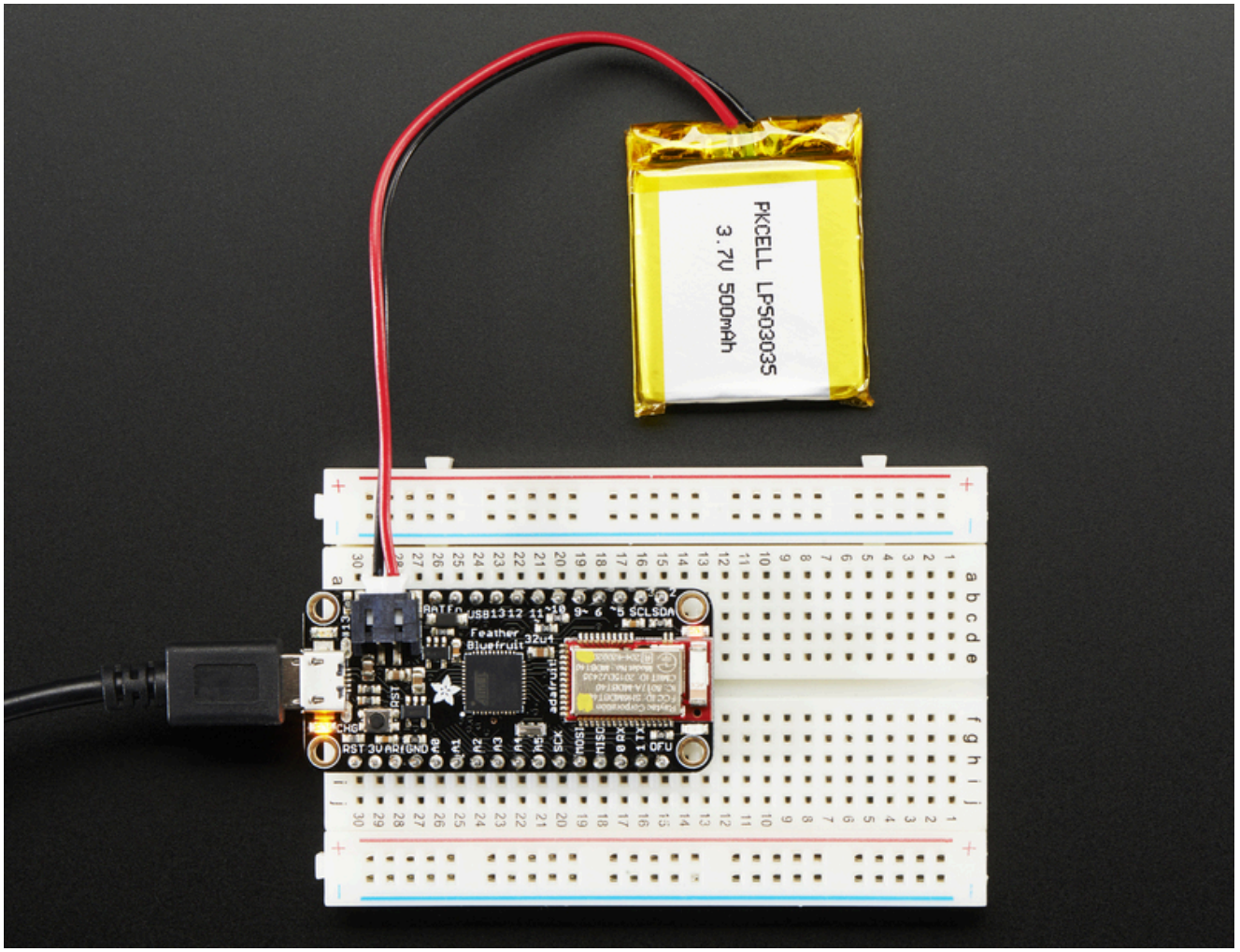


NOTE: Tibi updated firmware of the Feather BLE device used for development at the Sensorica lab to 8.1 version, using the BLE Android app.

We are running on version XXX

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions



For portable projects, the board has a connector for a 3.7V Lithium polymer battery and built-in battery charging. You don't need a battery, it will run just fine straight from the micro USB connector. But, if you do have a battery, you can take it on the go, then plug in the USB to recharge. The Feather will automatically switch over to USB power when its available. The battery is tied thru a divider to an analog pin, so you can measure and monitor the battery voltage to detect when you need a recharge.

Bluetooth LE 4.0 with Arduino - wireless

During 2024, the stakeholders of Breathing Games (project lead network organization), i.e. Sensorica (hardware design, trust economy and regulatory approval lead organization) and Sainte Justine Hospital (sponsor and early adopter) have adopted the Adafruit Feather 32u4 Bluefruit microcontroller and the wireless mode of communication and have decided to move

Content on this document is *Creative Commons (BY SA)*.

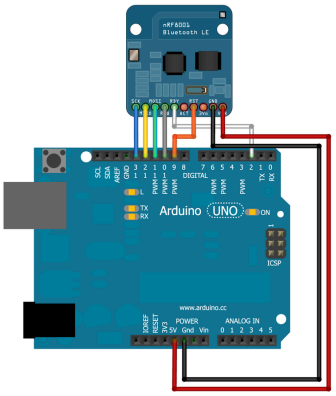
Log Contributions

games to mobile platforms. See the BLE communication section. The USB connection is not abandoned, it will be kept as an option.
The Arduino microcontroller platform has been abandoned for the moment.

Message written in 2024.

NOTE: the use of the Adafruit Feather microcontroller board has made this option obsolete.

The BLE protocol has been chosen due to its popularity and connectivity with Smartphones and Tablets. For the first prototype we are using a nRF8001 breakout board from adafruit mainly due to the reason that we had one in the lab :)



To test the board it was hooked up to an arduino uno r3 and loaded with a sketch that sends ASCII characters between the serial monitor and a device paired with the BLE radio. No smartphone was available for this first test so a BLE dongle from Texas Instruments was used along with Btool.

Code on Github

Download

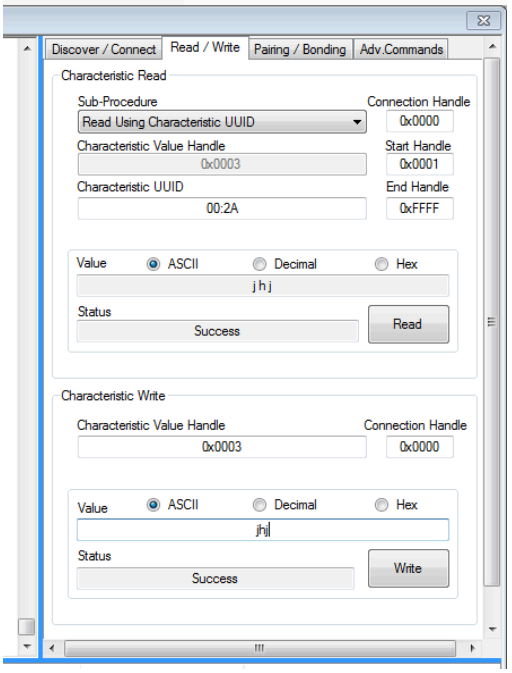
The CC2540 BLE USB dongle adds BLE functionality to any PC and can also be used as a packet sniffer.

Here are the tools to uses this device:

SmartRF protocol packet sniffer

BTool in BLE-Stack (required)

Here is BTool in action using the callbackEcho sketch. A character in ASCII is written to the paired device and it is echoed back to the dongle. You can view the different attributes and profiles in the BLE protocol with this tool if you want to explore the architecture of the BLE stack.



Content on this document is *Creative Commons (BY SA)*.

Log Contributions

! NOTE that the BLE 4.0 is not compatible with other versions of Bluetooth. It is not a successor either. It has different features which set it apart from the other protocols: smaller packets, longer range and extremely low energy consumption. This makes it ideal for sensors and battery powered applications. The developers had posters of coin cell batteries on the walls while designing this version (no joke!) and life expectancy is usually measured in weeks (even up to a year).

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

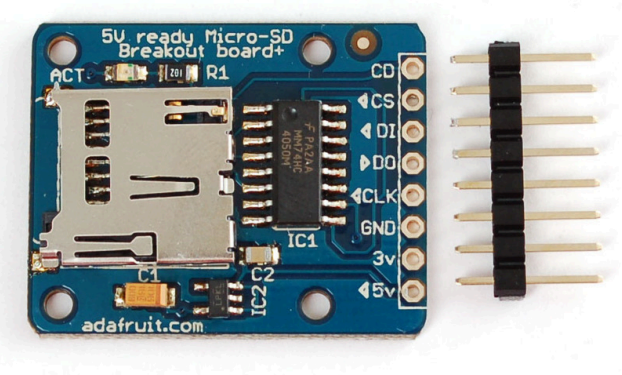
Data storage

A need to store data on the device has surfaced.

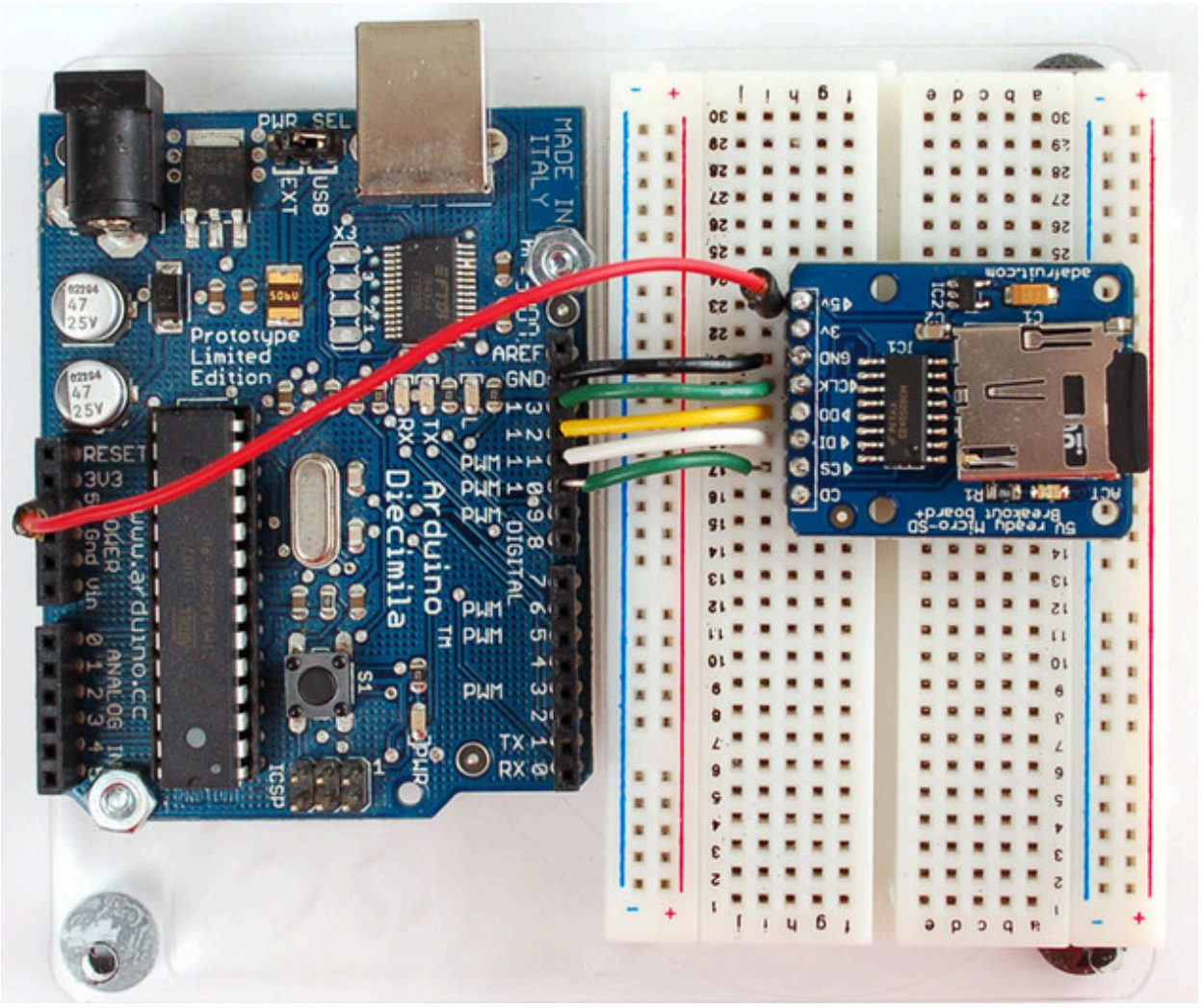
Let's explore some possibilities:

Micro SD Card Breakout Board+

Online supplier: Adafruit
Local supplier: Link to ABRA
SD Card formatter



Wiring



Also see Variables to measure on BG-8-1 Core document healthcare

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

Pressure during expirations from 0 to 100 (0 to 20 cm H_20), a 200 milliseconds = 25 data per 5 seconds expiration

Trigger to automatically start the device when pressure increases

Sleep mode if pressure is null after 15 minutes

Structure of data

```
<?xml version="1.0"?>
<cycle time="timestamp" id="n">                    timestamp
  <pressure>10</pressure>                          tinyint, 0 to 100
</cycle>
```

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

Data processing

For data processing we used Arduino Uno and Nano and Adafruit Feather. The Adafruit Feather 32u4 BLE has the advantage of having integrated low energy bluetooth, which provides the easy option of connecting the pressure device to any mobile device or desktop computer for a game.

Adafruit Feather 32u4 Bluefruit LE data processing

NOTE: If you experience problems with uploading sketches to the Adafruit Feather 32u4 BLE using Arduino IDE, if you're running Linux get help from here; if you're using Windows, make sure you have installed the proper drivers. If these links don't work, open the downloaded pdf. One trick that you should know is, after you click upload on the Arduino IDE for the sketch, press the board's reset button twice to start the bootloader. You can find this advice here.

Arduino nano data processing

This possibility has been explored already. See sections above.

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

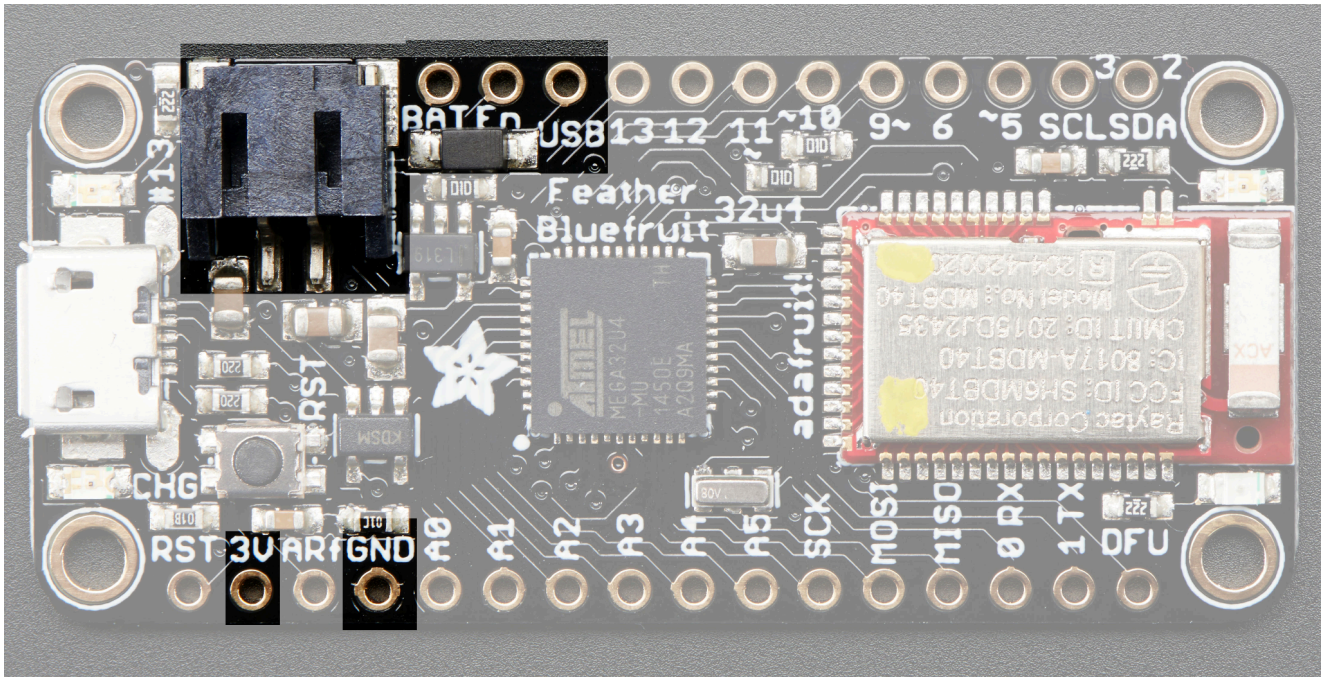
Power & power management

During the October Game Jam at Sainte Justine Tibi spoke to patients to get feedback about power switches ON and OFF to save battery, when the device is used unplugged with mobile devices. One design problem that we need to solve is to accommodate forgetfulness of switching the device OFF once used, to preserve the battery. We can simply use a SPST switch to manually disconnect the battery from the device, which requires conscious physical action. But patients are kids, … they can be distracted and forget to turn OFF.

Battery for Adafruit Feather 32u4 Bluefruit LE

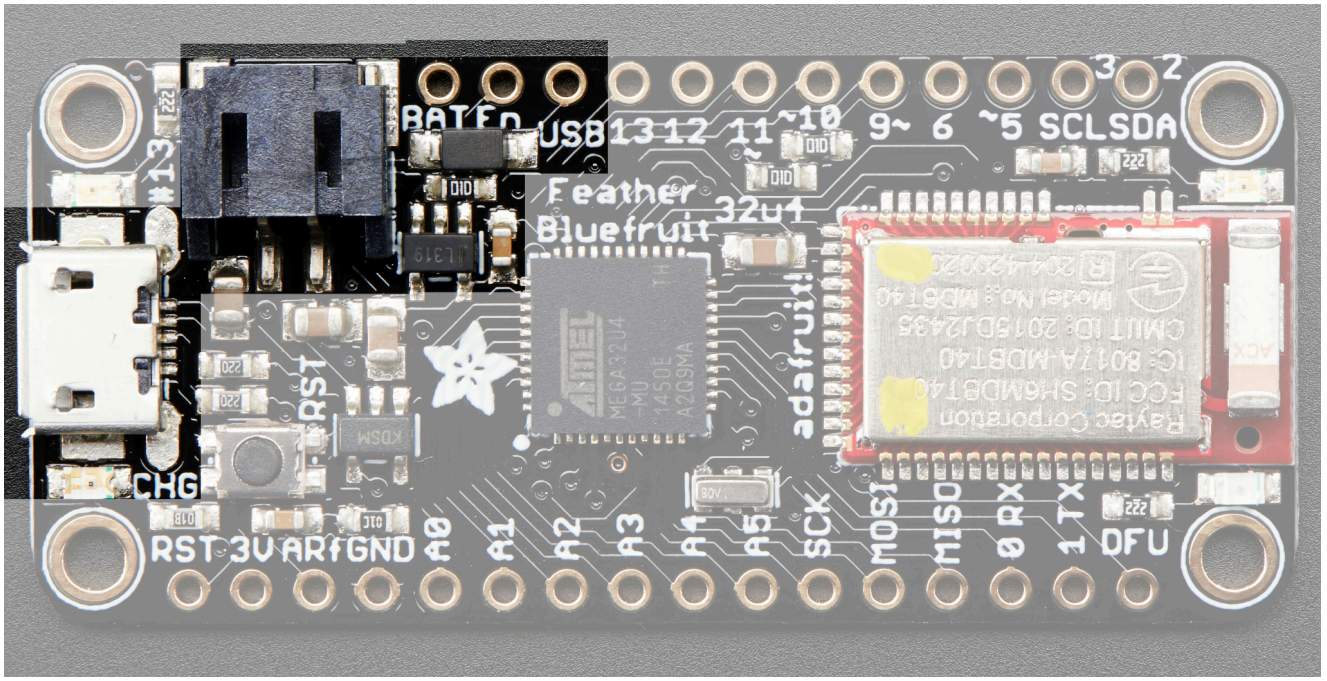
There's two ways to power a Feather:

You can connect with a USB cable (just plug into the jack) and the Feather will regulate the 5V USB down to 3.3V. You can also connect a 4.2/3.7V Lithium Polymer (LiPo/LiPoly) or Lithium Ion (LiIon) battery to the JST jack. This will let the Feather run on a rechargeable battery. When the USB power is powered, it will automatically switch over to USB for power, as well as start charging the battery (if attached). This happens 'hot-swap' style so you can always keep the LiPoly connected as a 'backup' power that will only get used when USB power is lost. See power management page for Feather



Content on this document is *Creative Commons (BY SA)*.

Log Contributions



The above shows the Micro USB jack (left), LiPoly JST jack (top left), as well as the 3.3V regulator and changeover diode (just to the right of the JST jack) and the LiPoly charging circuitry (to the right of the Reset button).

There's also a **CHG** LED next to the USB jack, which will light up while the battery is charging. This LED might also flicker if the battery is not connected, it's normal.

We use a Lithium Ion Polymer Battery 3.7V 500mAh Rechargeable

- Size: 1.4" x 1.15" x 0.19" (36mm x 29mm x 4.75mm)
- Weight: 10.5g
- Power lead length: 50mm / 1.97"
- Output: 500mAh at 3.7V nominal

Power management

There are two simple options to save battery when the device is not in use, a more complicated one.

Using the EN (enable) pin

End of October 2024, Tibi and Richard deliberated on Discord (Sensorica server and Breathing Games server) about what power management solution to implement. This one was chosen for the first devices delivered at Sainte Justine Hospital in early

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

November 2024, which were a modified version of the devices that were made in 2023. Here's a video explaining how this device functions.

The first simple option, recommended by Adafruit, is to turn off the 3.3V regulator, by pulling the **EN**(able) pin to ground, which disables the 3V regulator. The **BAT** and **USB** pins will still be powered.

Tibi made an experiment, when the EN pin is connected to GND of the board or to the negative wire of the connected battery, while the Feather board is NOT connected via OSB to a 5V power source, the entire board stops working. For example, the BLE doesn't work, as seen with the BLE Android App (it disappears from available devices list) and the red LED that tells the microcontroller turns off. So it seems like stopping the 3V regulator disables power to the board.

