



*Open-source platform
for beautifully responsive
interactive audio*

EPSRC

centre for digital music



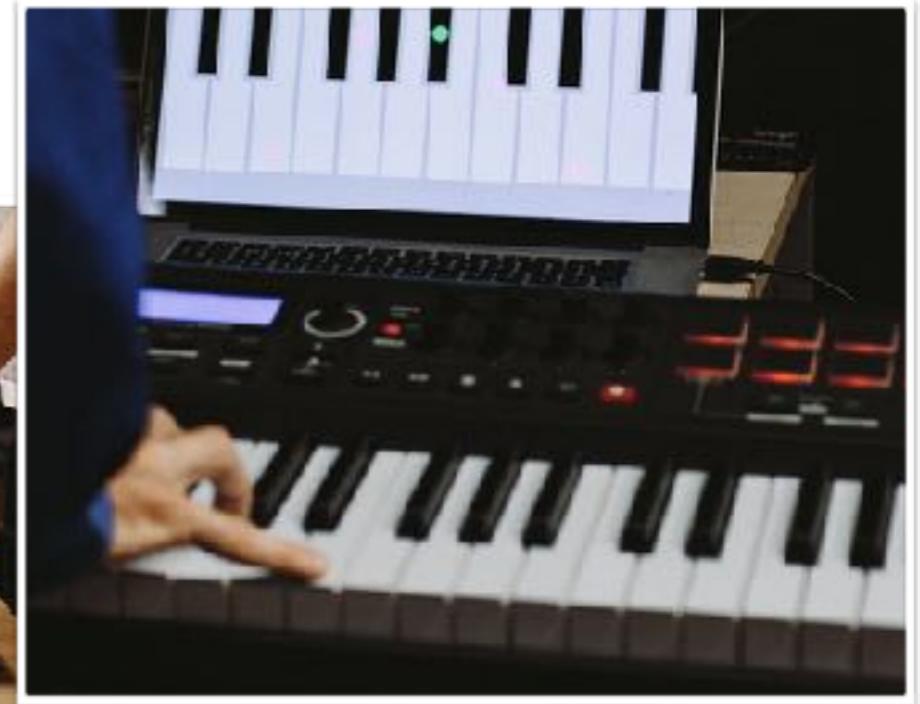
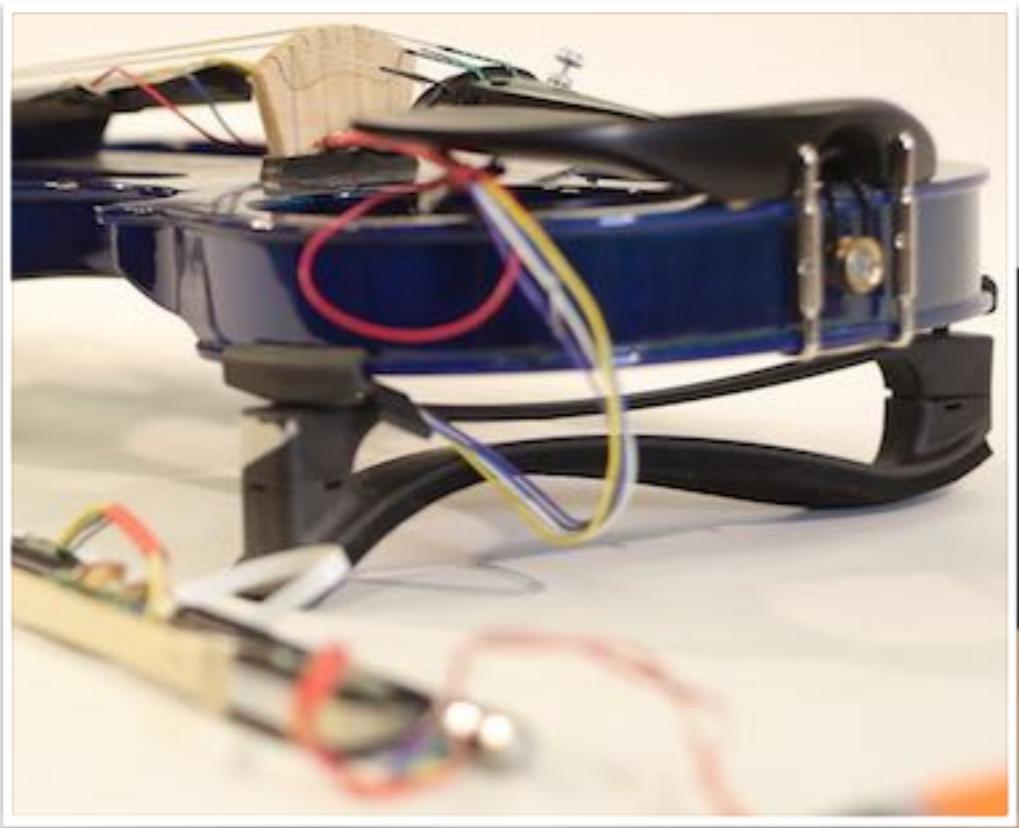
Queen Mary
University of London



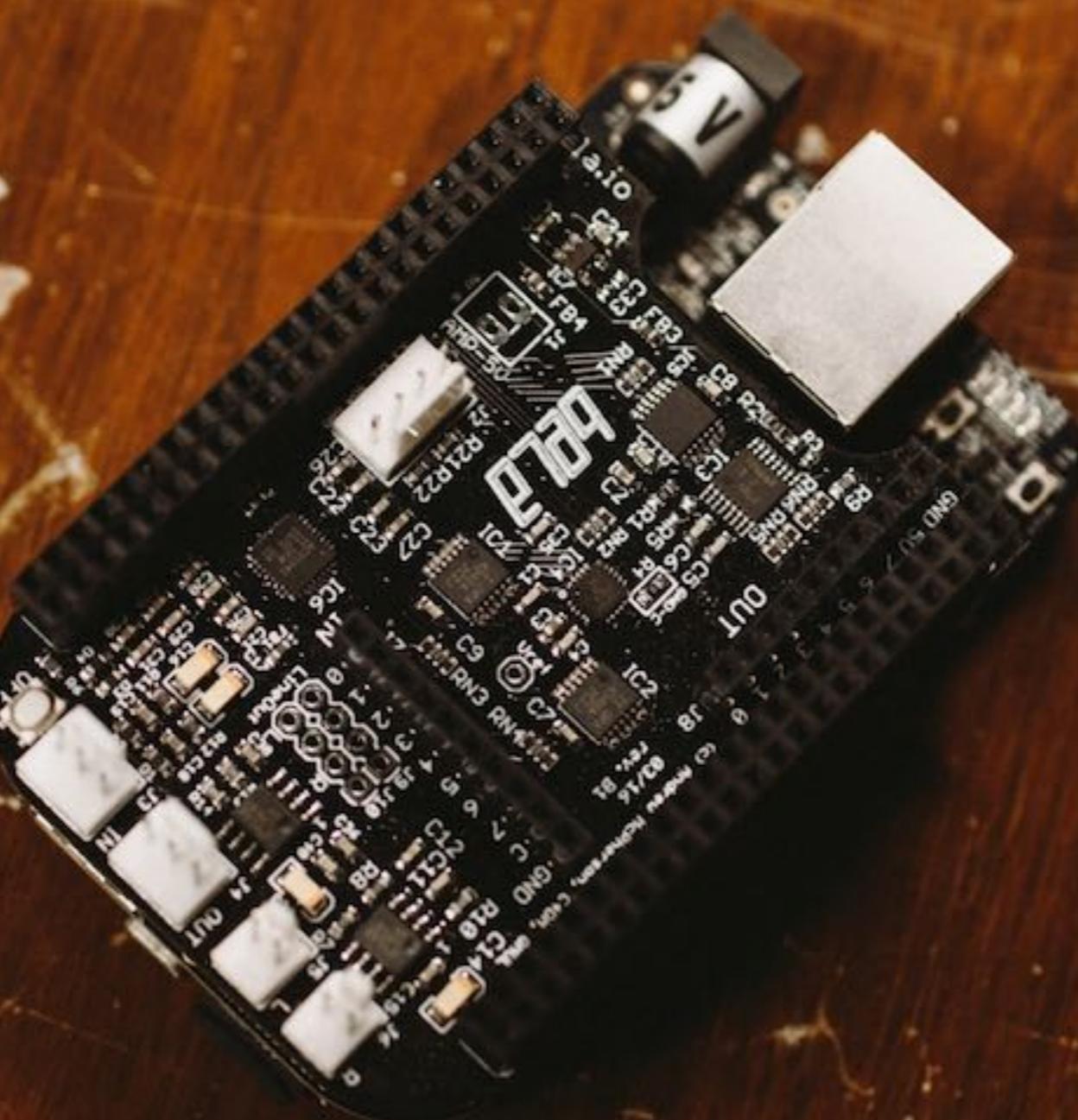
bela.io



A project of the Augmented Instruments Laboratory:
<http://instrumentslab.org>

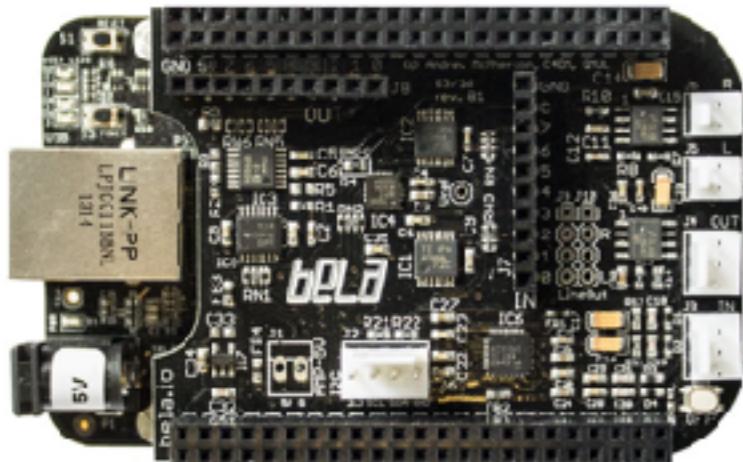


What is Bela?

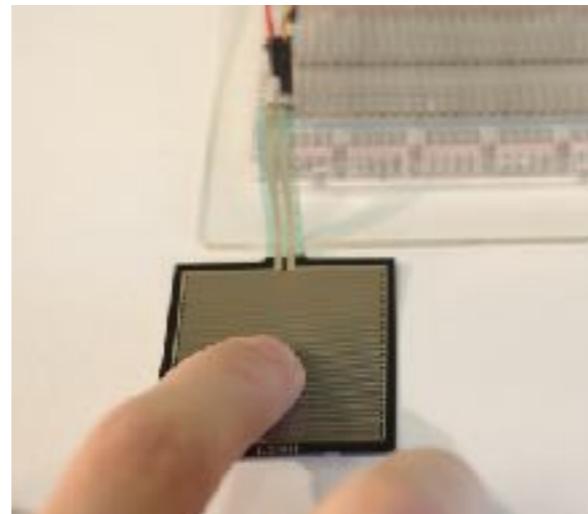


What is Bela?

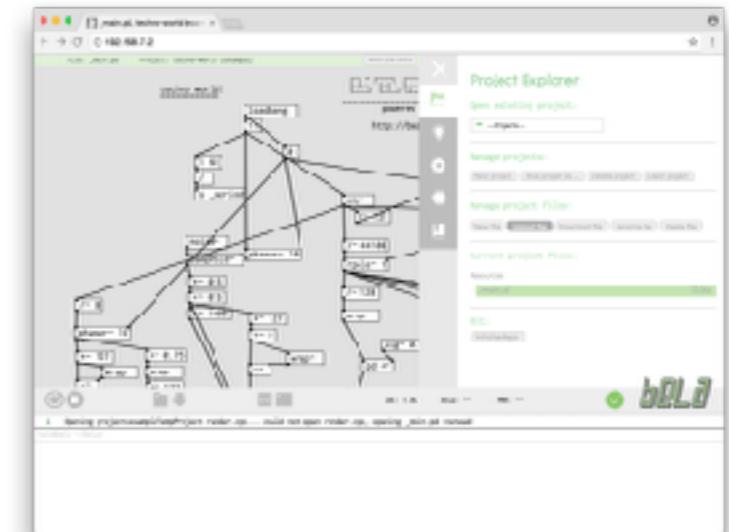
Embedded computer
designed for
interactive audio



New approach
to high-bandwidth
sensor processing



Open-source
maker
platform



- Power of a single board computer
- Connectivity of a micro-controller
- Combines the benefits of both

- Analog, digital I/O sampled at audio rate
- Ultra low action-sound latency
- Jitter-free alignment between audio and sensors

- Open hardware and software
- Targeted at musicians, artists
- Online community resources: forum, wiki, blog.

Latency

Delay between action and reaction,
a fundamental property of digital interactive systems

Sensor In

Audio Out

Latency

Does it matter? Yes!

- Long latency causes an audible delay
 - But even an imperceptible delay may make an instrument feel less responsive to play
 - How low is “low enough”?

“We place the acceptable upper bound on the computer’s audible reaction to gesture at 10 milliseconds. ... Low variation of latency is critical and we argue that the range of variation should not exceed 1 ms.”

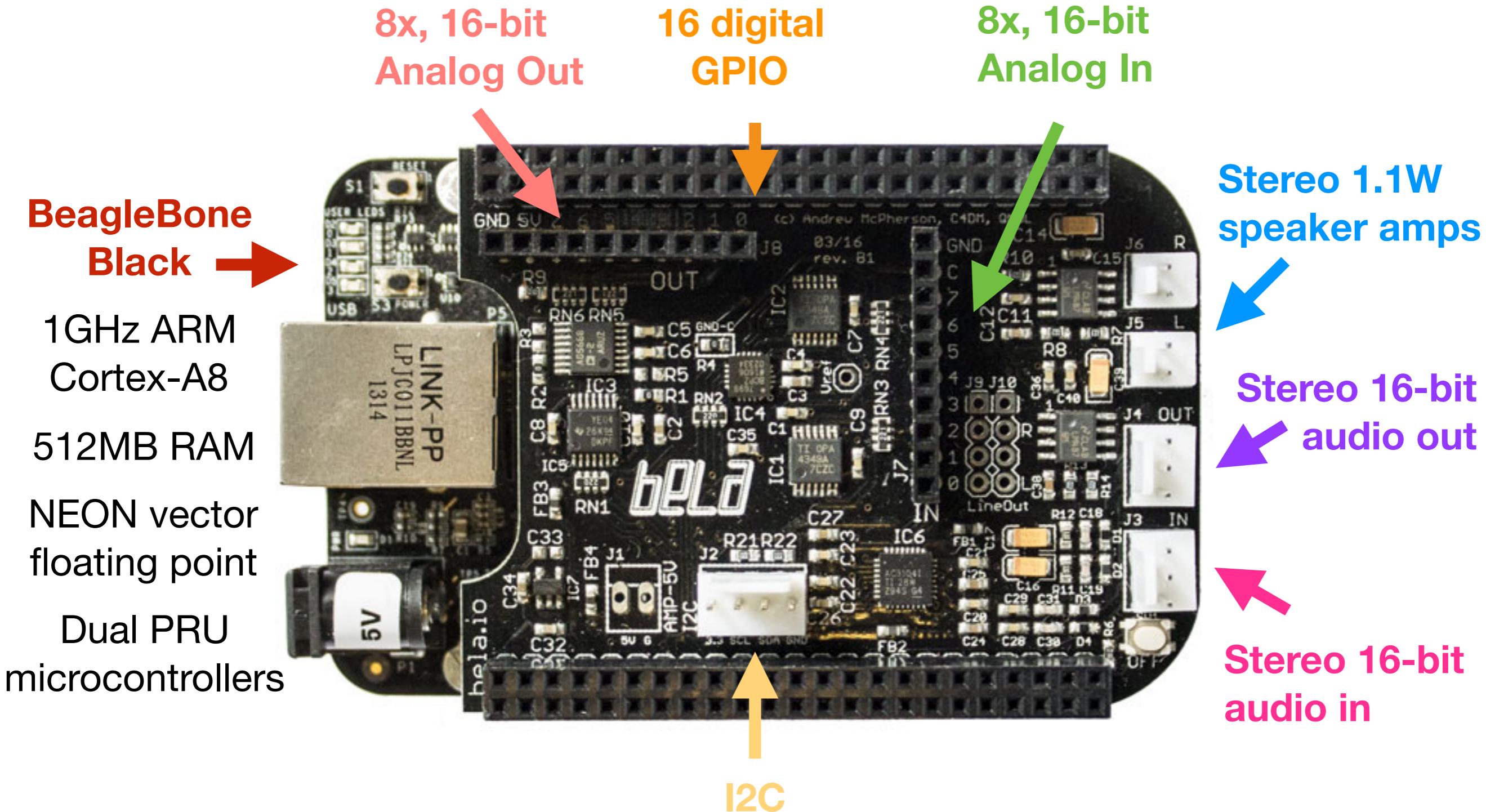
-- David Wessel & Matthew Wright, “Problems and Prospects for Intimate Musical Control of Computers”, Computer Music Journal, 2002.

- Surprisingly few systems for building digital musical instruments meet this threshold!

A. McPherson, R. Jack and G. Moro. “Action-Sound Latency: Are Our Tools Fast Enough?” Proc. NIME, 2016.

କେବଳ

Bela hardware



Bela software

- Bela uses the Xenomai real-time Linux extensions to run audio code at higher priority than the entire OS
- Allows buffer sizes as small as 2 audio samples
- Digital and analog sensors sampled at audio rate, synchronously to audio clock for near-zero jitter

Bela features

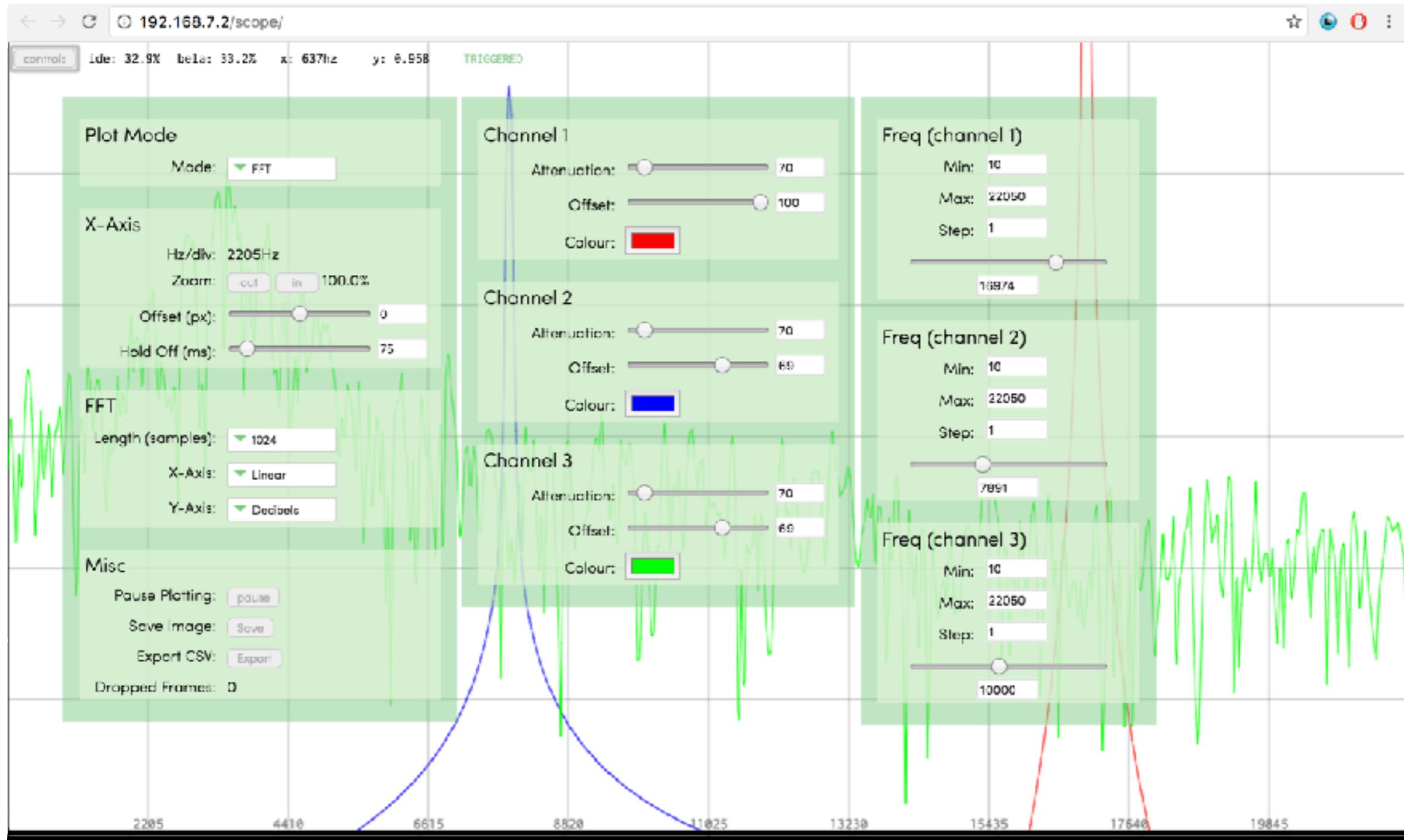
- High sensor bandwidth:
 - Digital I/Os sampled at 44.1kHz
 - Analog I/Os sampled at $\geq 22.05\text{kHz}$
- Jitter-free alignment between sensors and audio
- Hard real-time audio/sensor performance but full Linux APIs still available
- USB, network, filesystem, etc.

