Open-source maker platform for beautifully responsive interactive audio

Adan L. Benito

@BelaPlatform

EPSRC
centre for digital music
Queen Mary University of London

bela.io

Programmable Audio Workshop

1st of December, 2018

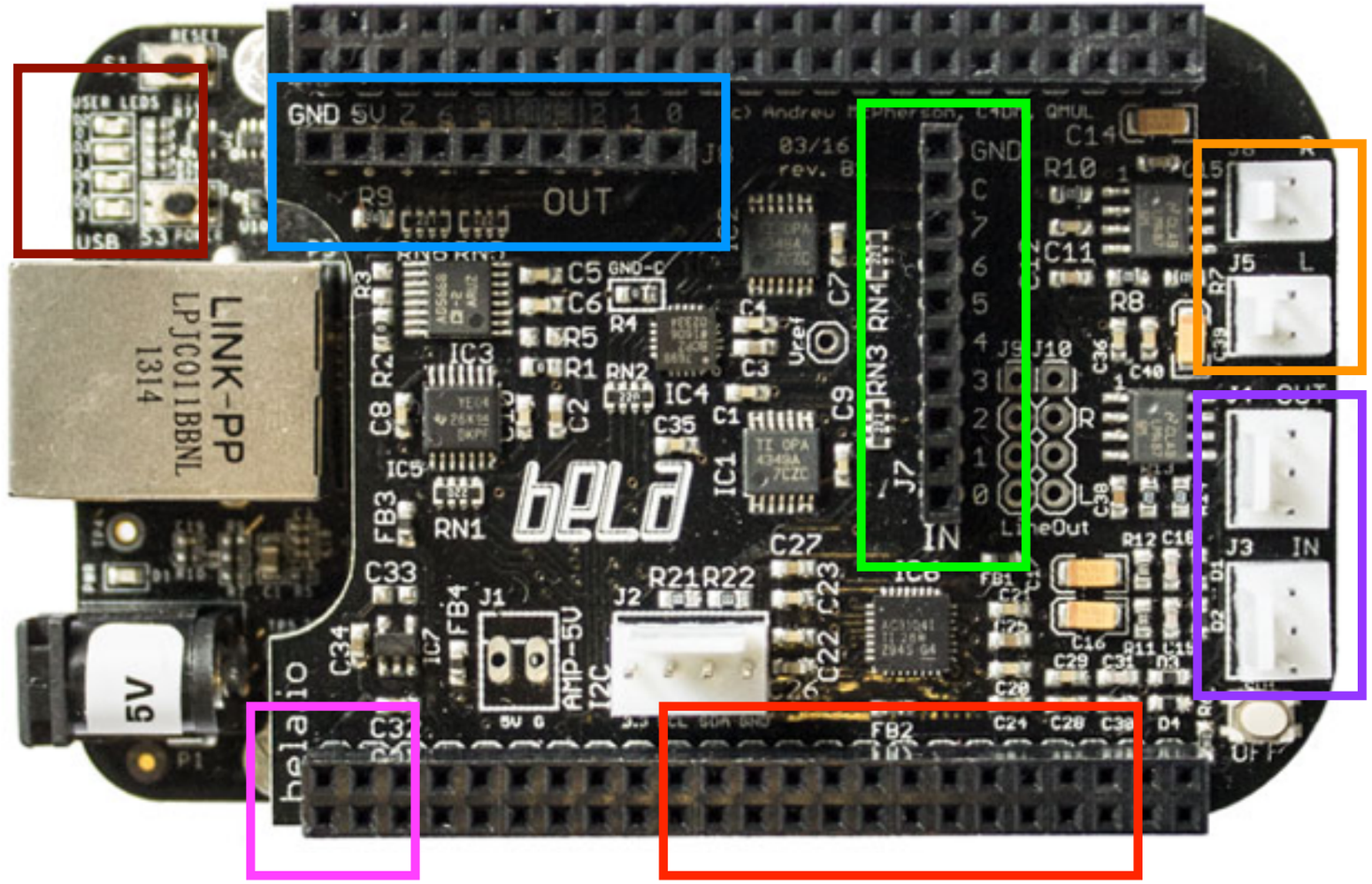
GRAME-CNCM, Lyon

Bela hardware

Analog outputs

Analog inputs

USB



Stereo speaker amps

Stereo audio in /out

Power / Ground

Digital inputs/outputs

Required software

Firefox/Chrome browser

The IDE may or may not work in other browsers.

