C++ Real-Time Audio Programming with Bela

Dr Andrew McPherson

Centre for Digital Music
School of Electronic Engineering and Computer Science
Queen Mary University of London

Founder and Director, Bela

Course topics

Programming topics		Music/audio topics
Working in real time		Oscillators
Buffers and arrays		Samples
Parameter control		Wavetables
Classes and objects		Filters
Analog and digital I/O		Control voltages
Filtering	Today	Gates and triggers
Circular buffers		Delays and delay-based effects
Timing in real time		Metronomes and clocks
State machines		Envelopes
MIDI		ADSR
Block-based processing		MIDI
Threads		Additive synthesis
Fixed point arithmetic		Phase vocoders
ARM assembly language		Impulse reverb

Lecture 6: Analog I/O

What you'll learn today:

Analog inputs and outputs on Bela
Working with breadboards
Basics of control voltages

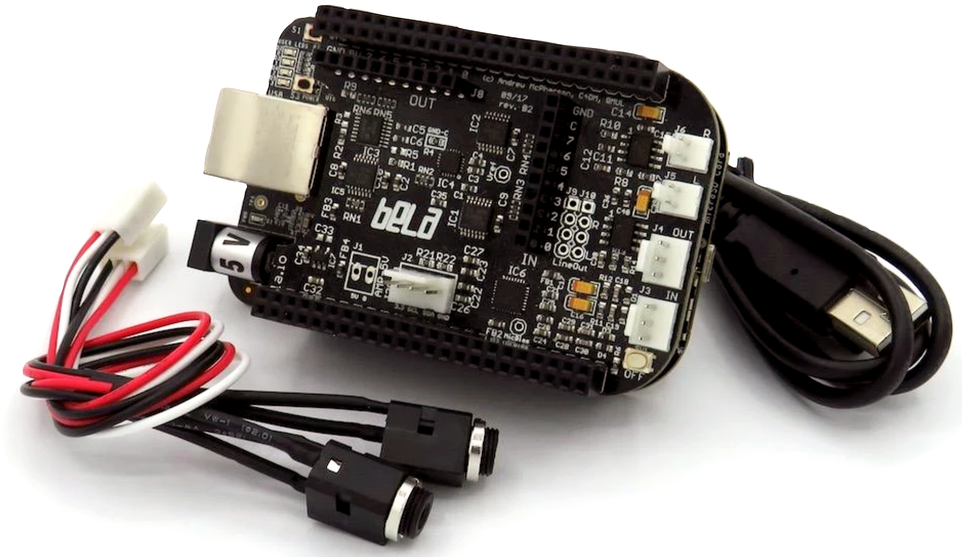
What you'll make today:

A VCO and a simple musical instrument

Companion materials:

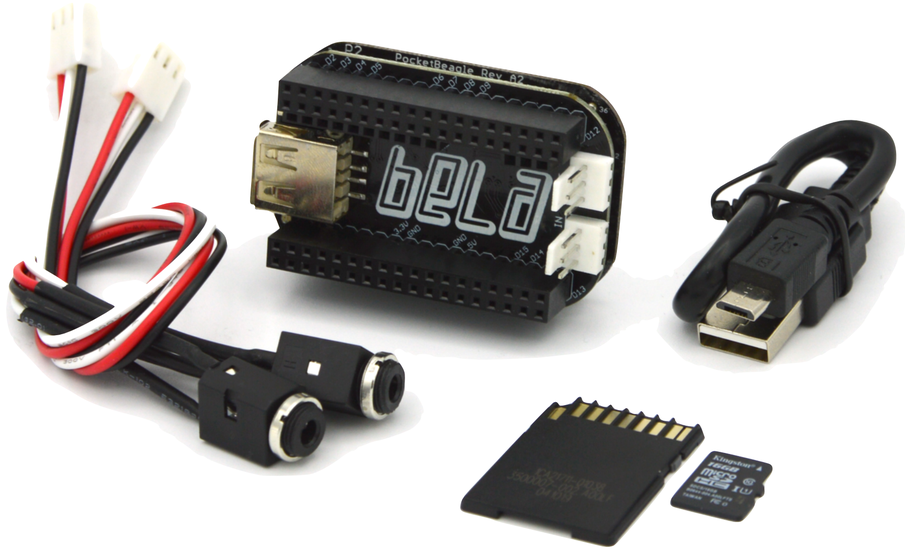
github.com/BelaPlatform/bela-online-course

What you'll need



Bela Starter Kit

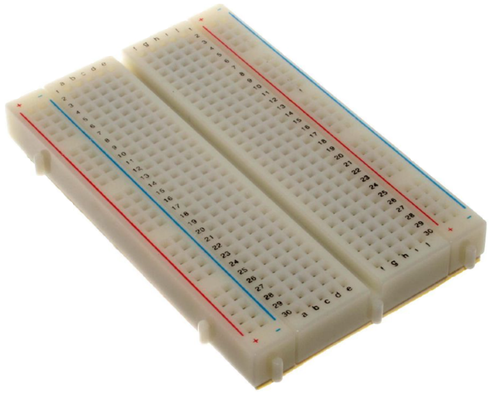
or

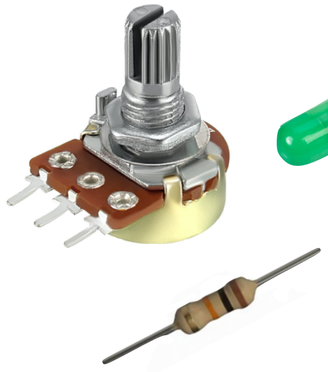


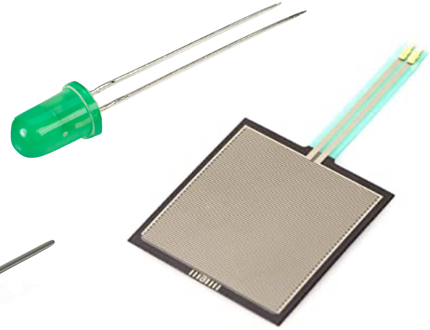
Bela Mini Starter Kit

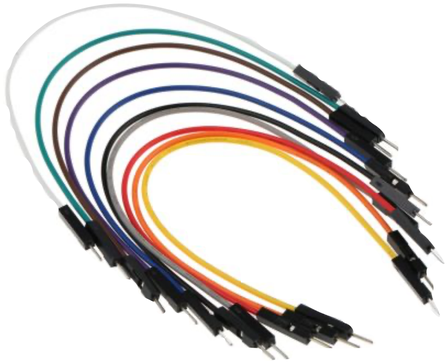
[shop.bela.io]

Also needed for this lecture:







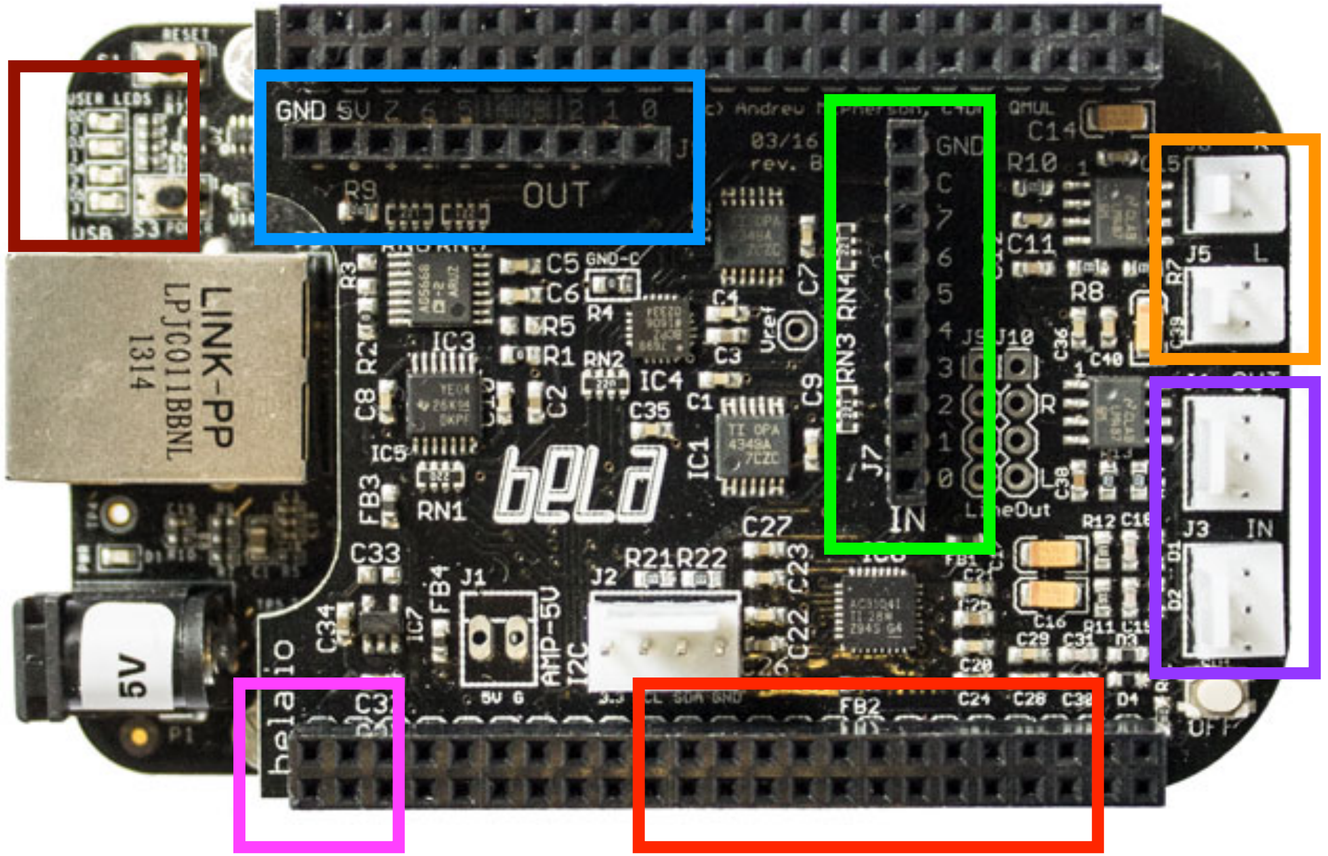


Bela hardware

Analog outputs

Analog inputs

USB to computer



R speaker

L speaker

Audio out (headphones)

Audio in

Power / Ground

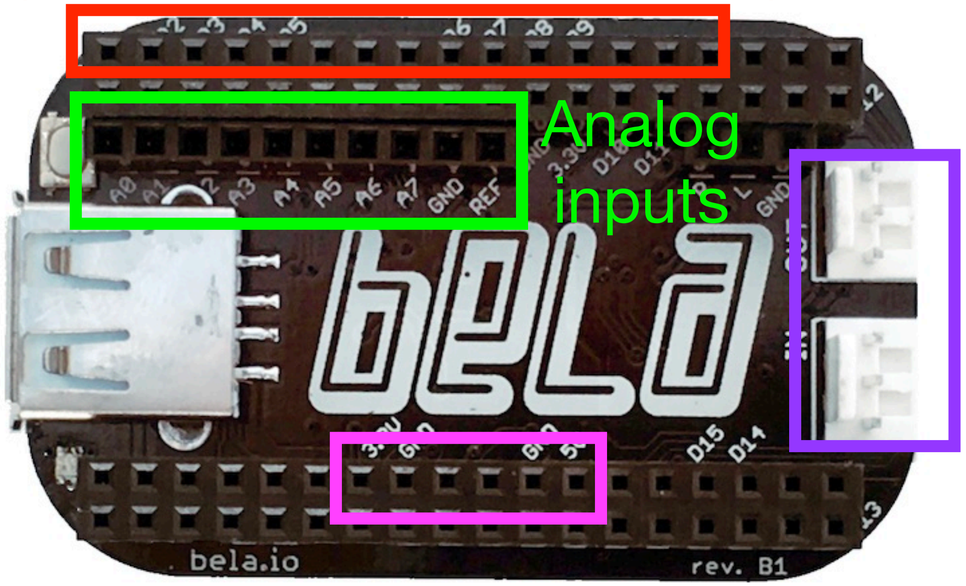
Digital inputs/outputs
(various locations around board)

Bela Mini hardware

Digital inputs/outputs
(various locations around board)

Analog inputs

USB to computer (below)



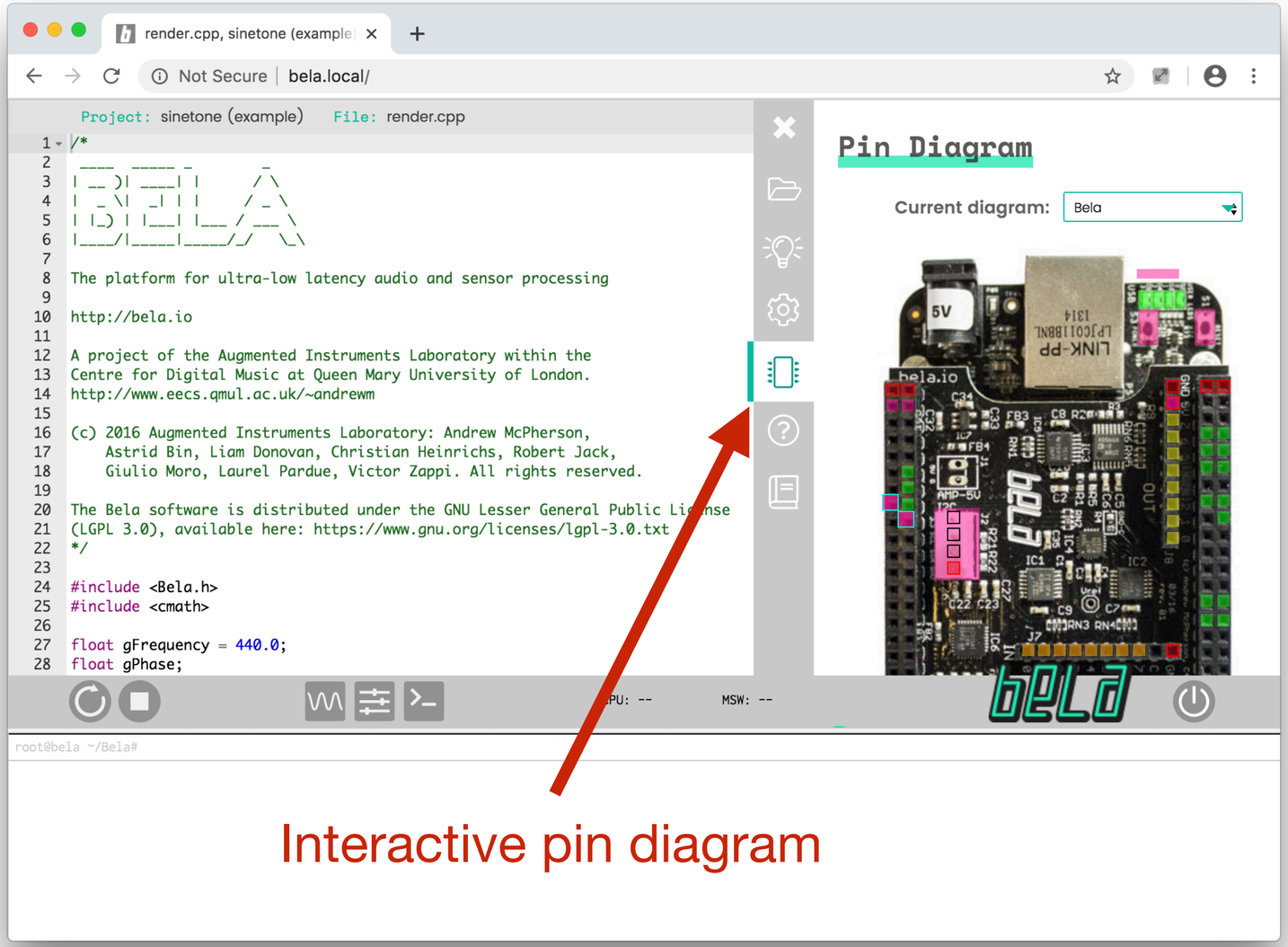
Audio out (headphones)