Audio (2 ch)	L	R	L	R	L	R	L	R
Digital (16 ch)	16 I/Os	16 I/Os	16 I/Os	16 I/Os				
Analog (8 ch)	0	1	2	3	4	5	6	7

Sampled automatically each block

Bela features

Built-in, browser-based IDE with oscilloscope



Bela features

- Programming options:
 - **C/C++** with built-in IDE (or terminal scripts)
 - **Pd** using either libpd or Heavy Audio Tools
 - **FAUST** DSP language (contributed by Stéphane Letz and Yann Orlarey)
 - **Pyo** python audio module (contributed by Olivier Bélanger)
 - **SuperCollider** (with Dan Stowell, Marije Baalman)
 - **Csound** (still in alpha; with Victor Lazzarini and Bernt Isak Wærstad)
- Open-source hardware and software
- Online repo, wiki and support forum:
 - <http://github.com/BelaPlatform>
 - <http://forum.bela.io>

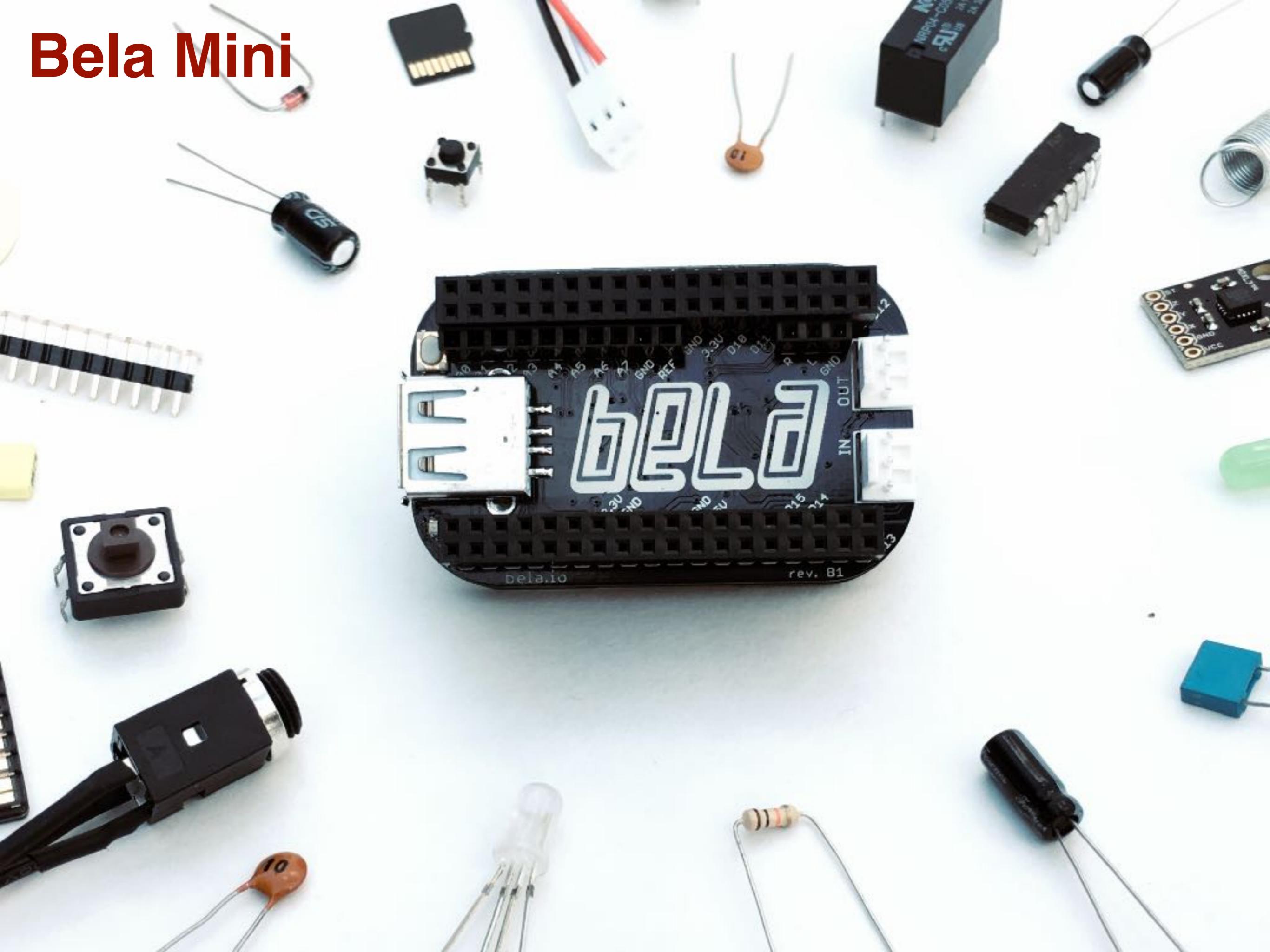
Pd on Bela



libpd

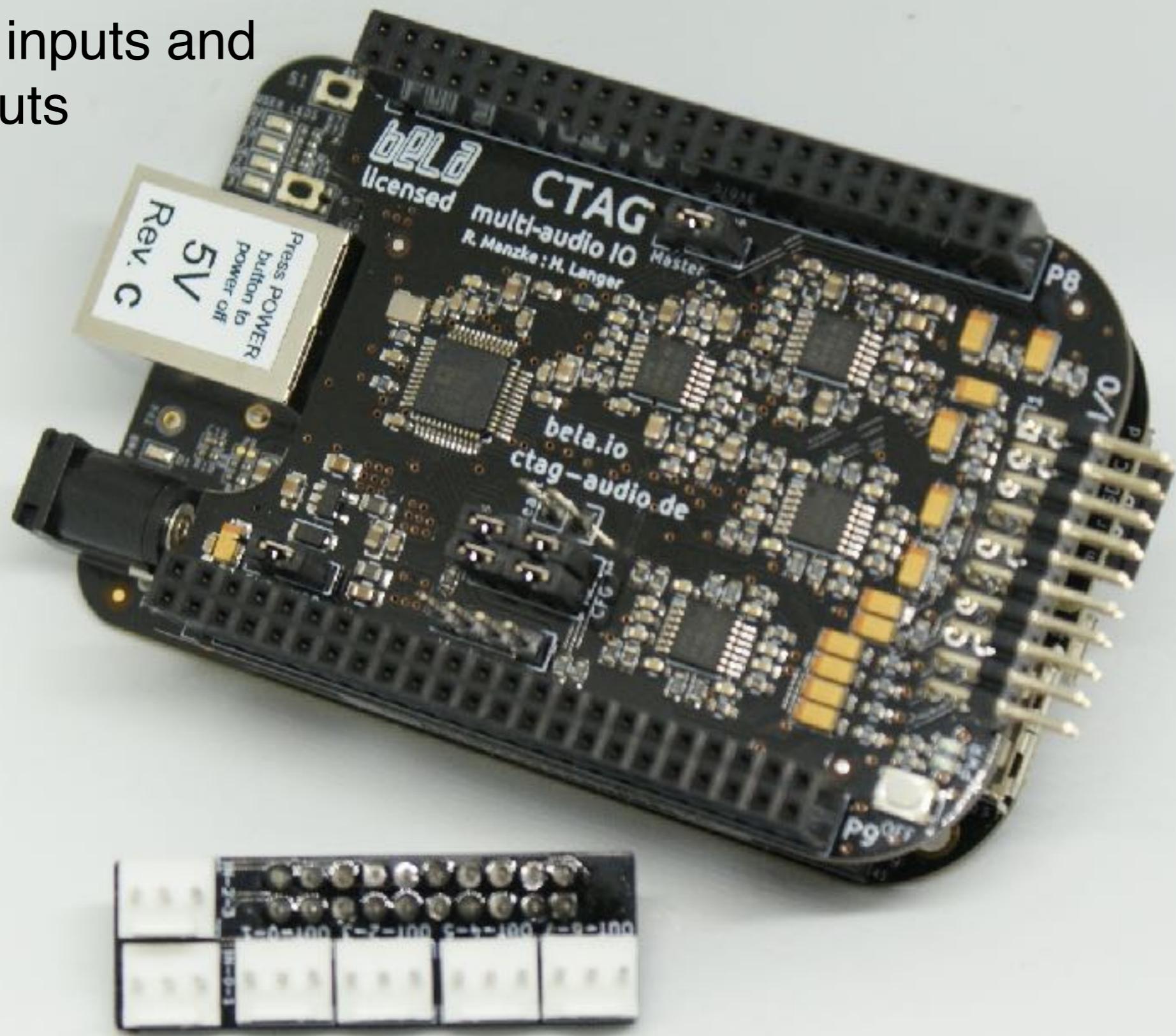
- Data flow computer music language
- Sketch signal and data flows
- Quickly prototype interactive audio systems
- Integrated with the browser-based IDE
- Currently no live patching in the browser

Bela Mini



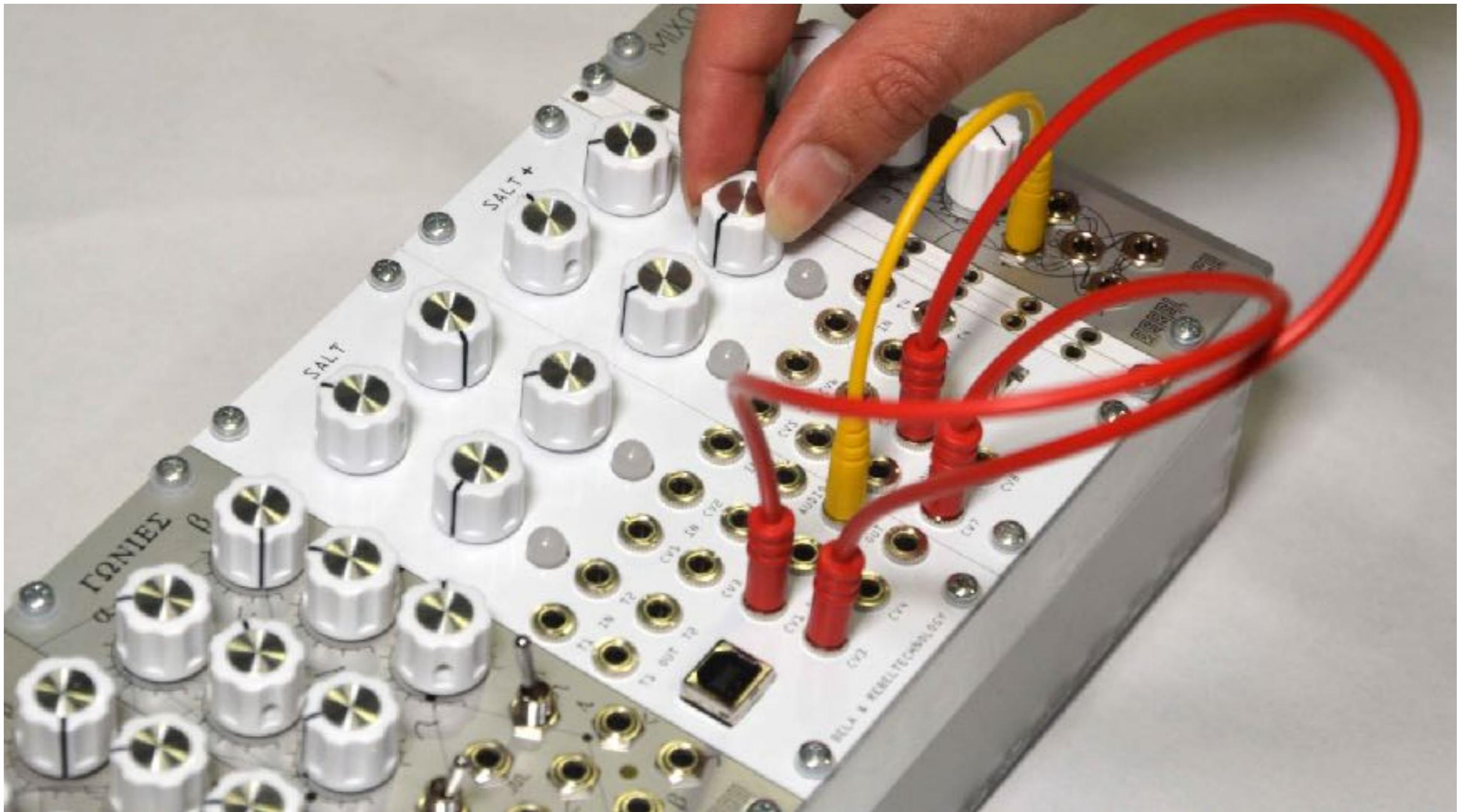
CTAG FACE and BEAST

Up to 8 audio inputs and
16 audio outputs



SALT: the Bela modular

a joint project with Rebel Technology (rebeltech.org)



REBEL TECHNOLOGY

SALT: the Bela modular

2 audio in
2 audio out

4 CV in
4 CV out
(outputs:
-5V to 5V)

**CV input
offsets**

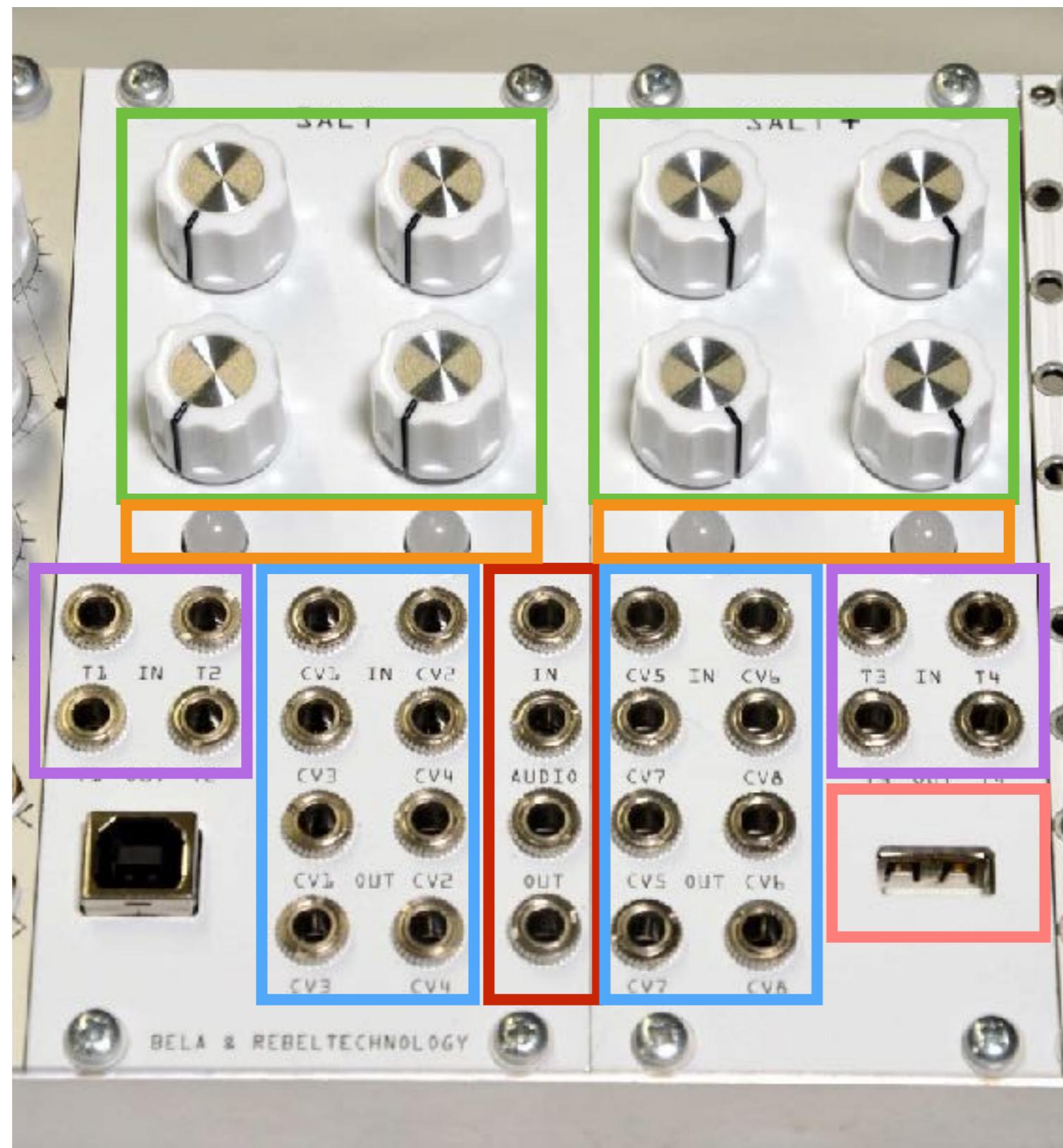
(from -10V to 0V
to 0V to 10V)

2 trigger in
2 trigger out
(0V to 5V)

2 buttons
(connected
to trigger ins)
2 LEDs
(bi-colour)