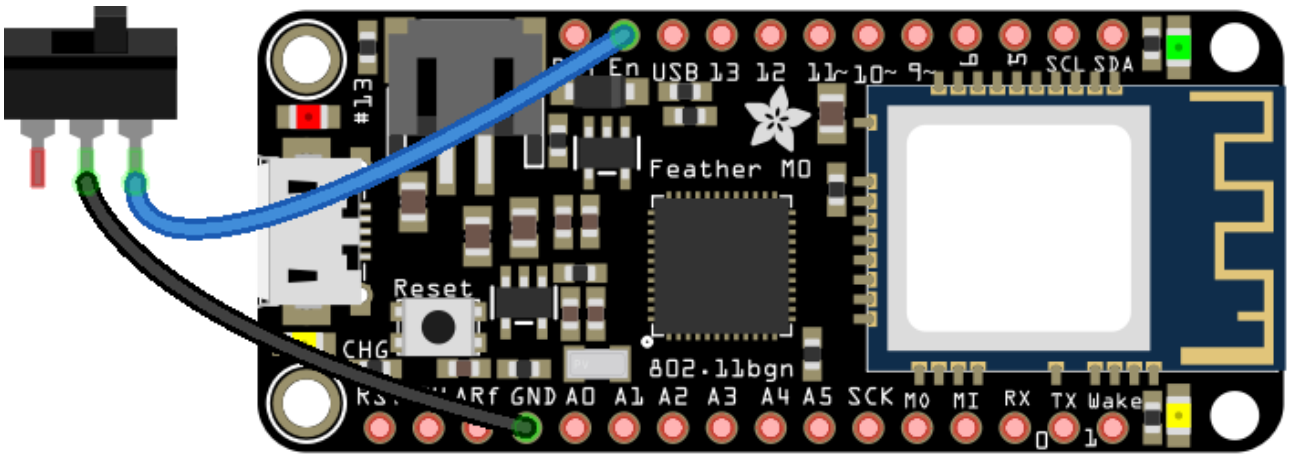
Here's a quick tip from @ladyada that will show you how you can use the **EN** (enable) pin to control the power on your Adafruit Feather with a SPDT slide switch.

Here is how to connect the switch to your Feather (reference):

- Feather **GND** -> Center pin on the switch
- Feather **EN** -> One of the outer pins on the switch

You can now turn your Feather off by flipping the switch to the side that is connected to the **EN** pin.

For more info about the **EN** (enable) pin, check out this guide.



Richard made some measurements and calculations for this case. This following picture shows the current consumption of the Pressure Sensor using the Feather 32u4 in operation. It looks spiky, which is due to the bluetooth radio transmitting. The values are pretty consistent whether connected or disconnected. There may be a couple more mA due the LED that is placed on the sensor board. So maybe if we take the average + 2mA then 18mA total. I think your battery is 500mAh so 500/18 hours operation if left on = 27 hours.

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

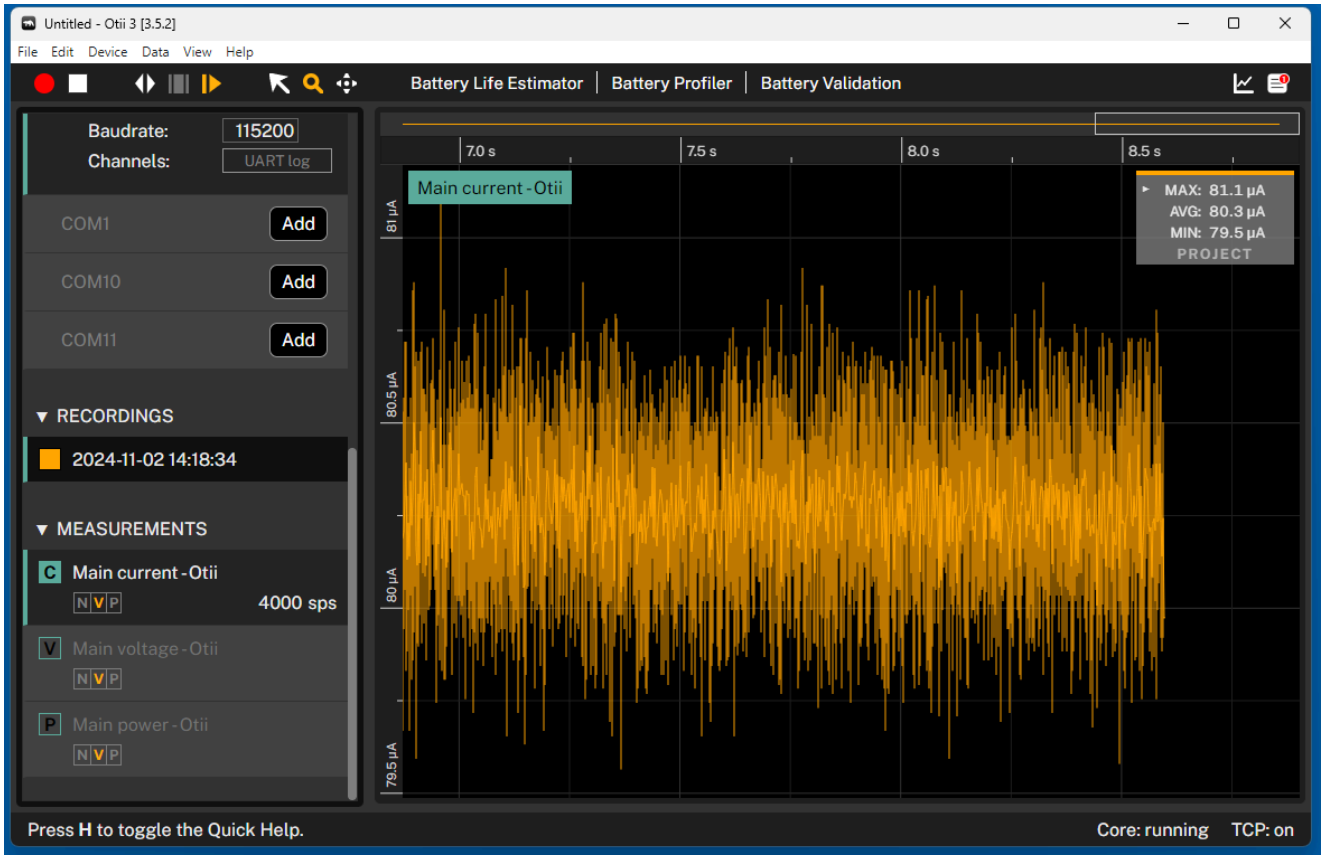
This needs to be validated in practice.



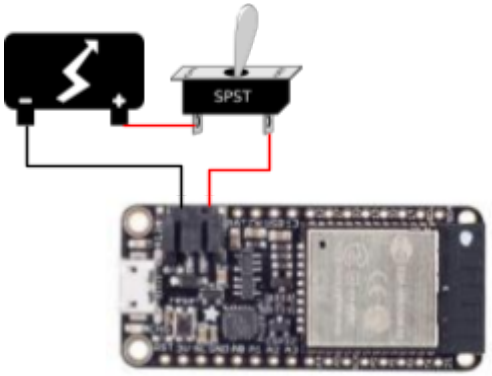
When Richard connected the En pin to ground then the device powers off and the current is above at around 80uA. This would give a battery life of 500/.08 = 6250 hours, 260 days.

Content on this document is *Creative Commons (BY SA)*.

Log Contributions



Separating the battery
The second simple option is to use a simple latch or two state ON / OF switch in line with the hot wire from the battery. We can also use a hidden, non-contact magnetic switch for this purpose.



Problem When the user wants to charge the device, he/she needs to turn ON the switch. The EN pin option doesn't require the user to do anything for charging, even if the 3V regulator is off the battery can still be charged.

We have purchased two types of switches.

Content on this document is *Creative Commons (BY SA)*.

Log Contributions



Reed magnetic switch, normally open, closed when magnet in proximity (SPST NO). The advantage is no contact, no holes into the box need to be made.



Latch ON OFF SPST push button.

Programmatically turn off the 3V regulator

NOTE: This was tried by Tibi on Oct 31th 2024 at the Sensorica lab and it didn't work.

We may want to turn the Feather device OFF programmatically, related to a timeout function, for example 5 minutes of unusage. In that case, we could try a risky solution, to use a digital I/O pin, set as input, connected to the EN (enable) pin, which is by default set to high using an internal pull up resistor, to low. In other words, we use a digital I/O to sink some current.

An I/O pin can sink only so much current. For a GPIO bank, 100mA max, one pin recommended 20mA. It is a good idea to use a resistor between the digital I/O pin and the EN pin. I tried 1.8 K ohms, but this can vary based on the specific requirements of your circuit.

Example of Arduino code for Adafruit Feather 32u4 BLE to implement this

```
#include <Arduino.h>
#include <SPI.h>
#include <SoftwareSerial.h>
#include "Adafruit_BLE.h"
#include "Adafruit_BluefruitLE_SPI.h"
#include "BluefruitConfig.h"

// Create the bluefruit object
Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS, BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);
```

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

```
int sensorPin = A1; // Input pin for the pressure sensor

int sensorValue = 0;

int scaledPressure = 0; // the output of the device remapped from 1 - 100

int batteryPin = 12; // declare and initialize  digital I/O pin number to control battery

int deltasensorValue = 90; // declare and initialize constant to account for small variations in sensorValue

int lastsensorValue = 0; // declare and initialize lastsensorValue to store the previous value read.


// Declare and initialize the timing variables for the timeout_function as unsigned longs because the time, measured in milliseconds, will quickly become a bigger number than can be stored in an int.

unsigned long lastuseTime = 0;  // declare and initialize lastuseTime that sensorValue changed

unsigned long timeDelay = 10000; // declare and initialize inactivity time to timeout and shot off the device


void setup() {

pinMode(batteryPin, OUTPUT); // set the digital I/O pin to output

digitalWrite(batteryPin, HIGH);  // set the digital I/O pin to HIGH, will keep the battery ON by letting the EN pin to which it is connected to stay HIGH.


// Initialize Serial for debugging

Serial.begin(115200);


// Initialize the BLE module

if (!ble.begin(false)) {

while (1);

 }


 // Perform a factory reset to ensure a clean state

ble.factoryReset();
```

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

```
// Set BLE device name
ble.sendCommandCheckOK(F("AT+GAPDEVNAME=FeatherBlue"));

// Set the mode to data mode (as opposed to command mode)
ble.setMode(BLUEFRUIT_MODE_DATA);
// Start advertising

ble.sendCommandCheckOK(F("AT+GAPSTARTADV"));
}



void loop() {
// Check if BLE is connected
if (ble.isConnected())
{
// Read analog value from A1 pin
sensorValue = analogRead(sensorPin);
scaledPressure = constrain(map(sensorValue,140,580,0,100),0,100);
// Send the sensor value as a string over BLE
ble.print(scaledPressure);

// Small delay to prevent flooding
delay(100);
}

 // Implement timeout logic
if(sensorValue > (lastsensorValue-deltasensorValue) && sensorValue < (lastsensorValue+deltasensorValue)) {
  Serial.println("device not in use");

  if ((millis() - lastuseTime) > timeDelay) {
        Serial.println("battery will stop!!!");
```

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

```
            delay(1000);
            digitalWrite(batteryPin, LOW);  // set the digital I/O pin to LOW
            delay(3000);
        }
        } else {
            lastuseTime =  millis();
            Serial.println("device in use");

            }

    // Always send data via Serial
    /*
    sensorValue = analogRead(sensorPin);
    scaledPressure = constrain(map(sensorValue,140,580,0,100),0,100);
    Serial.println(scaledPressure);
    */
    Serial.println(lastuseTime);
    Serial.println(millis());
    Serial.println(lastsensorValue);
    Serial.println(sensorValue);
    delay(300);
    lastsensorValue = sensorValue;

    }
```

I loaded the code by connecting the Feather to the PC with the USB cable. I used this connection to debug the logic and then I did the tests unplugging the USB cable, with the battery connected to the Feather device. **It didn't work.**

I also tried to set the batteryPin as INPUT mode and set it to LOW when the timeout condition was met, **still didn't work**. I thought that the resistor was probably too high to drain the EN pin quickly. So I connected the digital I/O pin directly to the EN pin with a wire, **still didn't work**.

The reason is that a digital I/O pin is not put to ground, it is supposed to be high impedance, not what you want. Also, probably once the 3V regulator turns off the digital I/O pin goes to a normal state and the effect vanishes almost immediately, the board recovers.

