bela

EXPERIMENTER'S KIT

KICKSTARTER 2016

Dear Kickstarter Backer,

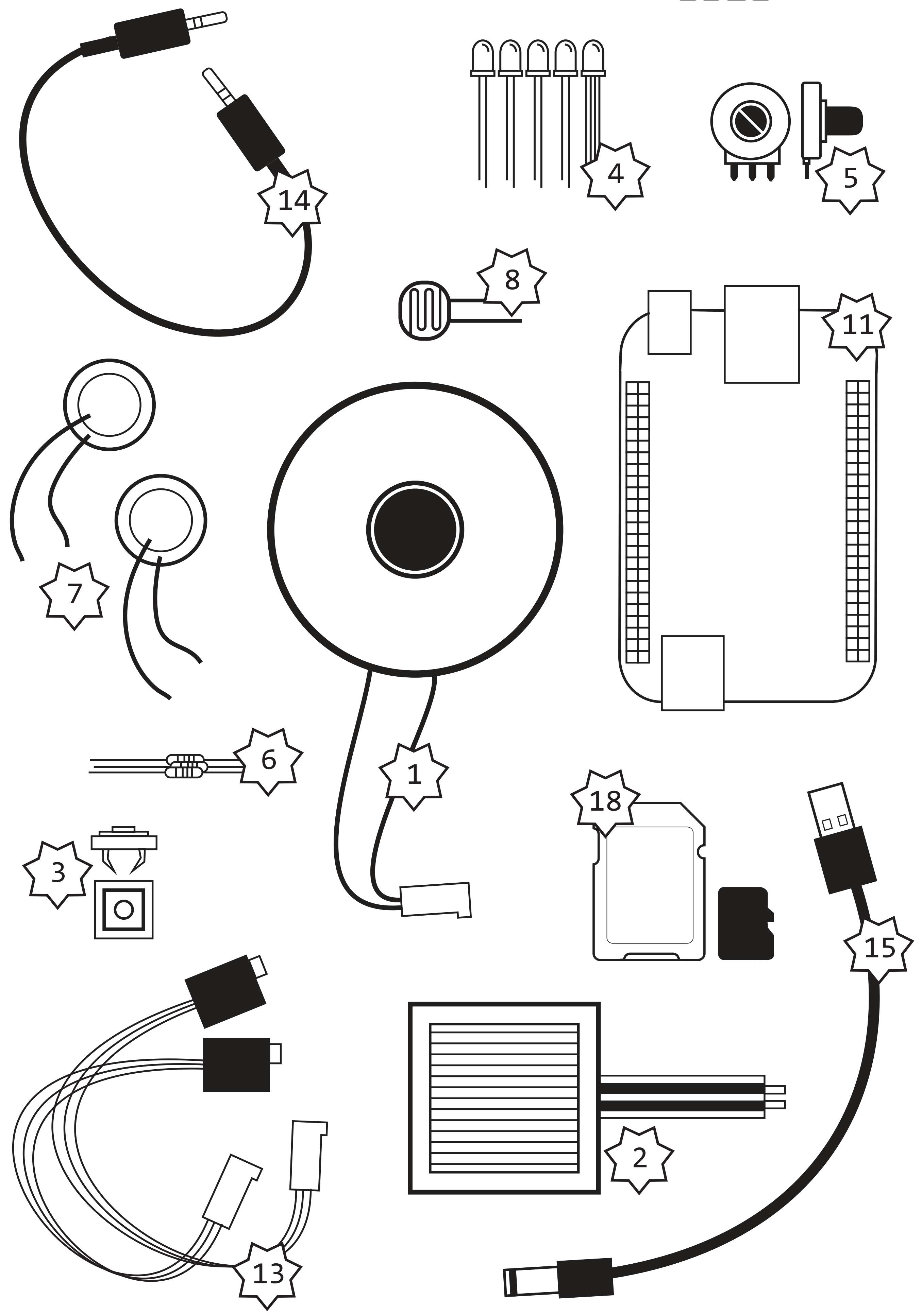
We are thrilled to deliver your **Bela Experimenter Kit**, the reward you selected during our Kickstarter campaign in March 2016. Over the past few months we have been building, developing, documenting and improving this platform, and we are proud to present the latest version of Bela to our Kickstarter backers.

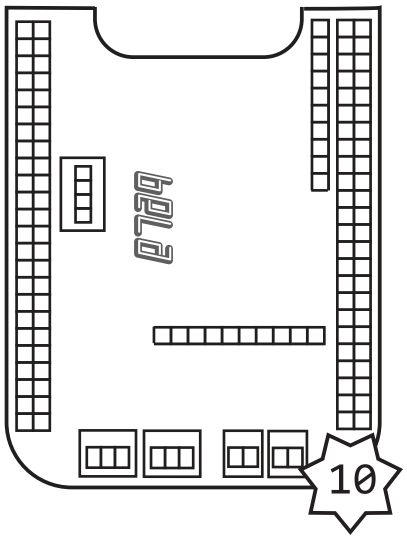
In this kit you will find a wide array of sensors and add-ons, as well as everything you need to make projects with Bela (see a complete list of what's included on the next pages). In this book you will find out how to get up and running with the hardware and running your code. We have also included a wide array of example projects and diagrams to get you started.

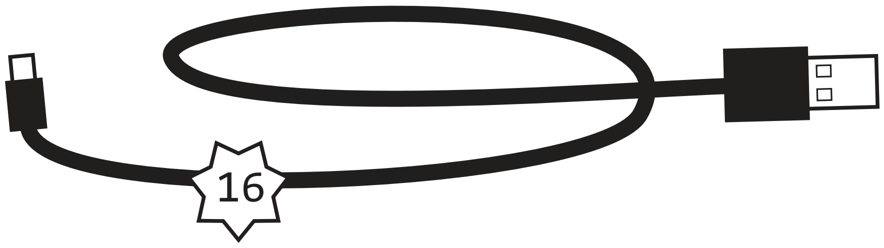
We encourage you to join our Forum (**forum.bela.io**), where you can get support, ask questions, and show off what you've built. You can also find all of our code and examples on Github (**bela.io/code**).

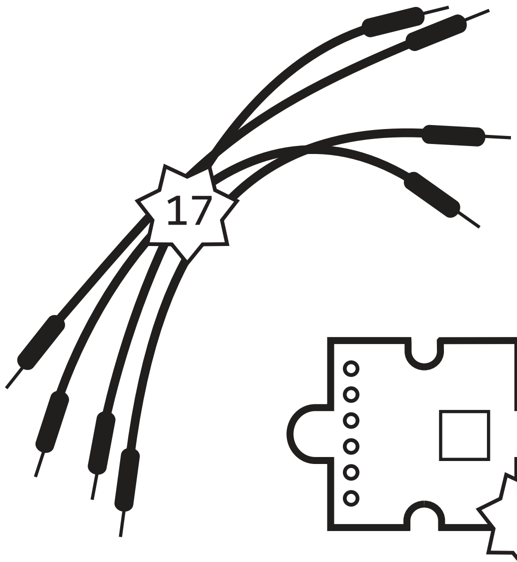
We hope you enjoy Bela as much as we've enjoyed making it.