SALT: 12hp

SALT+: 10hp



+4 CV in
+4 CV out
**with input
offsets**

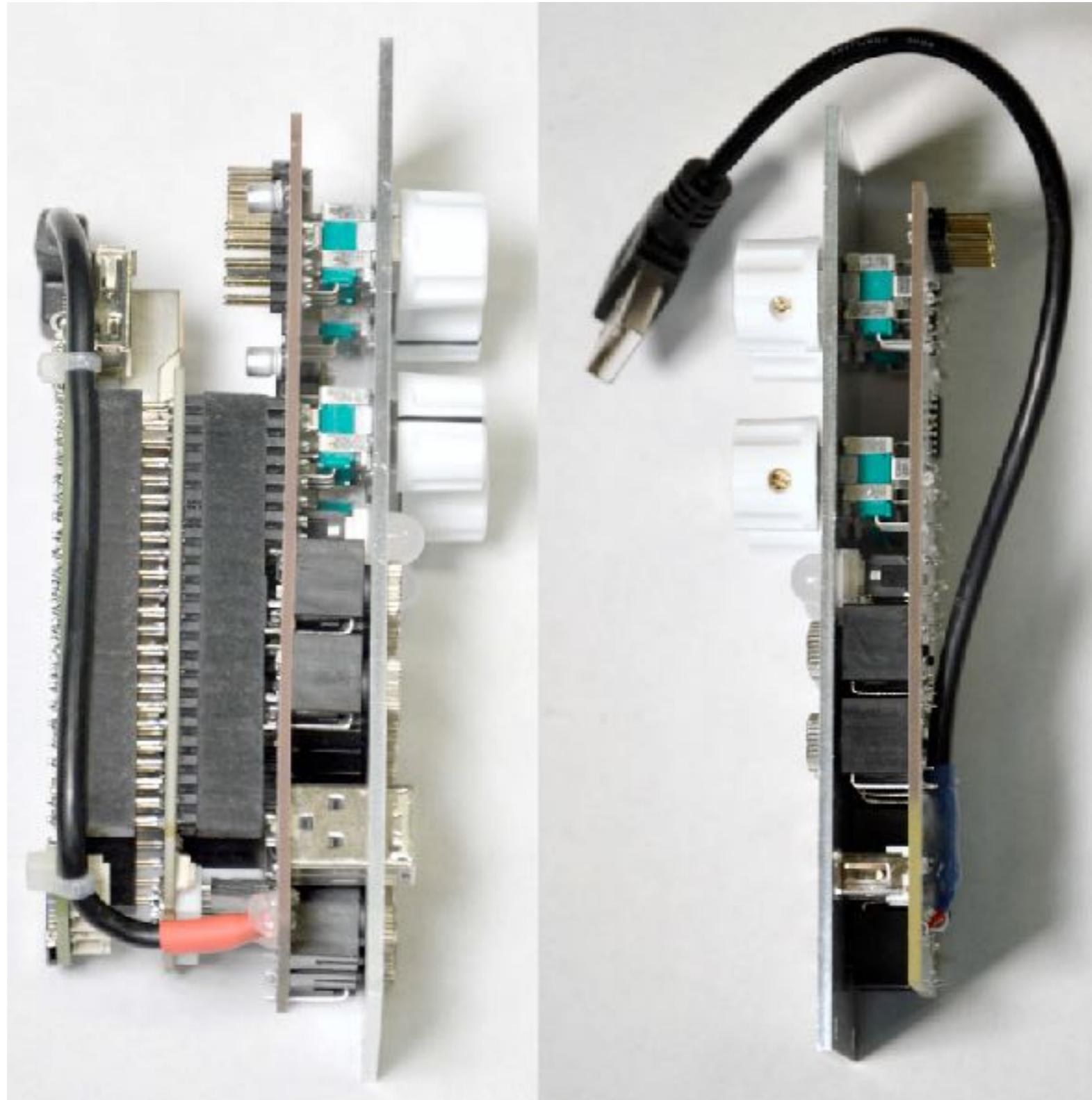
+2 trigger in
+2 trigger out

+2 buttons
+2 LEDs

USB host

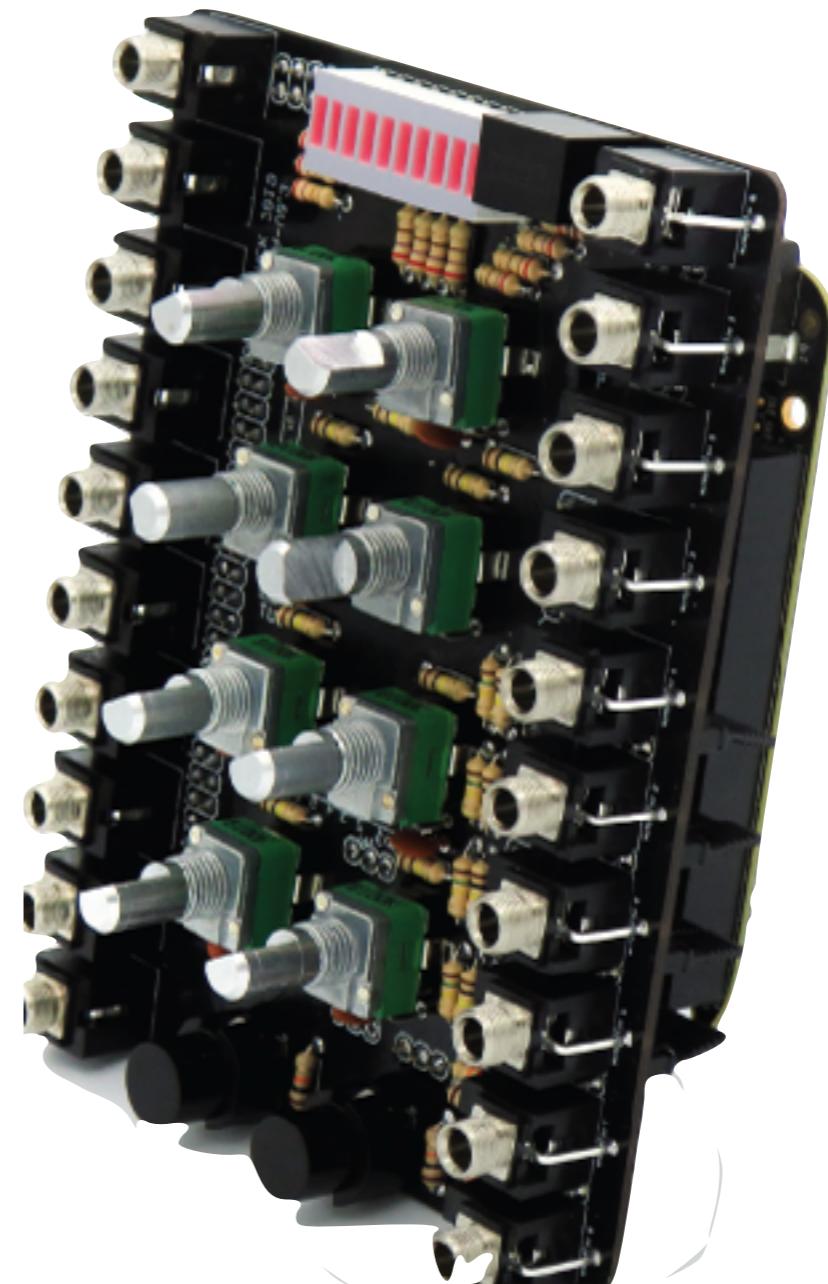
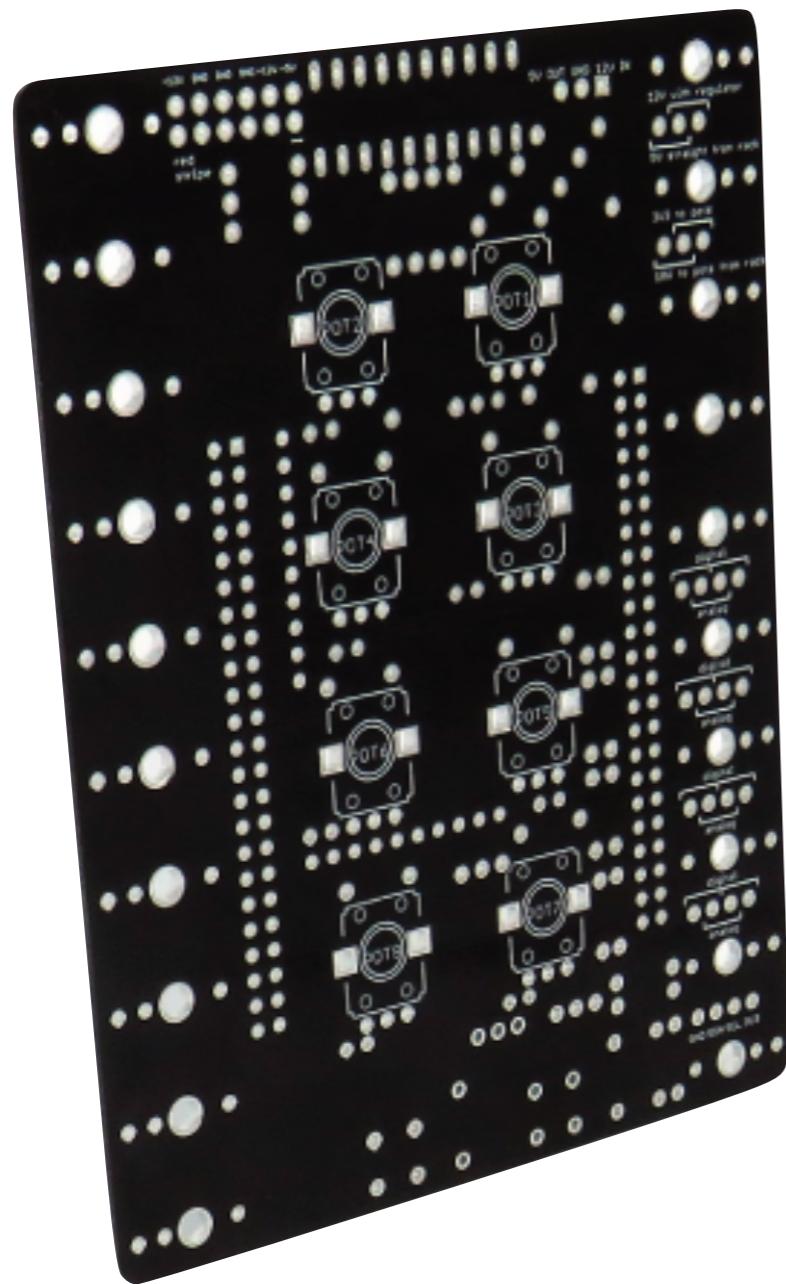
SALT: the Bela modular

More info: <http://bela.io/salt/>

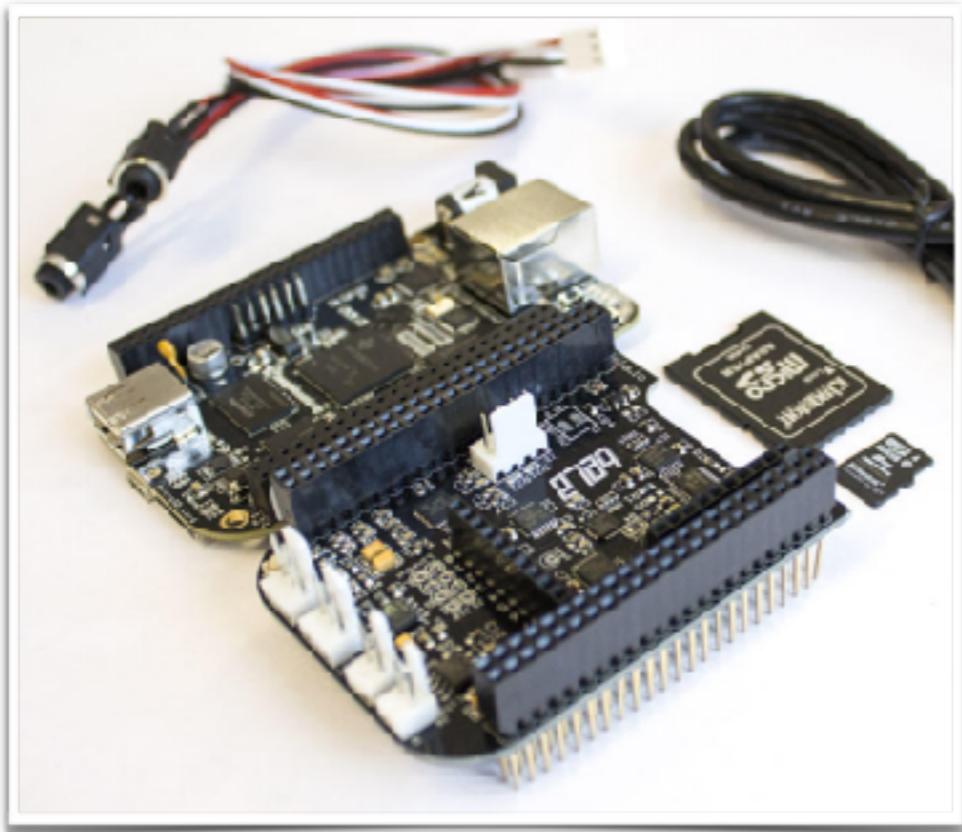


PEPPER

passive breakout to easily connect Bela to your rack
PCB with easily sourced DIY components



Getting started: hardware



Bela starter kit:

- BeagleBone Black
- Bela cape
- 2 audio adapter cables
- Mini-USB cable

Workshop extras:

- Breadboard
- Jumper wires
- Electronic components

Getting started: software



Chrome browser

The IDE may or may not work in other browsers



Pd software [<http://puredata.info>]

Needed to edit Pd patches (*use Pd-vanilla*)



beaglebone

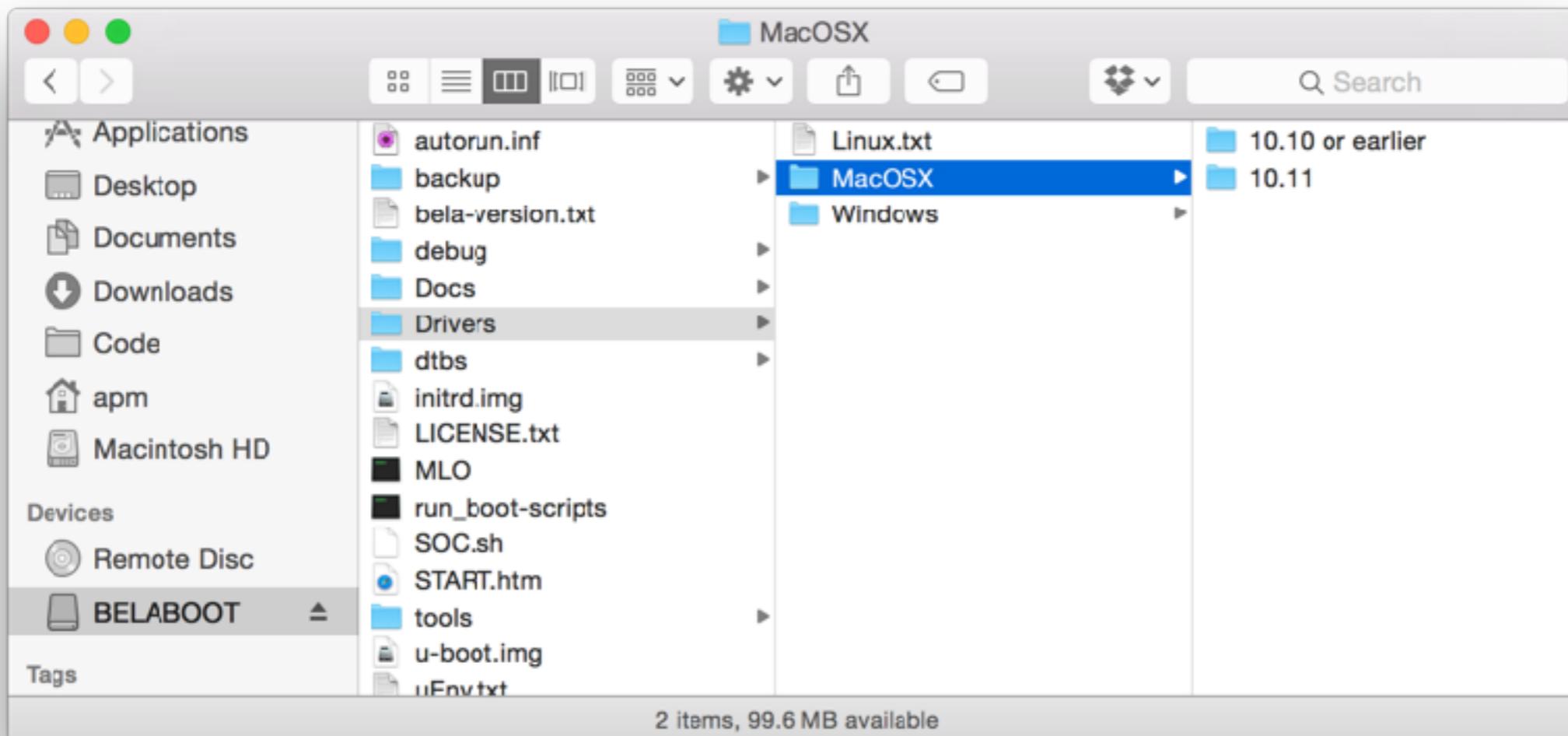
BeagleBone Black drivers

Mac Yosemite and above, Windows Vista and above, all Linux: no drivers required.

For older operating systems the drivers can be found on the board

Install drivers (*usually not needed)

- Step 1: Plug in Bela by USB
 - Wait for it to boot
 - You should see a drive **BELABOOT** come up



- Step 2: Install software from **Drivers** directory
- Step 3: Reboot computer

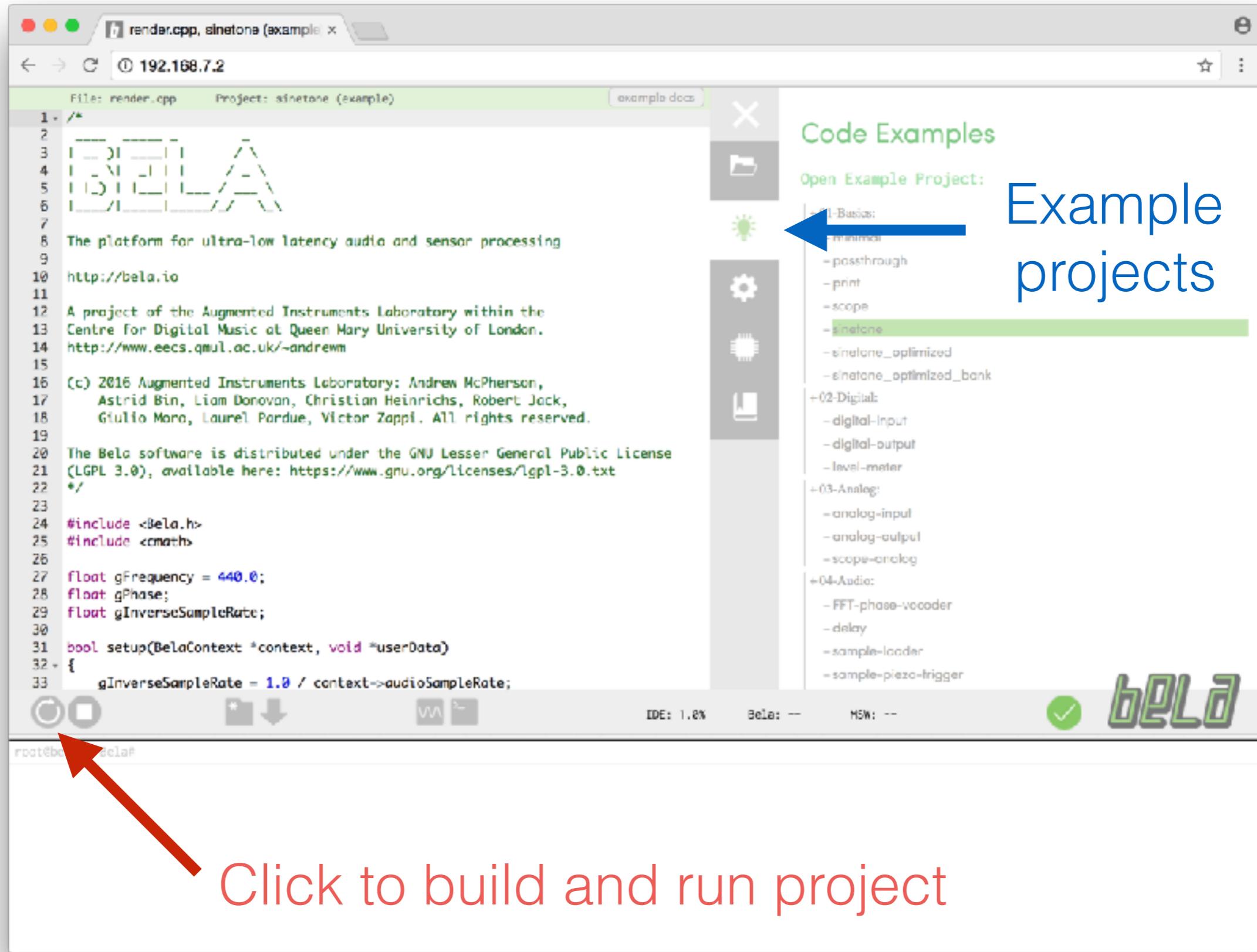
Copy workshop examples

As we need to edit Pd patches using Pure Data on our machine it is best to have a local copy.