Audio in

Power / Ground

Analog input

- You can always find the pinout for the analog input and other connectors in the IDE's Interactive Pin Diagram

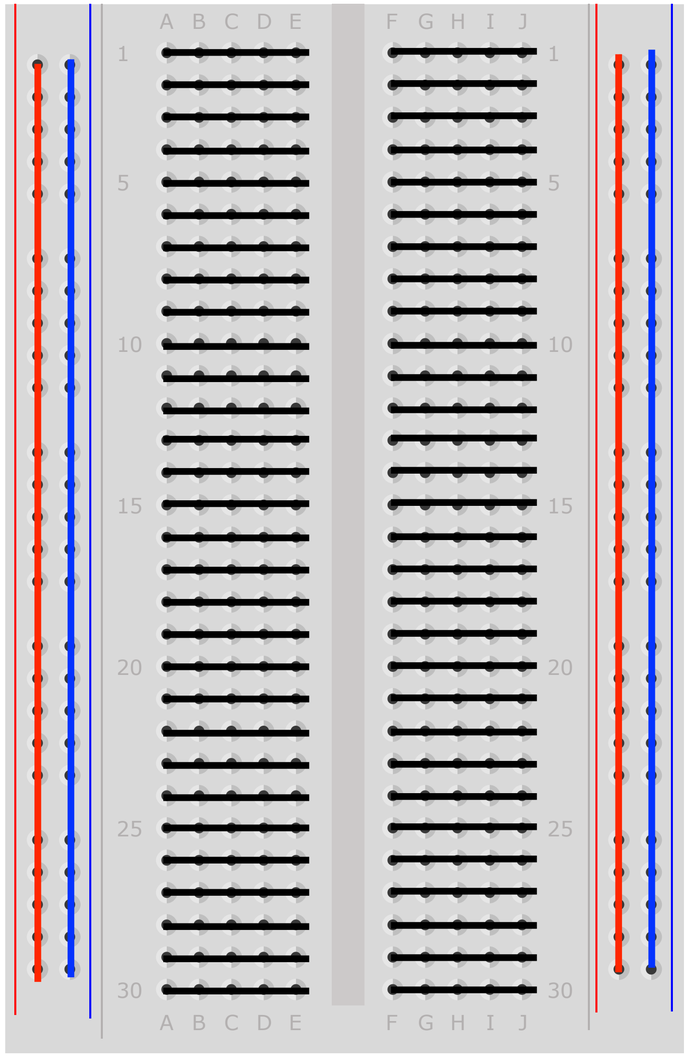


Using a breadboard

Each column of 5 holes connects together...

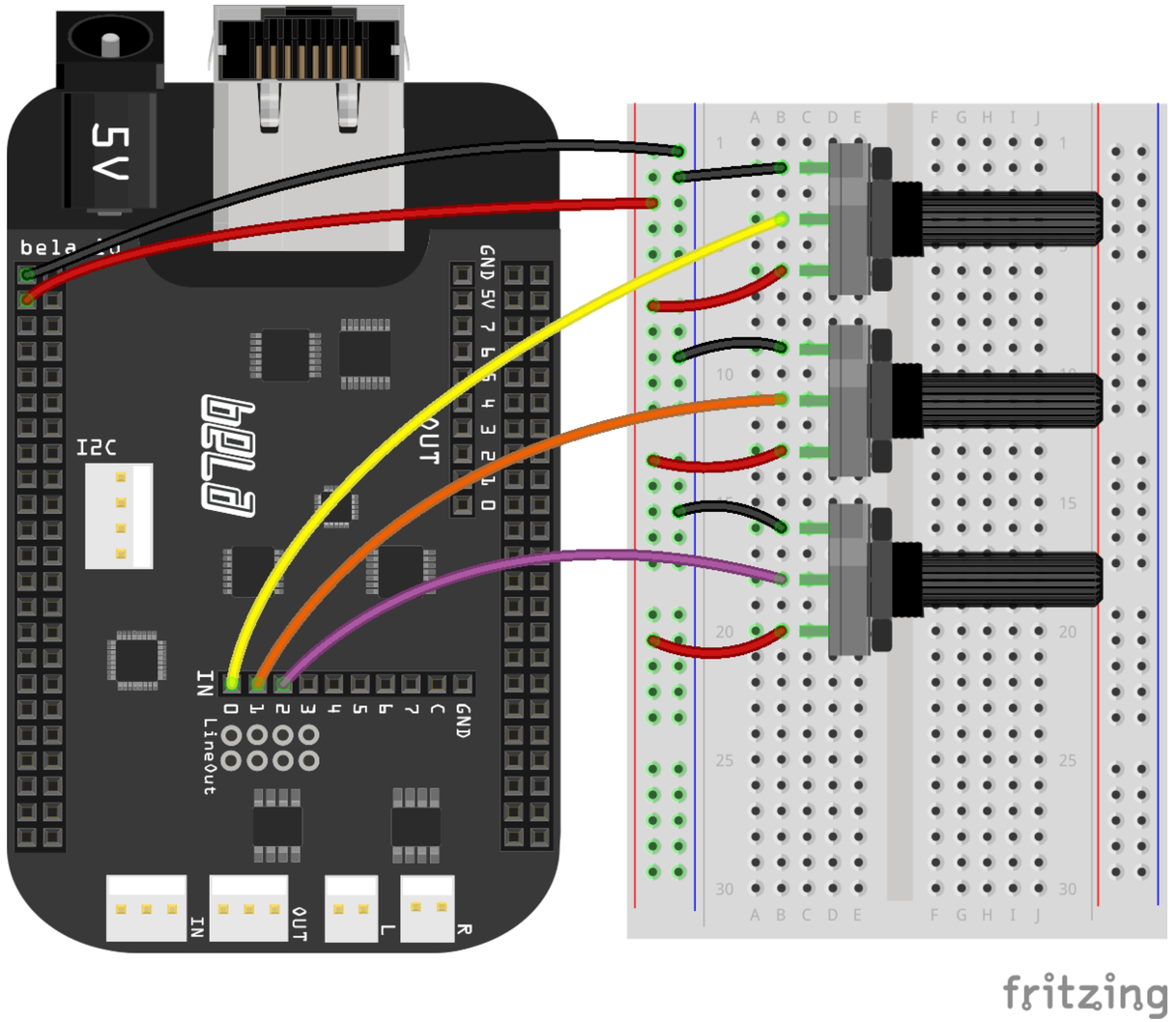
...but not across the break in the middle.

Long rows at the ends each connect together. Typically used for power and ground



Connect two potentiometers

- Bela has sources of 3.3V and 5V on board
- We will **always use 3.3V** in this course as 5V will damage the digital inputs!
- Wire up the circuit on the right and open the scope-analog example in the Examples tab of the Bela IDE