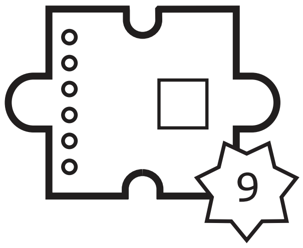
Your friends at
The Augmented Instruments Laboratory
(Andrew, Liam, Astrid, Giulio, Laurel, Chris & Robert)

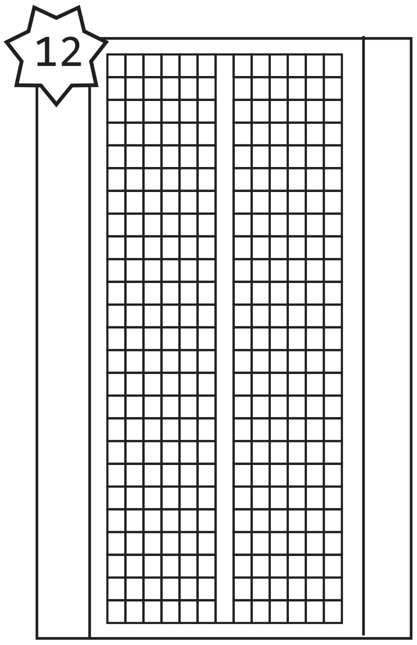

14
4
5
8
11
7
1
6
3
18
15
13
2











WHAT'S INSIDE:

1. 8 Ω Speaker
2. Force sensing resistor
3. Tactile buttons (2)
4. LEDs (2 x red/green /blue/yellow, 1 x RGB)
5. Potentiometers (2)
6. Resistors (5 x 10KΩ, 10 x 220Ω)
7. Piezos (2)
8. Light dependent resistor
9. Accelerometer
10. Bela cape
11. BeagleBone Black
12. Breadboard
13. Audio adapter cables (2)
14. Mini jack cable
15. Barrel jack cable
16. USB power cable
17. Jumper wires
18. SD card + adapter

*Not pictured: acrylic base plate, screws, spacers.
Please not that due to availability your parts might look a little different from the illustrations.

Assembling Bela

The first thing we need to do is assemble the Bela kit. Find your Bela cape, your BeagleBone, your acrylic base plate, your screws and spacers, and your breadboard.

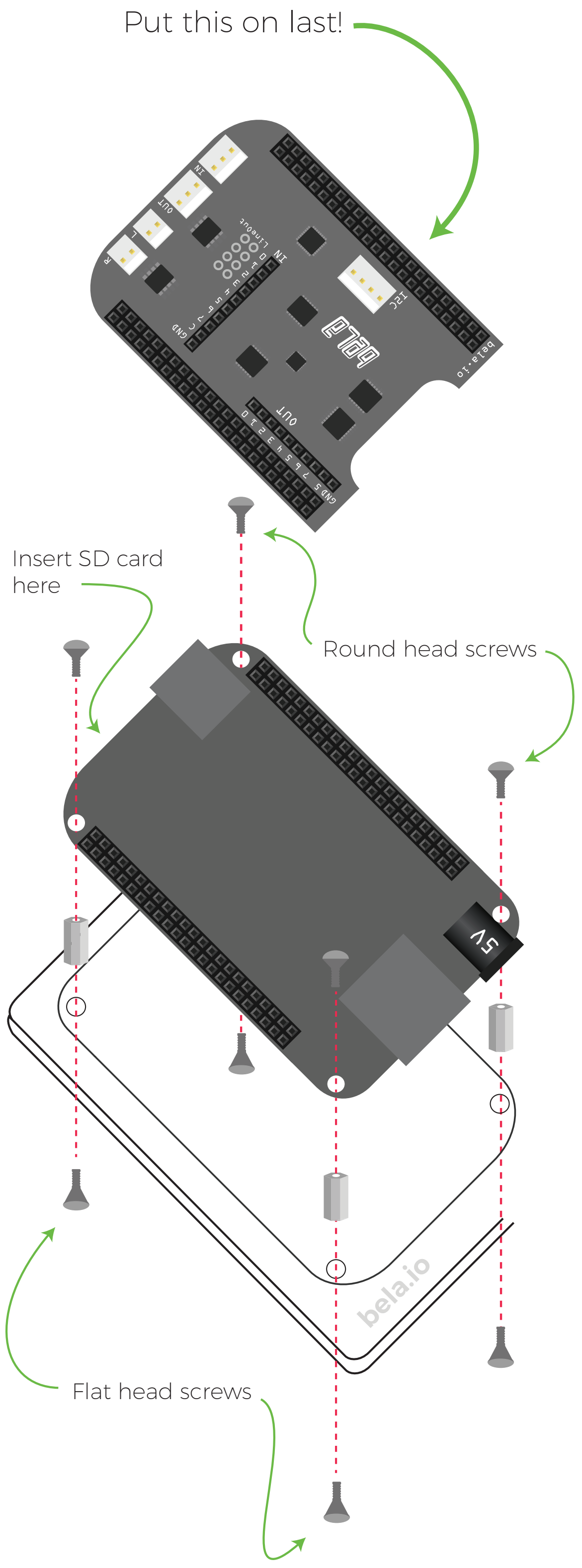
1. **Don't put your Bela cape on the BeagleBone straight away** - this makes things really hard later on. We'll do that last.

2. **Find the 4 flat head screws and the 4 hexagonal metal spacers.** Put the screws up through the bottom of the acrylic plate and affix the spacers on top.

3. **Then, locate the 4 round head screws.** Place your BeagleBone on top of the spacers, aligning the holes, and attach it to the base using the screws.

4. **Next, put the Bela cape in place.** Line up the pin headers and holes carefully and ease the cape in a little at a time.

5. **Finally, affix your breadboard.** Peel the adhesive off the back and affix to the right side of the acrylic plate beside your Bela setup. Now all you need to do is insert the SD card, and you're ready to go!



Running your first sketch

A sine tone is audio's Hello World.

1. Power it up.

Now that your Bela is assembled, attach one audio adapter cable to your audio out pins (see bela.io/belaDiagram if you can't find it) and plug in your headphones.

Then, plug the USB cable into Bela and connect to your laptop. Once iBela finished booting (this usually takes a couple of minutes) access the IP address of your board in a browser (we recommned using Google Chrome) by going to this address (the IDE will load automatically):

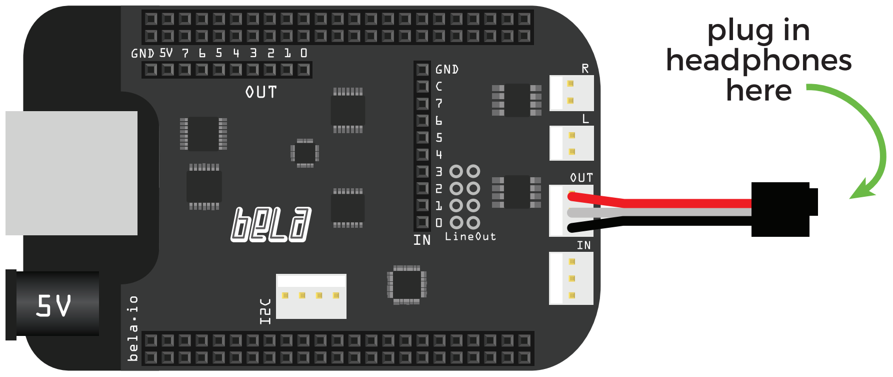
http://192.168.7.2

2. Open the example.

Open the tab on the right and click the Examples tab (look for this symbol: 💡) and choose the project 01-basics -> sinetone.

3. Run the code.

Click the Run button in the bottom left ⟳. It will turn yellow while it's building (this may take a minute on the first go), and then turns green when it's working. If you hear a sine tone, congratulations! Everything is working and you're ready to build some example projects.



A tour of the IDE

Write code, right in your browser.