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

What happens when a GPIO digital pin of a microcontroller board is set to low? It is internally connected to ground?

When a GPIO digital pin of a microcontroller is set to low, it is not necessarily internally connected to ground. Here are the key points to understand:

1. Configuration matters:
    - As an output pin, setting it low means it will be actively connected to ground (GND) through the pin's output driver circuitry.
    - As an input pin, setting it low means it will be in a high-impedance state, not actively connected to ground.
2. Tri-state logic:
    - Modern GPIO pins are implemented as tri-state buffers, allowing three states:
        - Logical 0 (connection to ground)
        - Logical 1 (connection to VCC/VDD)
        - High-impedance (floating)
3. Push-pull output mode:
    - When configured as a push-pull output, setting the pin low will actively sink current to ground.
    - The pin will have a defined voltage level close to ground.
4. Open-drain output mode:
    - When configured as an open-drain output, setting the pin low will only sink current to ground, but the pin will still be in a high-impedance state.
    - An external pull-up resistor is typically required to achieve a logical high state.
5. Input mode:
    - As an input, setting the pin low puts it in a high-impedance state, **not actively connected to ground**.
    - The pin will float to whatever residual voltage exists on the pin.

In summary, whether the pin is actively connected to ground depends on its configuration as an output versus an input, and the specific output mode (push-pull vs open-drain) if configured as an output. As an input, setting it low does not mean it's actively connected to ground.

Perhaps the proper way to do this is to use a relay.

NOTE : The following in this section was not tested !!

Key Considerations

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

1. The battery is connected to the device via a short electrical pulse, such as from a push button or magnetic switch.
2. It should remain connected until triggered by another pulse, generated by the microcontroller, driven by a timeout functionality, after a period of inactivity.
3. The user should be able to reconnect the battery by activating a push button.

Using a bistable latching relay, with a push button to turn the device ON and the microcontroller to turn the device OFF. In this implementation the push button can be a mechanical push button or a magnetic switch that we only activate momentarily, as a push button. This scheme has two solutions: Using a latching relay, see example G5RLU-1-E DC3 available at DIGIKEY.

More on latching relay.

A latching relay (also known as a bistable, keep, impulse, stay relay, or simply a "latch") is defined as a two-positional electromechanical switch. It is an electrically actuated switch used to maintain its position without power applied to the coil.

A latching relay is used to control the large flow of current with a smaller current. The coil of the latching relay consumes power only while the relay is switched ON. And its contact remains in position after the switch has been released. See the latching relay circuit diagram below for more details on how this works.

- **Latching Relay Definition**: A latching relay is a type of relay that maintains its contact position without continuous power application, allowing for efficient control of circuits.
- **Circuit Diagram**: The latching relay circuit diagram shows how Button-1 and Button-2 control the energizing and de-energizing of the relay, respectively.
- **Operation Mechanism**: By pressing Button-1, the relay is energized and remains so even after the button is released, until Button-2 is pressed.
- **Efficiency and Use**: Latching relays are energy efficient as they require power only for changing states, not for maintaining them.
- **Practical Application**: They are commonly used in setups requiring the circuit to remain active without continuous power, such as home lighting systems and industrial conveyors.

Solid state relay (SSR)

1. Optocoupled SSRs: These are ideal for your application as they can be turned on/off with a short electrical pulse. They typically use an infrared LED and photodiode to create an optical isolation barrier.

Content on this document is *Creative Commons (BY SA)*.

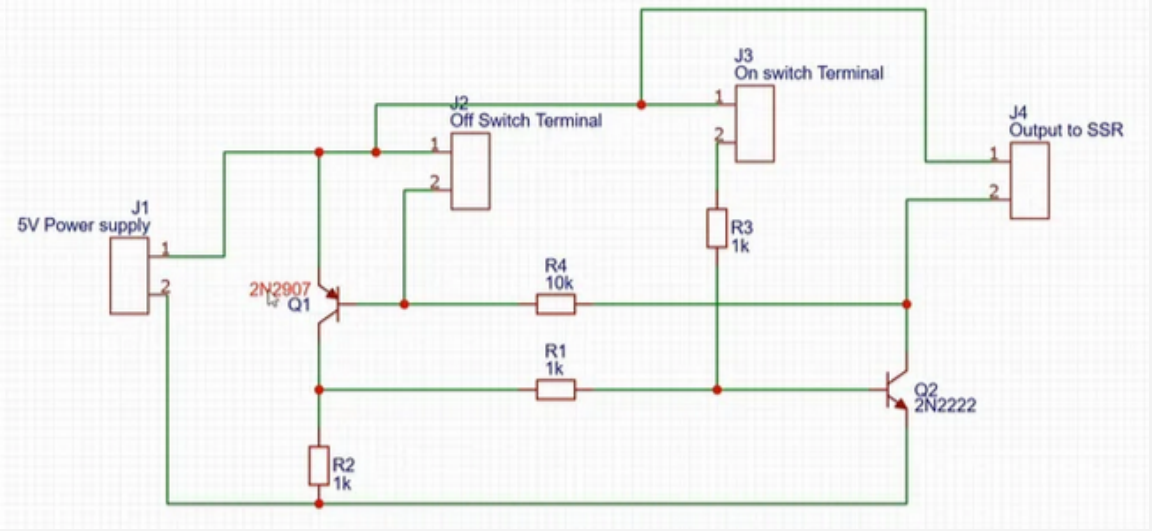
Log Contributions

2. Triac-based SSRs: These can be controlled with a short pulse and maintain their state until reset. They are commonly used in power control applications. See video and this one.
3. MOSFET-based SSRs: These are highly reliable and can be controlled with a logic-level signal, making them compatible with microcontrollers.

Implementation Steps

1. Choose an appropriate SSR based on your voltage and current requirements.
2. Connect the SSR between the battery and load circuit.
3. Implement a microcontroller program to:
    - Monitor for button presses
    - Control the SSR state (on/off)
    - Set up a timer for inactivity detection
    - Disconnect power after timeout period
4. Add a push button connected to the microcontroller input to allow manual reconnection.

Building a solid state latching relay see reference.



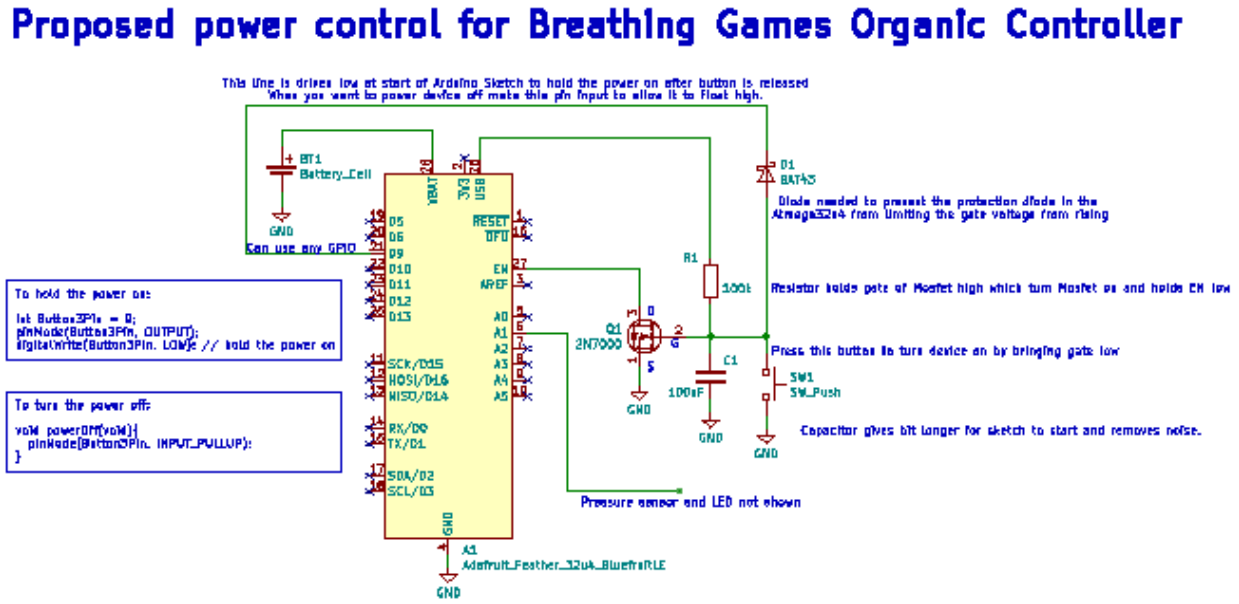
Richard's solution

Richard tested the following scheme to programmatically do power management using the EN (enable) pin and it seems to work OK on both USB and battery. The only quirk is that holding the power ON relies on the Arduino sketch. This is no problem during normal operation.

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

Problem: Unlike with the latching relay (see above), when uploading code in the device the user must hold the power ON momentary button ON to keep the power on. Once the code starts running everything will be OK. The advantage is that this solution is more cost effective, as Mosfets are much cheaper than relays. Since the problem is not on the user's side, only on the side of the fabricator, it's OK, as we expect fabricators to be a little more sophisticated than the users.



Rationale explained:

> To power off the device you need to pull EN (enable) pin low against the 100k pullup. This can't be done by the processor because as soon as the processor powers down the GPIO goes to floating and the device turns on again. You need an *N channel Mosfet* to hold the EN pin low to ground. At the gate of the Mosfet would be a resistor to VBus of say 100k to hold the Mosfet to begin. The resistor will add another 50uA to the quiescent load on the battery, but this should still be OK. You might try a larger resistor. Also on the gate of the Mosfet would be a momentary button (normally OFF, ON when pressed) to ground to turn on the device. Also the gate should be routed to a GPIO pin. As soon as the sketch begins you set this GPIO as output and low. This keeps the power ON to the device after the button is released. A capacitor on the gate of say 100nF may also be needed.

Pros and Cons table for Power Management

	Pros	Cons

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

Manually Using a switch.	Cutting battery OFF	Low cost. No battery consumption when device is turned OFF.	User must remember to turn the switch ON when charging the battery. User needs to remember to turn device OFF after use.
	Coupling EN pin to ground.	Low cost. User and fabricator don't need to follow other instructions, it just works.	Very low battery discharge time even when the device is OFF. User needs to remember to turn device OFF after use.
Programmatically Using EN pin + components	Mosfet Using EN pin	Low cost User doesn't need to remember to turn device OFF after use.	Very low battery discharge time even when the device is OFF. Fabricator must keep the momentary switch pressed to ON while uploading code to the device.
	Latching Relay Using EN pin.	No battery consumption when device is turned OFF. User doesn't need to remember to turn device OFF after use.	Expensive

USB cables and adapters

Short micro USB male to USB c female adaptor, 2.3 cm length Aliexpress



Content on this document is *Creative Commons (BY SA)*.