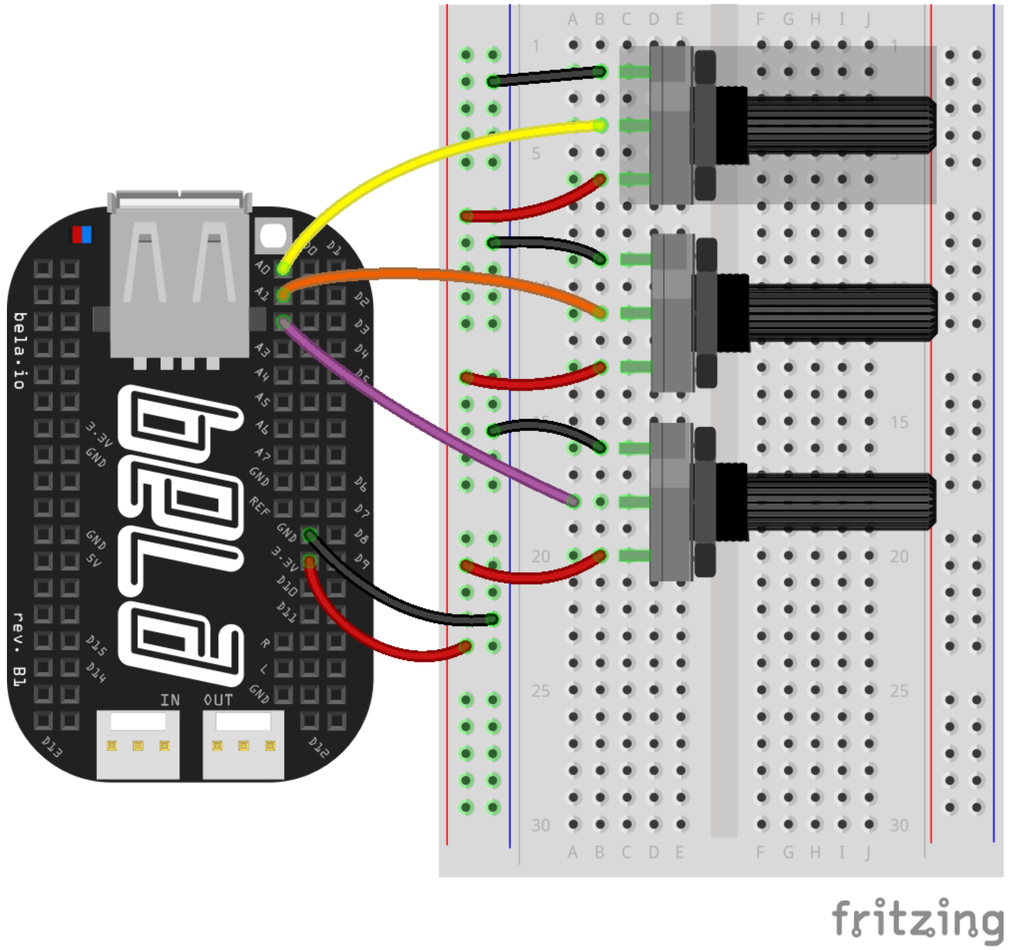


Ground (P9_1) and +3.3V (P9_3) to long rows

Middle wires of potentiometers go to Analog In 0 and Analog In 1

Connect two potentiometers

- Bela has sources of 3.3V and 5V on board
- We will **always use 3.3V** in this course as 5V will damage the digital inputs!
- Wire up the circuit on the right and open the scope-analog example in the Examples tab of the Bela IDE

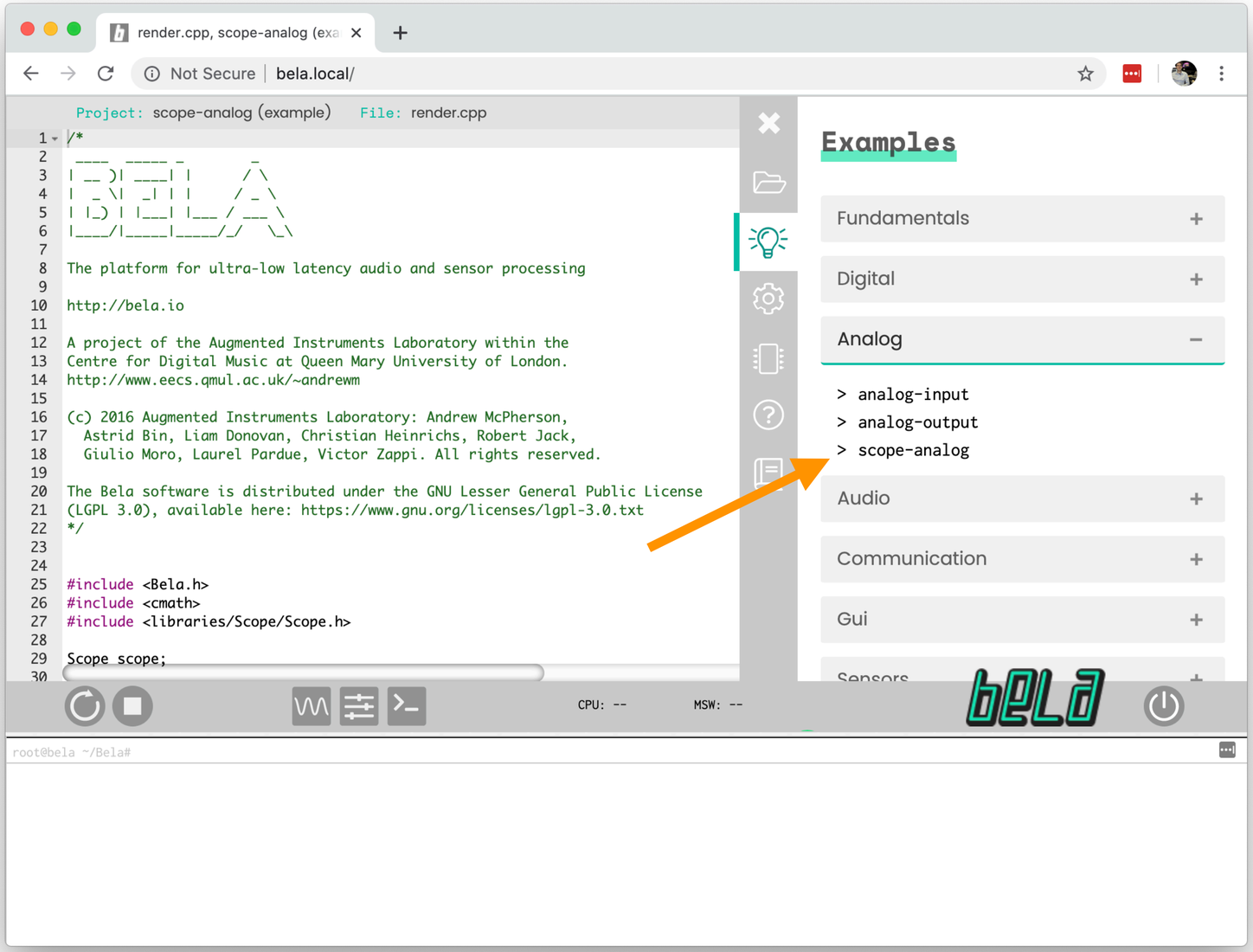


Ground (P9_1) and +3.3V (P9_3) to long rows

Middle wires of potentiometers go to Analog In 0 and Analog In 1

Connect two potentiometers

- Bela has sources of 3.3V and 5V on board
- We will **always use 3.3V** in this course as 5V will damage the digital inputs!
- Wire up the circuit on the right and open the scope-analog example in the Examples tab of the Bela IDE



Working with analog data

- Analog inputs on Bela are sampled signals, just like audio
  - You don't have to ask the Bela hardware to take the readings; they are captured automatically in a buffer at a constant sample rate
  - Thus, we automatically have access to signals representing voltage versus time

Voltage

Analog inputs are DC-coupled

16-bit resolution

Usable input range 0-4.096V

Software values 0-1 (as a `float`)

→

0V corresponds to a reading of 0.0;
4.096V corresponds to a reading of 1.0

- What does a reading of 0.5 correspond to in voltage?
- **2.048V**
- What is the expected reading for a 3.3V input?
- 3.3V / 4.096V ≈ **0.806**

Working with analog data

- Analog inputs on Bela are sampled signals, just like audio
  - You don't have to ask the Bela hardware to take the readings; they are captured automatically in a buffer at a constant sample rate
  - Thus, we automatically have access to signals representing voltage versus time

Voltage

Analog inputs are DC-coupled

16-bit resolution

Usable input range 0-4.096V

Software values 0-1 (as a `float`)