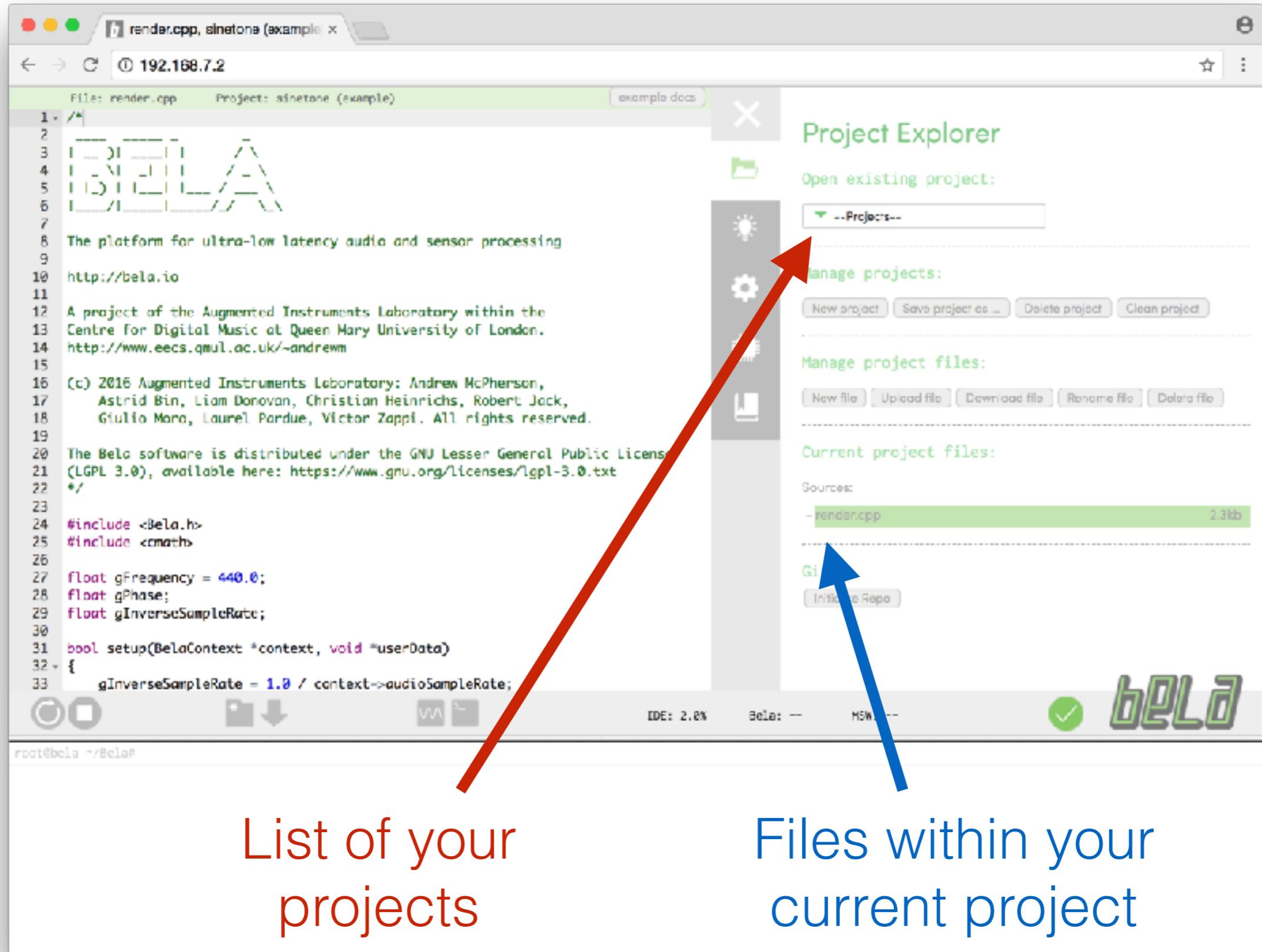
- Step 1: Copy [Workshop.zip](#) archive from circulating USB stick to computer
- Step 2: Unzip [Workshop.zip](#)

Bring up the Bela IDE:

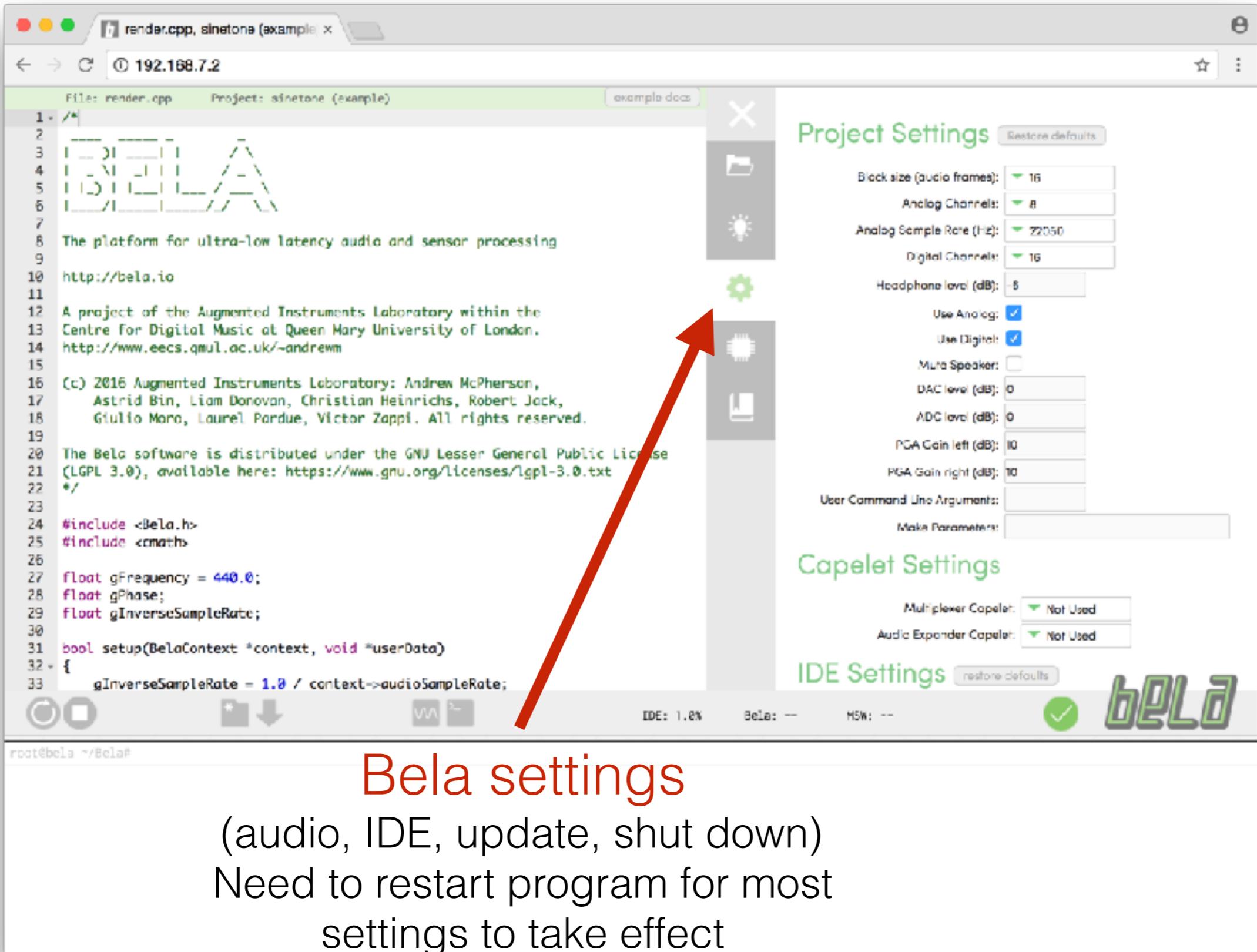
<http://bela.local/> (or 192.168.6.2 on Windows)



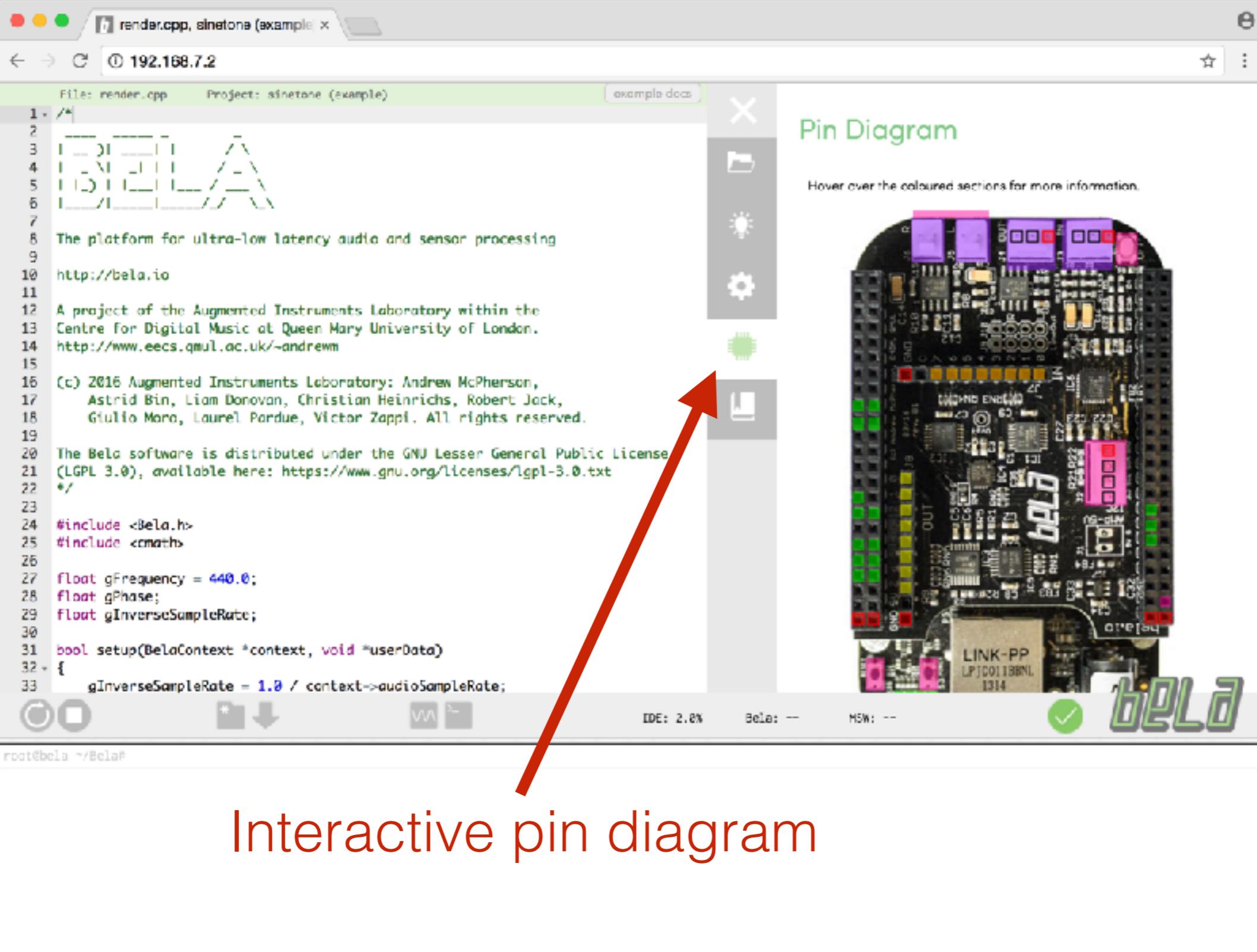
Bring up the Bela IDE: <http://bela.local/>



Bring up the Bela IDE: <http://bela.local/>



Bring up the Bela IDE: <http://bela.local/>

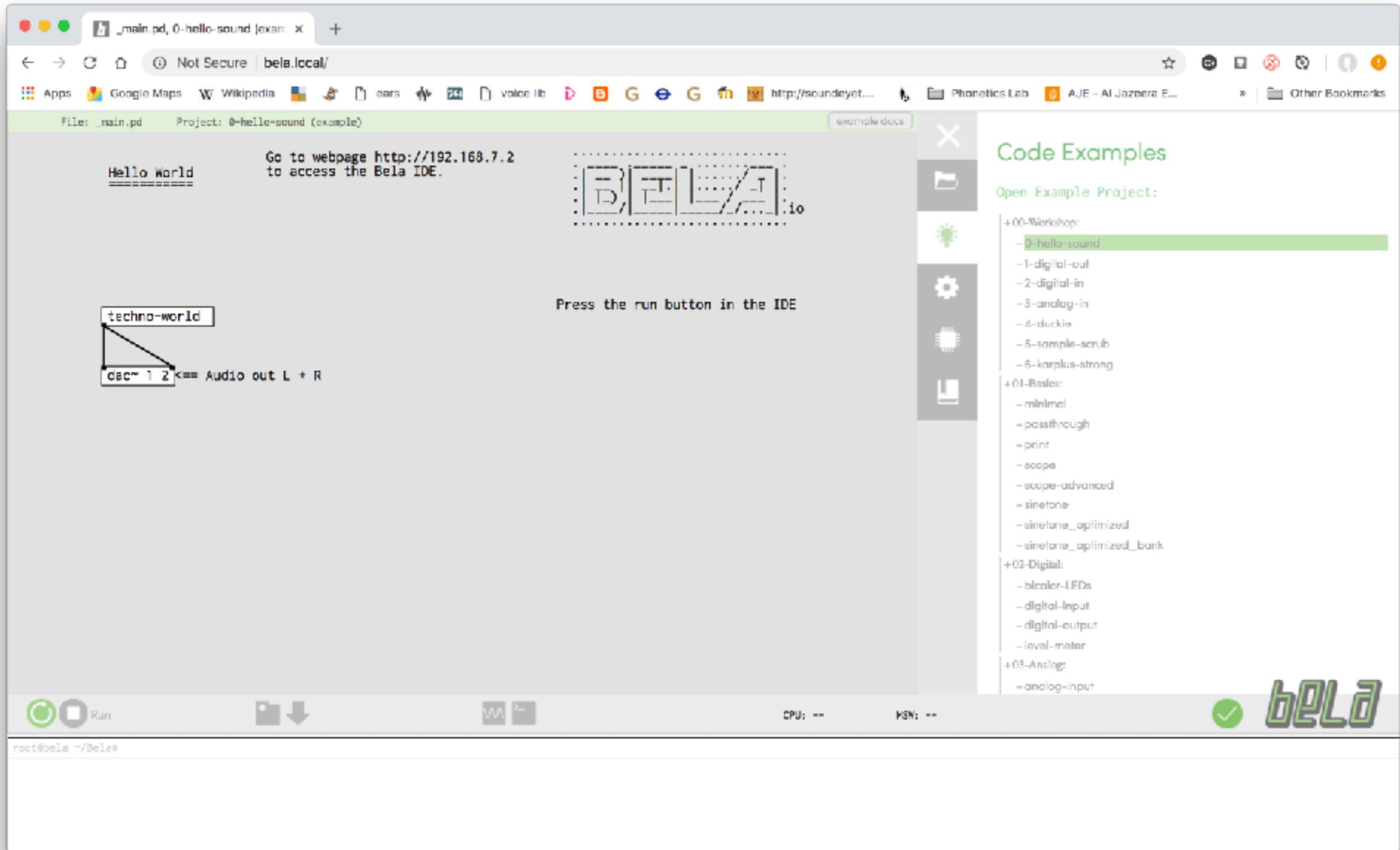


The screenshot shows the Bela IDE interface. On the left, there is a code editor window displaying the file `render.cpp` from the project `sinetone (example)`. The code is a C++ example for generating a sine wave. A red arrow points from the text "Interactive pin diagram" at the bottom to the pin diagram on the right. The pin diagram is titled "Pin Diagram" and shows a top-down view of the Bela Dev Board with various pins and components. Colored sections on the board represent different functional areas or components. The status bar at the bottom of the IDE window shows "IDE: 2.0%", "Bela: --", and "MSW: --".

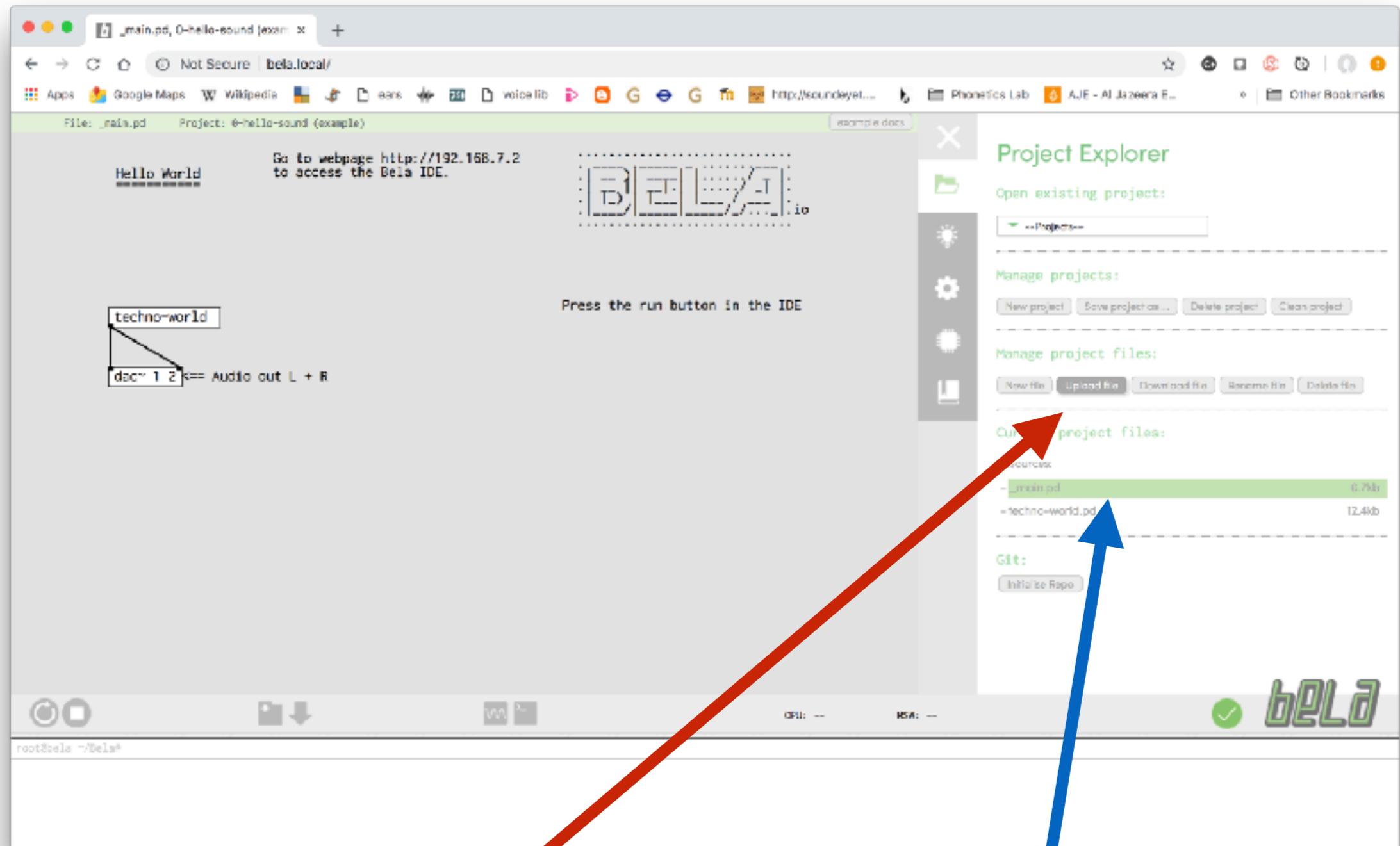
Interactive pin diagram

```
1 //*
2
3
4
5
6
7
8 The platform for ultra-low latency audio and sensor processing
9
10 http://bela.io
11
12 A project of the Augmented Instruments Laboratory within the
13 Centre for Digital Music at Queen Mary University of London.
14 http://www.eecs.qmul.ac.uk/~andrewm
15
16 (c) 2016 Augmented Instruments Laboratory: Andrew McPherson,
17 Astrid Bin, Liam Donovan, Christian Heinrichs, Robert Jack,
18 Giulio Mora, Laurel Pardue, Victor Zappli. All rights reserved.
19
20 The Bela software is distributed under the GNU Lesser General Public License
21 (LGPL 3.0), available here: https://www.gnu.org/licenses/lgpl-3.0.txt
22 */
23
24 #include <Bela.h>
25 #include <cmath>
26
27 float gFrequency = 440.0;
28 float gPhase;
29 float gInverseSampleRate;
30
31 bool setup(BelaContext *context, void *userData)
32 {
33     gInverseSampleRate = 1.0 / context->audioSampleRate;
```