Getting started

1. Plug in Bela to computer (USB)
2. Launch the browser
3. After around 30s go to the url:

MacOS, Linux	http://bela.local/
Windows	http://192.168.6.2

Troubleshooting: drivers

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

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

Troubleshooting: drivers

Step 1: Plug in Bela by USB

Wait for it to boot

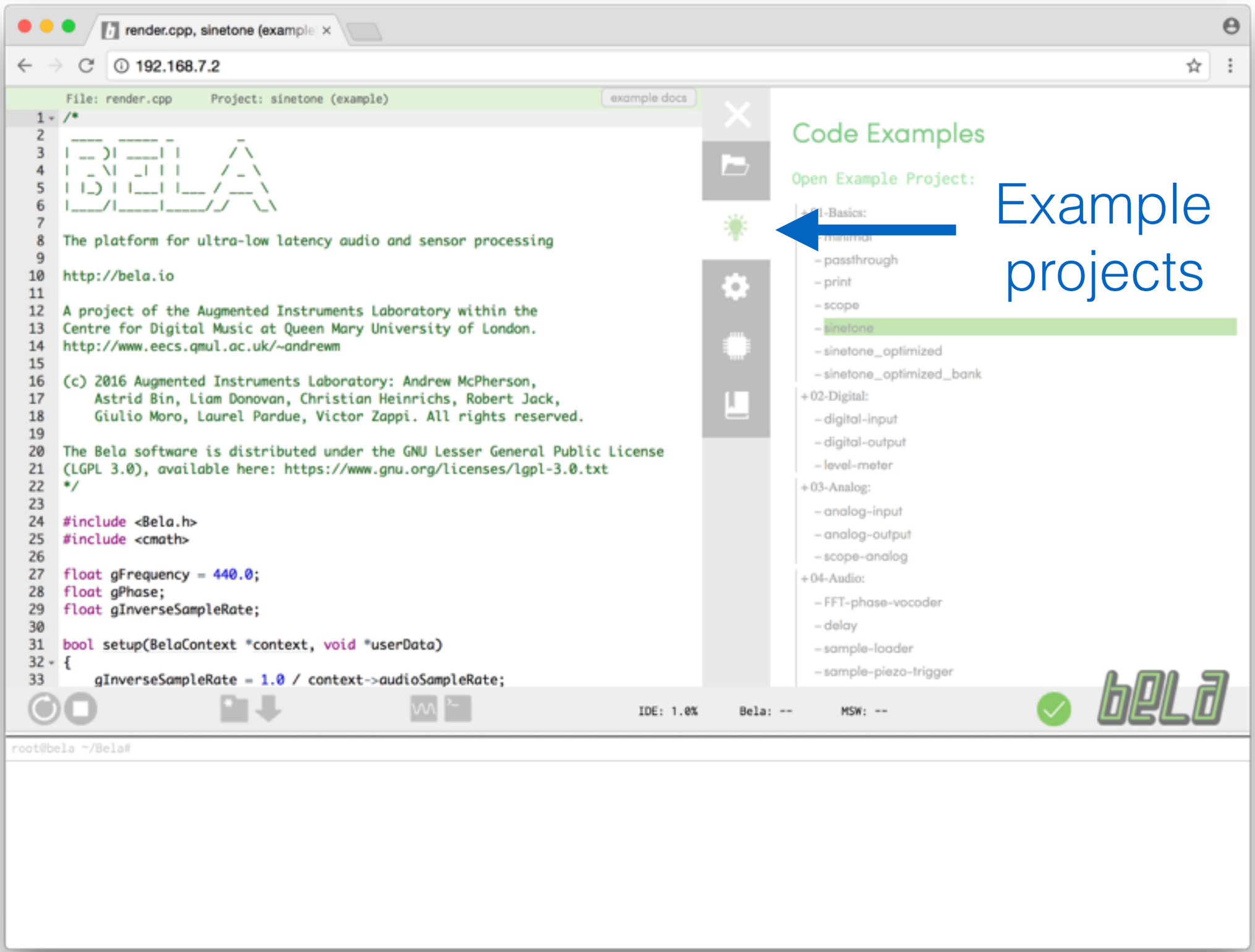
You should see a drive BELABOOT come up

Step 2: Install software from Drivers directory

Step 3: Reboot computer

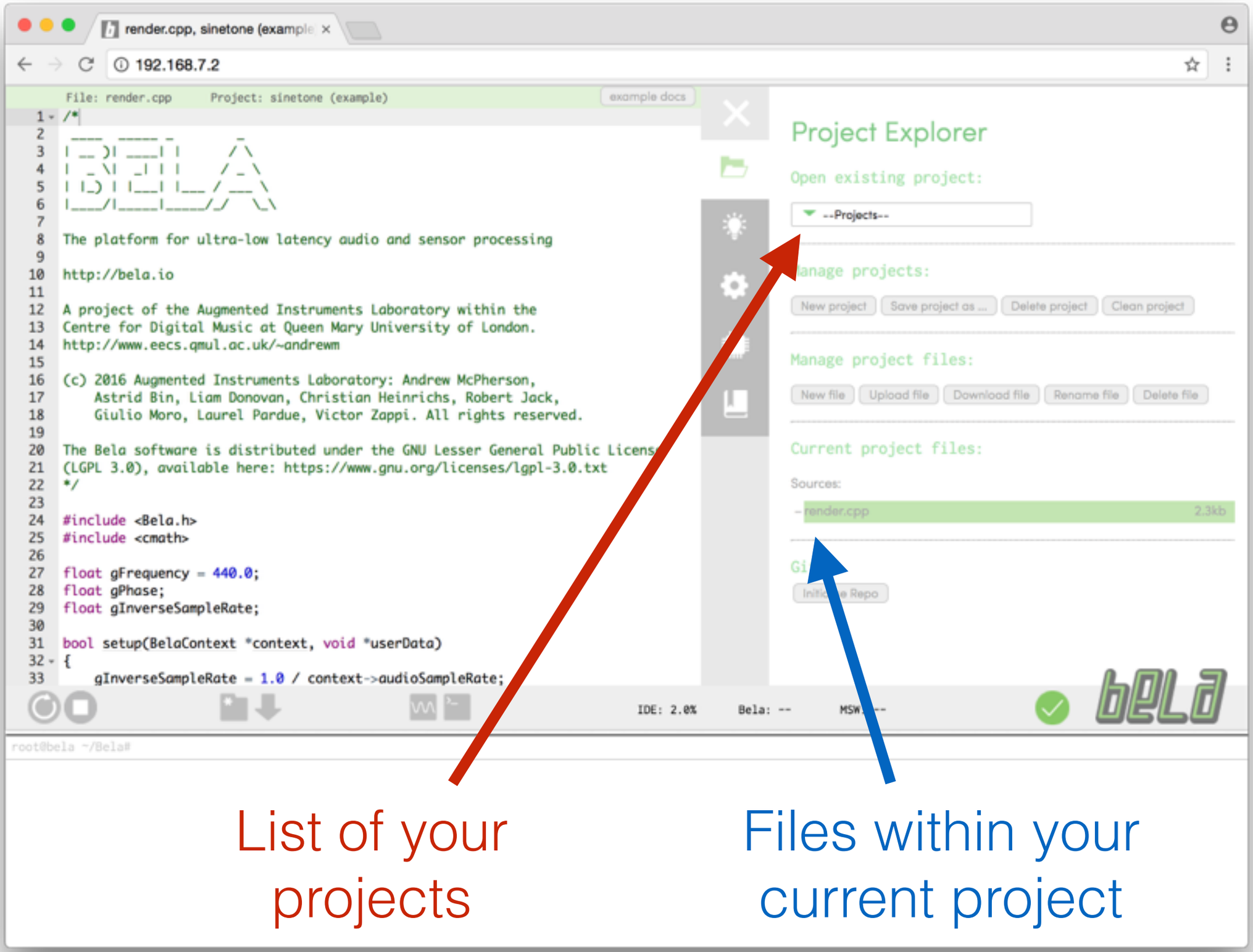
Bring up the Bela IDE:

http://bela.local/



Bring up the Bela IDE:

http://bela.local/

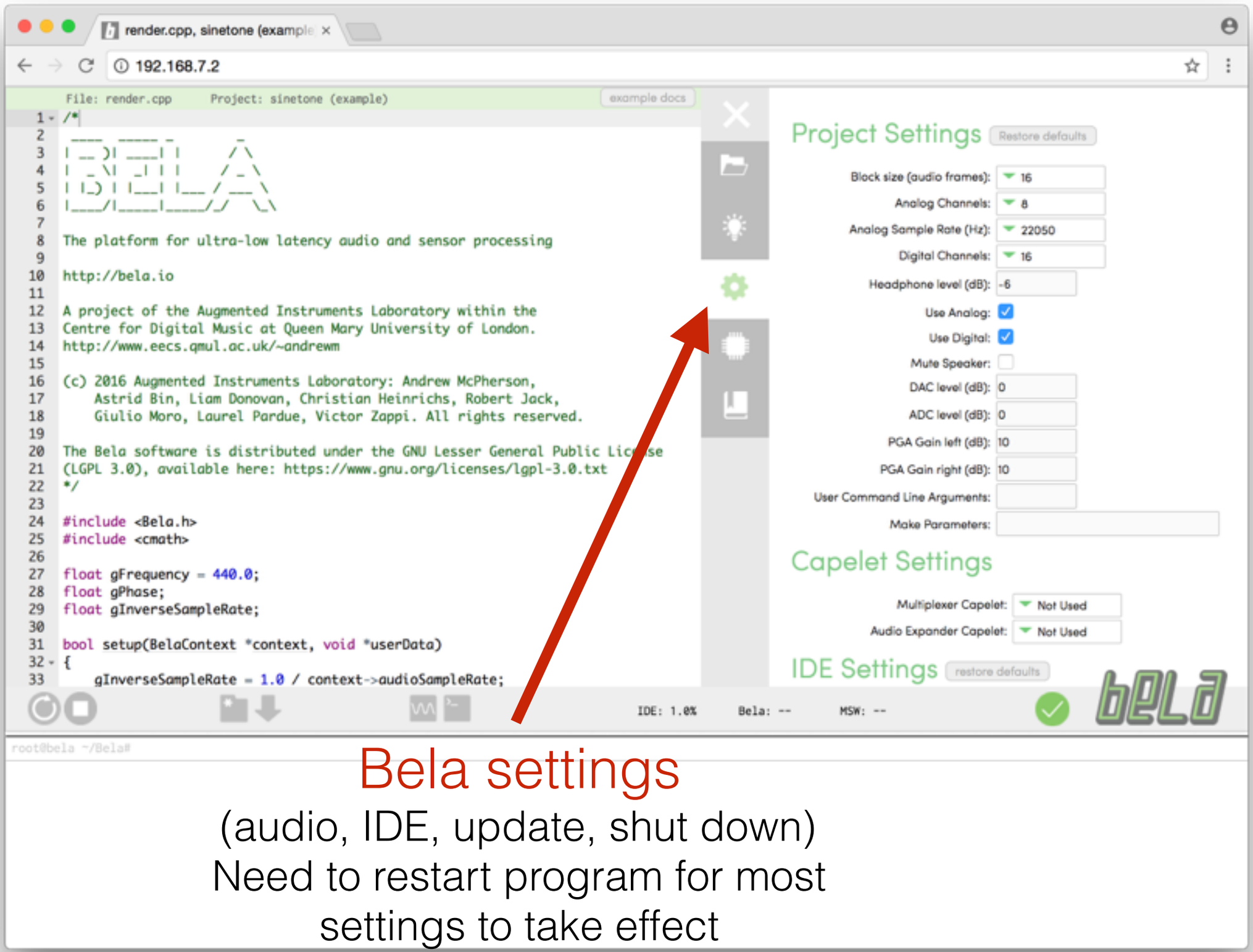


List of your projects

Files within your current project

Bring up the Bela IDE:

http://bela.local/

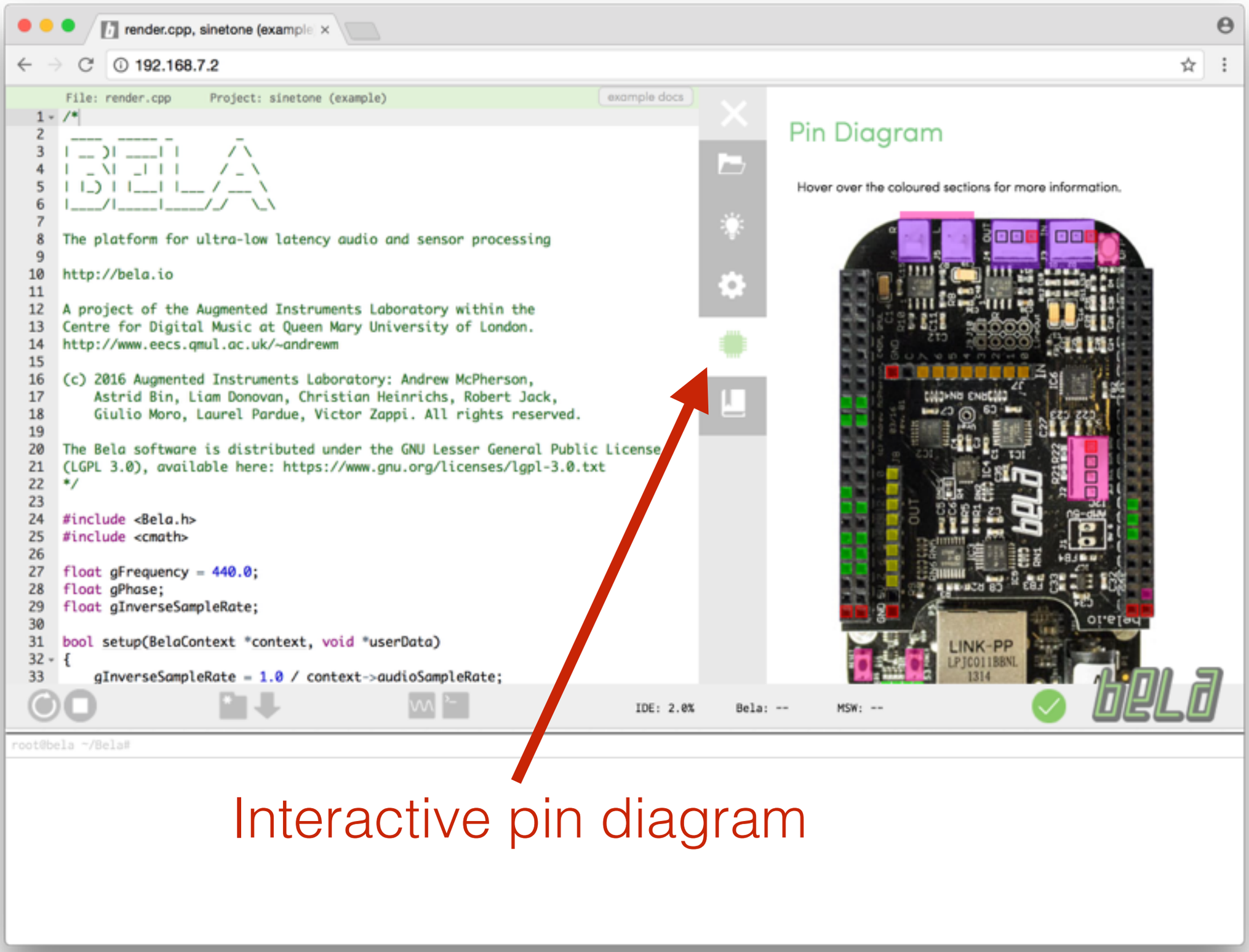


Bela settings

(audio, IDE, update, shut down)
Need to restart program for most settings to take effect