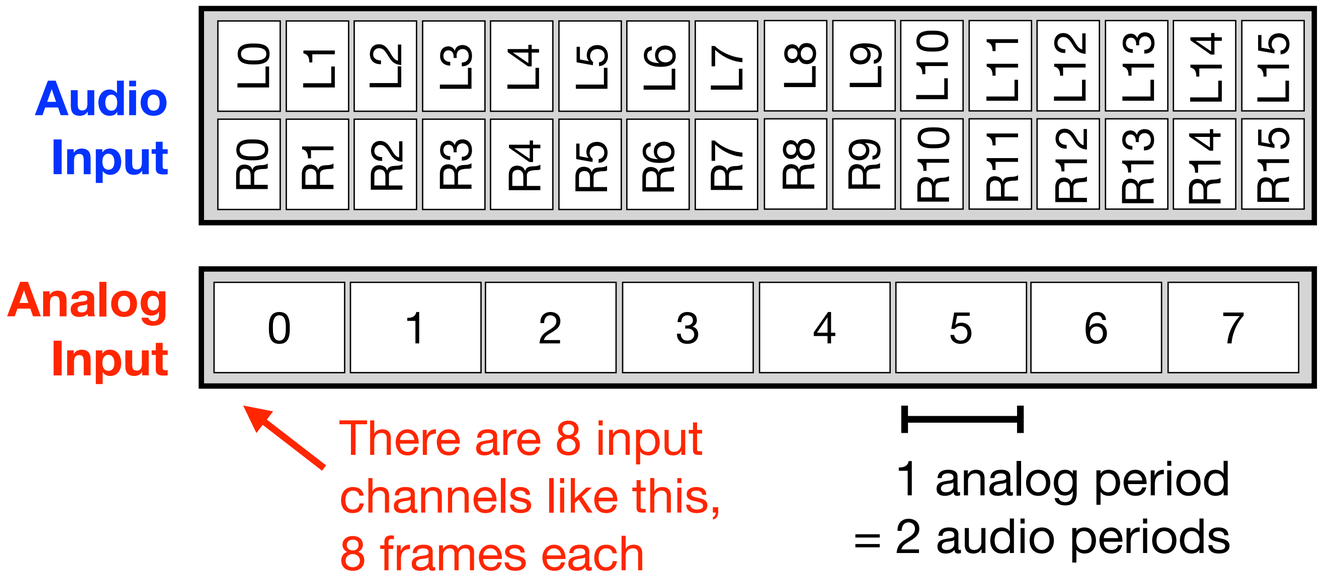
Time

Sample rate depends on the number of input channels:

8 channels (default)	4 channels	2 channels
↓	↓	↓
22.05kHz	44.1kHz	88.2kHz
(half audio rate)	*(audio rate)*	*(double audio rate)*

Analog and audio frames

- The default analog input sample rate is 22.05kHz
  - Audio is sampled at 44.1kHz. That's a period of 22.7µs.
  - What is the sampling period of the analog input? 45.4µs
  - At this sample rate, how many audio frames elapse for each analog frame? 2
- Each real-time block covers a particular slice of time
  - If there were 16 audio frames in the block, how many analog frames?



Given a particular audio frame number, how do we find the corresponding analog frame?

divide by 2

Why doesn't this result in a fractional index?

we use `int` for indexing

Analog API

- Bela analog buffers are interleaved, holding data that is already sampled
- `analogRead()` is a convenience function that accesses the analog buffer
  - Pulls out the particular frame and channel we want

```
float analogRead(BelaContext *context, int frame, int channel) {
    return context->analogIn[frame * context->analogInChannels + channel];
}
```

Reading is in the return value

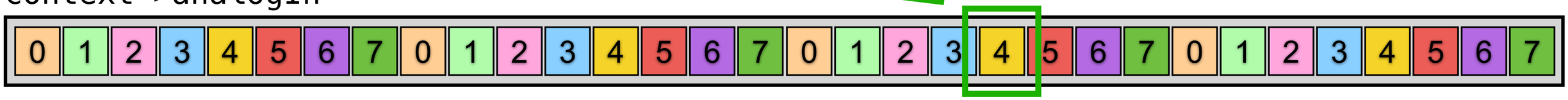
Reference to the `BelaContext` structure, which holds the actual data we want

Which frame (i.e. sample) within the buffer to read

Which channel (e.g. 0, 1, ..., 7) to read

For example: `float reading = analogRead(context, 2, 4);`

`context->analogIn`



Audio frames and analog frames

- Here is the scope-analog code for reading the analog inputs in `render()`:

```
for(unsigned int n = 0; n < context->audioFrames; n++) {

    if(gAudioFramesPerAnalogFrame && !(n % gAudioFramesPerAnalogFrame)) {
        // Read analog inputs and update frequency and amplitude
        gIn1 = analogRead(context, n/gAudioFramesPerAnalogFrame, 0);
        gIn2 = analogRead(context, n/gAudioFramesPerAnalogFrame, 1);
        gAmplitude = gIn1 * 0.8f;
        gFrequency = map(gIn2, 0, 1, 100, 1000);
    }

    float out = gAmplitude * sinf(gPhase);
    // ...
}
```

???

`analogRead()` takes the frame and channel and gets a value from the `BelaContext`

`map()` rescales the range from 0-1 to 100-1000 (Hz)

- The variable `gAudioFramesPerAnalogFrame` is calculated in `setup()`:

```
if(context->analogFrames)
    gAudioFramesPerAnalogFrame = context->audioFrames / context->analogFrames;
```

Audio frames and analog frames

- Suppose we knew the analog sample rate was half the audio rate
  - We could write:

```
for(unsigned int n = 0; n < context->audioFrames; n++) {

    if((n % 2) == 0) {  // Read analog on the even audio samples
        // Read analog inputs and update frequency and amplitude
        gIn1 = analogRead(context, n/2, 0);
        gIn2 = analogRead(context, n/2, 1);
        gAmplitude = gIn1 * 0.8f;
        gFrequency = map(gIn2, 0, 1, 100, 1000);
    }

    float out = gAmplitude * sinf(gPhase);
    // ...
}
```

Modulo (`%`) means remainder after division. Is `n` even?

Convert audio frame number (`n`) to analog frame number (`n/2`)

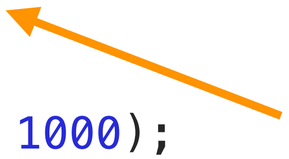
Audio frames and analog frames

- Suppose we knew the analog sample rate was equal to the audio rate
  - We could write:

```
for(unsigned int n = 0; n < context->audioFrames; n++) {

    // Read analog inputs and update frequency and amplitude
    gIn1 = analogRead(context, n, 0);
    gIn2 = analogRead(context, n, 1);
    gAmplitude = gIn1 * 0.8f;
    gFrequency = map(gIn2, 0, 1, 100, 1000);

    float out = gAmplitude * sinf(gPhase);
    // ...
}
```



- We use `gAudioFramesPerAnalogFrame` because we don't know the rate

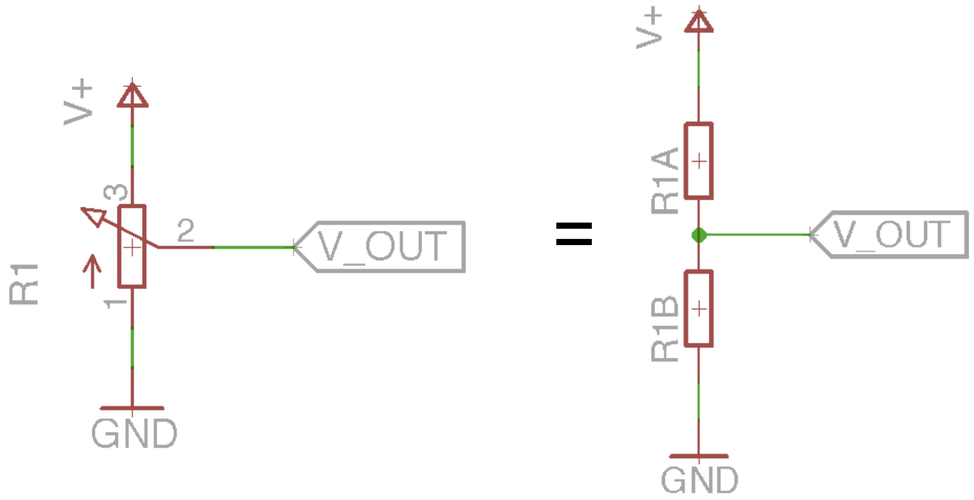
```
if(gAudioFramesPerAnalogFrame && !(n % gAudioFramesPerAnalogFrame)) { }
```

Voltage-controlled oscillator

- We have been making a VCO (voltage-controlled oscillator)
  - This is a standard component of analog synths
  - The control voltage (CV) input changes the frequency of the oscillator
    - The analog input is what reads the control voltage
- A common standard in the synth industry is 1V/octave
  - Every additional 1V on the CV produces a doubling of the frequency
  - On Bela, analog input takes a range of 0 to 4.096V, so what is 1V? 1 / 4.096 = .2441
- Task: using the vco example,
  - Change the frequency, amplitude and detune controls from GUI sliders to analog inputs
  - Then, make the control a 1V/octave range, with a minimum frequency of 55Hz
  - First rescale the analog input to volts, then implement the formula: $f_{out} = f_{ref} 2^V$