Open example **00-workshop/0-hello-sound**
In IDE, you can **view** and run but **not edit** Pd patches



Edit your patch in the Pd program on the computer
Save, then use [Upload File](#) button or drag file to browser window



Upload / Download
File

Notice: file always
called [_main\(pd\)](#)

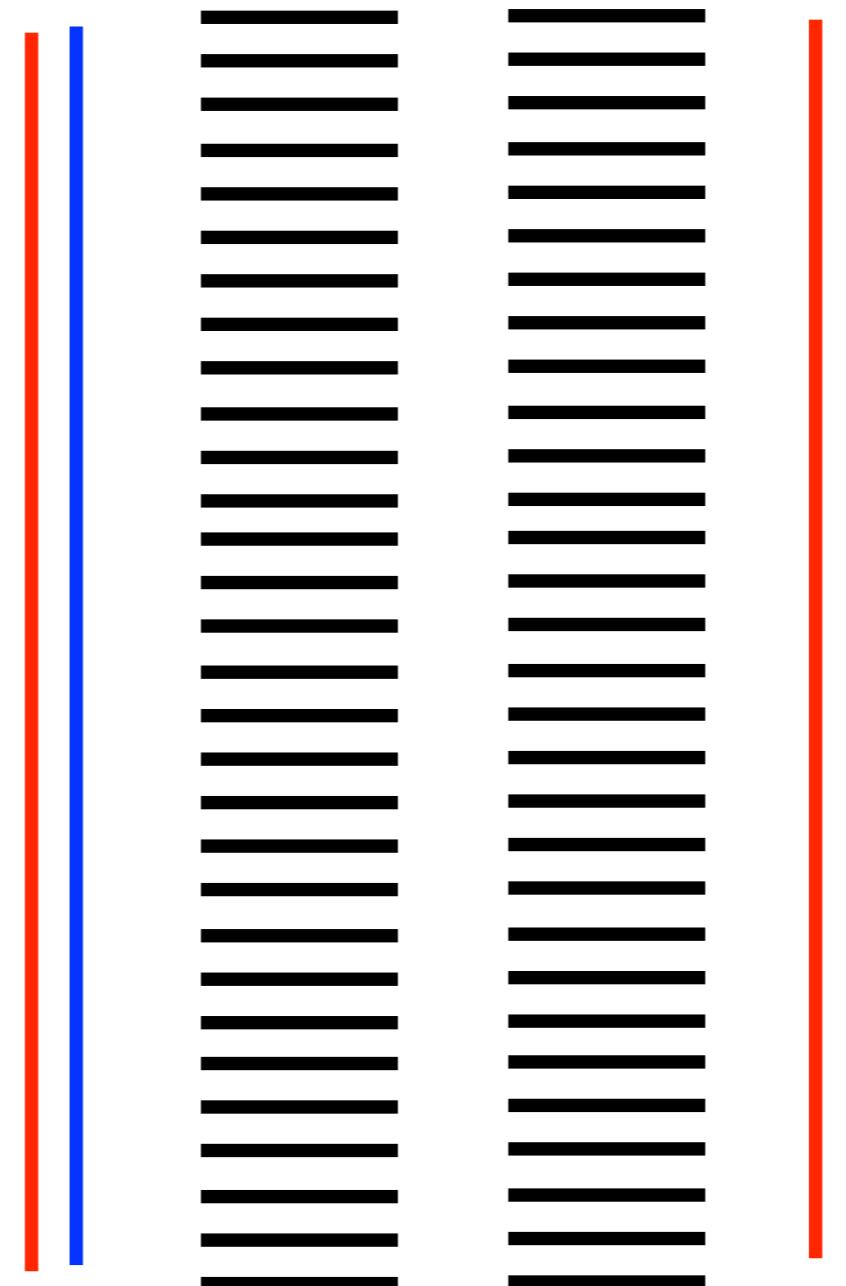
Using the Breadboard

Each column of 5 holes
connects together

...but not across the
break in the middle.

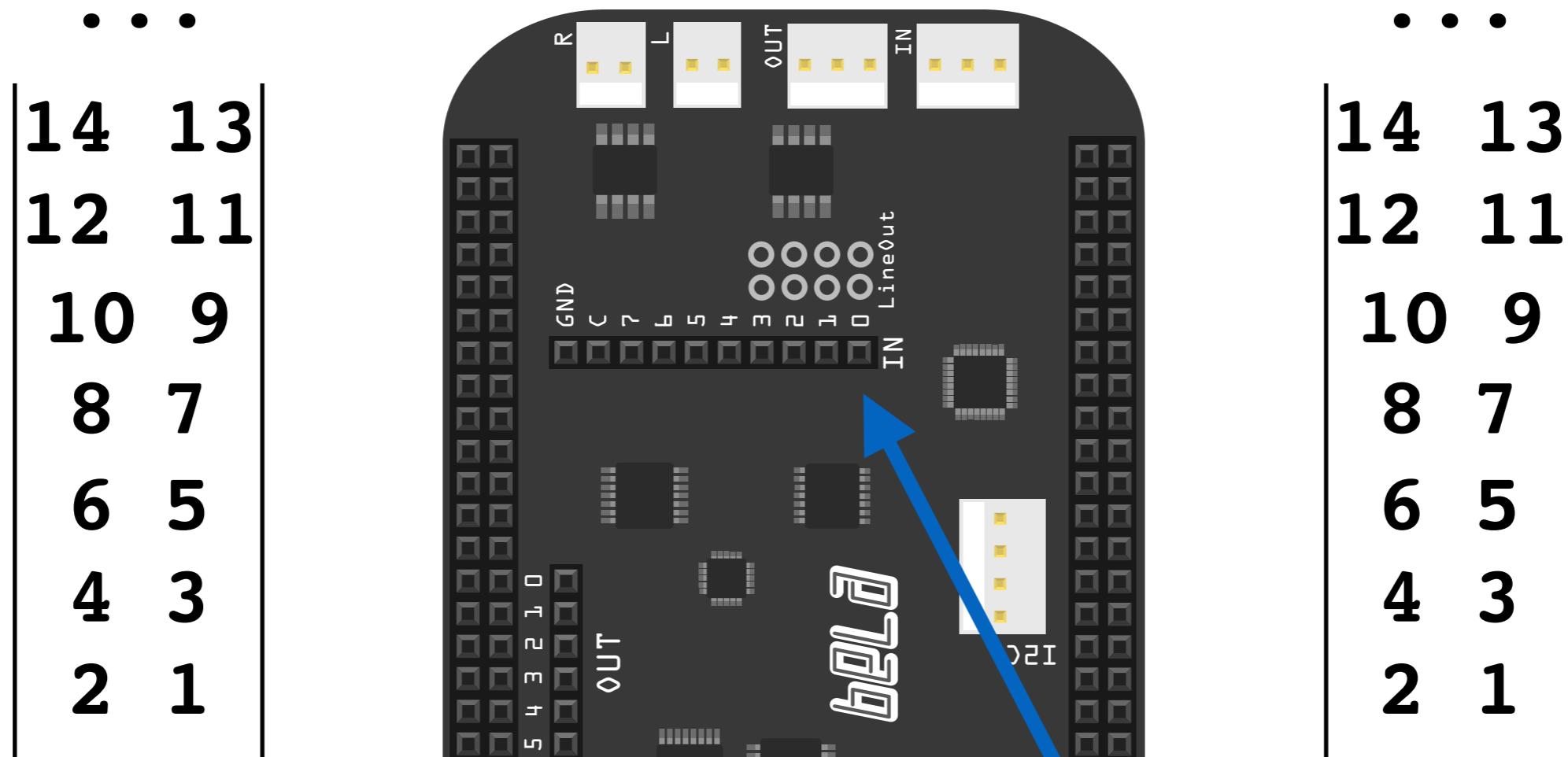
Long rows at the ends
each connect together.

Typically used for
power and **ground**



Bela pin numbering

Important: **NEVER** use 5V with digital I/Os!



P8 →

Analog Out

also 5V (+100Ω), ground

← P9

Analog In

also ground

Digital Output

examples/00-Workshop/1-digital-out

*Important: **short**
lead of LED to ground*

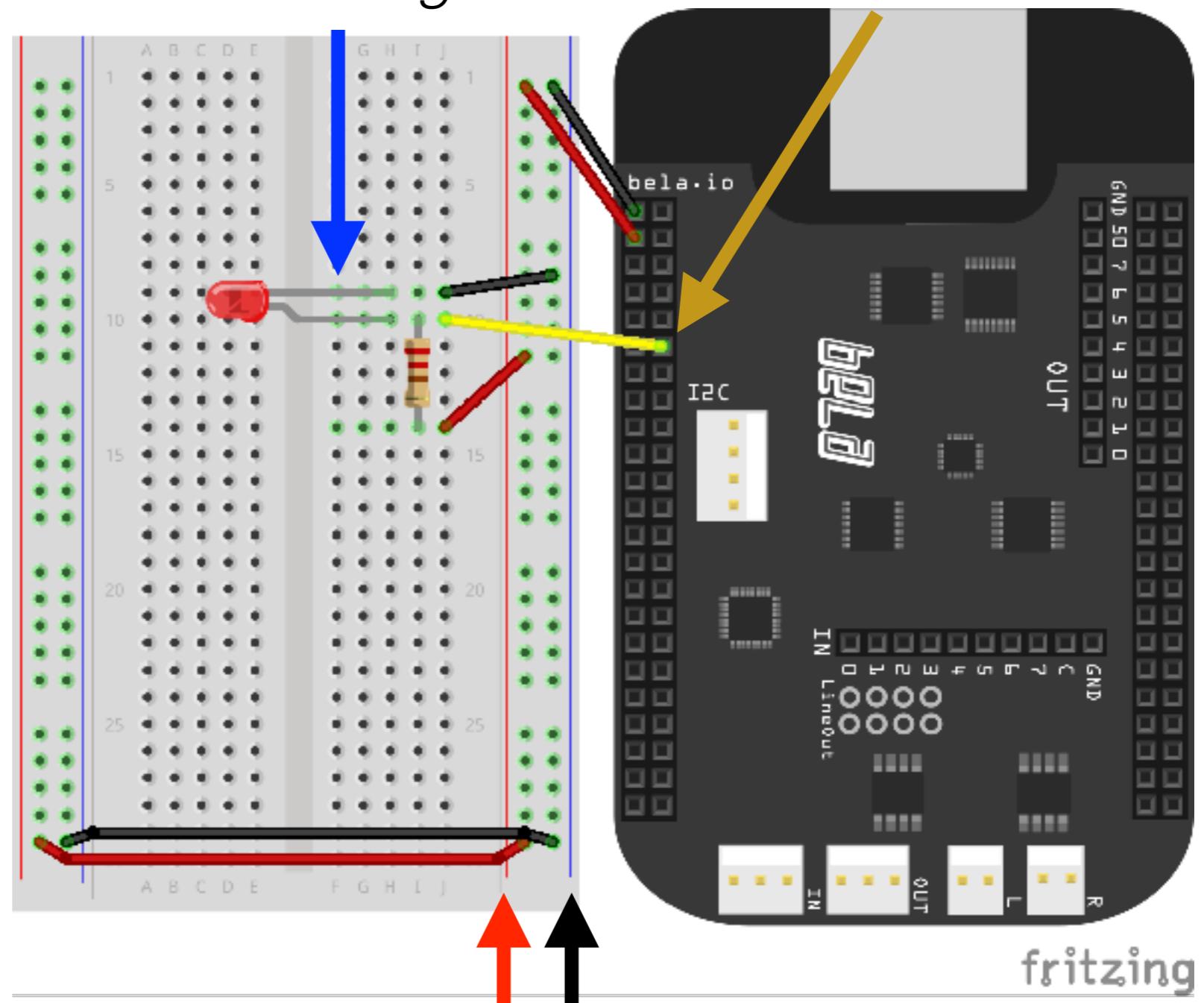
LED on P9 pin 12

Digital Output

```
loadbang  
out 17 initialise the pin to output  
s bela_setDigital
```

Digital out, message rate

```
loadbang  
metro 200  
X Blink that LED!  
s bela_digitalOut17
```



P9 pin 3: +3.3V P9 pin 1: ground

Digital Output

examples/13-Workshop/1-digital-out

*Important: **short** lead of LED to ground*

Digital Output

loadbang

out 17 initialise the pin to output

s bela_setDigital

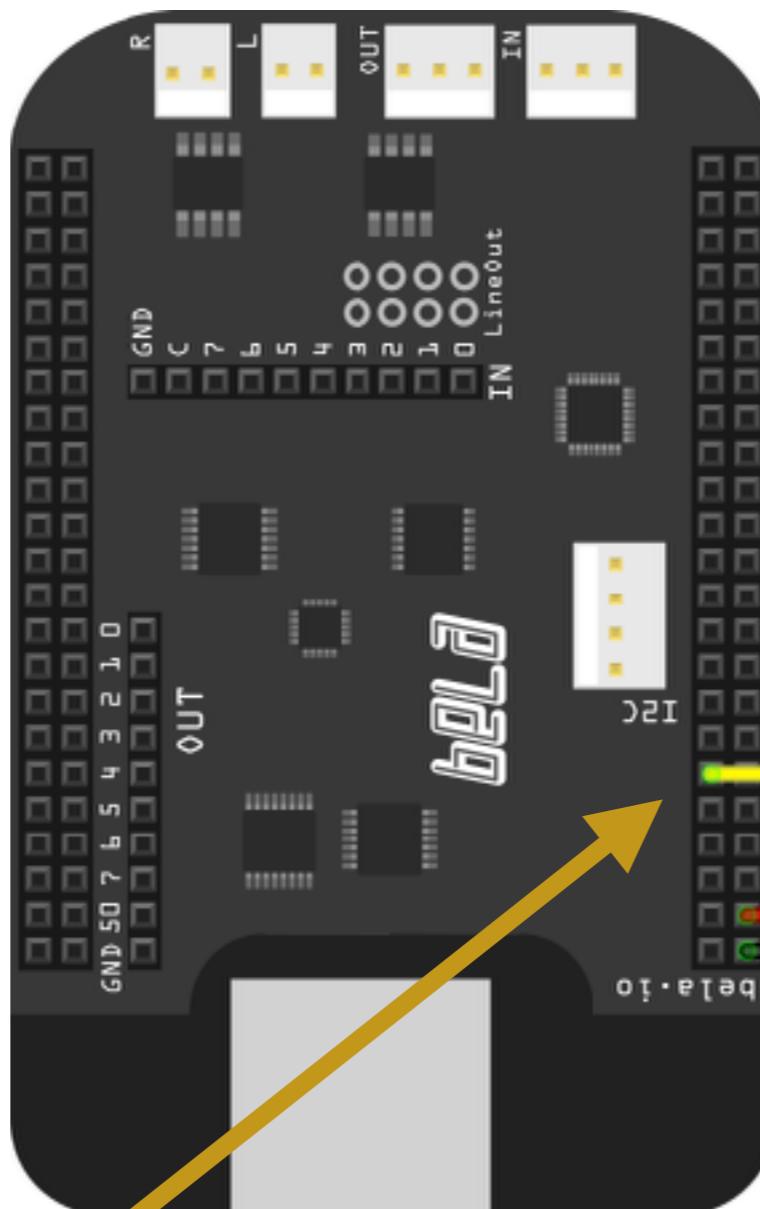
Digital out, message rate

loadbang

metro 200

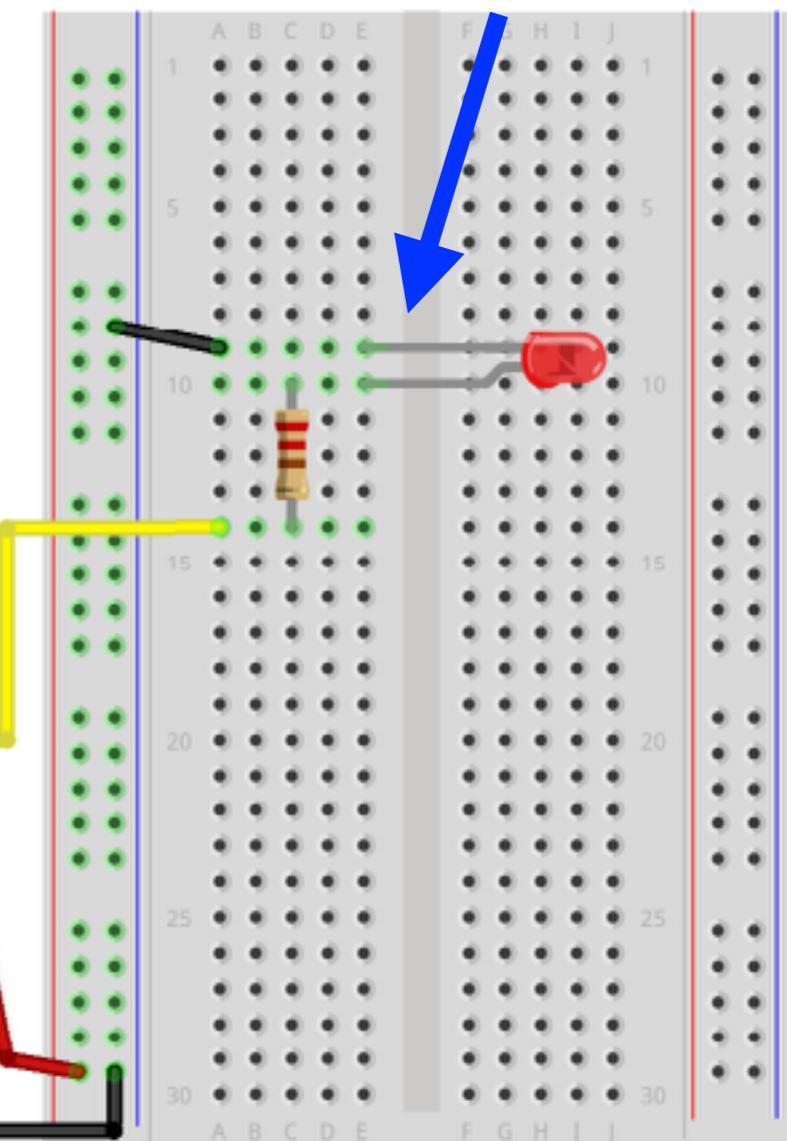
X Blink that LED!