Log Contributions

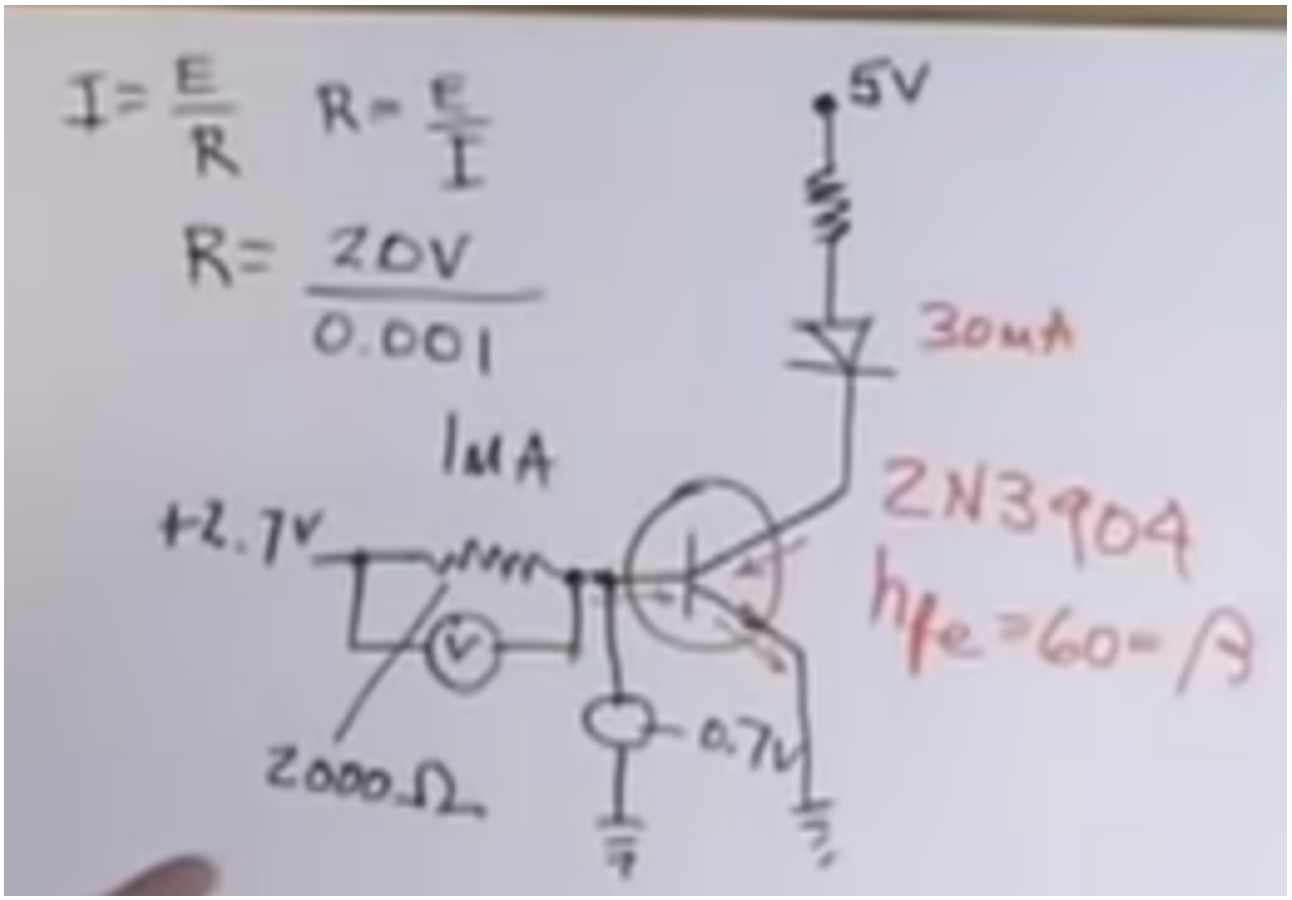
Others

Visual feedback

Using a hardware-driven LED to provide raw visual feedback of pressure. Requirements: the LED is run directly from the 5V power supply and the pressure sensor and is independent of the microcontroller (Feather or arduino).

We use a NPN (2N) 3904 transistor. See this video for the theory.

The LED visual signal can be used for debugging of the hardware.

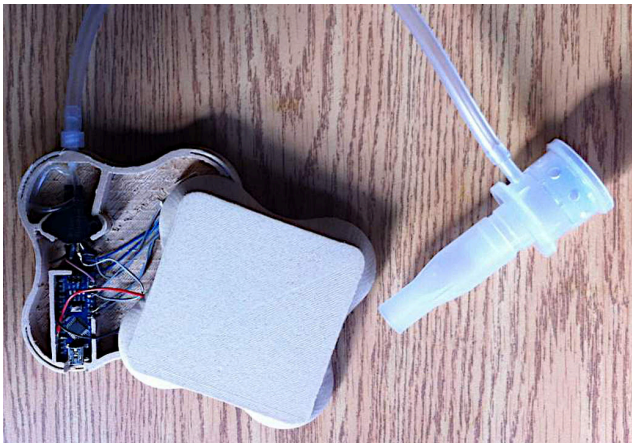


Mechanical

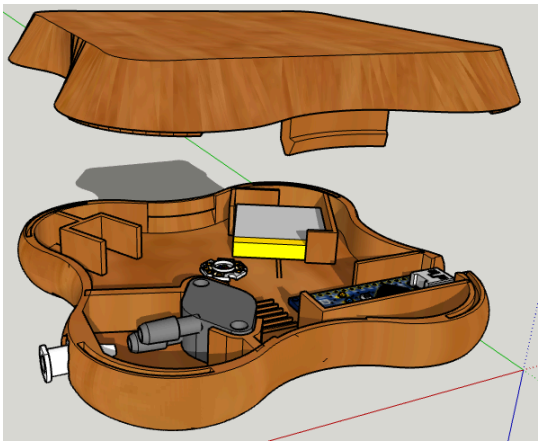
Organic Box

Content on this document is Creative Commons (BY SA).

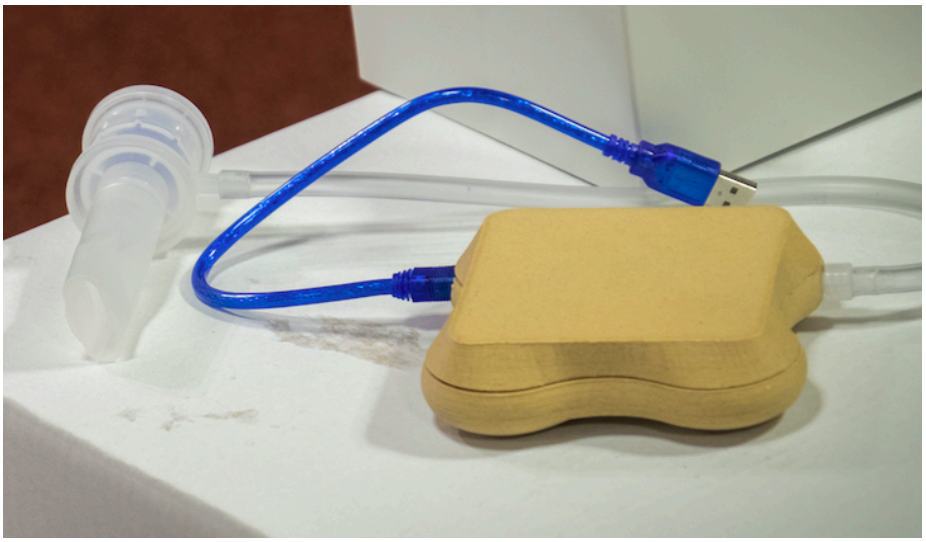
Log Contributions



Get 3D model



Sketchup version



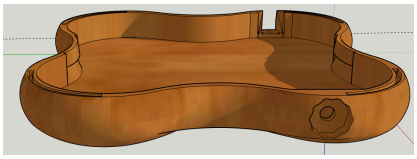
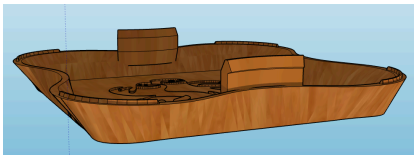
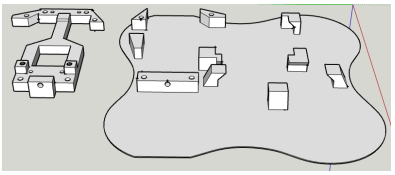
Watch a video documenting the first production of this enclosure for what was called the b-organix-3D-box device (see Gitlab repo).

This box was designed by John, with the help of Daniel. Daniel 3D printed the first prototype, which was used with an arduino nano. Click on the links above and spin them around and provide feedback. This first design was then modified by Tibi in 2023 to be used in prototyping for Saint Justine, you can find them in the Sensorica's Google Drive, here. See also Tibi's modified model on 3D Warehouse.

The 2023 organic box model is made of 3 parts, bottom, top cover and middle. The middle part is a thin sheet with pins for holding the electronics within the box. Tibi has decided to separate these holding features from the bottom part of the box in order to make it easily adjustable to changes in electronics design and to make it better printable upside down.

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

 bottom	 top cover	 middle

3D printing the box with PETG

RECOMMENDED!

Less warping and less post-processing work required and less toxic.

A cartesian FDM 3D printer was used, with 1.75mm PETG filament (color green), through a nozzle of 0.4 mm.

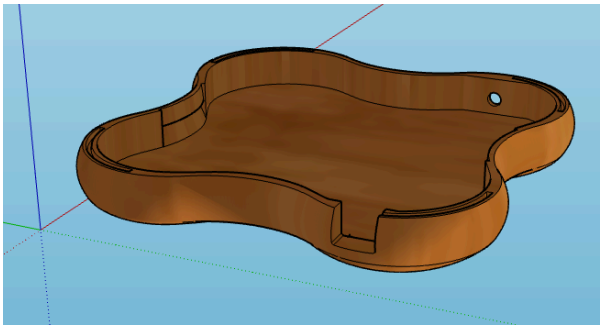
Preparation of the printing bed

The development was performed at the Sensorica lab and all parts were printed with a bed temperature of 95C and hot end temperature at 235C. The bed was covered with kapton tape.



Printer parameters and printing process

The **bottom part** is printed with the opening facing down (upside down) and it requires support, which is 1.5 mm separated from the walls of the part (in X and Y) and 0.3 mm separated in Z from the flat horizontal surface of the part. No brim or skirt or raft was used. Bed temperature was 95C, hot tend (extrusion) temperature was 235C. Printing this part takes between 3 to 4 hours.

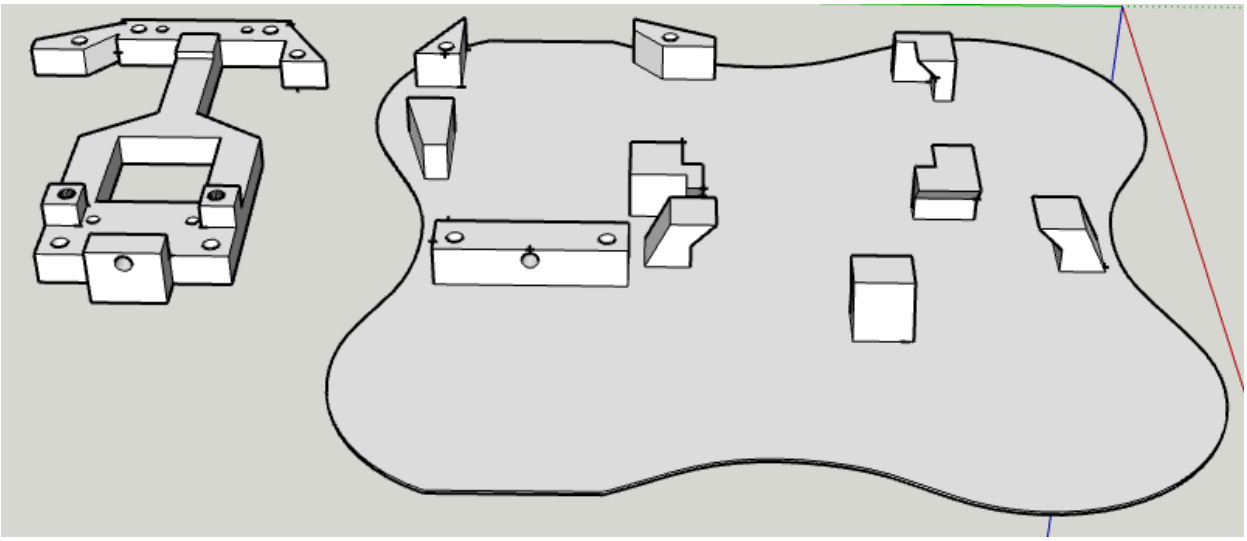
	

Content on this document is ***Creative Commons (BY SA)***.