Understanding potentiometers

- How does the potentiometer work?
  - Outer terminals form a resistor (10kΩ in our case)
  - Connect the outer terminals to +3.3V and ground
- Inner terminal is the wiper which contacts an intermediate part of the resistor
  - i.e. there is one resistance from +3.3V to wiper; another from wiper to ground
- Two resistors form a voltage divider
  - Ratio of top and bottom resistors is what matters



=

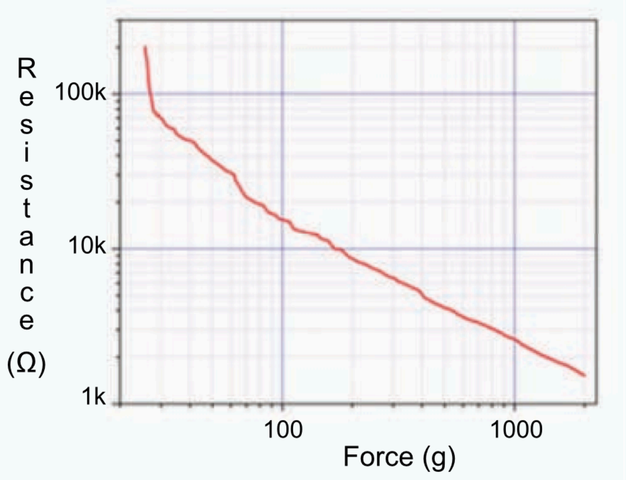
$$V_{out} = 3.3\text{V}\,\frac{\text{R1B}}{\text{R1A} + \text{R1B}}$$

$$\text{R1A} + \text{R1B} = 10\text{k}\Omega$$

Force-sensing resistors

- A force-sensing resistor (FSR) measures pressure on its surface
  - They come in various shapes and sizes, but all work similarly
  - Increased pressure ⟶ decreased electrical resistance
    - Where would we find more details?
    - Read the datasheet!

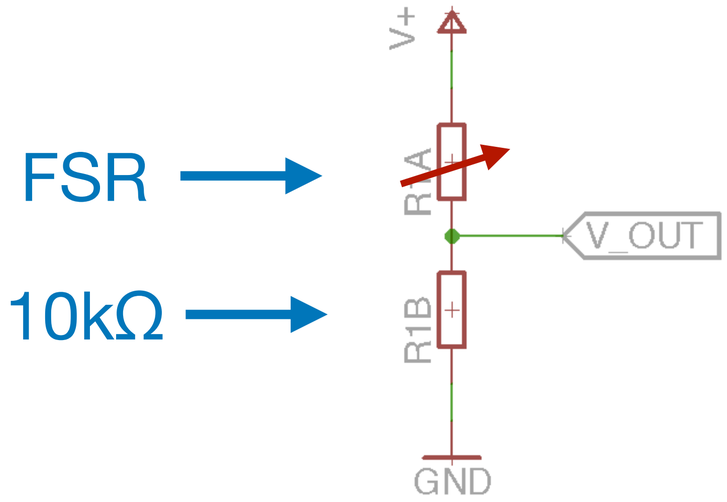
Interlink FSR400 series:



Device Characteristics	
Actuation Force*	~0.2N min
Force Sensitivity Range*	~0.2N – 20N
Force Resolution	Continuous (analog)
Force Repeatability Single Part	+/- 2%
Force Repeatability Part to Part	+/- 6% (Single Batch)
Non-Actuated Resistance	>10 Mohms
Hysteresis	+10% Average (RF+ - RF-)/RF+

Force-sensing resistors

- How do we convert resistance into a voltage we can read?
  - Use a voltage divider circuit
  - One side is a fixed resistor (say 10k), the other side is the FSR

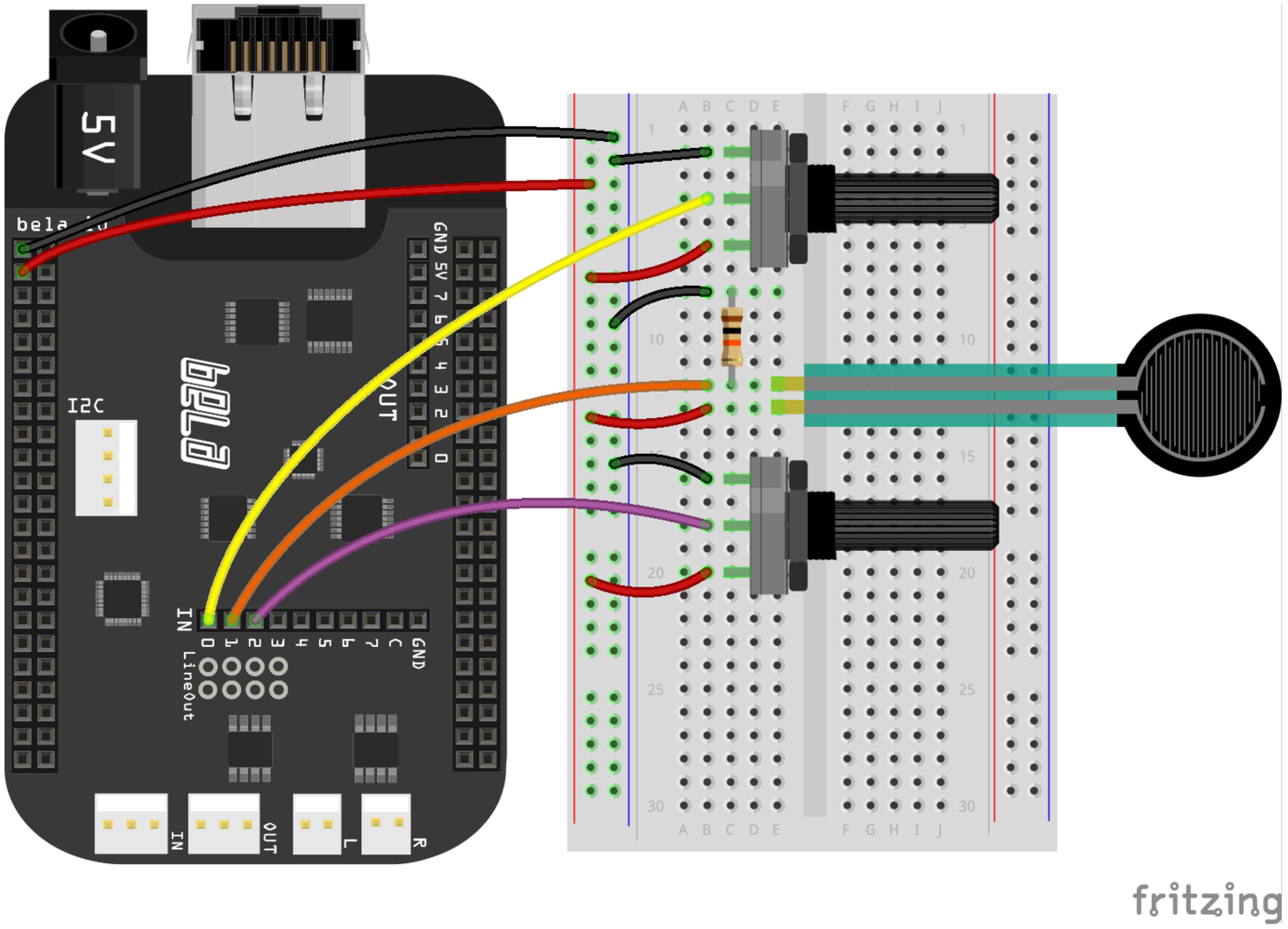


$$V_{out} = 3.3\text{V}\frac{\text{R1B}}{\text{R1A} + \text{R1B}}$$

- The voltage at the midpoint moves toward the side with lower resistance
  - So in this case, the harder we press, the higher the voltage