Bring up the Bela IDE:

http://bela.local/

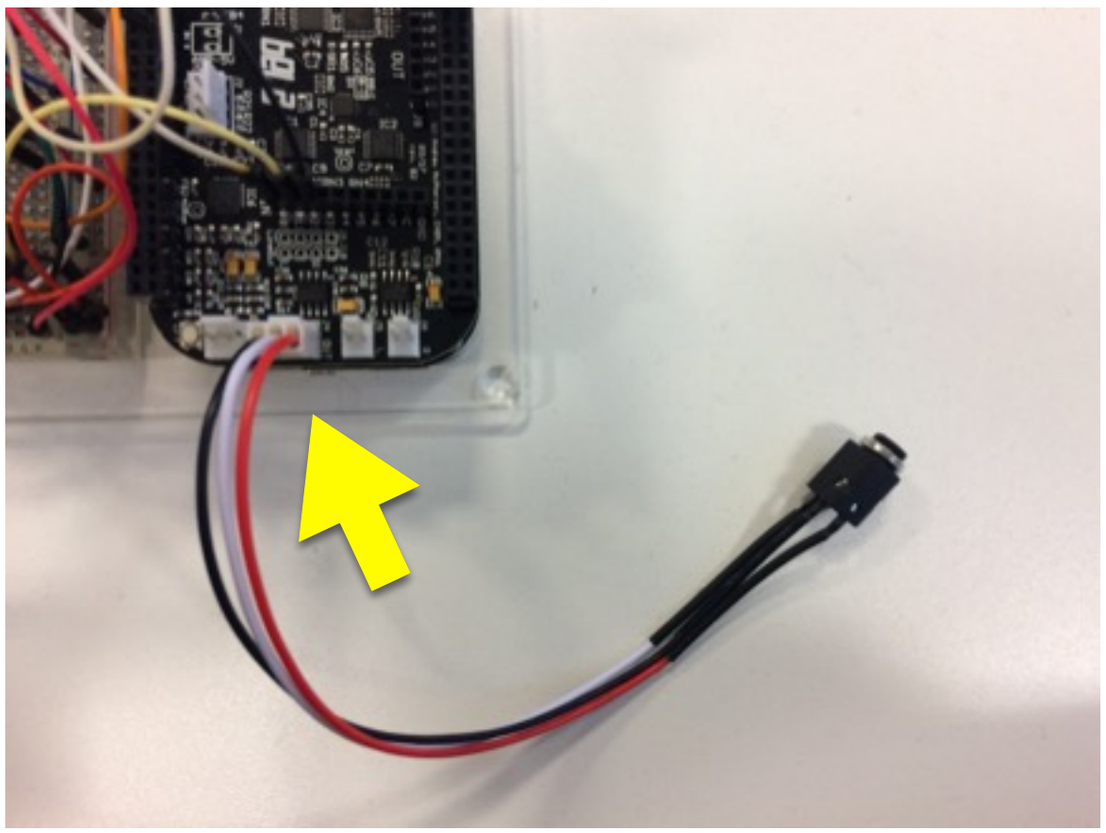


Interactive pin diagram

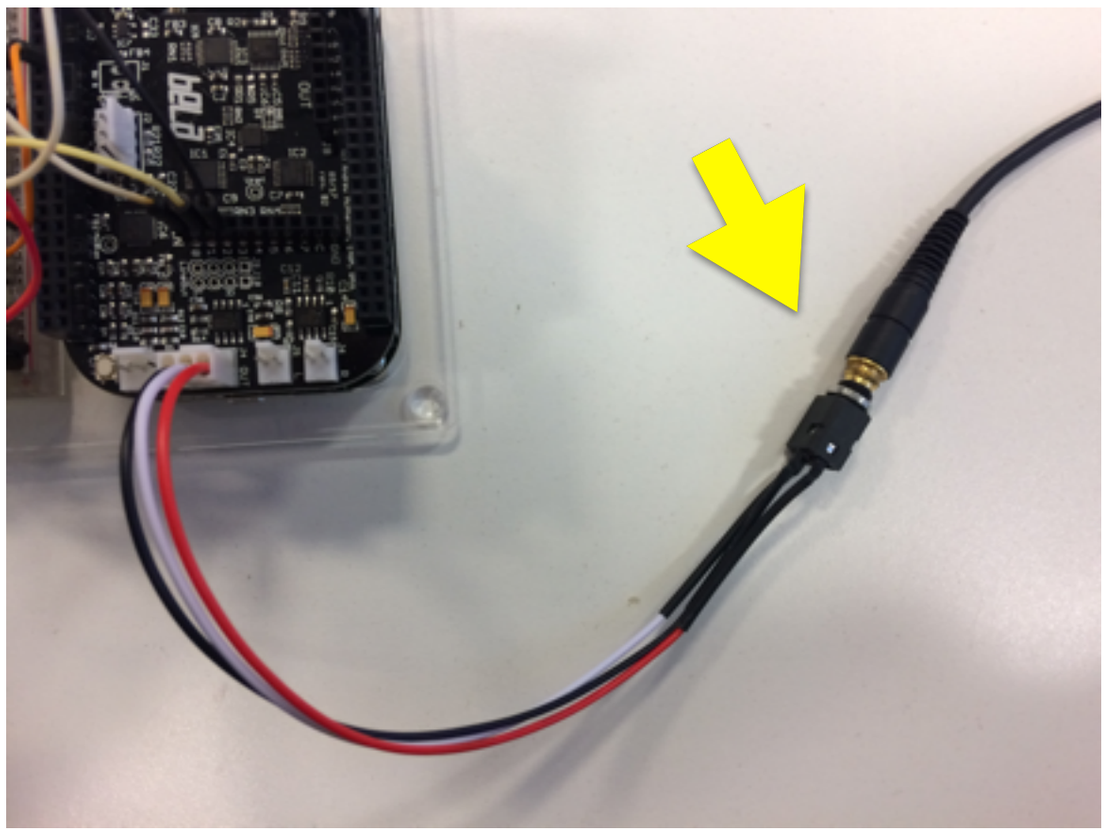
A C++ project is now running **on the board**
not on your personal computer

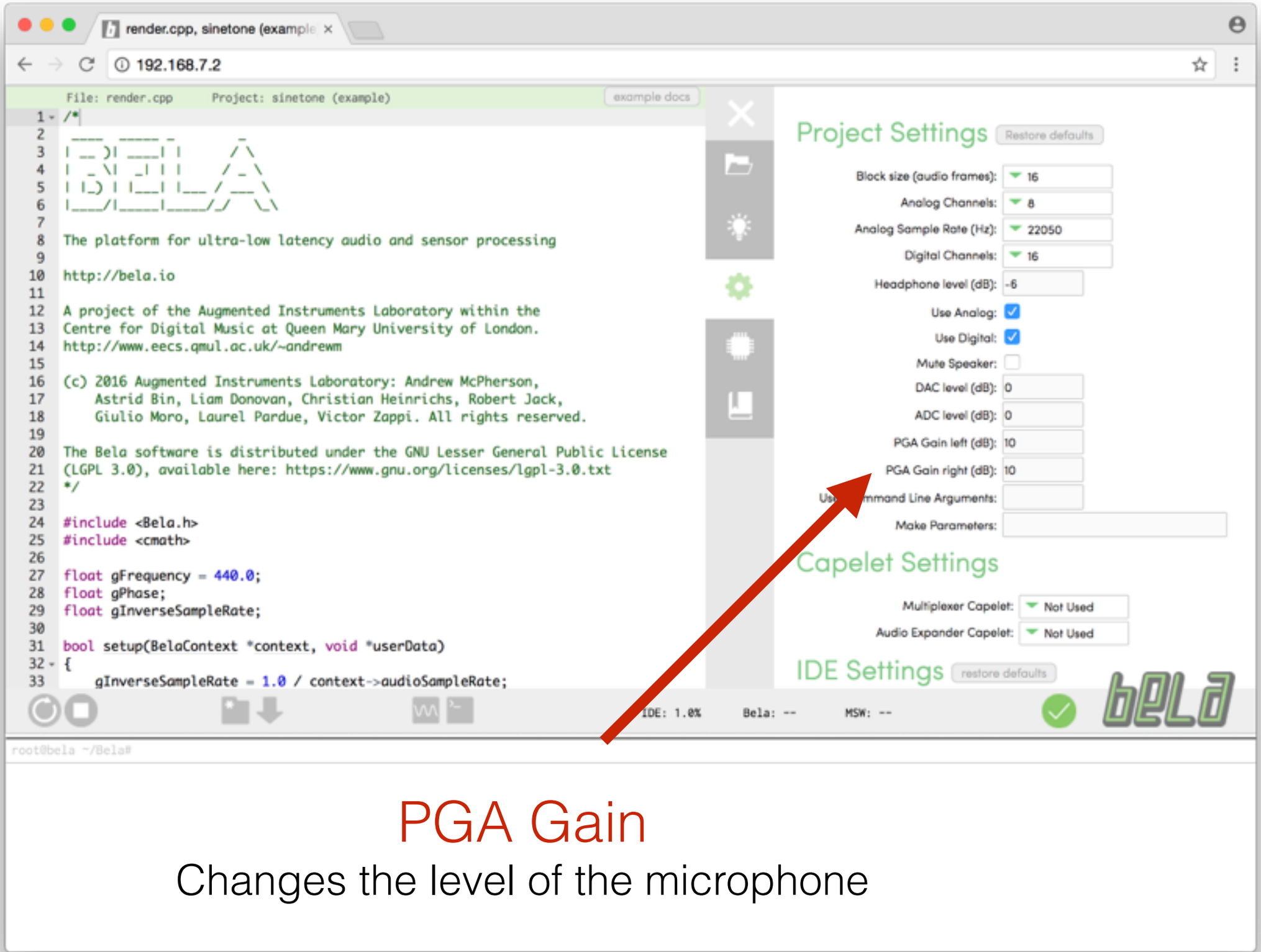
To listen:

Plug the audio adapter cable into the 2nd audio out molex connector



Plug headphones into the audio adapter cable





PGA Gain

Changes the level of the microphone

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

Hello world: 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);
        }
    }
}
```

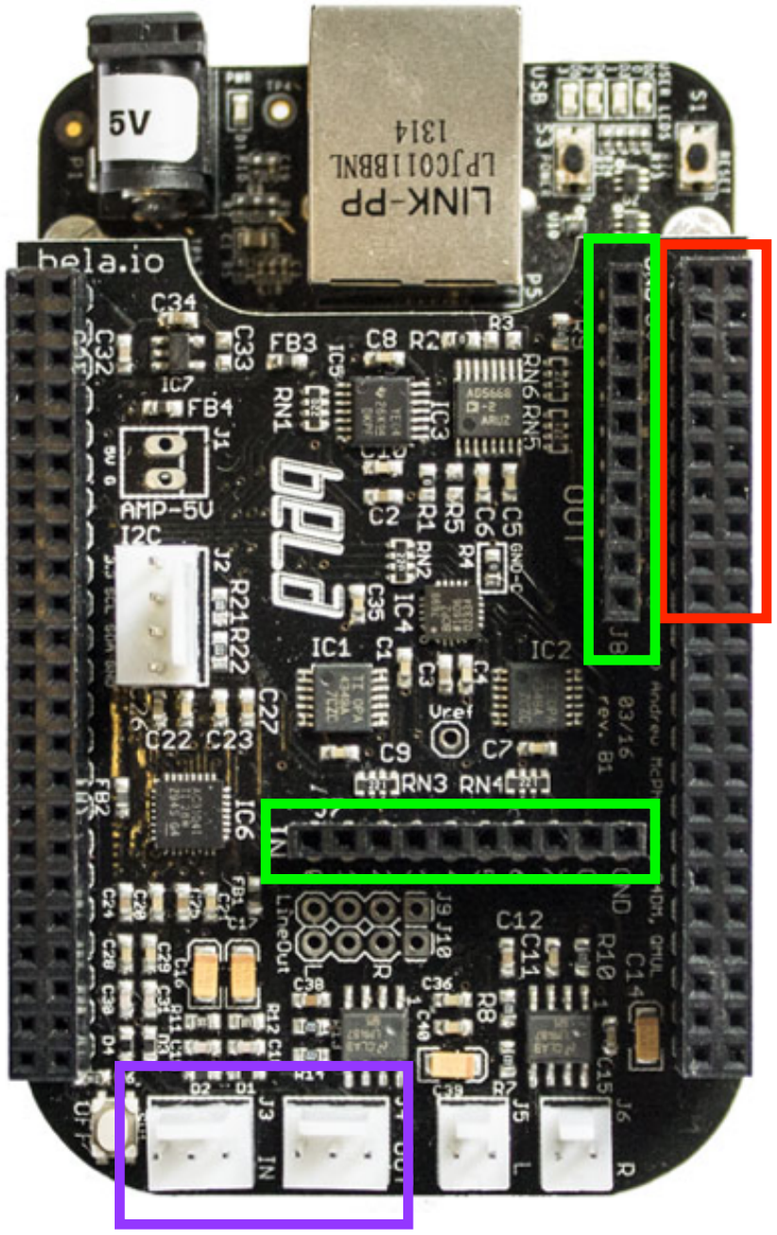
This runs once per block

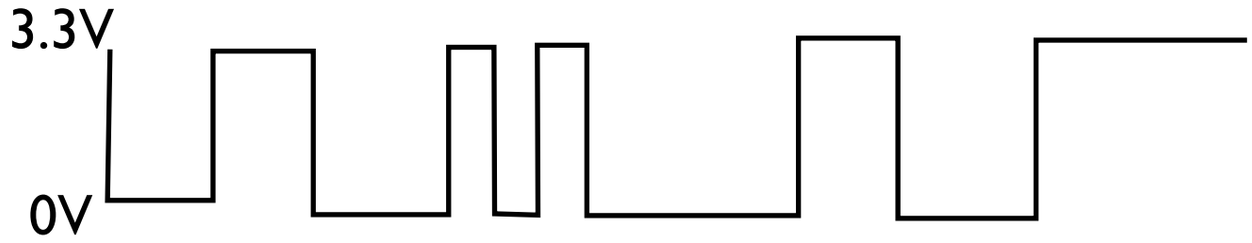
This runs once per sample in the block (audioFrames gives the number)

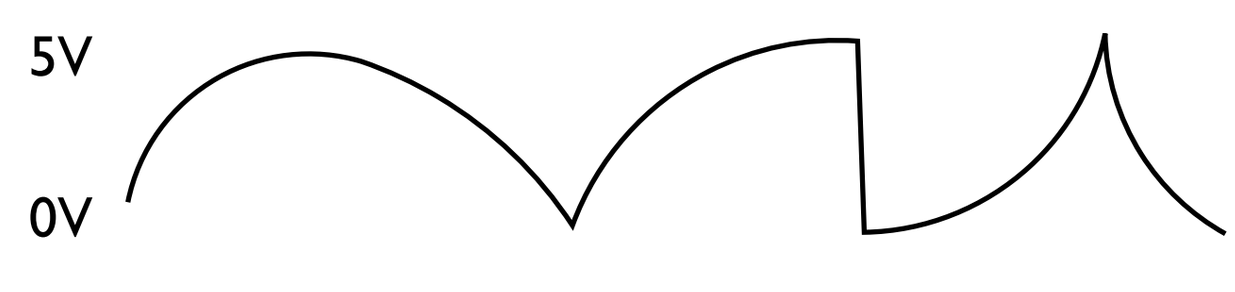
This runs twice per frame, once for each channel

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

Bela hardware





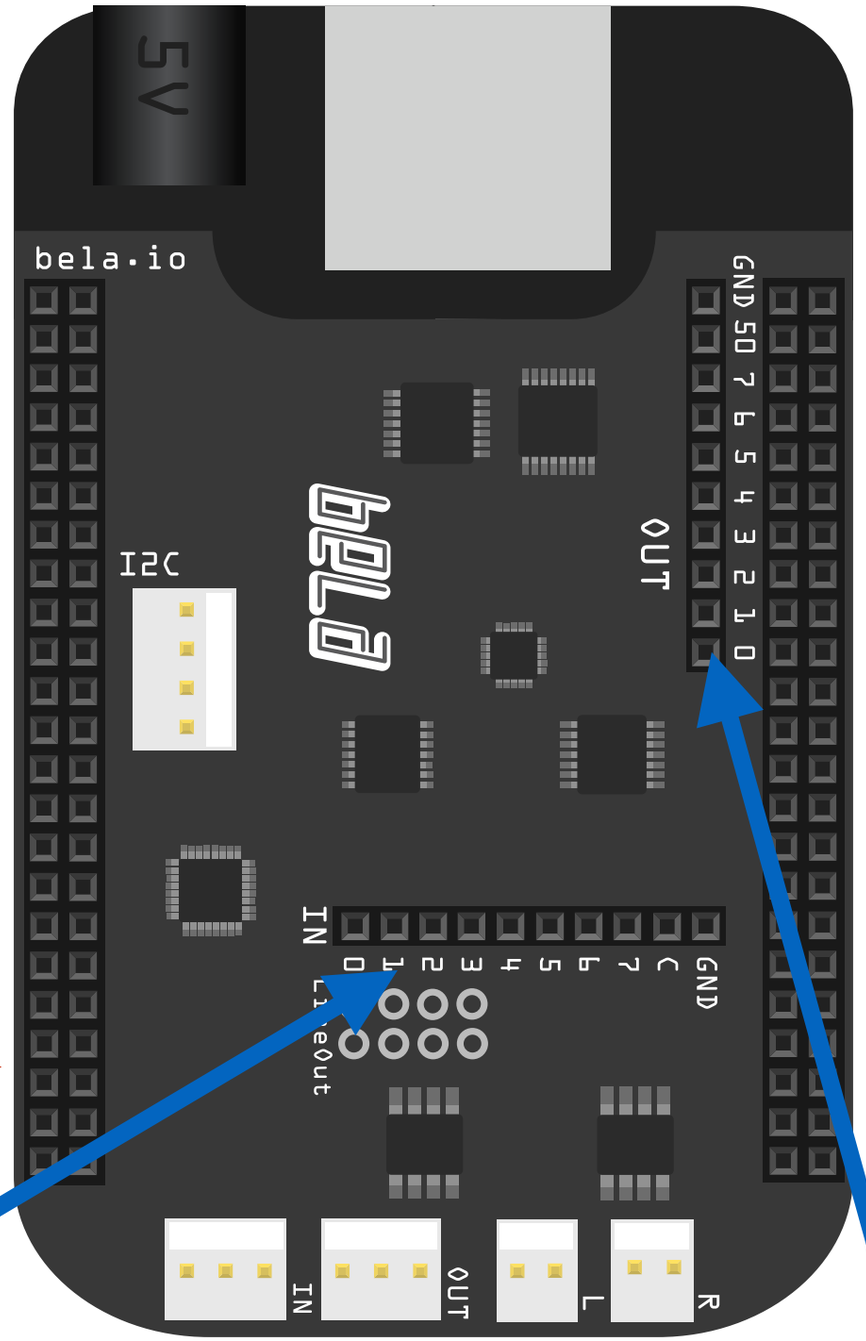


Bela pin numbering

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

1	2
3	4
5	6