s bela_digitalOut17



LED on P9 pin 12

P9 pin 3: +3.3V P9 pin 1: ground



fritzing

Digital Input

examples/00-Workshop/2-digital-in

*Button requires
3 wires + 10k resistor*

Button on P9 pin 14

Digital Input

loadbang

in 18, out 17 initialise the pins to input and output

s bela_setDigital

Digital input and out, message rate

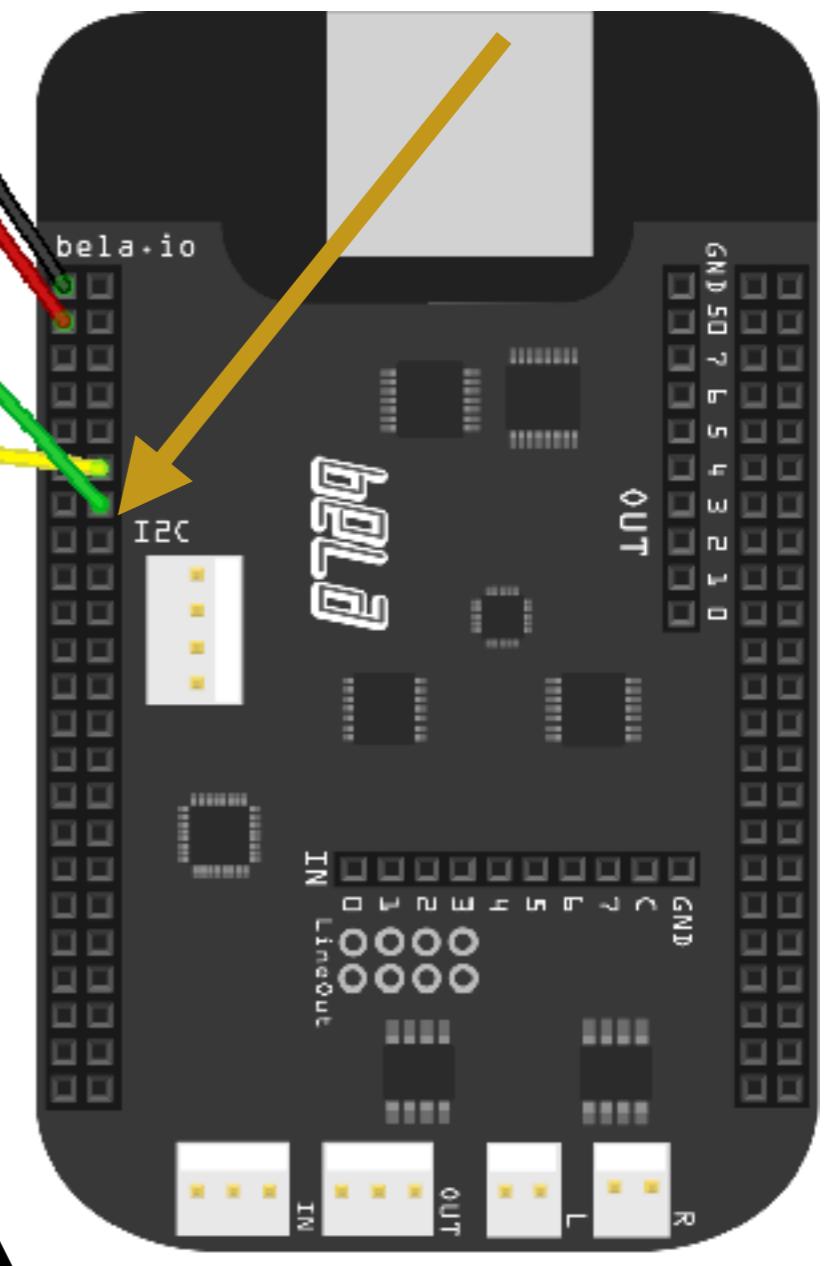
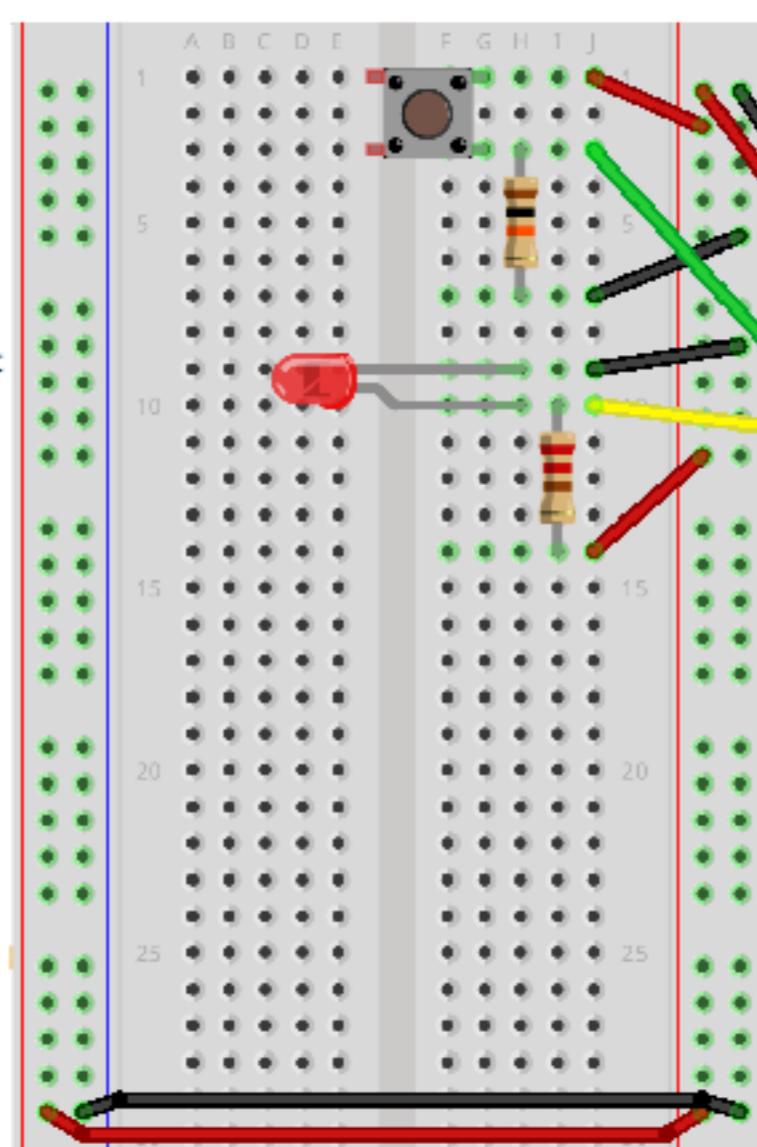
r bela_digitalIn18

Blink that LED!

Button pressed value: \$1

s bela_digitalOut17

print



P9 pin 3: +3.3V P9 pin 1: ground

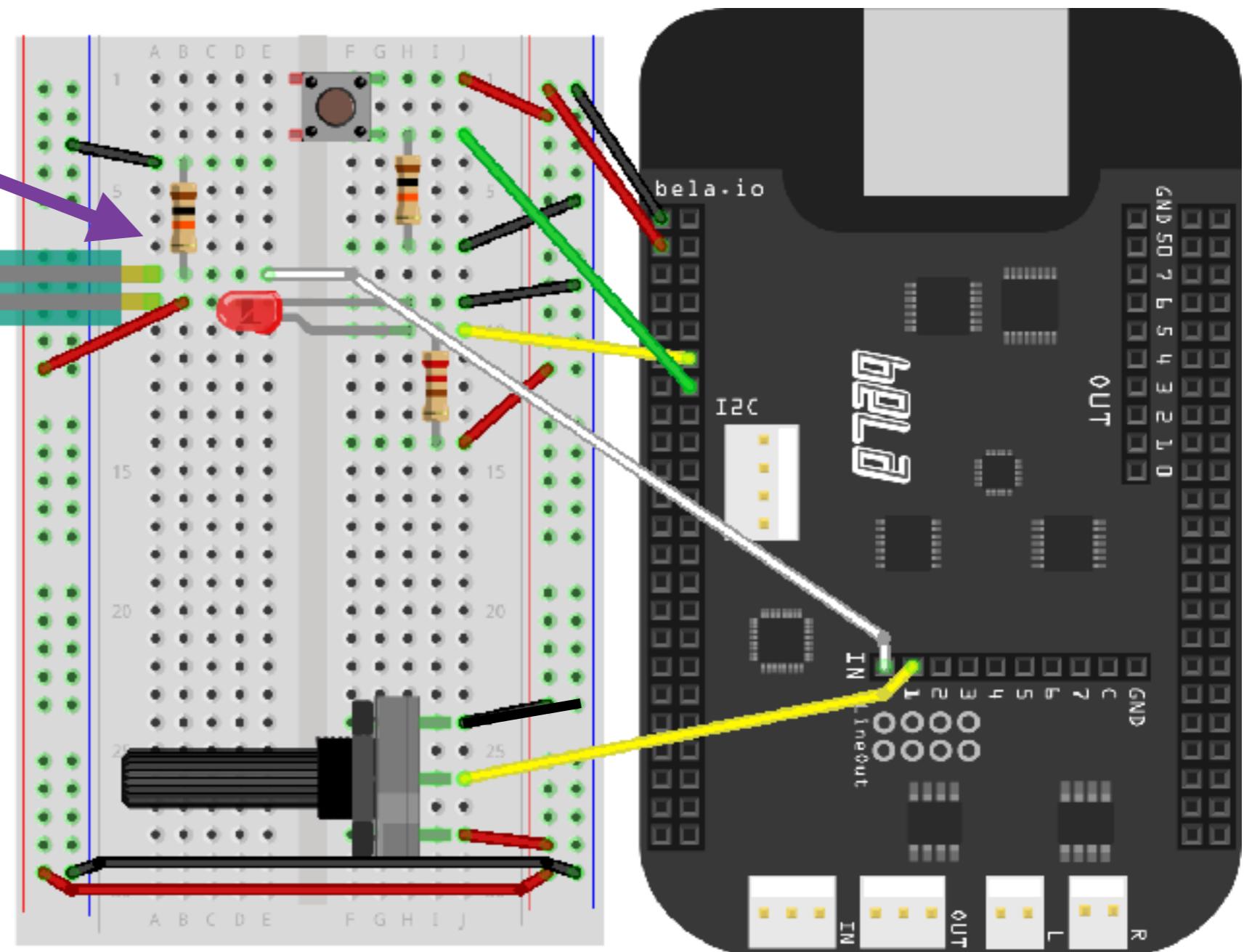
Analog Input

examples/00-Workshop/3-analog-in

Same row connects
to FSR, 10k resistor
and Analog In 0



Middle wire of
potentiometer
goes to
Analog In 1



Note: in Pd, audio is on channels 1 and 2,
analog on channels 3 to 10.
So here, [adc~ 3 4] → Analog In 0 and 1

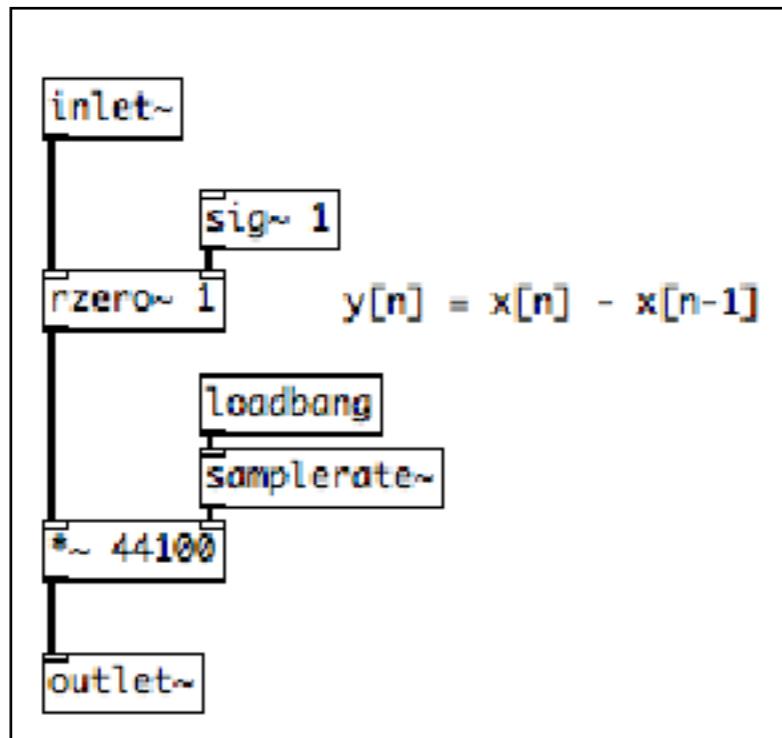
fritzing

Rubber Duckie

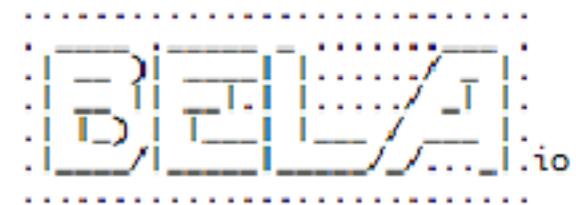
examples/00-Workshop/4-duckie

(same wiring as 3-analog-in)

Abstractions:
Subsidiary Pd patches



Rubber Duckie



adc~ 6

sigdelta

duckie

*~ 0.1

dac~ 1 2

<<< use FSR on analog input 0

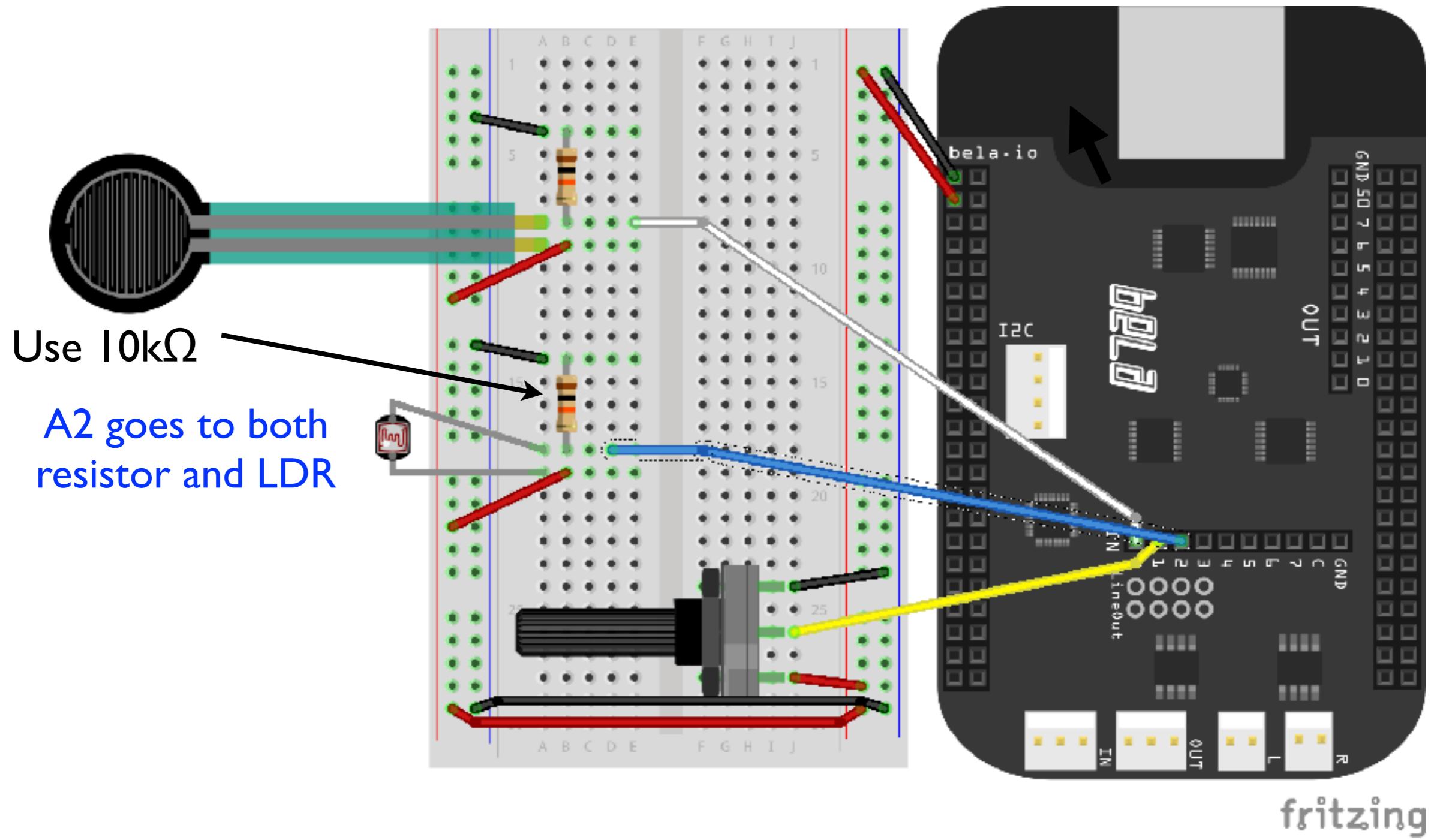
<<< calculate differential on input signal

<<< open this abstraction to see how the model works

Light sensor

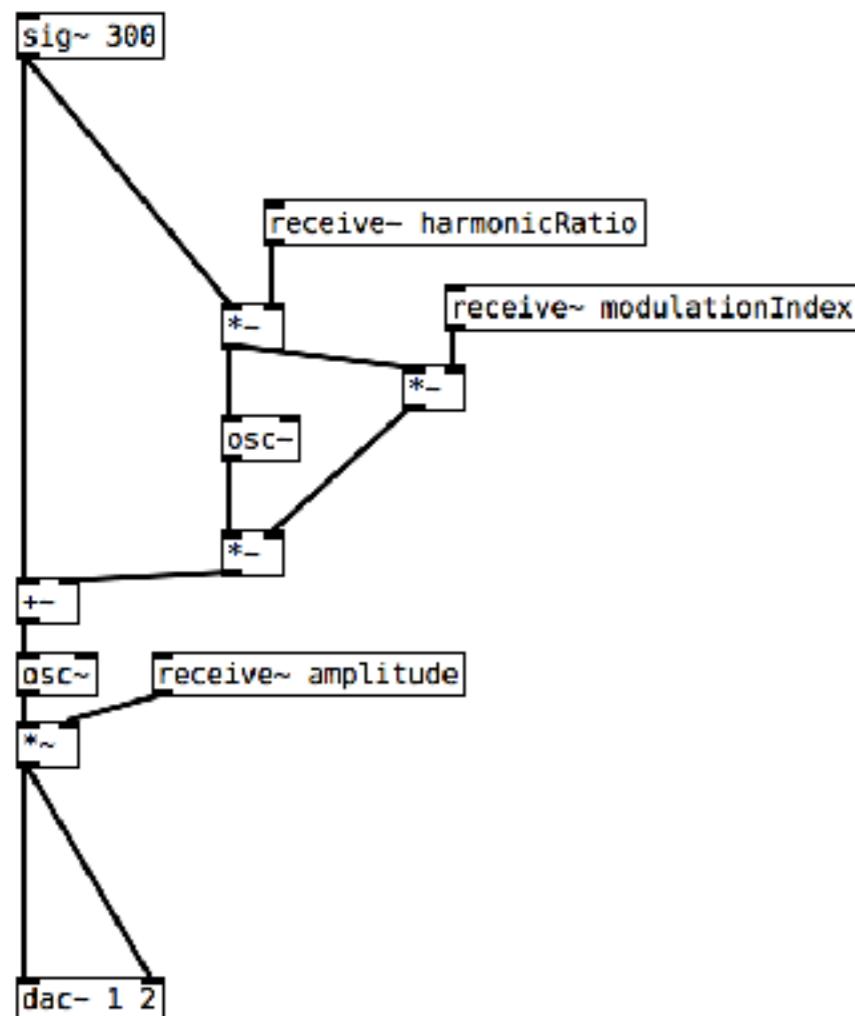
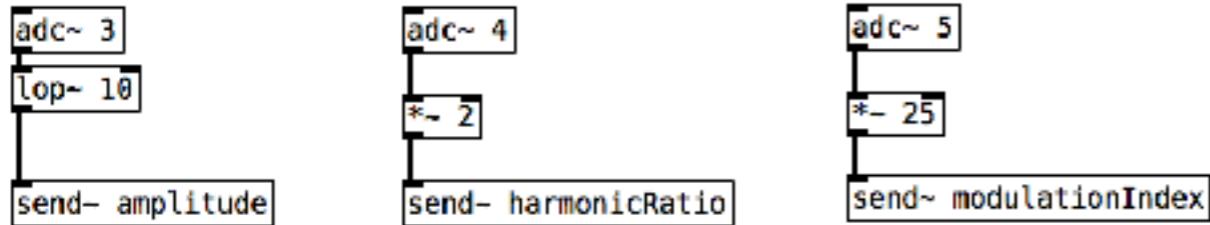
examples/00-Workshop/5-fm-synth

Add this to the existing components on the board



Light sensor

examples/00-Workshop/5-fm-synth



TASK 1: map the LDR to an appropriate range by using the scope

TASK 2: use the FSR to trigger an envelope once over a threshold

TASK 3: EXTRA try to connect simple-sequencer.