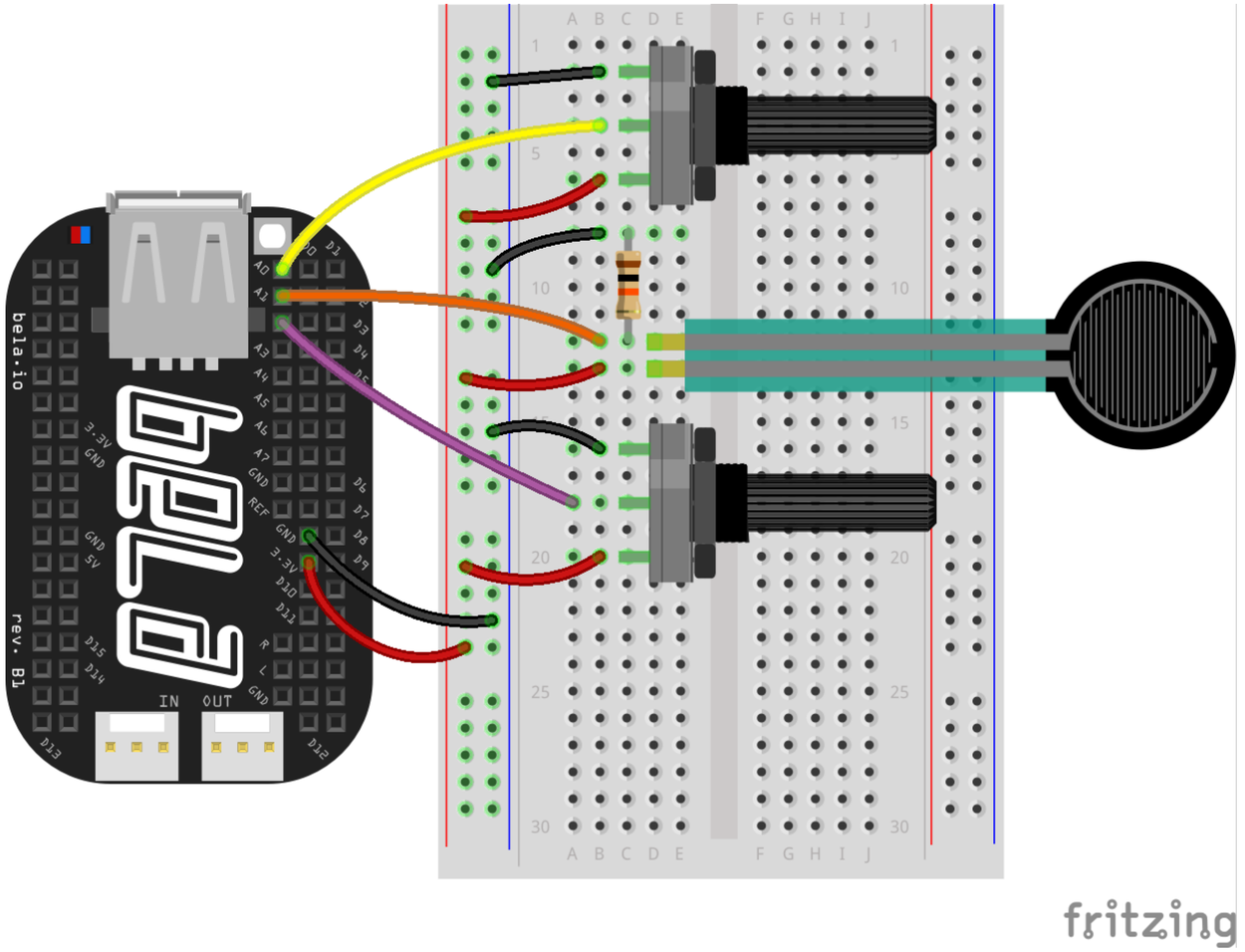
A simple instrument

- Task: change your circuit so that the amplitude control uses an FSR rather than a potentiometer
  - Don’t forget the 10k resistor!
  - Keep the code the same as before
  - How does the experience of playing it differ from before?
- Next, adjust the range for the FSR input to make a better playing experience



A simple instrument

- Task: change your circuit so that the amplitude control uses an FSR rather than a potentiometer
  - Don’t forget the 10k resistor!
  - Keep the code the same as before
  - How does the experience of playing it differ from before?
- Next, adjust the range for the FSR input to make a better playing experience



Analog output

- Analog output works equivalently to analog input
  - Samples are generated at a constant sample rate (default 22.05kHz for 8 channels)
  - DC coupled (like analog input): can output constant voltages
  - Signal range of 0 to 1 corresponds to 0V to 5V
    - Notice: this is different than analog input
- Main API:

```
void analogWrite(BelaContext *context, int frame, int channel, float value);
```

Reference to the `BelaContext` structure, which holds the actual data we want

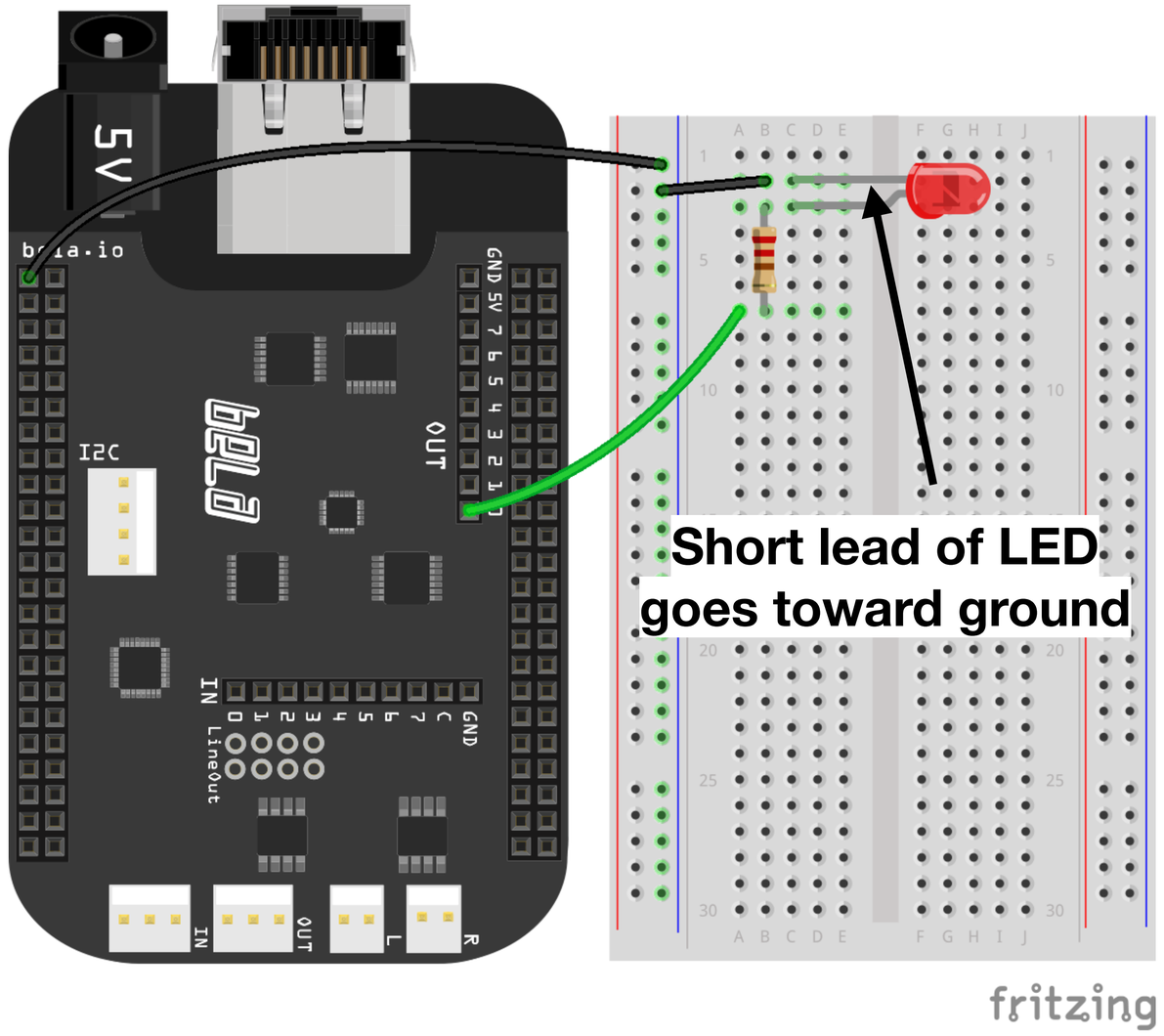
Which frame (i.e. sample) within the buffer to write