7	8
9	10
11	12
13	14
...	

P9



1	2
3	4
5	6
7	8
9	10
11	12
13	14
...	

P8

Analog In

also ground

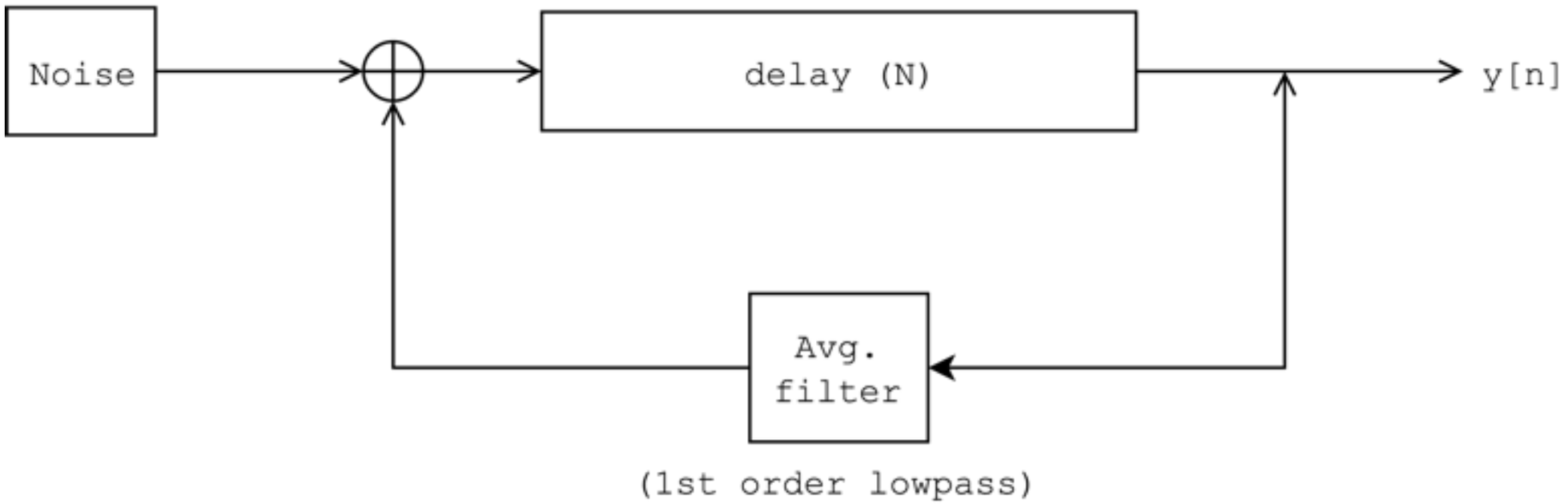
Analog Out

also 5V (+100Ω), ground

Keep everything on your breadboard!

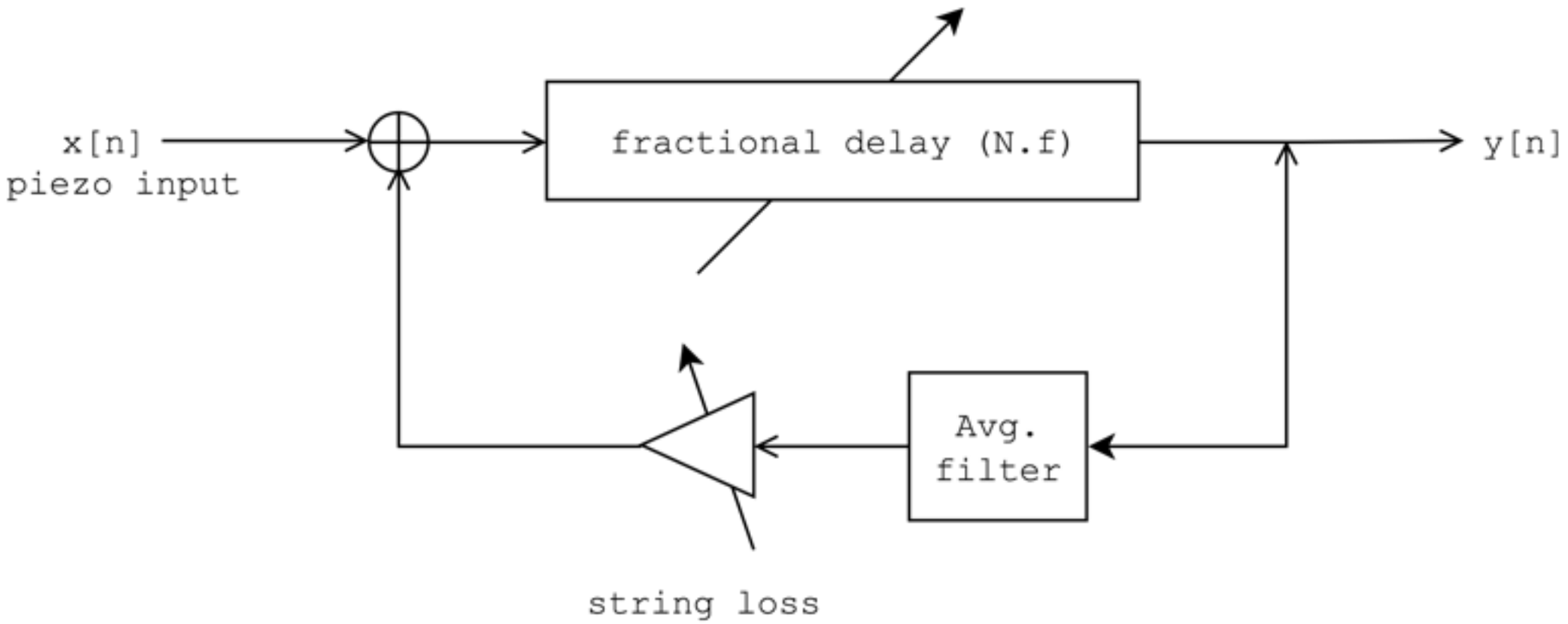
We will be using all sensors through the workshop

The Karplus-Strong algorithm



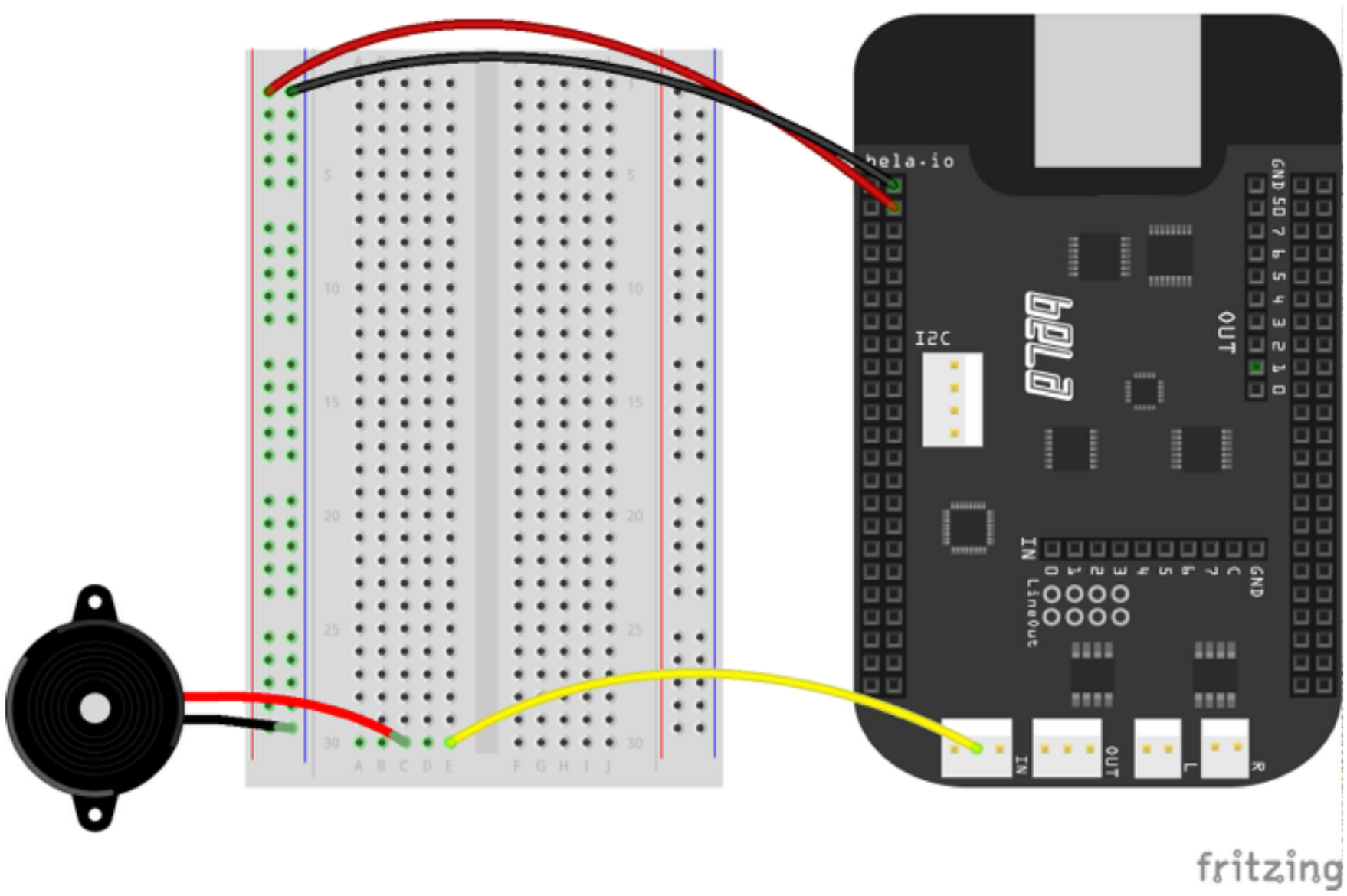
[1] Karplus, K. and Strong, A., 1983. **Digital synthesis of plucked-string and drum timbres**. *Computer Music Journal*, *7*(2), pp.43-55.
[2] Jaffe, D.A. and Smith, J.O., 1983. **Extensions of the Karplus-Strong plucked-string algorithm**. *Computer Music Journal*, *7*(2), pp.56-69.
[3] Karjalainen, M., Välimäki, V. and Tolonen, T., 1998. **Plucked-string models: From the Karplus-Strong algorithm to digital waveguides and beyond.** *Computer Music Journal*, *22*(3), pp.17-32.