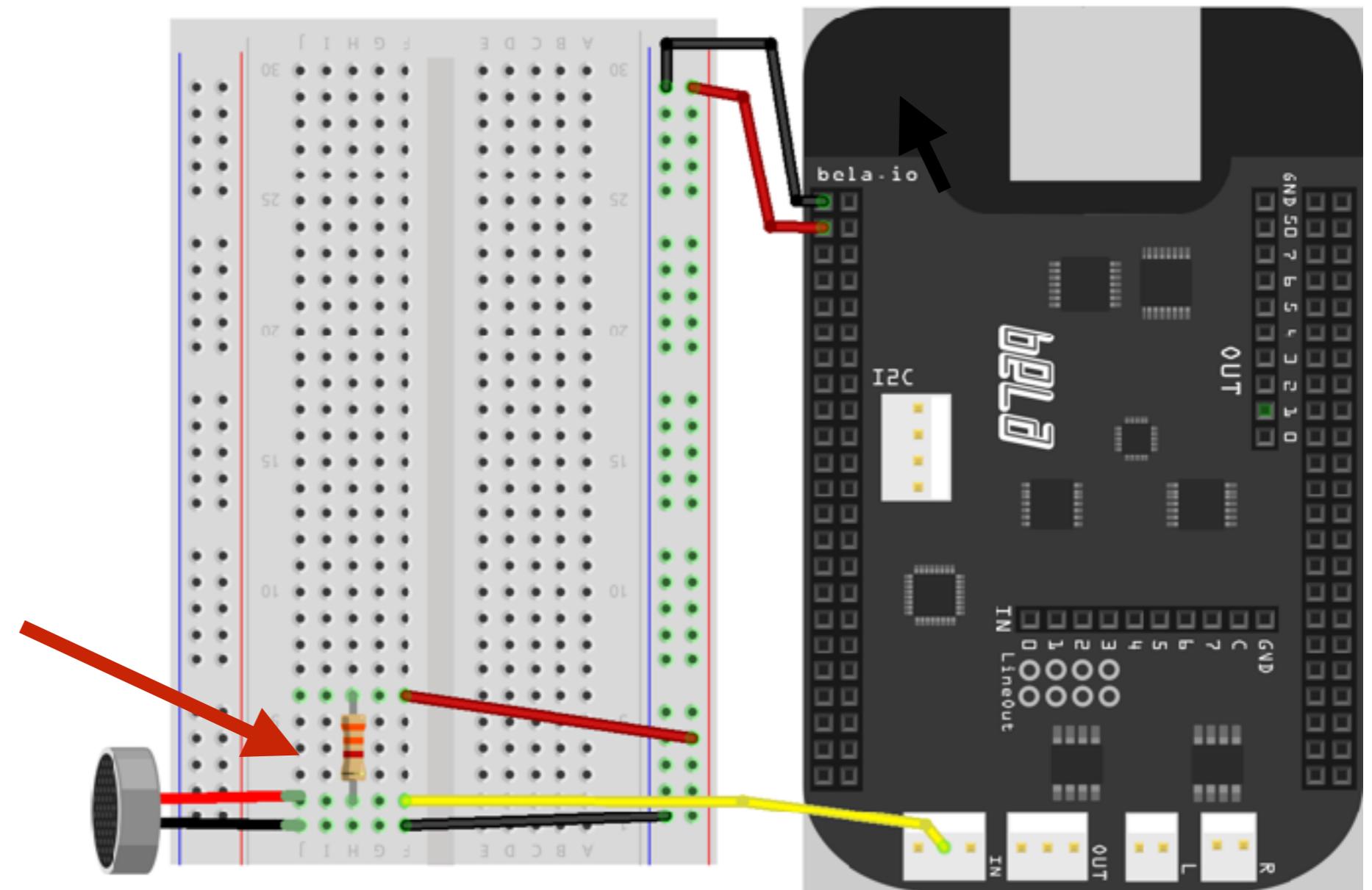
Microphone

examples/00-Workshop/6-mic-test

Add this to the existing components on the board

Pin attached to metal case goes to **ground**

Microphone needs resistor to +3.3V



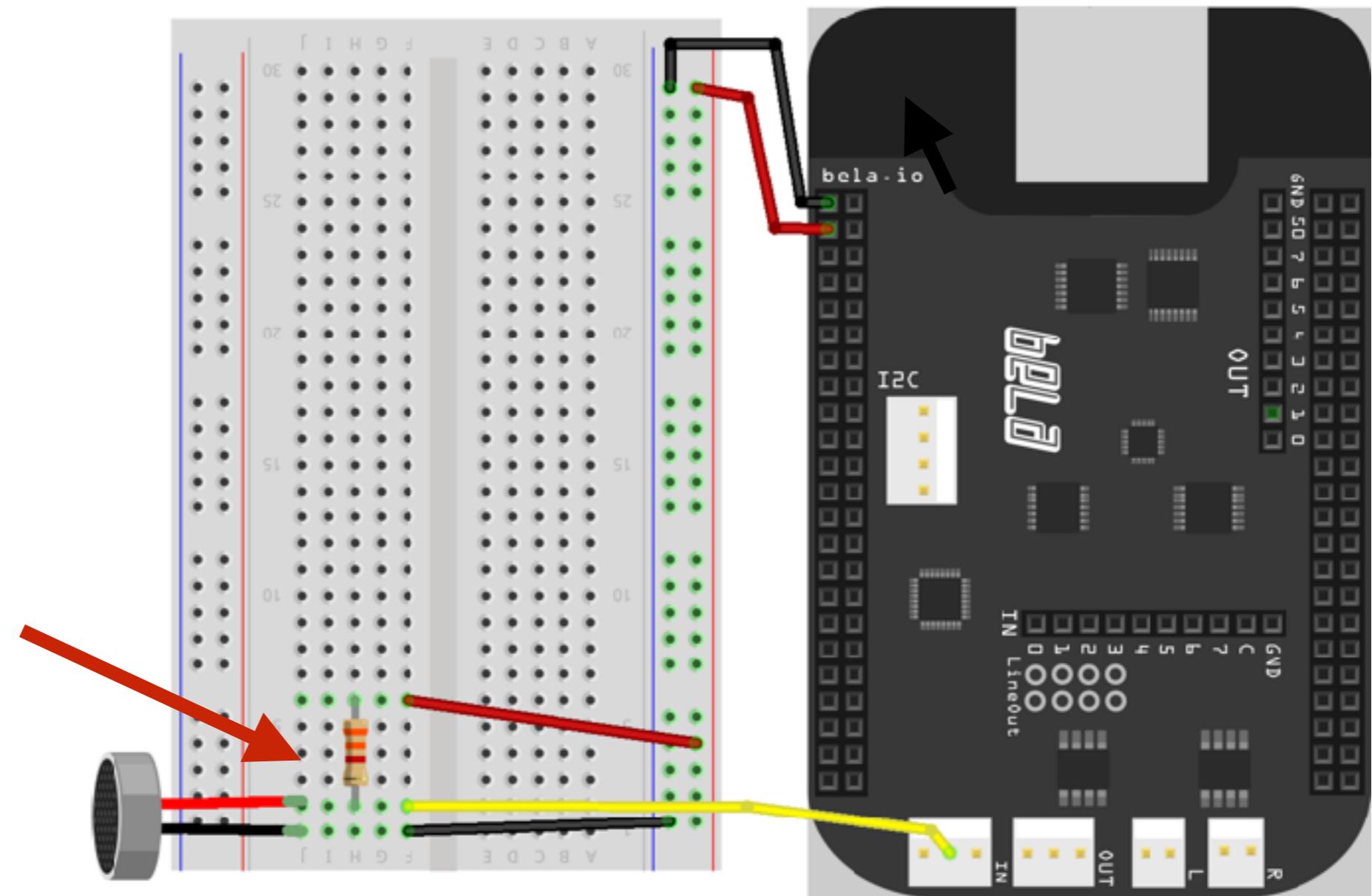
Sampler

examples/00-Workshop/7-sample-scrub

Using the same setup as the last example
(microphone plus other components)

Pin attached to
metal case goes
to **ground**

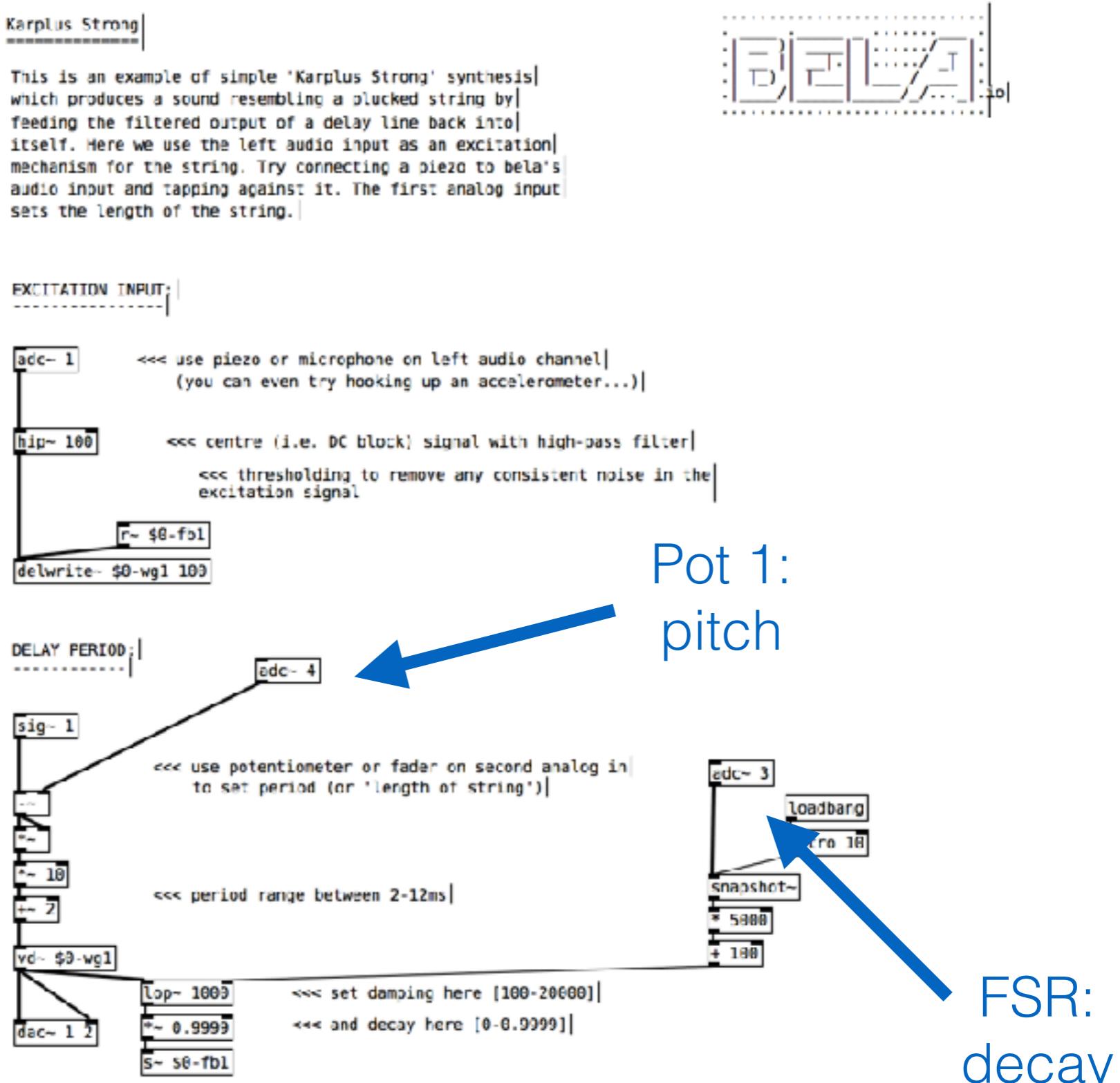
Microphone needs
resistor to +3.3V



Karplus-Strong (string synthesis)

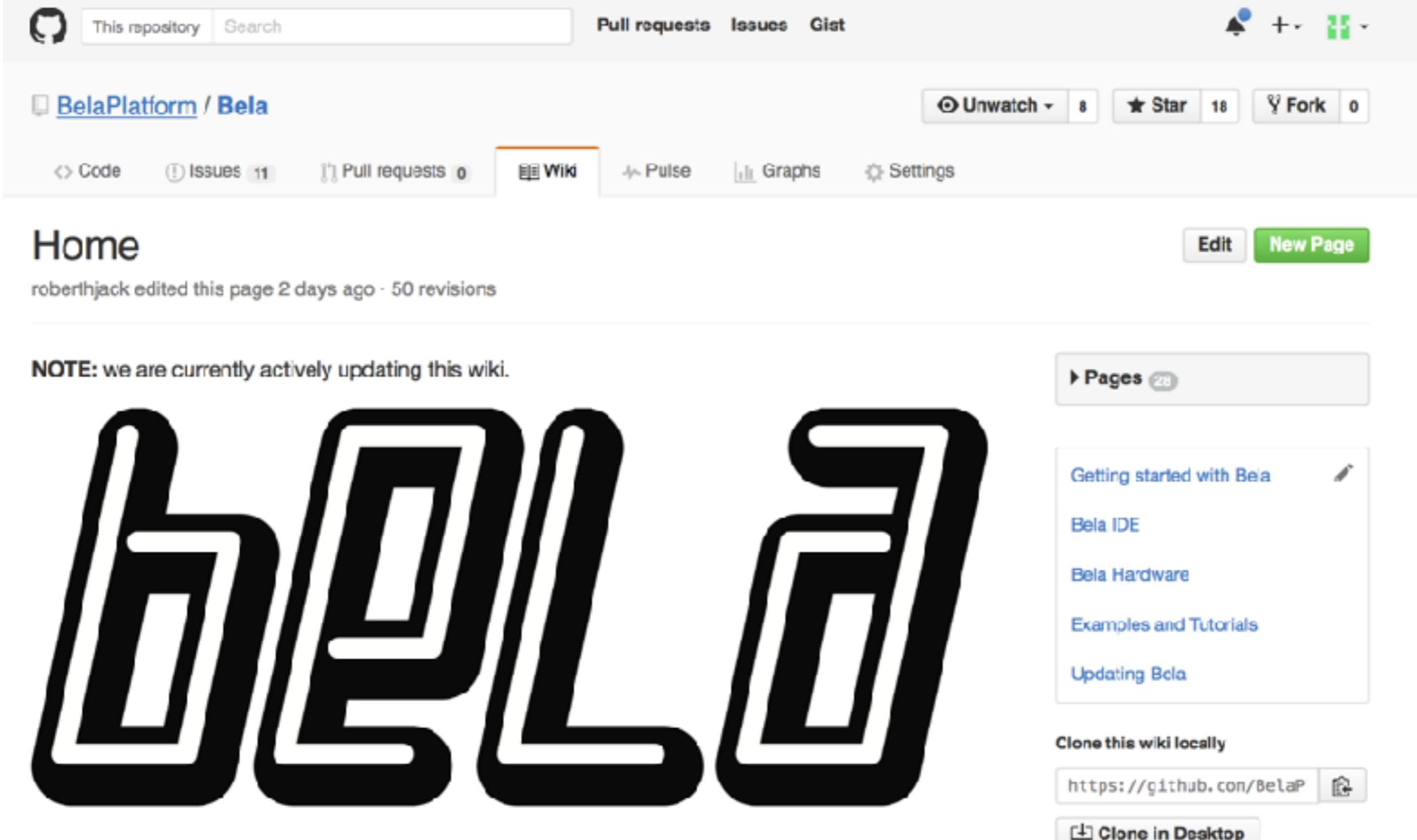
examples/00-Workshop/8-karplus-strong

Keep wiring!



Support resources: GitHub repo and wiki

<http://bela.io/code>



The screenshot shows the GitHub repository page for 'BelaPlatform / Bela'. The top navigation bar includes links for 'Pull requests', 'Issues', 'Gist', and a search bar. Below the header, the repository name 'BelaPlatform / Bela' is displayed, along with statistics: 8 unwatched, 18 stars, and 0 forks. A navigation bar below the header offers links to 'Code', 'Issues (11)', 'Pull requests (0)', 'Wiki' (which is selected), 'Pulse', 'Graphs', and 'Settings'. On the right side of the main content area, there are buttons for 'Edit' and 'New Page'. A note at the top of the page states: 'NOTE: we are currently actively updating this wiki.' To the right, a sidebar titled 'Pages (28)' lists several pages: 'Getting started with Bela', 'Bela IDE', 'Bela Hardware', 'Examples and Tutorials', and 'Updating Bela'. At the bottom of the sidebar, there are links for 'Clone this wiki locally' (with a URL: <https://github.com/BelaP/Bela>) and 'Clone in Desktop'.