One of Bela's unique features is an IDE (Integrated Development Environment) that you access through a browser window. No internet connection is required; the IDE runs on the board, we just use the browser to access it. (We recommend using the latest version of Chrome right now; we're working on cross browser compatability but that's still in progress, and we can't guarantee functionality in other browsers.) To access the IDE, go to:

http://192.168.7.2

We will briefly explain the useful stuff in the tabs, but first, a couple of features for your enjoyment:

Your code is automatically saved. Every time you stop typing, your work is saved to the board. No need for ctrl+s, and when you open up another project your last one is saved exactly as you left it.

Constant syntax checking. At the right of the toolbar you'll see a status indicator: ✓ for ok, ⋯ for in progress, and ✕ for error. These indcate the stauts of your code, which is checked every time you stop typing, and will tell you if your code is ok to run, or if there's a problem with it. (If there is a problem, a red X in the left

A toolbar integrated into your workflow. The toolbar and its functionality - running/stopping your code, status, and so on - is in the middle of the screen, so you don't have to mouse up to the top of the window to do the things you need to do most often.

Download your project, any time. By clicking the Download button, your current project is saved to your computer as a zip file. (We recommend downloading your code regularly in case of SD card failures).

Project explorer

This tab allows you to manage the files of your current project (make a new file, upload/download/rename/delete files) and make new projects. It also lists the files contained in your current project, and you can open these files in the editor by clicking on the file name. Further, you can initialise a Git repo on the board for your project, and make commits and send Git commands straight from the interface.

Examples

We have loaded Bela with lots of example projects. which provide templates and techniques for programming Bela with C++. There are also Pd examples, and although you can't edit Pd patches in the IDE, you can click on them and they will open an image in the editor, showing you which objects are used and how the patch works.

Settings

The settings tab provides an easy way to adjust the settings for your project, from basic things like headphone level to advanced settings like make parameters, command line arguments, and audio block size. You can also download all your projects with one click.

Pin Diagram

The interactive pin diagram highlights active pins and connectors, and you can see what they are by mousing over them. No more guesswork, no more cheat sheets.

Documentation

The Bela API uses three basic functions - setup(), render() and cleanup() - and you can find explanations of these, plus the functions used for things like addressing analog and digital pins. The Full Documentation button opens our complete documentation in a new browser tab.

Getting started with Pd

Bela: Not just for C++ anymore.

As well as C++, Bela can also run patches created with Pure Data.

Pure Data, or Pd, is a graphical programming language. We'll describe how to run patches with Libpd here, but you can also use Enzien Audio's Heavy audio compiler (see enzienaudio.com for details). Libpd is a GUI-less version of Pd which allows you to embed Pd patches into other programs. Libpd runs all of Pd Vanilla objects, almost all of which are supported on Bela. Here's how to use it:

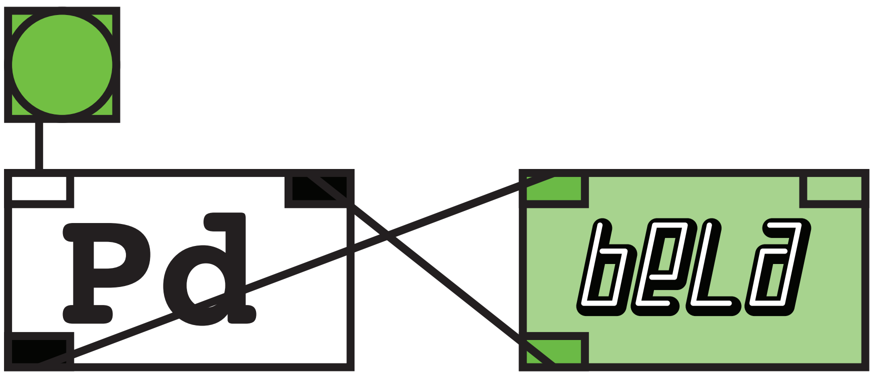
1. Download Pd Vanilla (http://puredata.info/downloads) and make an example patch, saving it as _main.pd.

2. In the in-browser IDE, go to File Explorer -> New Project

3. A dialogue box will ask you if you want to create a C++ or Pure Data project. Choose Pure Data, and give your project a name.

4. Now you can drag and drop your Pd patch into the browser and click the run button to run it!

Visit our Github wiki to read more about running Pd patches on Bela, and how to use Pd to address analog and digital pins.



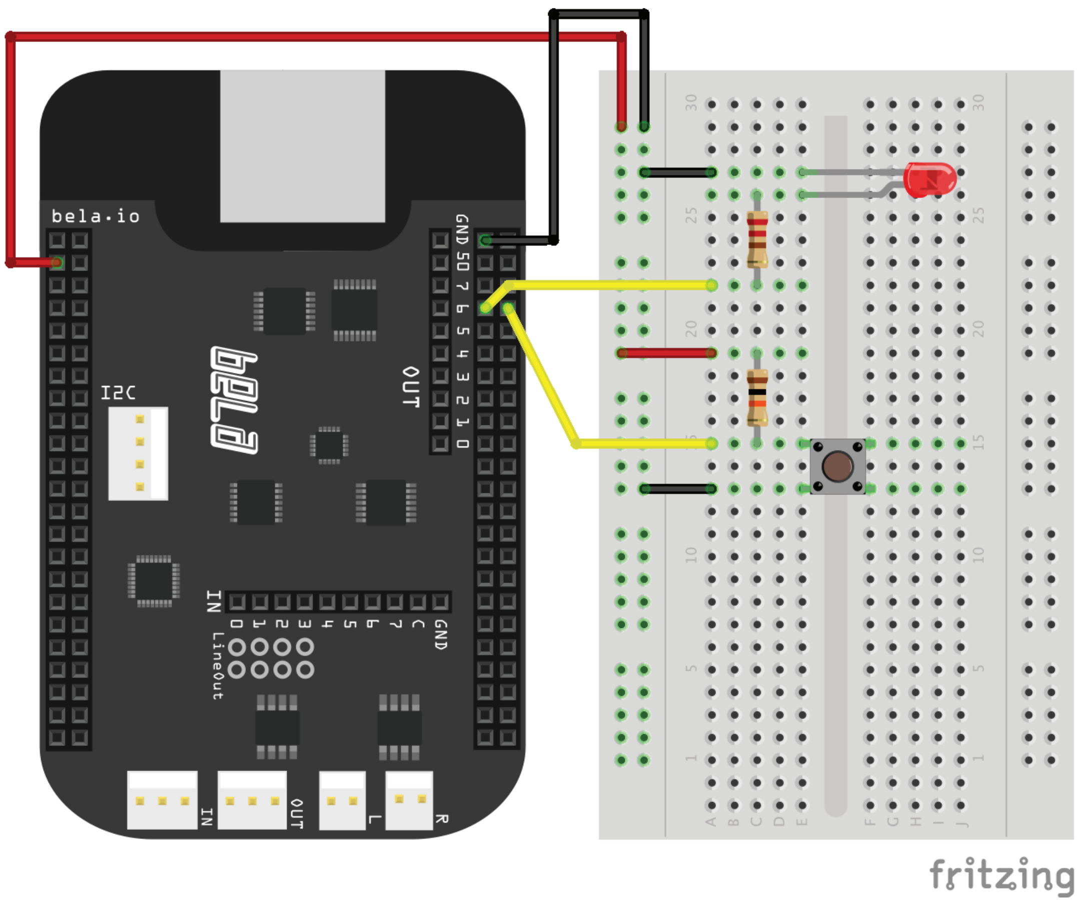
Using a switch

Using a digital switch to turn an LED on and off
Found in Examples -> 02-Digital -> digital-input

How it works:
This example brings together digital input and digital output. The program will read a button and turn the LED on and off according to the state of the button.