[3] http://amid.fish/karplus-strong

Karplus-Strong(ish)



Karplus-Strong(ish)

Step 0



examples/00-Workshop/karplus-strong

Karplus-Strong(ish)

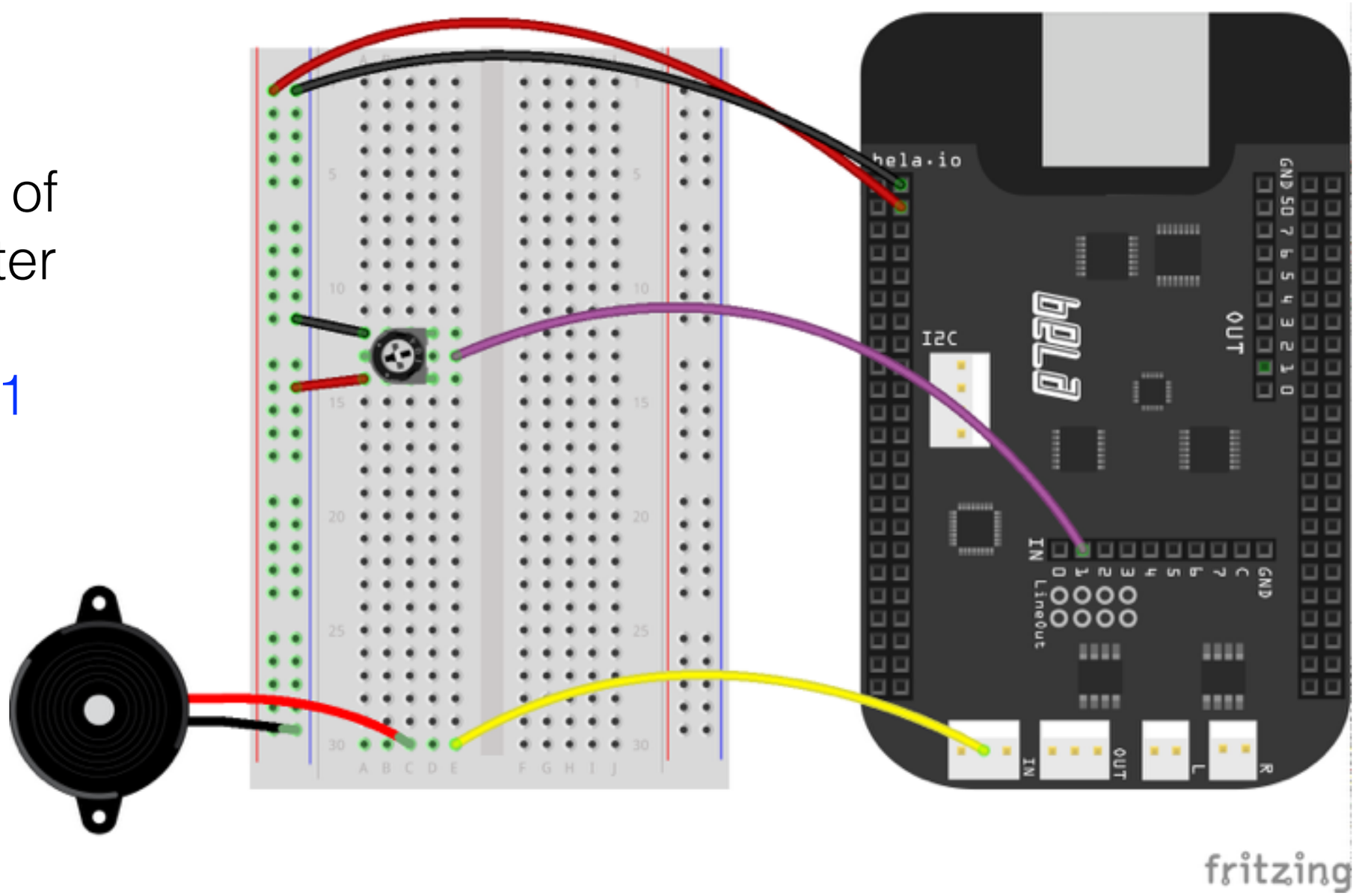
Step 0

- Piezo connected on audio channel 0
- Tap the piezo and listen to K-S being triggered
- Open scope and look at piezo input and K-S output
- Adjust input gain to adjust clipping

Karplus-Strong(ish)

Step 1

Middle wire of potentiometer goes to Analog In 1



Karplus-Strong(ish)

Step 1

- Potentiometer connected to analog input 1
- Add code to render function:

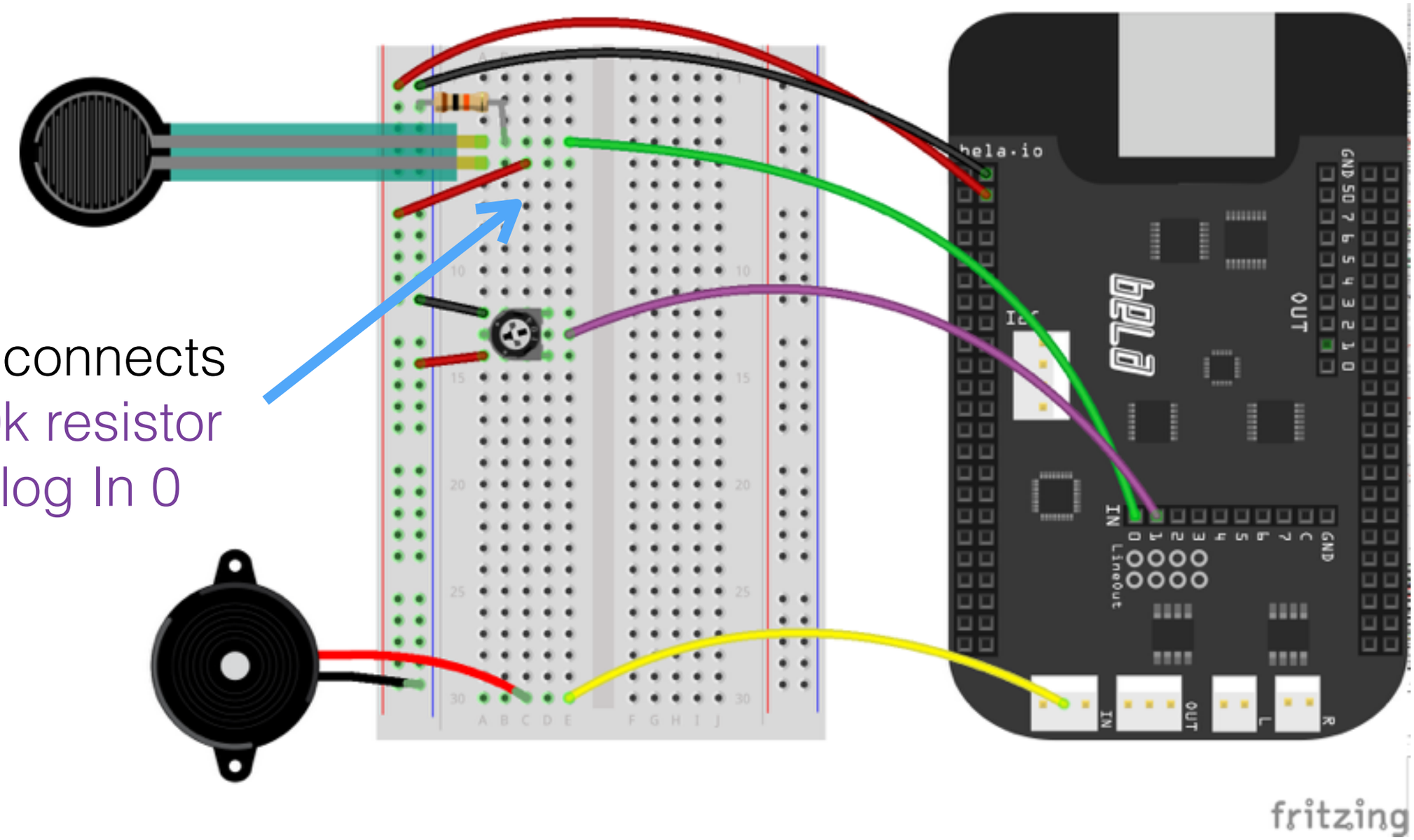
```
// read analog inputs (at audio rate) and update frequency
float potVal = analogRead(context, n, gPotChannel);

