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

Working in real time
Buffers and arrays
Parameter control
Classes and objects
Analog and digital I/O
Filtering
Circular buffers
Timing in real time
State machines
MIDI
Block-based processing
Threads
Fixed point arithmetic
ARM assembly language

Today

Music/audio topics

Oscillators
Samples
Wavetables
Filters
Control voltages
Gates and triggers
Delays and delay-based effects
Metronomes and clocks
Envelopes
ADSR
MIDI
Additive synthesis
Phase vocoders
Impulse reverb

Lecture 7: Digital I/O

What you'll learn today:

Digital inputs and outputs on Bela
Working with push buttons
Gates and triggers

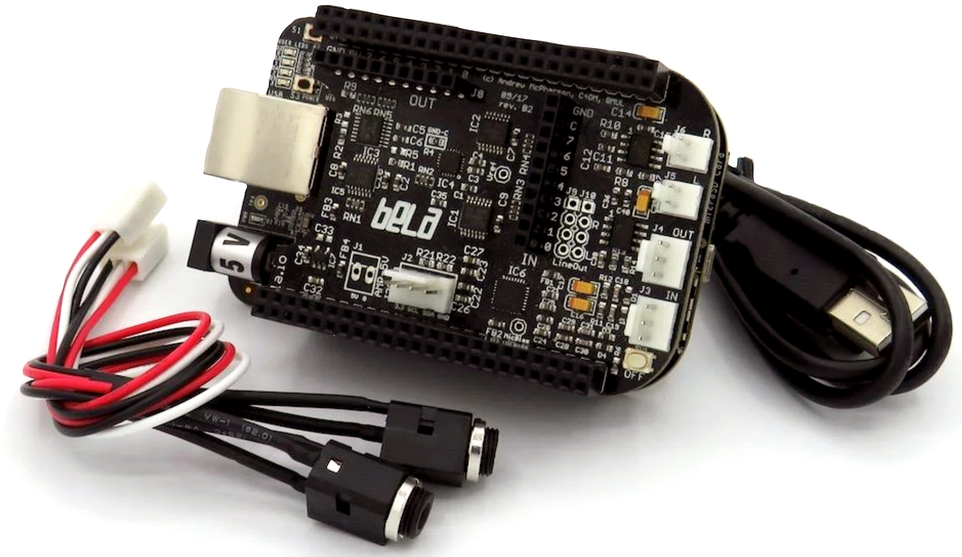
What you'll make today:

A noise generator and a step sequencer

Companion materials:

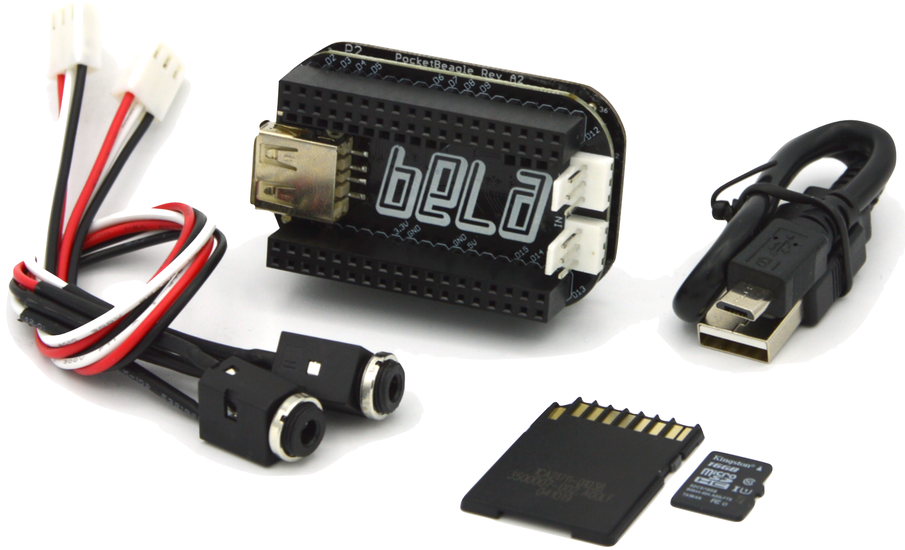
github.com/BelaPlatform/bela-online-course