Introducing Bela

Bela is an open-source embedded system for real-time audio processing with ultra-low latency. Based on the BeagleBone Black single-board computer and featuring a custom hardware and software environment, Bela integrates audio processing and sensor connectivity in a single high-performance package. Bela has been designed to be a flexible platform for musicians, instrument designers, audio enthusiasts and interactive artists. On this Wiki you'll find all the information you'll

Support resources: Bela forum

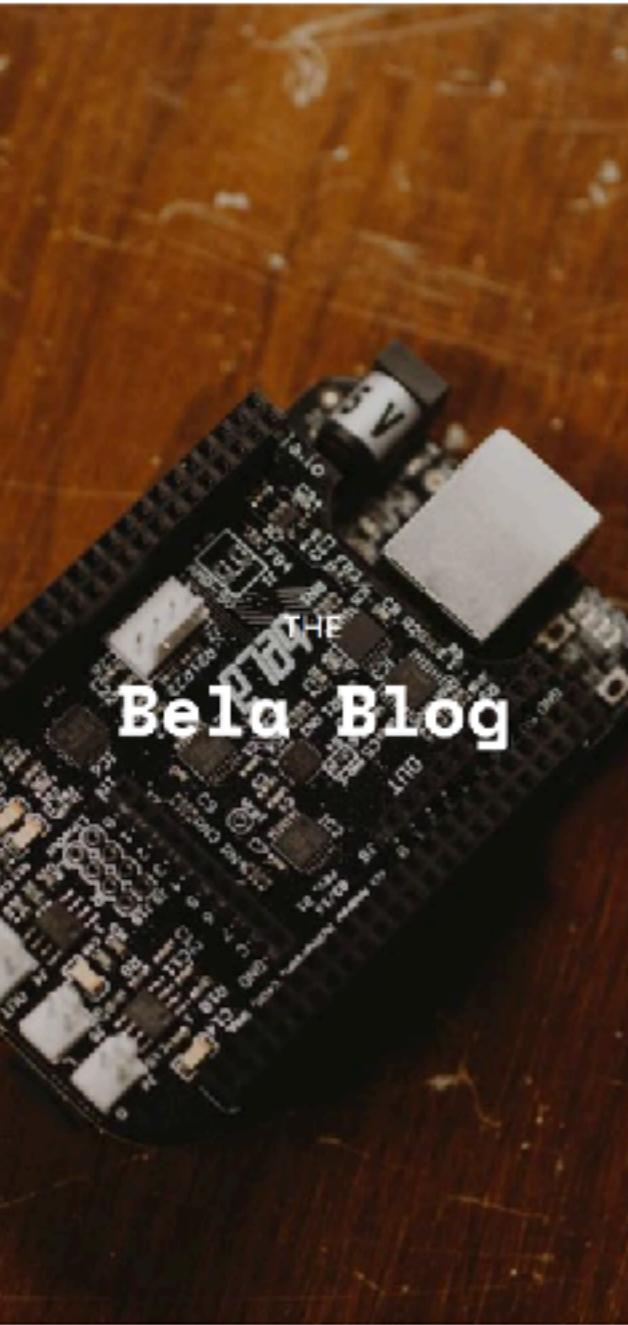
<http://forum.bela.io>

The screenshot shows the Bela forum homepage with the following layout:

- Header:** Includes the Bela logo, navigation links (Home, Guidelines, Code, Docs, bela.io), a search bar, and a user profile for "andrew".
- Left Sidebar:** Features a "Start a Discussion" button, a "Latest" dropdown menu set to "Latest", and a sidebar with links for "All Discussions", "Following", "Tags", and categories like FAQ, General, Getting Started, Interactivity, Audio, Hardware, Software, and Show and Tell.
- Discussion List:** A list of 12 forum posts arranged vertically, each with a user icon, title, last reply, and tags.
 - Eurorack Input** by UlrichH (Hardware, Eurorack) - 3 replies
 - Code blocks** by korhanerel (Getting Started) - 5 replies
 - Hooking up an LCD** by nuromantik (Hardware) - 7 replies
 - PRU vs McASP** by andrew (Hardware) - 3 replies
 - Critter&Guattari Organelle and Bela** by andrew (Hardware) - 1 reply
 - MIDI Interface** by giuliomoro (Hardware) - 3 replies
 - Instructions for newbies, laypeople, etc** by bela_robert (General) - 1 reply
 - How to multiplex digital inputs** by nuromantik (Hardware) - 2 replies
 - Bela API: a new hope. Porting code from the old to the new stable API** by giuliomoro (Software, C/C++) - 0 replies
 - Forum Guidelines** by astrid (General) - 0 replies
 - Purchasing** by astrid (Getting Started) - 1 reply

Bela blog

[http://blog.bela.io](https://blog.bela.io)



A screenshot of a web browser displaying the Bela blog at <https://blog.bela.io>. The page features a header with three navigation links: "BLOG" (underlined), "CATEGORIES", and "ABOUT BELA". Below the header, there are two blog posts. The first post, dated Sep 14, 2018, is titled "Giraf by Hjalte Bested Møller: a self-contained polyphonic sampler" and includes a brief description and a "READ MORE" button. The second post, dated Sep 7, 2018, is titled "STEAMY by Rob Blazey" and includes a similar description and button. The browser's address bar shows the URL <https://blog.bela.io/2018/09/14/Giraf-Bela-Hjalte-Bested-Møller/>.

Sep 14, 2018

Giraf by Hjalte Bested Møller: a self-contained polyphonic sampler

This post introduces Giraf, a polyphonic sampler created Hjalte Bested Møller. The instrument uses Bela in combination with a distance sensor, piezo disc, and embedded speakers to create a mobile...

[READ MORE](#)

Sep 7, 2018

STEAMY by Rob Blazey

In this post Rob Blazey introduces us to STEAMY, a solar-powered interactive sound-sculpture that makes use of Bela. Over to Rob: This piece was made for an exhibition at Cragside...

<https://blog.bela.io/2018/09/14/Giraf-Bela-Hjalte-Bested-Møller/>



Stay tuned! Join the announcement list at

<http://bela.io>

£10 off starter kits thru 31 May using code **SUPERBOOTH19**

<http://shop.bela.io>

C++ API

- In `render.cpp`....
- Three main functions:
- **setup()**
*runs once at the beginning, before audio starts
gives channel and sample rate info*
- **render()**
*called repeatedly by Bela system ("callback")
passes input and output buffers for audio and sensors*
- **cleanup()**
*runs once at end
release any resources you have used*
- Code docs available in sidebar of IDE, or at docs.bela.io

Example: sinetone

```
#include <Bela.h>
#include <cmath>

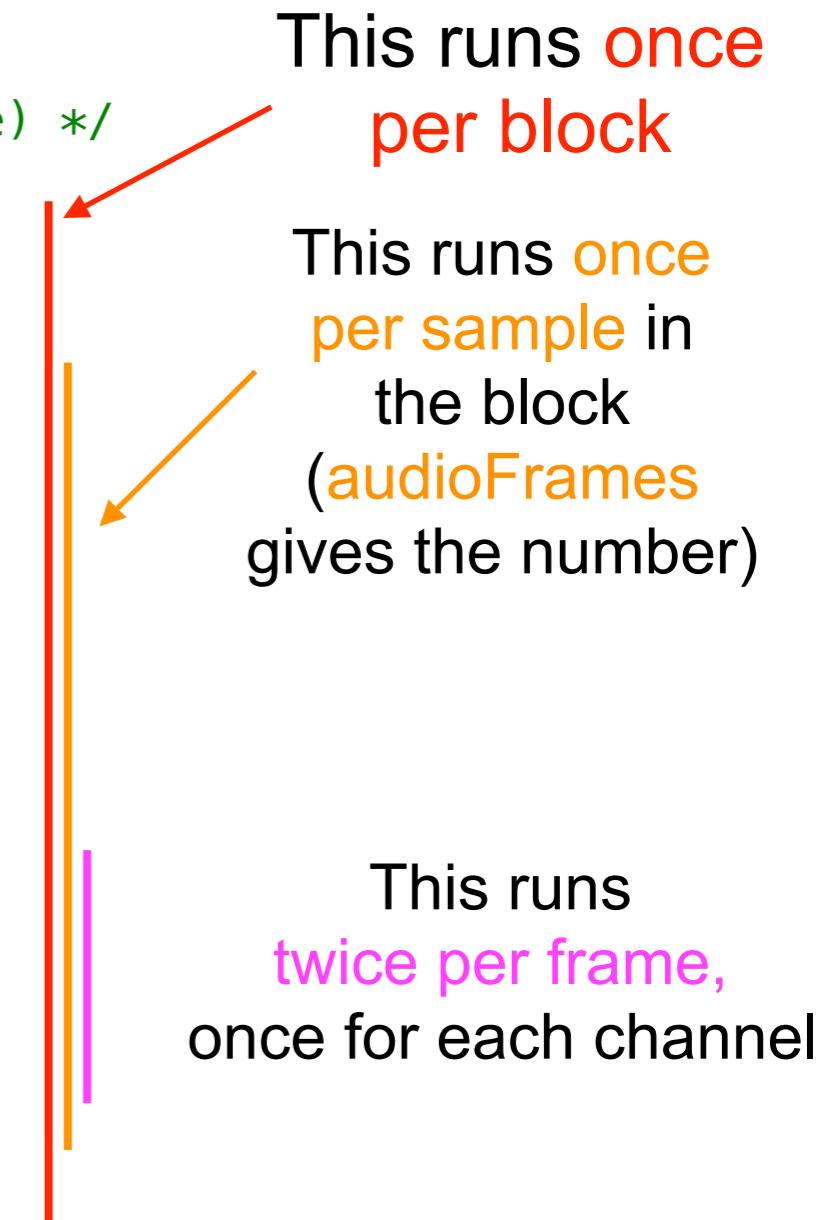
float gPhase = 0.0; /* Phase of the oscillator (global variable) */

void render(BelaContext *context, void *userData)
{
    /* Iterate over the number of audio frames */
    for(unsigned int n = 0; n < context->audioFrames; n++) {
        /* Calculate the output sample based on the phase */
        float out = 0.8 * sinf(gPhase);

        /* Update the phase according to the frequency */
        gPhase += 2.0 * M_PI * gFrequency * gInverseSampleRate;
        if(gPhase > 2.0 * M_PI)
            gPhase -= 2.0 * M_PI;

        for(unsigned int channel = 0;
            channel < context->audioOutChannels; channel++) {
            /* Store the output in every audio channel */
            audioWrite(context, n, channel, out);
        }
    }
}
```

write to buffer of interleaved audio data,
specifying a **frame**, a **channel** and a **value**

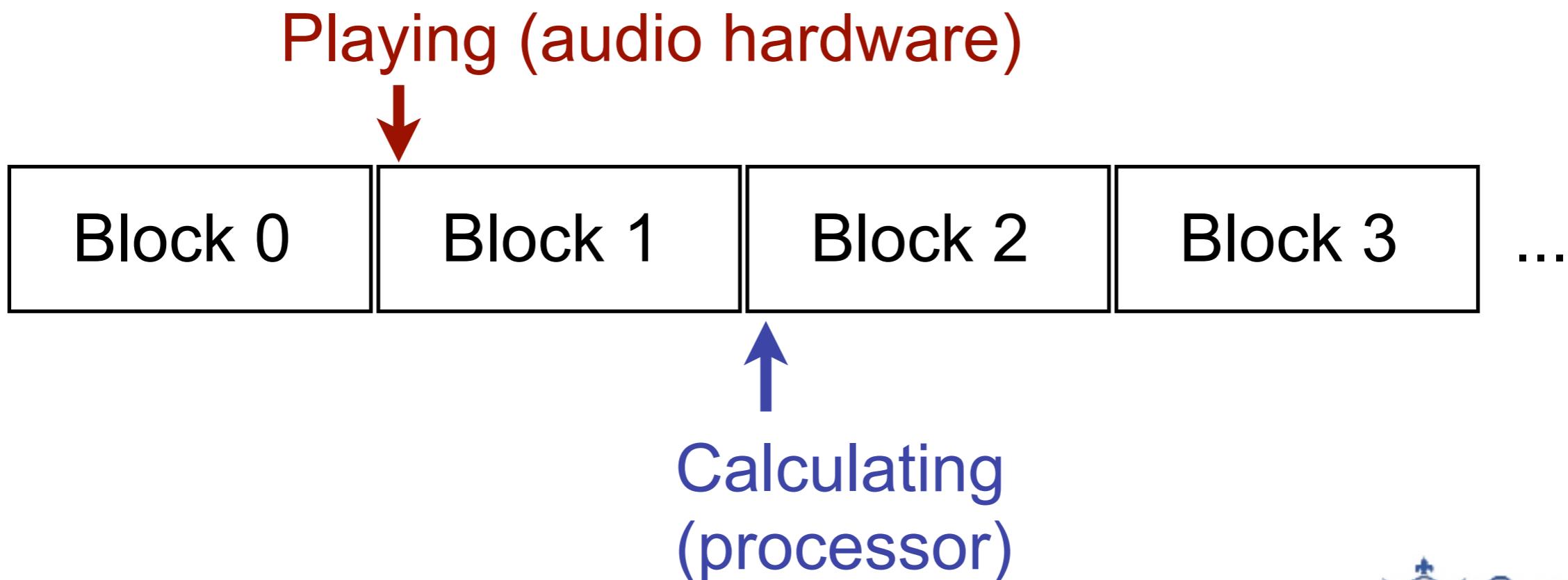


Real-time audio

- Suppose we have code that runs **offline**
 - ▶ (non-real time)
- Our goal is to re-implement it **online** (real time)
 - ▶ Generate audio **as we need it!**
 - ▶ Why couldn't we just generate it all in advance, and then play it when we need it?
- Digital audio is composed of **samples**
 - ▶ 44100 samples per second in our example
 - ▶ That means we need a new sample every $1/44100$ seconds (about every $23\mu\text{s}$)
 - ▶ So option #1 is to run a short bit of code every sample whenever we want to know what to play next
 - ▶ What might be some drawbacks of this approach?
 - Can we guarantee we'll be ready for each new sample?

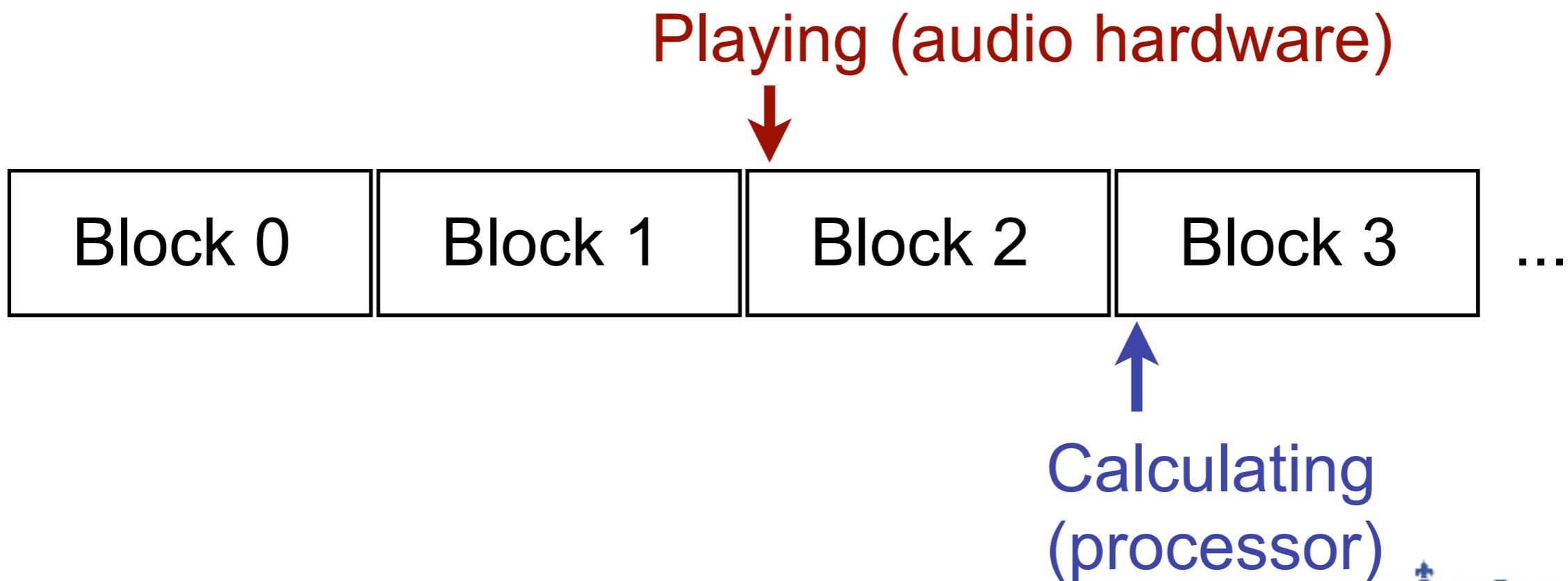
Block-based processing

- Option #2: Process in **blocks** of several samples
 - ▶ Basic idea: generate enough samples to get through the next few milliseconds
 - ▶ Typical **block sizes**: 32 to 1024 samples
 - Usually a power of 2 for reasons having to do with hardware
 - ▶ While the audio hardware is busy playing one block, we can start calculating the next one so it's ready on time:



Block-based processing

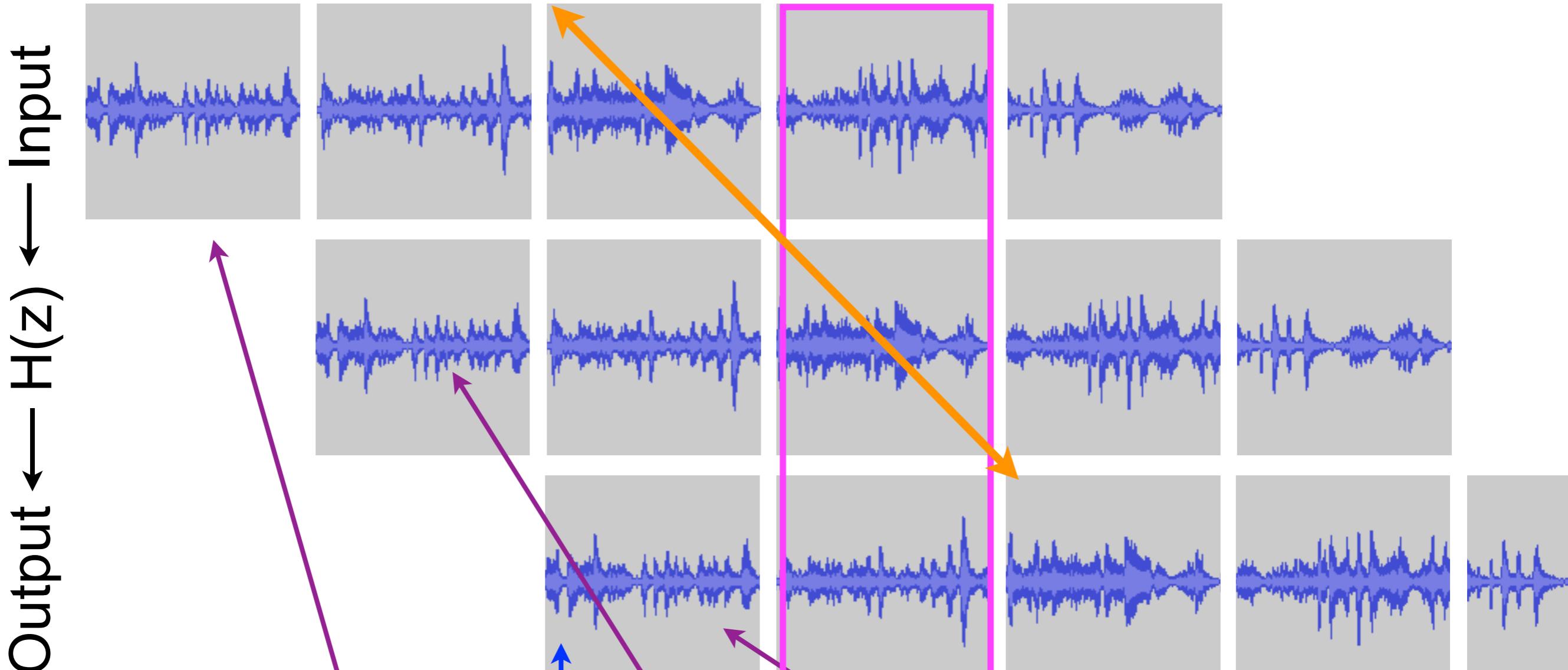
- Option #2: Process in **blocks** of several samples
 - ▶ Basic idea: generate enough samples to get through the next few milliseconds
 - ▶ Typical **block sizes**: 32 to 1024 samples
 - Usually a power of 2 for reasons having to do with hardware
 - ▶ While the audio hardware is busy playing one block, we can start calculating the next one so it's ready on time:



Block-based processing

- Advantages of blocks over individual samples
 - ▶ We need to run our function less often
 - ▶ We always generate one block ahead of what is actually playing
 - ▶ Suppose one block of samples lasts 5ms, and running our code takes 1ms
 - ▶ Now, we can tolerate a delay of up to 4ms if the OS is busy with other tasks
 - ▶ Larger block size = can tolerate more variation in timing
- What is the disadvantage?
 - ▶ Latency (delay)

Buffering illustration



1. First we fill up a buffer of samples

At any given time, we are reading from ADC, processing, and writing, we send this buffer to the output while the next one buffers to the output
Total latency is 2x buffer length