float frequency = map(potVal, 0, gAnalogInputRange, gFreqRange[0], gFreqRange[1]);

gPiezoString.setFrequency(frequency);
```

- Use pot to adjust frequency of K-S

Karplus-Strong(ish)

Step 2

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



Karplus-Strong(ish)

Step 2

- FSR connected to analog input 0
- Add code to render function:

```
// read analog inputs (at audio rate) and update loss factor

float fsrVal = analogRead(context, n, gFsrChannel);

float lossFactor = map(fsrVal, gAnalogInputRange, 0, gLossFactorRange[0],

  gLossFactorRange[1]);

gPiezoString.setLossFactor(lossFactor);
```

- Use FSR to adjust loss factor
- Is the FSR response suitable for this application?

Karplus-Strong(ish)

Step 2

- Log FSR to scope:
  - update number of channels in `setup()`
  - add `fsrVal` to `gScope.log()`
  - Let's clip the bottom range of the FSR, change/update code:

```
fsrVal = constrain(fsrVal, gFsrRange[0], gFsrRange[1]);

float lossFactor = map(fsrVal, gFsrRange[1], gFsrRange[0], gLossFactorRange[0],

    gLossFactorRange[1]);
```

  - Log `lossFactor` to scope

Karplus-Strong(ish)

Step 3

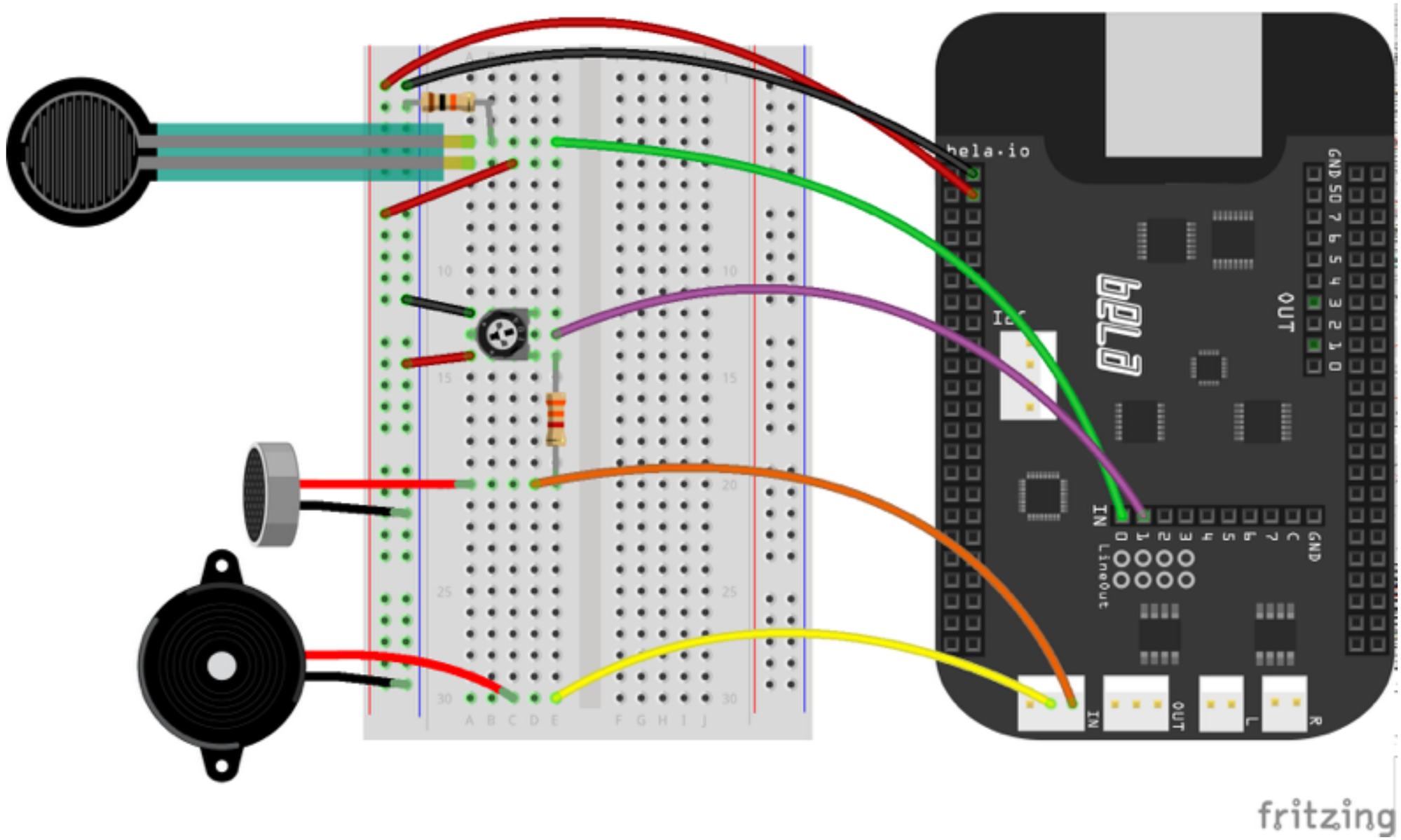
- Is this the response we want from the FSR?
- Can it be better?
- Try log mapping:

```
float fsrLog;

fsrLog = logMap(fsrVal, gFsrRange[1], gFsrRange[0], gLossFactorRange[0],

  gLossFactorRange[1]);

float lossFactor = fsrLog;
```

- **Log mapped FSR (`fsrLog`) to scope**

Karplus-Strong(ish)

Step 4



Karplus-Strong(ish)

Step 4

- Add/replace microphone input (audio channel 1)

```
...setup()...

        gMicString.setup(context->audioSampleRate, gFreqRange[0], 432.f *
gFreqRatio);

    ...render()...

    gMicString.setLossFactor(lossFactor);

    gMicString.setFrequency(frequency * gFreqRatio);

    ....

    float micInput = audioRead(context, n, gMicChannel);

    float micStringOut = gMicString.process(micInput);

    ....

    audioWrite(context, n, ch, gOutputGain * (piezoStringOut + micStringOut));

    ....
```

Karplus-Strong(ish)

Step 5

- Loss factor can be improved
- Add unable LP filter to feedback loop for more realistic damping
- In `KarplusStrong.h`, comment out :

```
#define KS_CONSTANT_LOWPASS
```

- In `render()`:

```
#ifdef KS_CONSTANT_LOWPASS

....

#else /* KS_CONSTANT_LOWPASS */

        fsrLog = logMap(fsrVal, gFsrRange[1], gFsrRange[0], gDampingRange[0],
gDampingRange[1]);

        float damping = fsrLog;

        gPiezoString.setDamping(damping);

        gMicString.setDamping(damping);