What you'll need



Bela Starter Kit

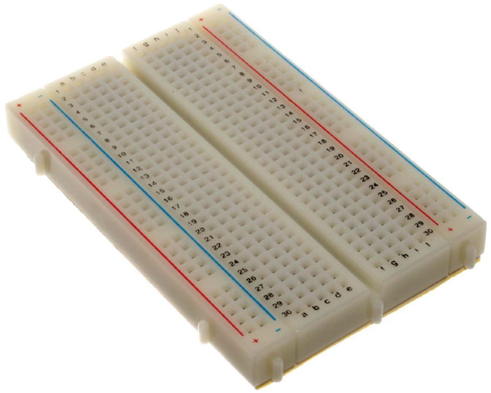
or

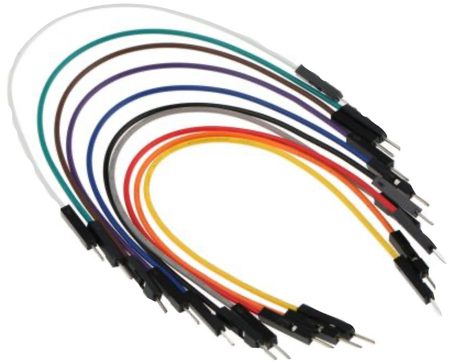


Bela Mini Starter Kit

[shop.bela.io]

Also needed for this lecture:



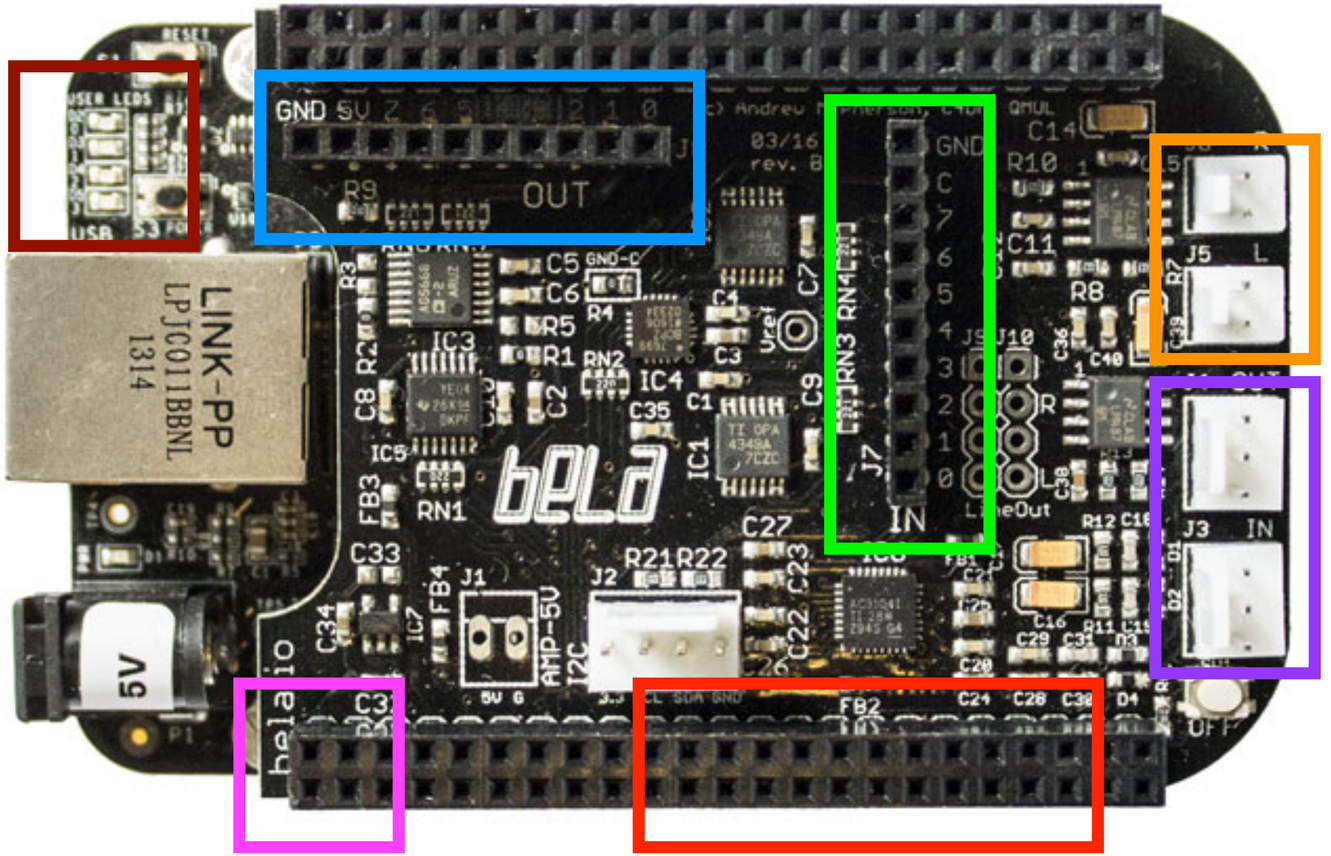


Bela hardware

Analog outputs

Analog inputs

USB to computer



R speaker

L speaker

Audio out (headphones)