Code:
Before using the digital pins we need to set whether they are input or output. This is done via pinMode(context, 0, P8_08, INPUT);

Note that there are two ways of specifying the digital pin: using the GPIO label (e.g. P8_07), or using the digital IO index (e.g. 0)

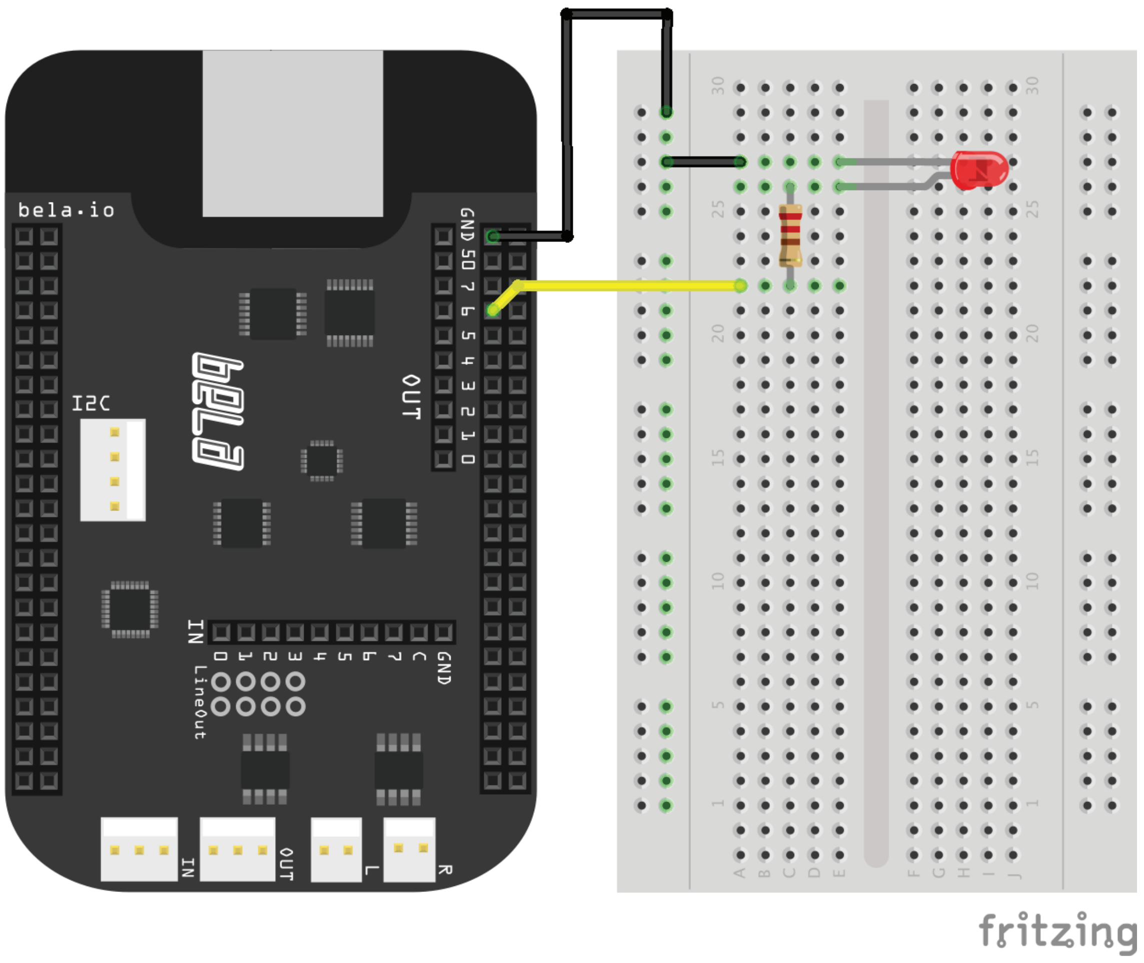


Blinking an LED

Blinking an LED on and off with digital pins.
Found in Examples -> 02-Digital -> digital-output

How it works:
The LED is set to blink on and off by setting the digital pin HIGH and LOW at regular intervals in render()

Code:
In setup() the pin mode must be set to output mode via pinMode(). For example: pinMode(context, 0, P8_07, OUTPUT). In render() the output of the digital pins is set by digitalWrite(). For example: digitalWrite(context, n, P8_07, status) where status can be equal to either HIGH or LOW. When set HIGH the pin will give 3.3V, when set to LOW 0V.



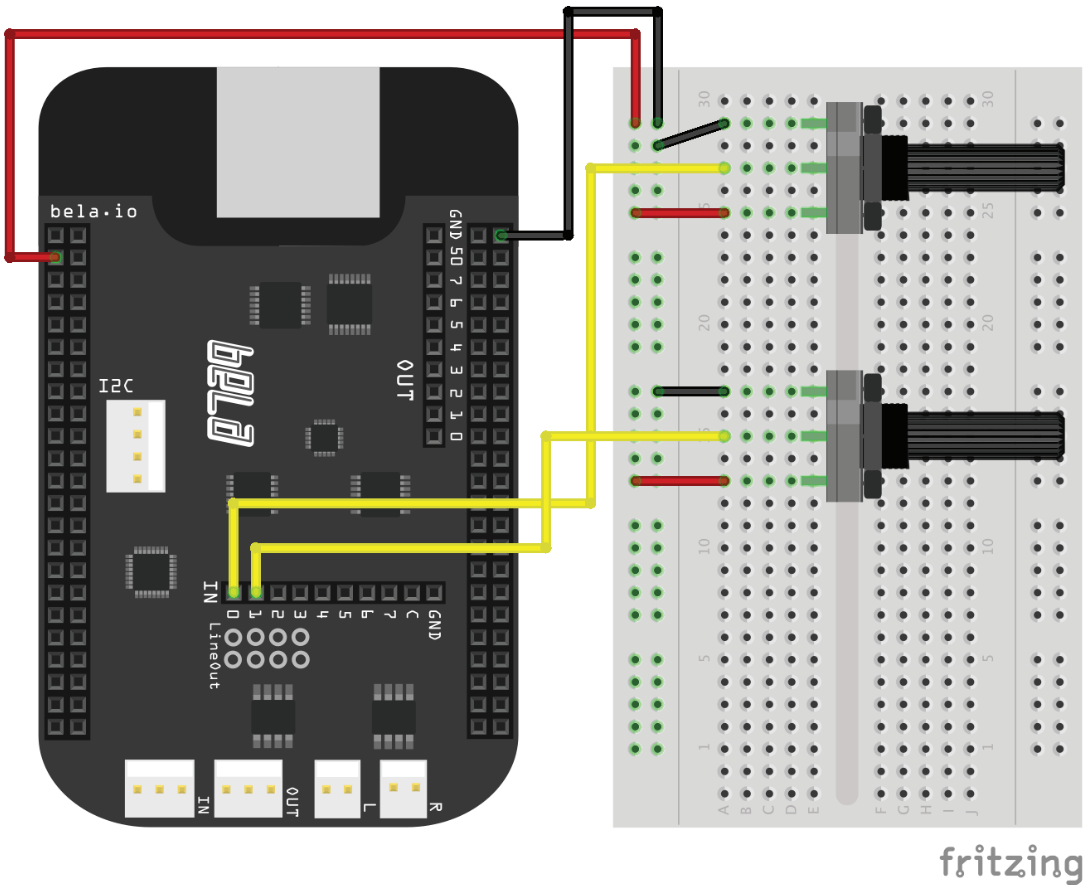
Using Potentiometers

Controlling analog inputs using potentiometers
Found in Examples -> 03-Analog -> analog-input

How it works:
This sketch produces a sine tone. Its frequency and amplitude are modulated by data received on the analog input pins.