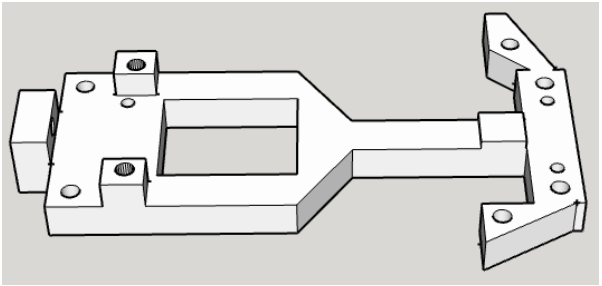
Log Contributions

The **upper part** (the cover) is printed with the opening up, and thus it does not require support. Same bed preparation and temperature was used as for the bottom part. No brim or skirt or raft was used. Printing this part takes between xxx to xxx hours.

The **middle part** is printed with the same bed preparation and temperature as for the bottom part. No brim or skirt or raft was used. Printing this part takes around 30 minutes.



The optical fiber holder



3D printer parameters are listed in this folder.

3D printing of box with ABS

NOT RECOMMENDED! Toxic, more warping.

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

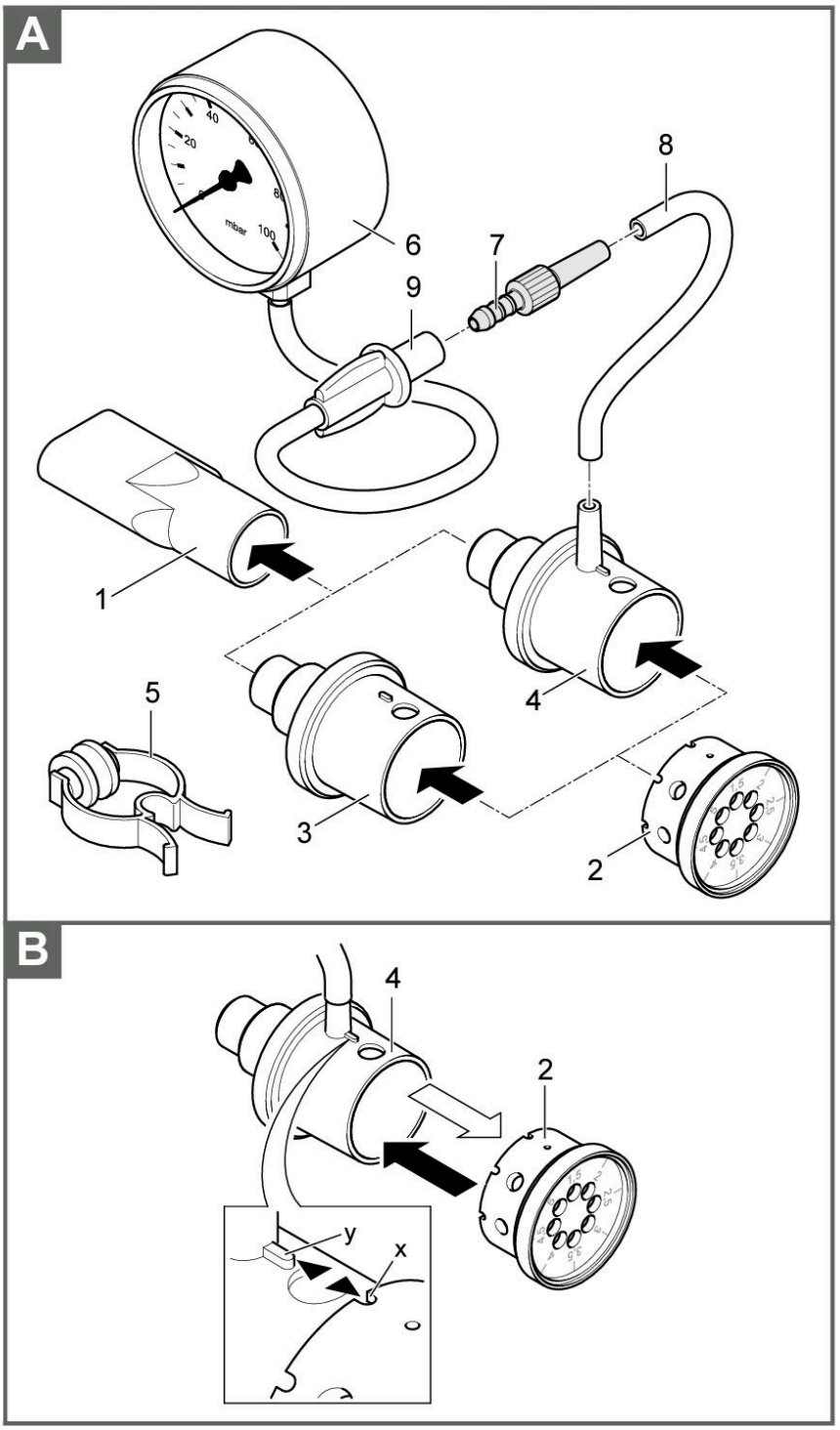
Mouthpiece

The patient will use the following device to blow into the pressure sensor / device. It connects to the sensor box with a ¼ ID elastic tube.



Content on this document is ***Creative Commons (BY SA)***.

Log Contributions



Content on this document is *Creative Commons (BY SA)*.

Log Contributions



Open doc for more info

Air tube

We use Tube-S5-1M translucent silicone, 5mm ID see BOM. This tube is very flexible and it doesn't present a medical problem.

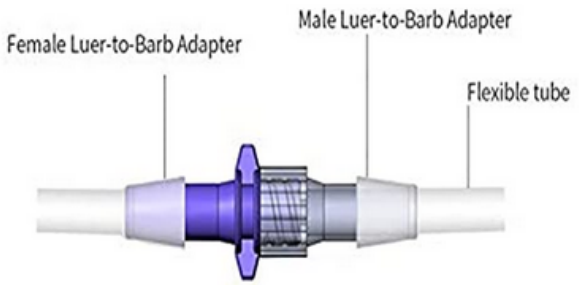
Tubing and filters

Respond to the Gitlab issue about water accumulation on the pressure sensor, how it can interfere with the measure, lifetime of the sensor and medical risks.

Air tubes connections

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions



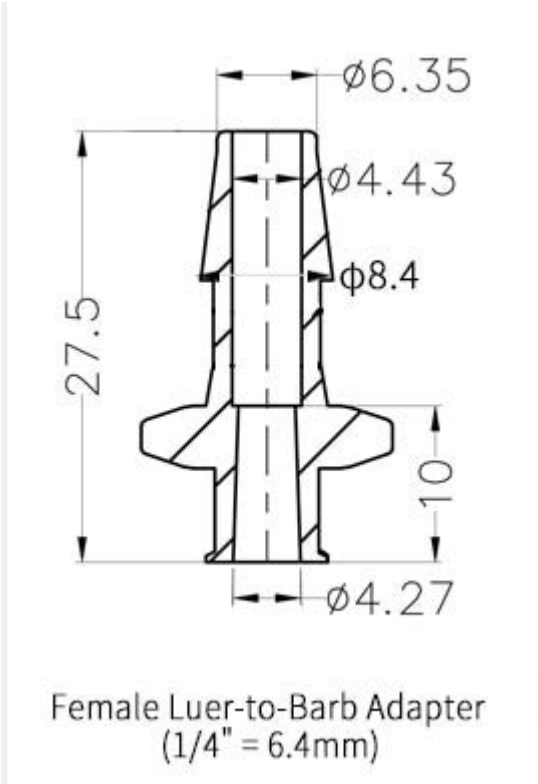
For the tube fittings, we use a Plastic Hose Fitting Luer Lock Connector Male/Female Luer Adapter Polypropylene

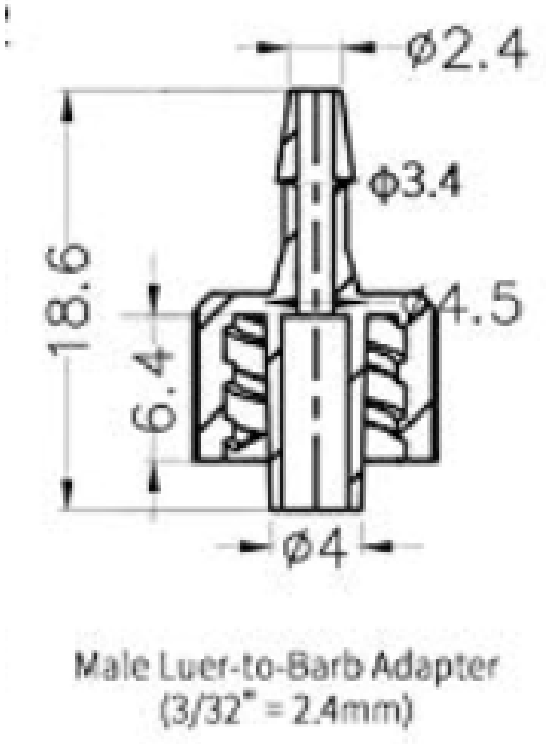
See on Amazon [link may go out of date]

The reason why we need a connector is to allow sharing between patients of the device, in which case the mouthpiece and the tube that connects 9t to the box must be changed for every patient. Thee connectors can support multiple connection cycles.

The exterior part is Female 1/4", it connects the mouthpiece to the box.

The interior part is recessed into the all of the box and glued. In place. It connects to the pressure sensor. It is Male 3/32, fits into a smaller tube..





Include battery.

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

Other developments: insert a battery.

Others

The idea of a raspberry pie has been raised. It does not seem strategic for domestic use as CF is mostly to be found in North America and Central Europe - countries that are well equipped. However, using it for follow ups in hospitals would be an asset as their infrastructure is often highly secured.

According to Jim:

> *The Raspberry Pi is not good for analog input. Its GPIO's are unprotected , take a max of 3.3V and aren't reliable for continuous sensor input since the cpu is running an OS and can't dedicate to mission critical tasks properly*

Testing this configuration

See our video

Get the parts list

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

Software

Firmware for Adafruit Feather 32u4 Bluefruit with USB cable operation

The current stakeholders of Breathing Games (project lead network organization), i.e. Sensorica (hardware design, trust economy and regulatory approval lead organization) and Sainte Justine Hospital (sponsor and early adopter) have adopted the Adafruit Feather 32u4 Bluefruit with BLE mode of communication and have decided to move games to mobile platforms. The USB cable connection for operation with games on PC is not abandoned, will be kept as an option.

Source code must be updated on GitHub under GNU General Public License.

Setting up the programming environment for the Feather Bluefruit LE 32u4 is well covered in the section above and you can also consult on Adafruit.

Output: absolute pressure in cm of water, from 0 to 20.

Firmware for USB game

```
/*************************************************************
 *                    Breathing Games                        *
 *                                                           *
 *          Peripheral device sketch on Arduino Nano         *
 *            written by Jim Anastassiou     v 0.2          *
 *************************************************************/

int sensorPin = A1;     // select the input pin for the pressure sensor
int sensorValue = 0;    // variable to store the value coming from the sensor
int scaledPressure = 0; // the output of the device remapped from 1 - 100
int pinBrightness = 0;  // pin brightness variable
int ledPin = 9;         // the pin that the LED is attached to

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop() {

// read the value from the sensor:
sensorValue = analogRead(sensorPin);
scaledPressure = constrain(map(sensorValue,140,580,0,100),0,100);
//Serial.print(sensorValue);
//Serial.print(",");
Serial.println(scaledPressure);
//Serial.print(",");
//Serial.println(pinBrightness);
switch (scaledPressure){
  case 0 ... 10:
```

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

```
    pinBrightness = 0;
    break;
  case 11 ... 25:
    pinBrightness = 15;
    break;
  case 26 ... 50:
    pinBrightness = 50;
    break;
  case 51 ... 75:
    pinBrightness = 125;
    break;
  case 76 ... 100:
    pinBrightness = 255;
    break;
  }
analogWrite(ledPin,pinBrightness);
delay(10);

}
```

See this code on Gitlab, make sure that you keep it up to date.