Which channel (e.g. 0, 1, ..., 7) to write

What value to write

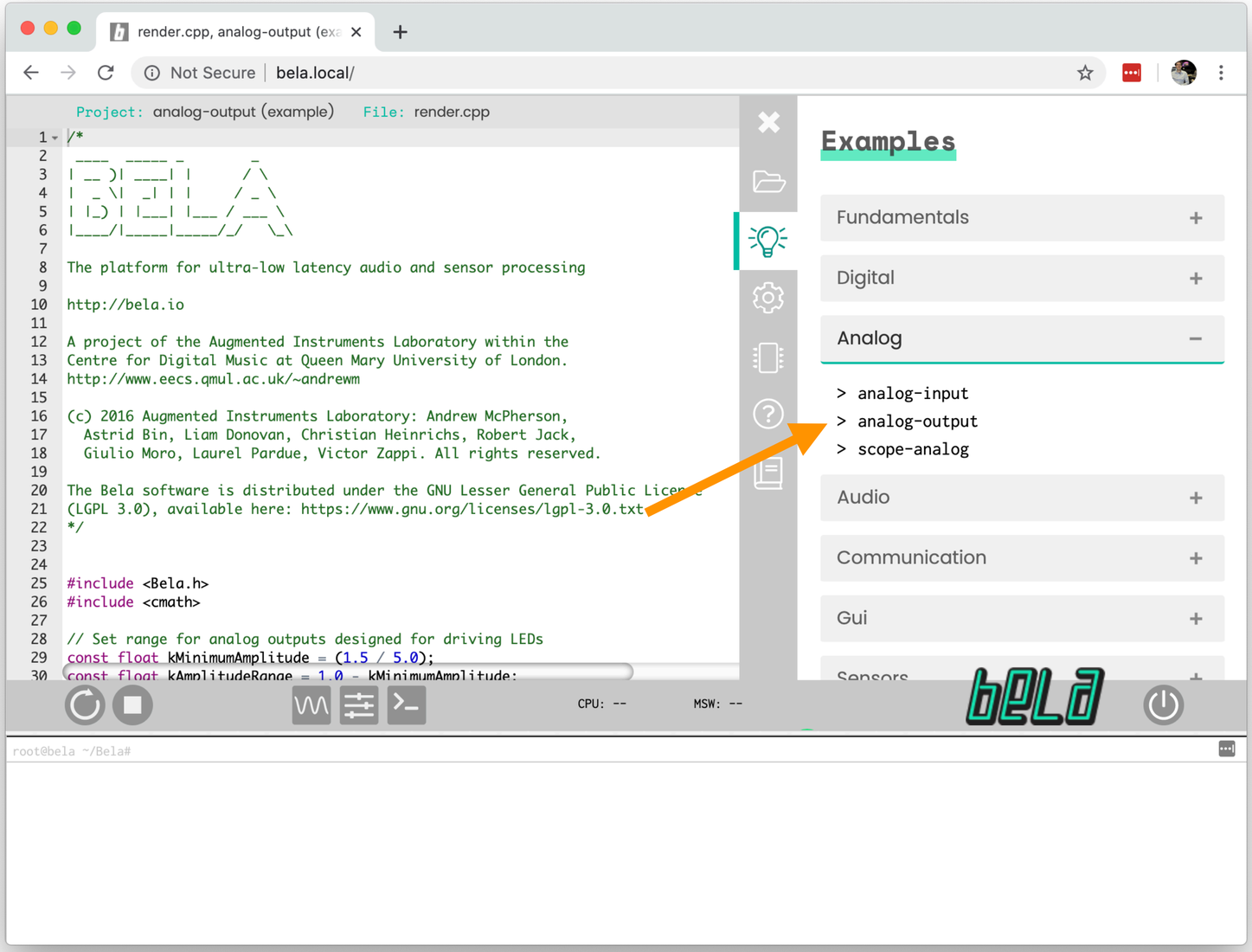
Connect an LED (Bela only)

- Add this circuit to the breadboard (don't remove the potentiometers or FSR)
- It's important to use a resistor in series with your LED to limit the current that flows through it
- Open the analog-output example in the Bela IDE
- In this example, if you use multiple LEDs they will fade on and off with different phase



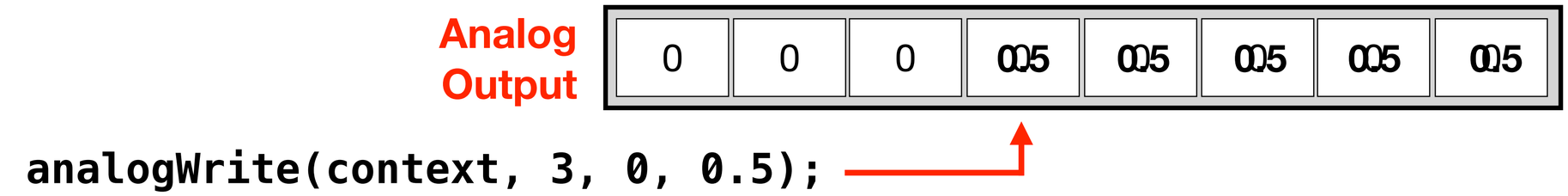
Connect an LED (Bela only)

- Add this circuit to the breadboard (don't remove the potentiometers or FSR)
- It's important to use a resistor in series with your LED to limit the current that flows through it
- Open the analog-output example in the Bela IDE
- In this example, if you use multiple LEDs they will fade on and off with different phase



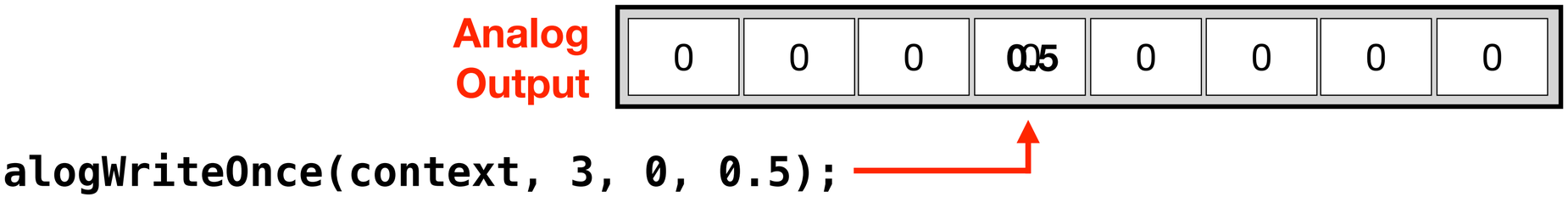
analogWrite() and analogWriteOnce()

- When writing an analog output, we might want the value to persist
  - Calling `analogWrite()` for a particular frame will also set the value for all future frames, until we call `analogWrite()` again
  - This is how writing a digital or analog pin on typical microcontrollers would work




```
analogWrite(context, 3, 0, 0.5);
```

- There is also a more efficient function that sets only the current frame
  - Use `analogWriteOnce()` if you will be writing a new value every frame anyway




```
analogWriteOnce(context, 3, 0, 0.5);
```

VCO with LED

- The two detuned oscillators produce beat frequencies:

$$\sin(x) + \sin(y) = 2\sin(\frac{x+y}{2})\cos(\frac{x-y}{2})$$

  - Given that we have: $f_1 = (1+r)f \qquad f_2 = (1-r)f$
  - What is the beat frequency in terms of *f* and *r*?
- Task: using the vco-analog-out example,
  - Add an analog output for the LED that pulses on and off at the beat frequency
  - Use `gLFO` to calculate the LED value, but at a lower frequency than the audio
  - Remember that the analog output has half the number of audio frames

Keep in touch!

Social media:

@BelaPlatform

forum.bela.io

blog.bela.io

More resources and contact info at:

learn.bela.io/resources