Code:
Before looping through each audio frame, we declare a value for the frequency and amplitude of our sine tone. We adjust these values by taking in data from analog sensors (in this case, potentiometers) with analogRead().

The important thing to notice is that audio is sampled twice as often as analog data. The audio sampling rate is 44.1kHz (44100 frames per second) and the analog sampling rate is 22.05kHz (22050 frames per second).

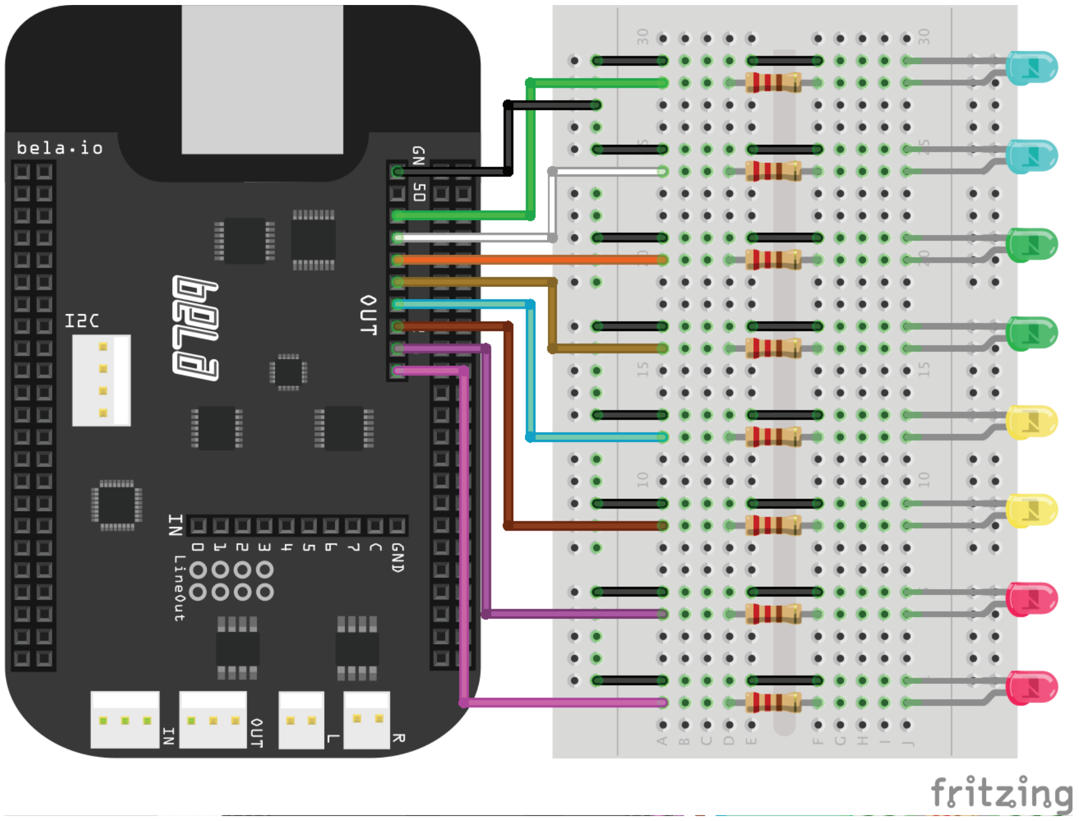


Fading LEDs

Using analog output to control the fade of LEDs
Found in Examples -> 03-Analog -> analog-output

How it works:
Each analog pin provides an output between 0 and 4V. If we vary this voltage up and down smoothly, we can fade LEDs accordingly.

Code:
The output on each pin is set with analogWrite() within the for loop that cycles through the analog output channels. This needs to be provided with arguments as follows analogWrite(context, n, channel, out). Channel is where the you give the address of the analog output pin (in this case we cycle through each pin address in the for loop), out is the variable that holds the desired output (in this case set by the sine wave) and n is the frame number (given by the outer for loop).

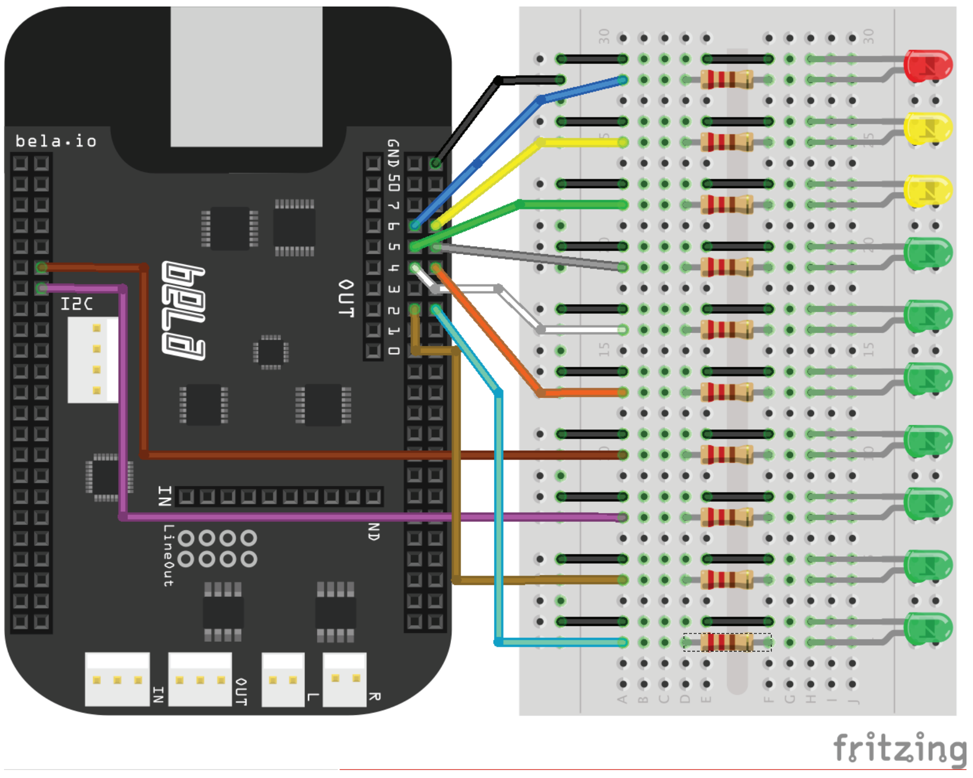


Level Meter

Visualising music with LEDs
Found in Examples -> 02-Digital -> level-meter

How it works:
LEDs light up sequentially with the level of the music, indicating its volume. Connect an audio adapter to the Audio In pins and use the audio cable to connect the adapter and your music source (such as a laptop).

Code:
The code performs peak detection on the audio input and lights the LEDs once the amplitude of the input signal passes a certain threshold. Each LED has its own threshold, in steps of 3dB. All digital pins are toggled on and off when the amplitude passes the threshold for that LED. The LEDs below the current peak value always remain lit to create a classic amplitude peak meter. The audio input is passed through to the output so you can listen as you watch the light show. (You can use any colours you like, and it's okay if you don't put in all 8 LEDs.)