Connect to Gitlab

Repo: https://gitlab.com/breathinggames/bg_chu_pep_tools

You can determine if Git is already installed on your computer by opening a terminal and running this command:

git --version

If your computer doesn't recognize git as a command, you must install Git.
https://git-scm.com/

Configure Git
To start using Git from your computer, you must enter your credentials. The username and email address should match the ones you use in GitLab, so create a GitLab account if you don't have one yet.
https://about.gitlab.com/

In your shell, add your user name:
git config --global user.name "your_username"

Add your email address:
git config --global user.email "your_email_address@example.com"

To check the configuration, in the terminal run:

git config --global --list

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

The *--global* option tells Git to always use this information for anything you do on your system. If you omit *--global* or use *--local*, the configuration applies only to the current repository.

To clone the project:

- On your computer, create an empty folder
- Download Git Bash => https://git-scm.com/downloads
- Open the command line (Git Bash) and go to the empty folder you just created. One quick trick to do so, in the address bar of the empty folder, write git and the option to open git.ink should appear.
- Clone the project => git clone git@gitlab.com:breathinggames/bg_chu_pep_tools.git

To check for updates, from the git bash command line:
1: *git status* (this will tell you if your version is up to date
2: if it isn't => *git pull*

Firmware for Adafruit Feather 32u4 Bluefruit with BLE wireless connection mobile

The current stakeholders of Breathing Games (project lead network organization), i.e. Sensorica (hardware design, trust economy and regulatory approval lead organization) and Sainte Justine Hospital (sponsor and early adopter) have adopted the Adafruit Feather 32u4 Bluefruit with BLE mode of communication and have decided to move games to mobile platforms. The USB cable connection for operation with games on PC is not abandoned, and will be kept as an option.

Expanding Game to BLE wireless operation

Porting a desktop game to mobile in Unity requires careful consideration of several factors to ensure a smooth transition and optimal user experience. Here are some important considerations:

1. **Performance Optimization:**
    - Mobile devices have limited resources compared to desktops. We can optimize our game for performance by reducing draw calls, optimizing meshes, using efficient shaders, and minimizing the use of expensive effects.
2. **UI/UX Adaptation:**
    - Mobile devices have smaller screens and different input methods (touchscreen instead of mouse and keyboard). The UI must be designed to fit smaller screens and make touch interactions intuitive and responsive. It should already work fine for the most

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

part, but this is a case-by-case approach and will require a lot of testing on different mobile platforms (Android, IOS, tablets)

3. **Control Scheme:**
    - Desktop games often use keyboard and mouse controls, while mobile games typically rely on touch gestures. Redesign our game's controls to work well with touchscreen inputs, ensuring they are comfortable and intuitive for mobile users.
4. **Aspect Ratio and Resolution:**
    - Mobile devices come in various screen sizes and aspect ratios. Ensure our game supports different resolutions and aspect ratios to provide a consistent experience across devices.
5. **Input Handling:**
    - Mobile devices offer various input methods, including touch, tilt, and accelerometer. Adjust our game mechanics to take advantage of these input methods or provide alternatives for mobile users.
6. **Memory Management:**
    - Mobile devices have limited memory compared to desktops. We need to optimize memory usage by reducing texture sizes, audio compression, and minimizing memory leaks to ensure smooth performance on mobile devices.
7. **Battery Consumption:**
    - Mobile games should be energy-efficient to avoid draining the device's battery quickly. Minimize CPU and GPU usage, optimize rendering, and reduce background processes to improve battery life.
8. **Platform-Specific Considerations:**
    - Each mobile platform (iOS, Android) has its own guidelines and requirements. We need to familiarize ourself with platform-specific features, APIs, and performance considerations to ensure compatibility and optimal performance on each platform. At the same time, the dev who will join us has to already know Unity as well as mobile development.
9. **Testing and Iteration:**
    - Test our game extensively on various mobile devices to identify and address compatibility issues, performance bottlenecks, and usability problems. Iterate on our design and implementation based on user feedback and testing results. QA is a large part of game development and if we go to mobile, we will need testers specialized in platform specific FQA. A Testrail composed of many different test cases as well as a DB for bug entry like airtable or jira is required going forward
10. **Monetization and Distribution:**
    - This does not really apply to us since we are not selling the game.

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

By carefully considering these factors and adapting our game accordingly, we can successfully port our desktop game to mobile and provide an enjoyable experience for mobile users.

From Yannick (February 2024)

> *Here are the main challenges for porting desktop games to mobile. Also, I have fragmented the game in 3 major parts from the start to ensure smooth transition in case we made the switch to Bluetooth earlier than expected.*
>
> *1: Core module - I mainly focus on this part and this is where the assets bought from the asset store are used. We cannot share that part (project folder) because if we do we need to transfer my personal license to the CHU for a commercial one and that will increase the cost of everything. We do not need to do that. The build done with it is freely distributable. We just can't share the assets in this part. Every parameter, parental control, data extraction, credentials for login, save, load, internal relational DB that can be extracted to an external one later, online features for later, localization, dialogue system and voice over system, etc. are here.*
>
> *2: Integration module - The part that connects what I do to what everyone else will do. I need to make that part before we start recruiting people. It is only designed, not implemented. This part is mainly code that links everything together and requires a very specific folder structure to allow the integration in the core system everything that will be done with the last part. We can share this but it makes no sense to do that since this is mainly code.*
>
> *3: PEP and Arduino module - this is everything about hardware, the controllers for the PEP games, the pep games themselves, the firmware and if the games are desktop or mobile. This is where everyone else will be working with the parents and the kids. The template will be the same for everyone and will be made so it can be extracted as a unity package and integrated in part 2 without any extra work from me. We can share 100% of everything here. No complex data manipulation is required here since my template will already link what is here to part 1. Our developers will focus mainly on making fun games with family and kids and doctors with hardware for specific platforms.*

Firmware (works with BLE and USB)

```
#include <Arduino.h> #include <SPI.h>

#include <SoftwareSerial.h>

#include "Adafruit_BLE.h"

#include "Adafruit_BluefruitLE_SPI.h"

#include "BluefruitConfig.h"

// Create the bluefruit object

Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS, BLUEFRUIT_SPI_IRQ,
BLUEFRUIT_SPI_RST);
```

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

```
int sensorPin = A1; // Input pin for the pressure sensor
int sensorValue = 0;
int scaledPressure = 0; // the output of the device remapped from 1 - 100
void setup() {
// Initialize Serial for debugging
Serial.begin(115200);
// Initialize the BLE module
if (!ble.begin(false)) {
while (1);
 }
  // Perform a factory reset to ensure a clean state
ble.factoryReset();
// Set BLE device name
ble.sendCommandCheckOK(F("AT+GAPDEVNAME=FeatherBlue"));
// Set the mode to data mode (as opposed to command mode)
ble.setMode(BLUEFRUIT_MODE_DATA);
// Start advertising
ble.sendCommandCheckOK(F("AT+GAPSTARTADV"));
}
void loop() {
// Check if BLE is connected
if (ble.isConnected())
{
// Read analog value from A1 pin
sensorValue = analogRead(sensorPin);
scaledPressure = constrain(map(sensorValue,110,550,0,100),0,100);
// Send the sensor value as a string over BLE
ble.print(scaledPressure);
// Small delay to prevent flooding
delay(500);
}
// Always send data via Serial
```

Content on this document is *Creative Commons (BY SA)*.

Log Contributions

```
sensorValue = analogRead(sensorPin);
scaledPressure = constrain(map(sensorValue,110,550,0,100),0,100);
Serial.println(scaledPressure);
}
```

From Tibi (February 2024)

> For the hardware, we just need to integrate the battery and a magnetic switch to operate in battery mode or wire mode. All the parts have been purchased during summer 2023 (this version), battery and switch. They just need to be connected to the board. This requires opening the box and making the wire connections. The new box was designed last summer to receive the battery and the magnetic switch can only be glued to the box.
>
> **In short**, just a bit of labor. I can do an estimate for the first 2 devices and for the rest of the devices.
>
> **Question**: Do we need new devices?

Data management

Open document

Identification of device



Need a way to identify every device. Should we extend the identification to important parts of the device? Perhaps a corresponding number should be assigned to a device and to the pressure sensor to which the device belongs, since the pressure sensor is the medically relevant piece, important for cross-contamination but also for consistency of data?

Log the stable or permanent solution on Gitlab to close the issue.
This solution must become part of the **DIY Instructible 2023 doc** and DIY instructables - BG pressure device 2024.

The idea was to print some information on a piece of material and fix it on the pressure sensor box. The organic box was used. We considered heating to insert a metal plate into the plastic walls, but didn't try it because it seemed not too well suited. In the end, we opted for using transparent two parts epoxy or resin.

An image was produced using Inkscape, as an svg file

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions



In terms of materials, we considered the following:

1. **Laser printing on aluminum tape:** Works pretty well.
2. **Laser printing on copper tape:** Doesn't work well.
3. **Laser printing on transparent tape:** Works well

We tried 3, which doesn't really work well. For fixing to the box, the transparent tape ends up warping upwards when placed on top of the uncured resin, and it does the same even if we cover it with epoxy. This ended up being a tedious job, because one needs to push these ends down as the epoxy cures, to make sure that they don't stick up too much, and touching the tape all repetitively compromises it. Also, we see bubbles under the tape, since the surface of the box is not completely flat, it's 3D printed.

Regulatory compliance

Open the BG Regulatory Approval Doc

Validation of critical components

The idea is to list critical components of the pressure sensor and design electronic methods for validating their authenticity.

A critical component is one that can harm the patient. Components that have only two possible states, work/no_work and their failure cause a total failure of the device are not considered critical, since the device cannot be used if they fail. On the other hand, the pressure sensor can degrade, its sensitivity can change and provide an erroneous reading. Since the pressure is an important parameter in the PEP therapy, and since a malfunction of the pressure sensor could pass unnoticed by the patient, it is important to make sure that this component works properly.

One could rely on critical components by trusting the specs from the manufacturer. One problem arises when a knockoff, low-cost version of the component is used, which can be less stable / reliable. Thus, it becomes important to provide the means to a user to verify the type of component used, after fabrication, in case the fabrication is performed by another party that may have an incentive to cheat, not respect the BOM associated with the device.

Content on this document is ***Creative Commons (BY SA)***.

Log Contributions

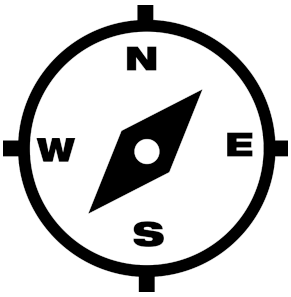
Message to NXP, fabricator of the MP3V5010DP pressure sensor

Content on this document is ***Creative Commons (BY SA)***.

Current development

In 2024, hardware development activities were coordinated within the Sensorica OVN. This is part of the Deliverables Work Package, more on the PEP Master venture's webpage.

You can engage in these activities by following the links below.