#endif /* KS_CONSTANT_LOWPASS */
```

Bela + Karplus-Strong Instruments

Kalichord Strum

Dan Moses

Strummi

Jacob Harrison, Robert H. Jack

Air Harp

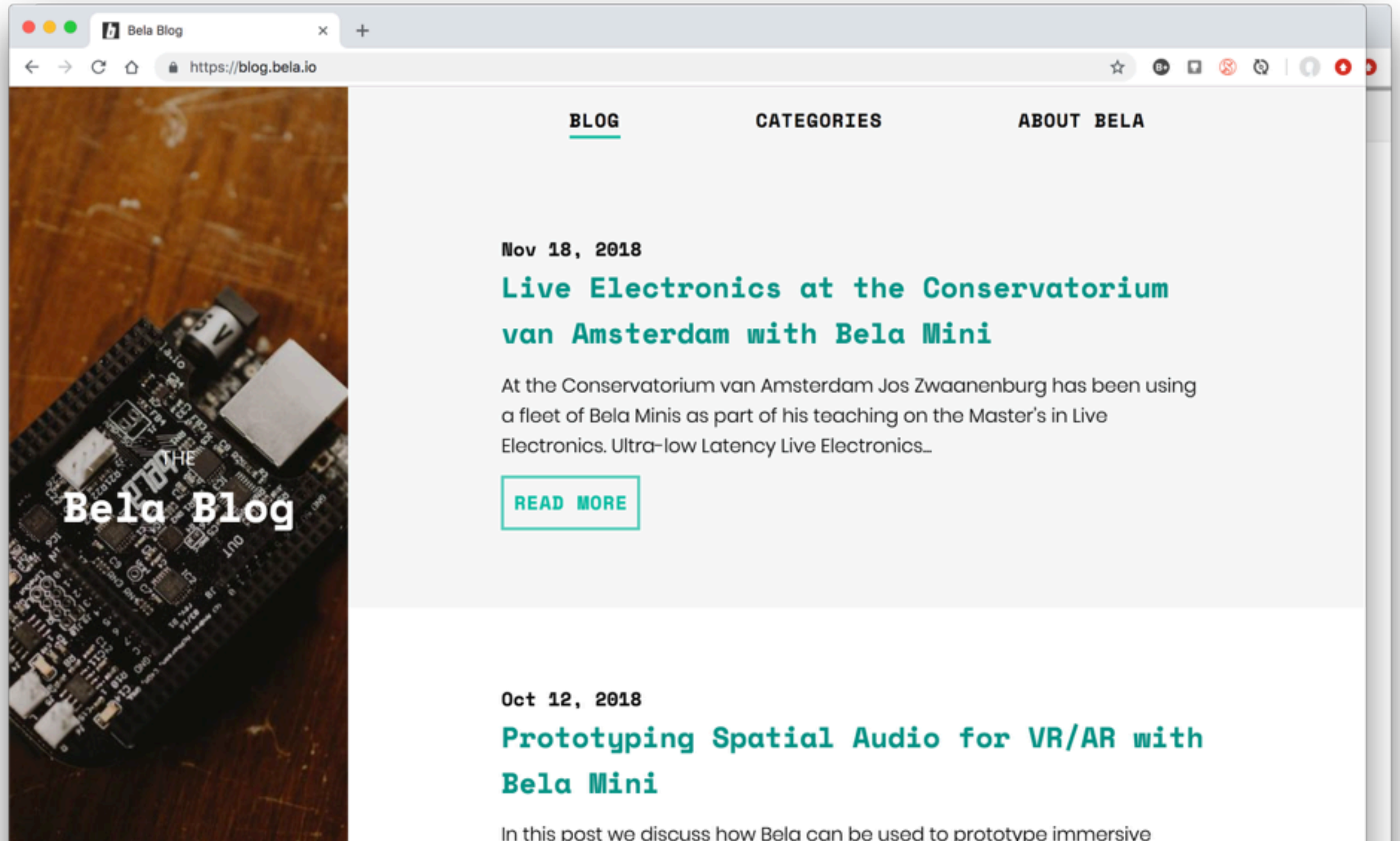
Chris Heinrichs

KSSynth

Rej Poirier

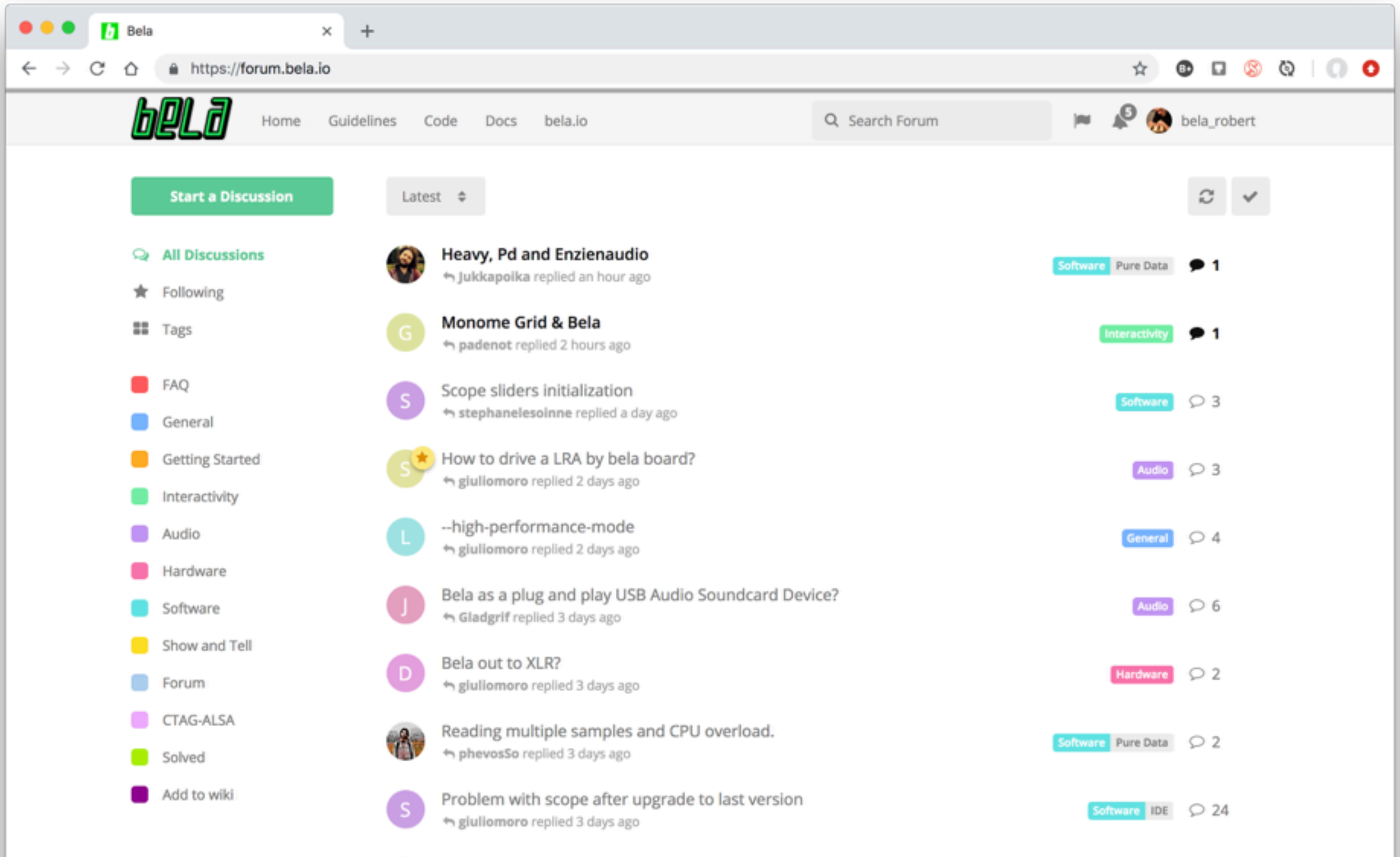
Finding out more

http://blog.bela.io



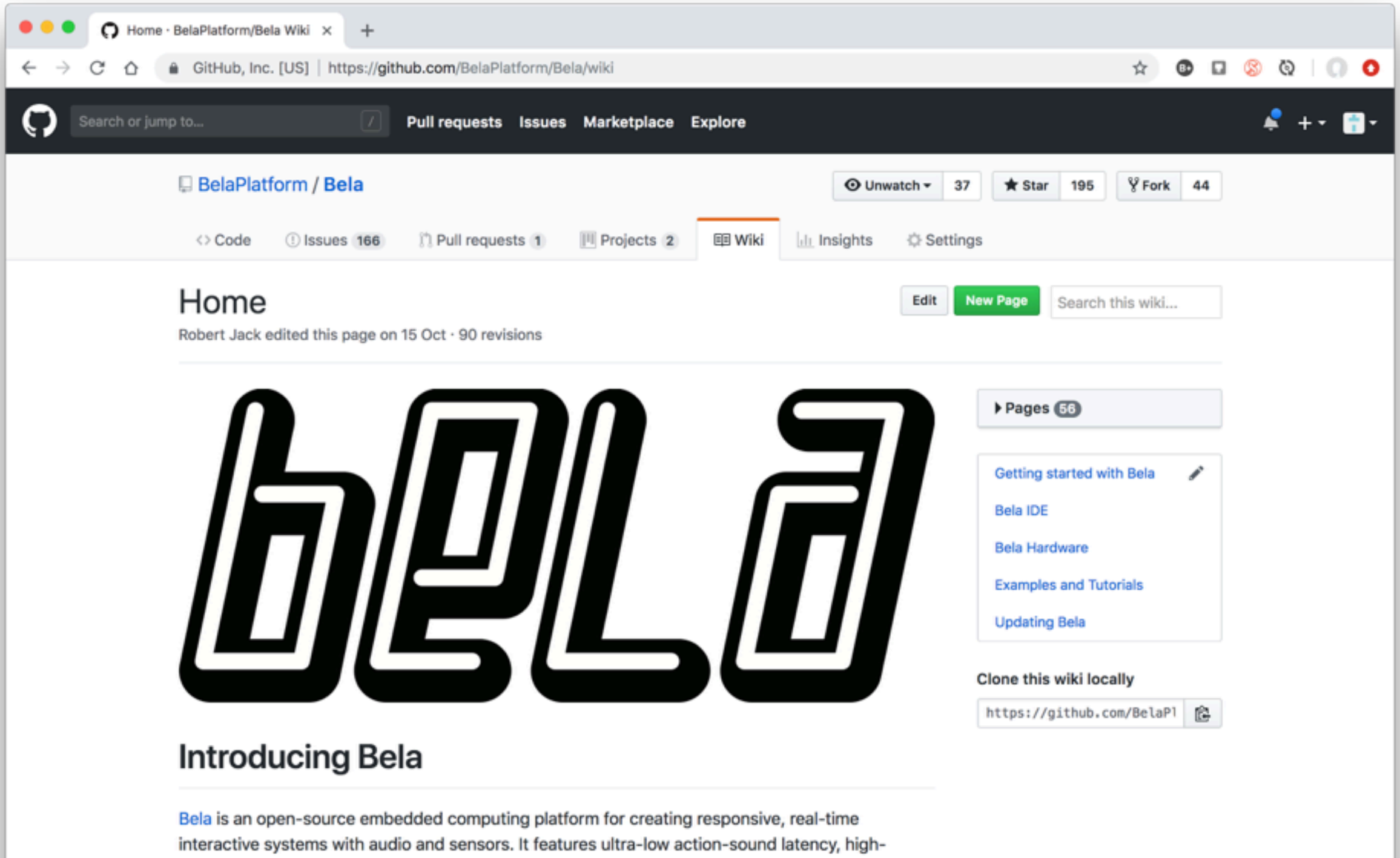
Finding out more

http://forum.bela.io



Finding out more

https://github.com/BelaPlatform/Bela/wiki



Where can I get one?

https://shop.bela.io/