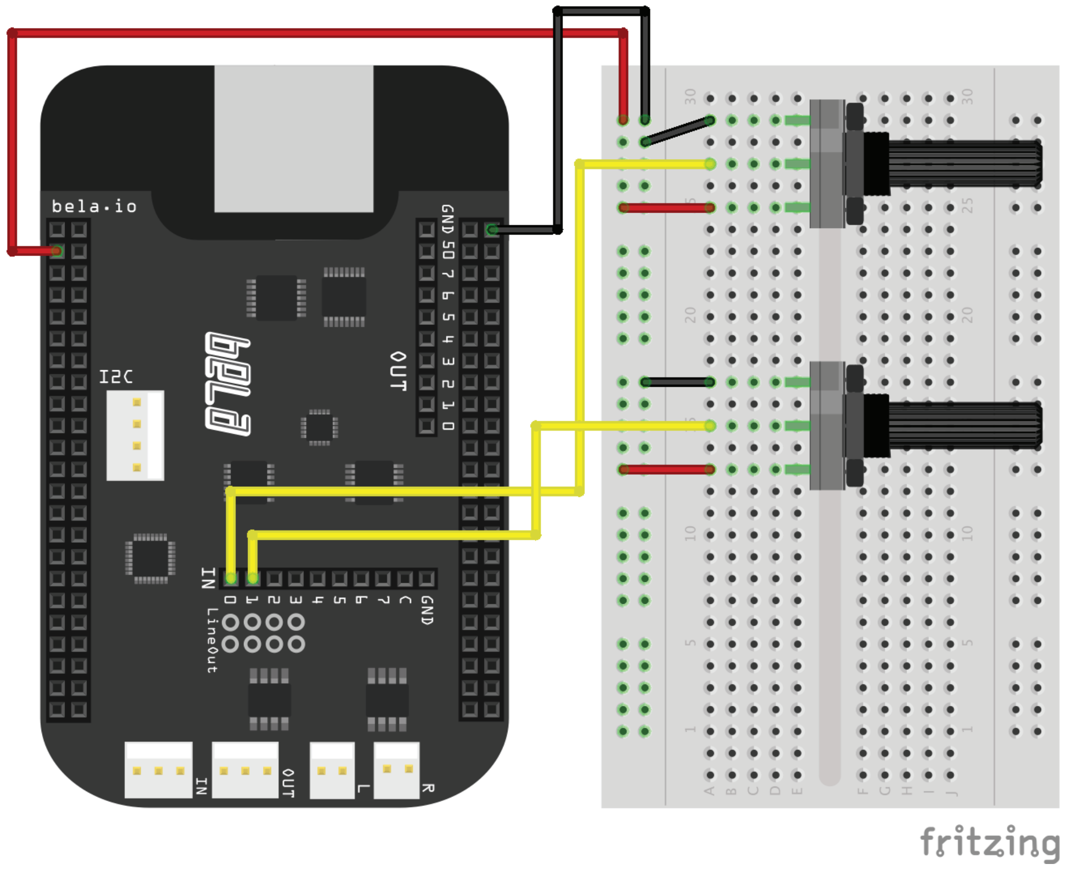
The oscilloscope

See what your signal is doing, right in your browser.

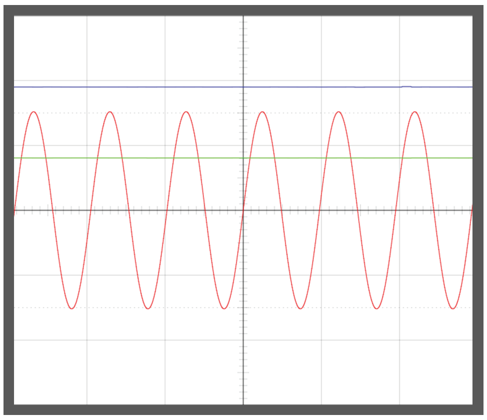
One of the Bela IDE's most exciting features is an in-browser oscilloscope. This allows you to see what the signal from your audio code or sensor is doing (or if there's any signal at all). We love oscilloscopes but they're also expensive, heavy, and can be hard to set up, so we built one that you can use in your browser window.

Build a two-potentiometer setup as in the diagram below, and open the example 03-Analog -> scope-analog. This sketch involve two potentiometers, one which controls the signal's frequency, and the other which controls the signal's amplitude.

Once you have the sketch running, open up the oscilloscope by clicking the Scope button in the toolbar ( ).



What we're looking at



On the left is a capture of the scope output (you can make a screen cap with the Save Image button). You can see the three lines change if you adjust the potentiometers.

In an oscilloscope, the horizontal axis is time and the vertical axis is signal amplitude. In our example, the potentiometers are the blue (pot 1) and green (pot 2) lines, and the red line is the generated sine wave. As we turn the potentiometers the green and blue lines move with their values, and affect the generated sine wave.

Here's how to plot your output to the oscilloscope:

1. Make a scope object.
At the top of the sketch we declare a scope object, and in setup() we determine how many inputs it will have (in our case, 3) and the sample rate at which the data should be logged.

2. Give the scope object signal to plot.
Our three signals - the sine wave, pot 1, and pot 2 - are in the render() function as out, gIn1, and gIn2. We log these to the scope with scope.log(out, gIn1, gIn2) to produce our visual output.

Explore the scope controls by adjusting features such as the colour of the lines representing each signal, as well as the horizontal offset and the y-axis amplitude.

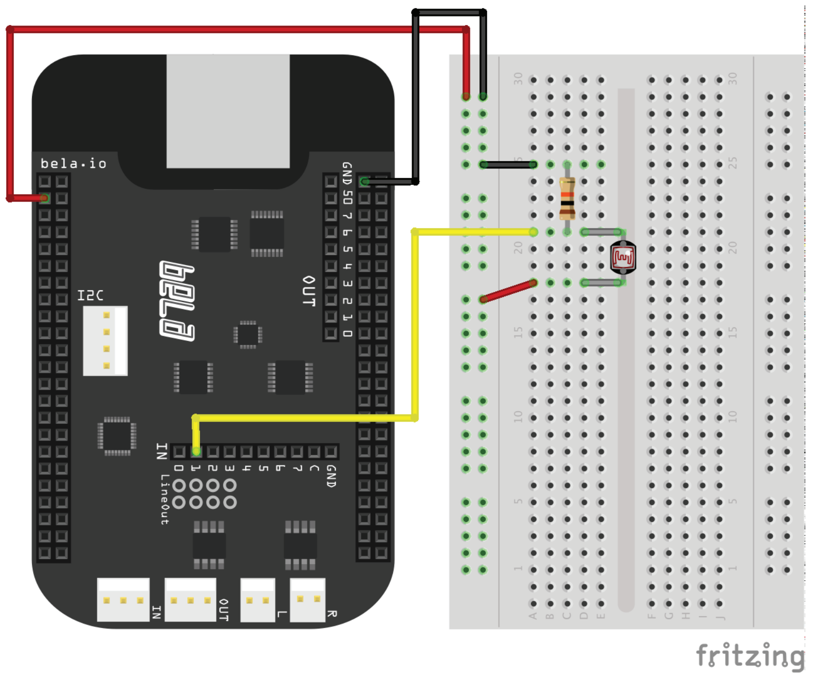
Using light as a controller

Control the volume of white noise with an LDR
Found in Examples -> 06-sensors -> LDR

How it works:
A light dependent resistor changes its resistance depending on how much light hits it. We can use this to move the amplitude of the white noise up or down.

Code:
In order to use the LDR as a volume control we need to set the thresholds of the resistor for ambient light, maximum light, and maximum darkness. There is a chunk of code in render(); uncomment it and run the sketch, and wave a hand over the LDR and shine a torch on it. The LDR values will be printed in the console; make note of the minimum and maximum values.