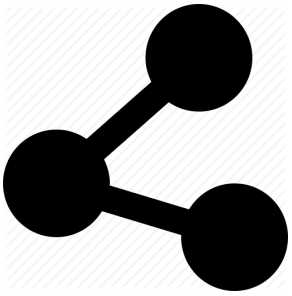


Get oriented



Discuss Collaborate


Document


Disseminate

Breathing Games Context

Since May 2014, various interfaces have been conceived to connect the positive expiratory pressure device (PEP) used in respiratory physiotherapy with a computer. A study was conducted among ten patients. We also built an alternative without device based on Flower for all, we developed a C# library, that captures with a micro / earphones set (iPhone) the sound of the PEP.

Here, we develop "Multi", a device for domestic use, which is going to be usable for different respiratory therapies with and without being connected to a computer.

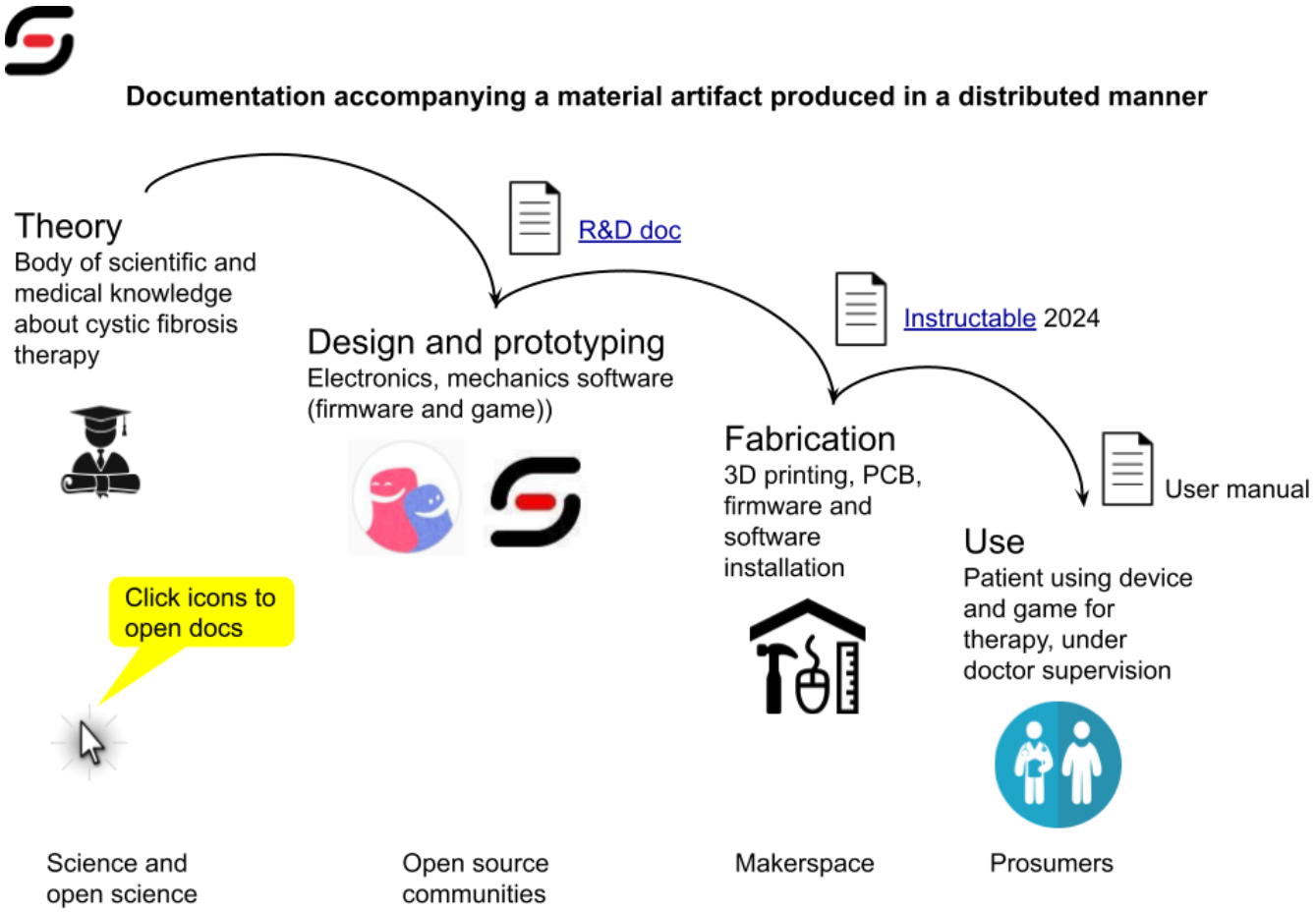
See BG-5-3 Flowmeter theory

Development ecosystem and documentation

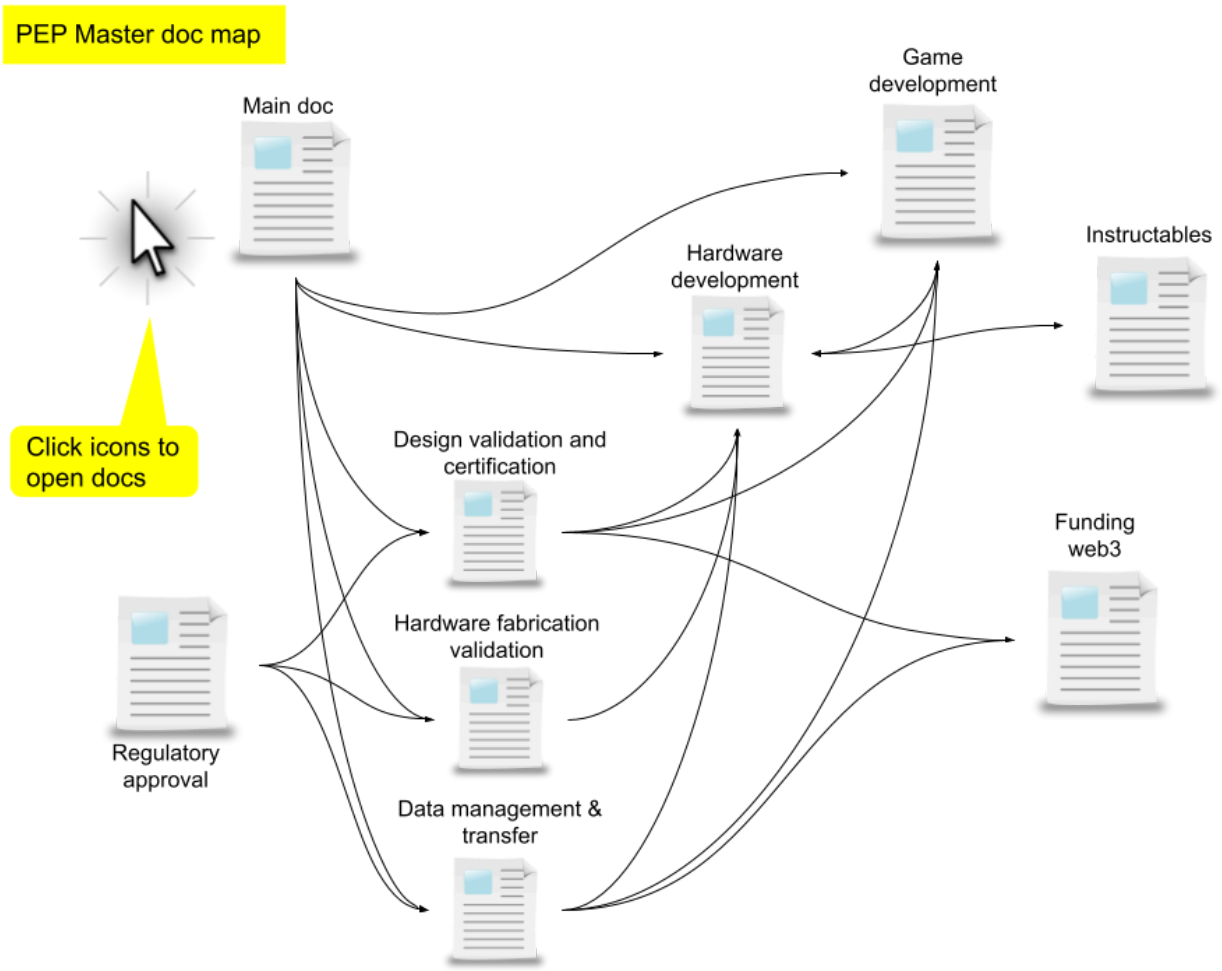
The schematic below presents various types of actors that engage in various types of activities, evolving an idea to a material artifact. This picture doesn't show everything, like all the feedback loops that connect these actors, for example the user being at the same time the fabricator and providing information about the design to designers. The goal here is not to present an exhaustive representation of the processes at play. The goal is only to show actors in relation with activities and the documentation that is produced in the process.

Learn more about documentation in the context of material peer production.

You can open the drawing below in a separate window and open the docs.



You are reading the R&D doc. The R&D activity is related to other development activities, as you can see in the drawing below. Open the drawing below in a separate window and open the docs.



BOM and prices

There are different possible versions of the pressure device, depending on sensors and power options. See complete inventory and BOM for every existing device version HERE.

You can order the device from Sensorica

Price for third party

Use this document to produce your own device. If you want the device to be produced by a third party, your local fablab or makerspace, the total cost of materials HERE, and add production costs. See example below:

Price > 10 units	$277.24
Description	Price
Production labor cost	136.50
Cost of materials for Sensorica MP3V5010DP pressure device	94.53
Production and materials sub-total	231.03
Revenue to innovators and creators - 15%	34.65
Custodian fees - 5%	11.55
price does not include your local taxes	
NOTE: Cost of parts is an example, change cost of parts cell from Overview sheet	

See also BG-8-1 Core document healthcare.

To the BOM, you might want to add the following items.

Part name	Function and description	Links	CAD
Med apparatus	Go to BG-8-1 Core document healthcare	Pari PEP-System I / S Flutter / Pari O-PEP Acapella RC Cornet with adapter Quake	21.00 80.00 45.00 95.00 86.00

ToDos

1 Design and test solution for sensor degradation caused by water vapors penetrating to the sensor. See issue on Gitlab, address the issue as progress is made.

- Assess presence of water close to the sensor, in normal use
- Dissect sensor after a long period of use. Estimate use time
- Assess medical danger in current conditions
- Decide if extra measures of protection are needed based on the previous assessment.
- Parked solutions - use an easy to refill and replace table salt filter (a container containing grained salt, through which the air passes) that can be connected between the box and the external tubing

2. Serial number or other type of identification, on each device. Can it be a hardware fingerprint?

3. Self-check for hardware.

- Contact Bosh to discuss forensic ways to determine with high accuracy that their sensor is in use, without opening the box.

{Signalization tools}



{symbol for process/status updates - use this to signal important milestones in the process}



{symbol for notes - use this to post reminders or short messages for self or to collaborators}



{symbol for important information - use this to attract collaborators' attention}



{symbol for ToDos - use this to signal to your collaborators about what they can do}



{symbol for alternatives: enumerates possible solutions to consider}



{symbol for reasoning: presents arguments about possible choices}



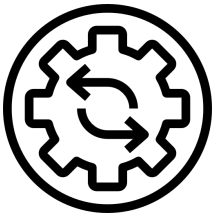
{symbol for Information: tells you how stuff works.}



{symbol for growing consensus: a summary of a section of this report}



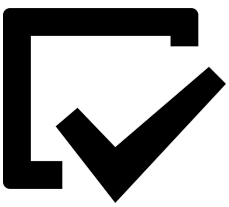
WP2: Capacity
“HR” and
additional funding



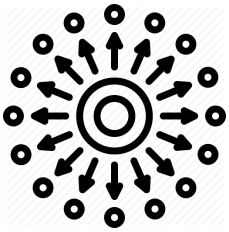
WP3: Process
“venture stearding”



WP4: Deliverables
design and
prototyping



WP5: Wrapping up



WP6: Dissemination

Find icons from the noun project
https://thenounproject.com/