Then re-comment out that chunk, and update the gLight and gDark variables with the minimum and maximum values. Re-run the sketch, and control the noise by waving a hand over the LDR.



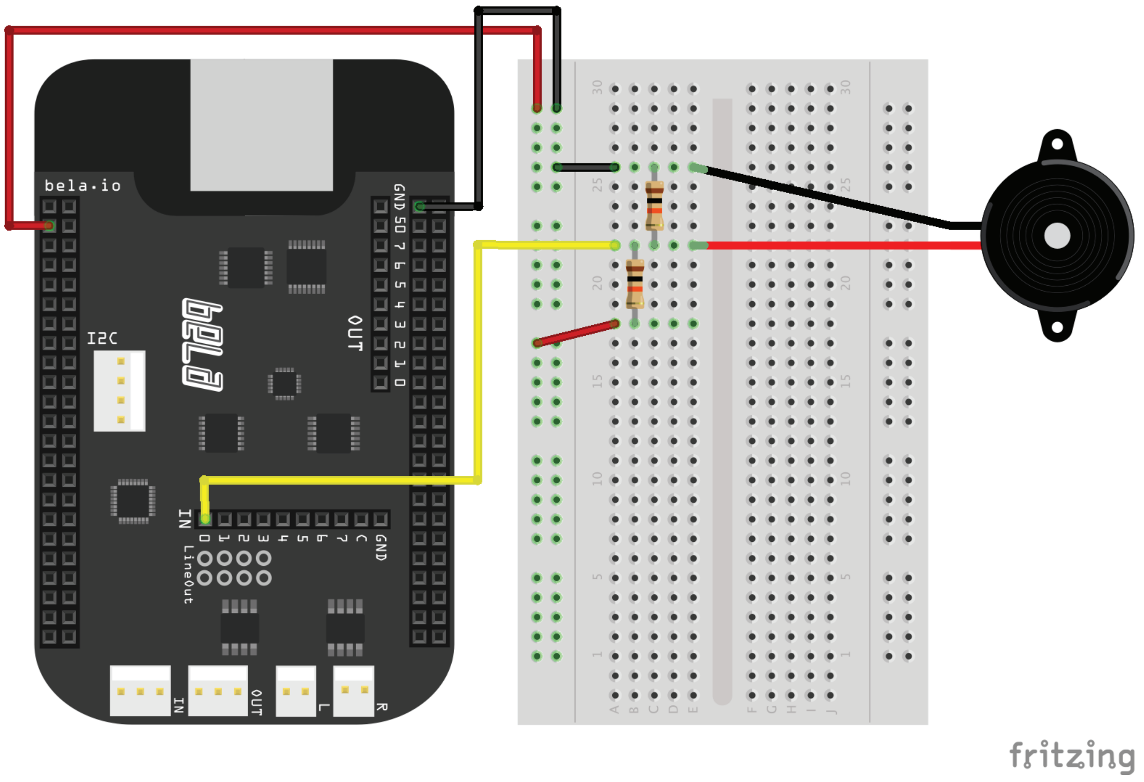
Piezo trigger

Trigger a WAV file sample with a piezo strike
Found in Examples -> 04-Audio -> sample-piezo-trigger

How it works:
An audio file is loaded into a buffer, and plays whenever the piezo detects a strike.

Code:
Getting coherent signal from the piezo element requires some signal conditioning. This is explained in detail in the project file comments.

In this example the sample is loaded into a buffer and then read with a read pointer, a variable that keeps track of where we are in the buffer. The read pointer is incremented every time and output sample is needed, advancing its position.

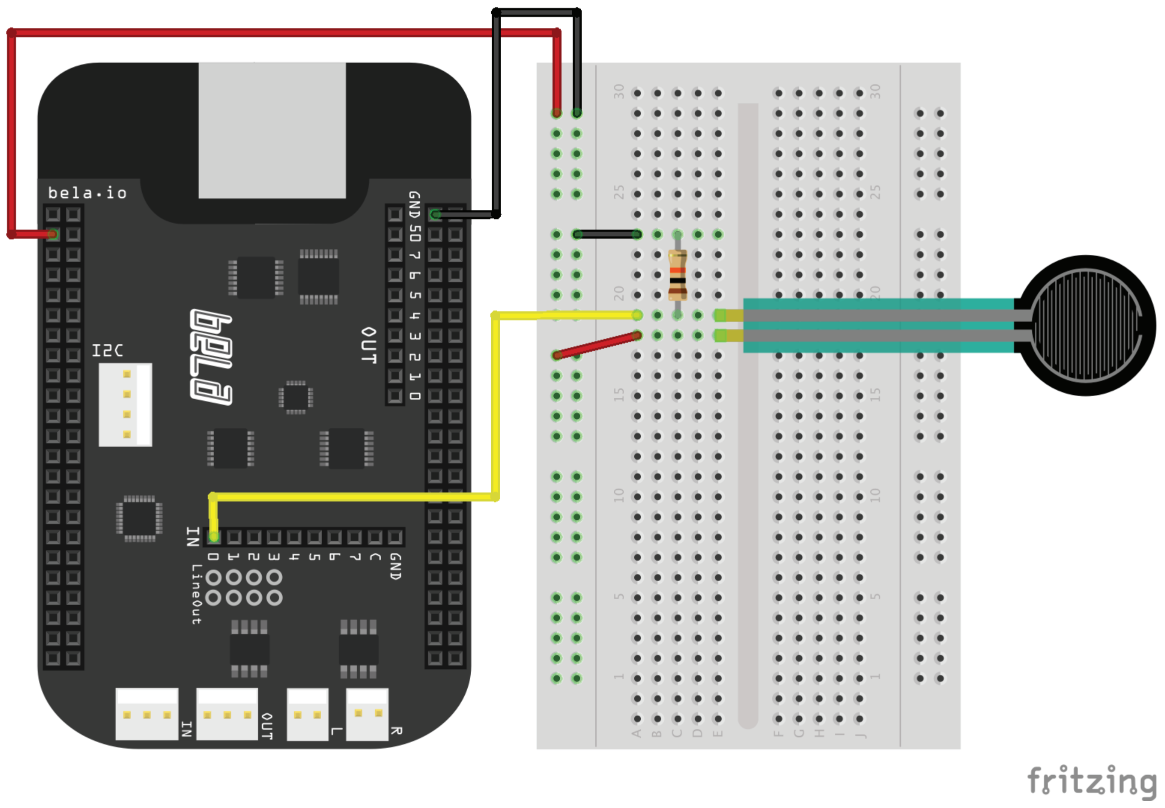


Rubber duckie

Control a squeaky sound model with an FSR
Found in Examples -> 08-PureData -> rubberDuckie

How it works:
Changing the resistance of the FSR by pressing on it changes the variables controlling the sound model, resulting in different duck sounds.

Code:
This is another example of a voltage divider circuit. As you press the FSR its resistance decreases, and we measure the change in voltage at the analog input. In the PureData patch we then calculate how much this signal changes and use this to drive the sound model. This gives the sound model a much more realistic response when the FSR is squeezed. Open up the patch in the IDE to find out more details about the sound model. Try removing the differential and listen to how it sounds.

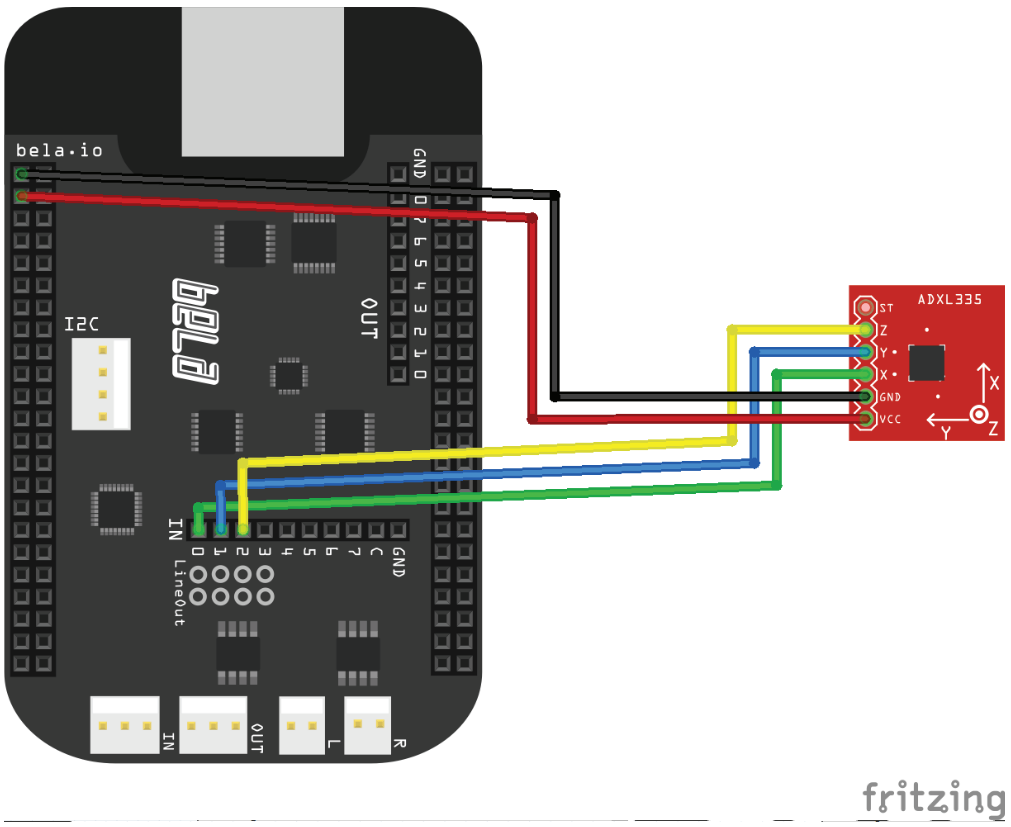


Air Harp

Hear plucked strings by moving an accelerometer.
Found in Examples -> 10-Instruments -> air-harp

How it works:
This project is a instrument that uses an accelerometer to sense movement. The movement of the accelerometer is used to excite a physical model of 9 strings.

Code:
As you tilt the accelerometer along one axis it will move a virtual mass across the strings, plucking each string as it passes. As well as tilting it you can try shaking it back and forth which will create a strumming effect like on a guitar. This shows the great variety of gestures that you can get from a simple accelerometer and also shows some techniques for using this sensor to control sound synthesis. Note that the panning of the strings changes as you move the sensor, high strings are in one channel and low strings in the other.



Using an external speaker

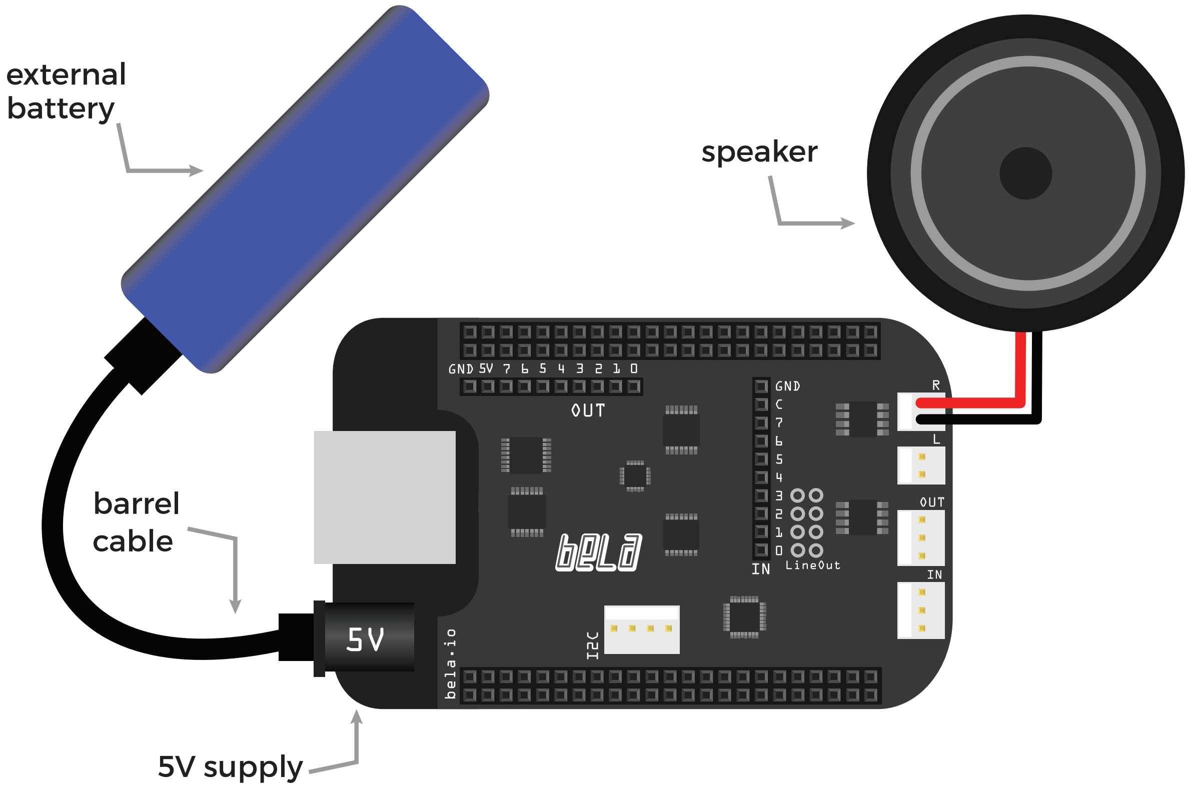
When headphones just aren't enough.

We have included an external 8Ω speaker in this kit so you can amplify the sounds you make with Bela. However, using this speaker requires a bit of a different setup.

To attach the speaker to Bela, attach the 2-pin connector to either one of the speaker headers (see bela.io/belaDiagram if you can't find it).

Next, you'll need an **external power source**. This is required because the speaker amplifiers require a higher current to work. We use external mobile phone batteries - these are inexpensive, widely available, and rechargable, but you could run Bela off any 5V battery source. This also means that you can run Bela without a laptop.

Once you're finished programming, go to Settings -> Run project on boot, and choose the sketch you'd like to run when Bela starts up. Then, detach Bela from the computer, attach to a battery via the 5V jack, and power it up - now Bela can be embedded almost anywhere, and can power its own speaker.



Before you go ...

Here's some useful links.

Bela code base on Github:

bela.io/code

Bela forum:

forum.bela.io

Our Getting Started guide:

bela.io/gettingStarted

An interactive Bela pin diagram:

bela.io/belaDiagram

IDE address:

http://192.168.7.2

bela.io

Bela is a project of the Augmented Instruments Laboratory, Centre for Digital Music, Queen Mary University of London. Bela is open source and licensed under LGPL.

bela.io