Audio in

Power / Ground

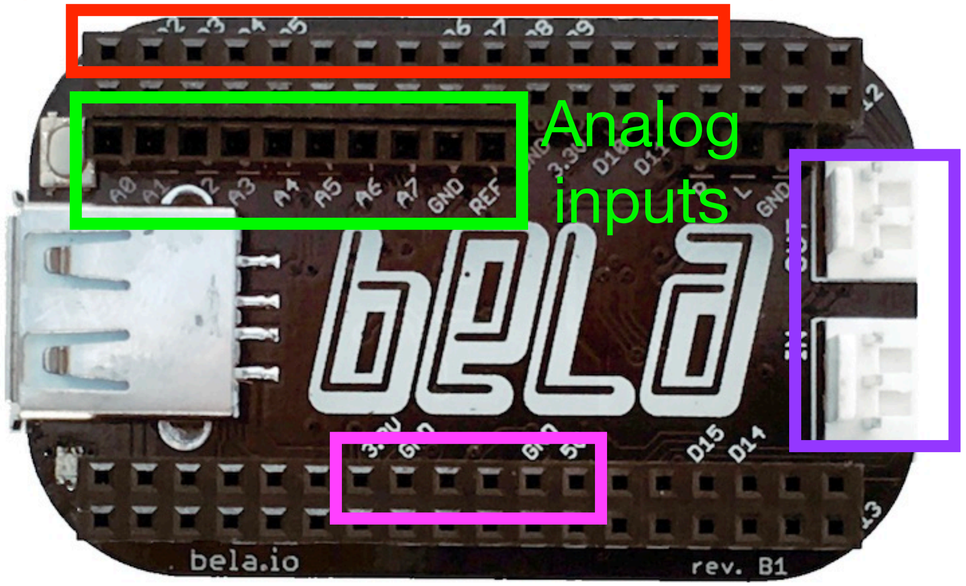
Digital inputs/outputs
(various locations around board)

Bela Mini hardware

Digital inputs/outputs
(various locations around board)

Analog inputs

USB to computer (below)



Audio out (headphones)

Audio in

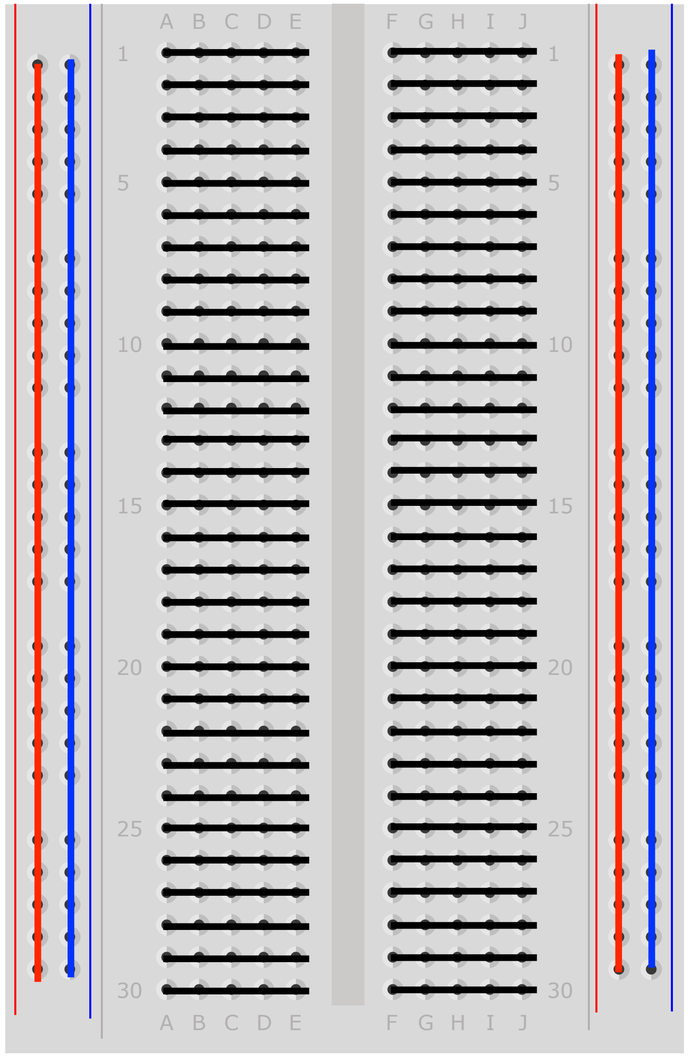
Power / Ground

Using a breadboard

Each column of 5 holes connects together...

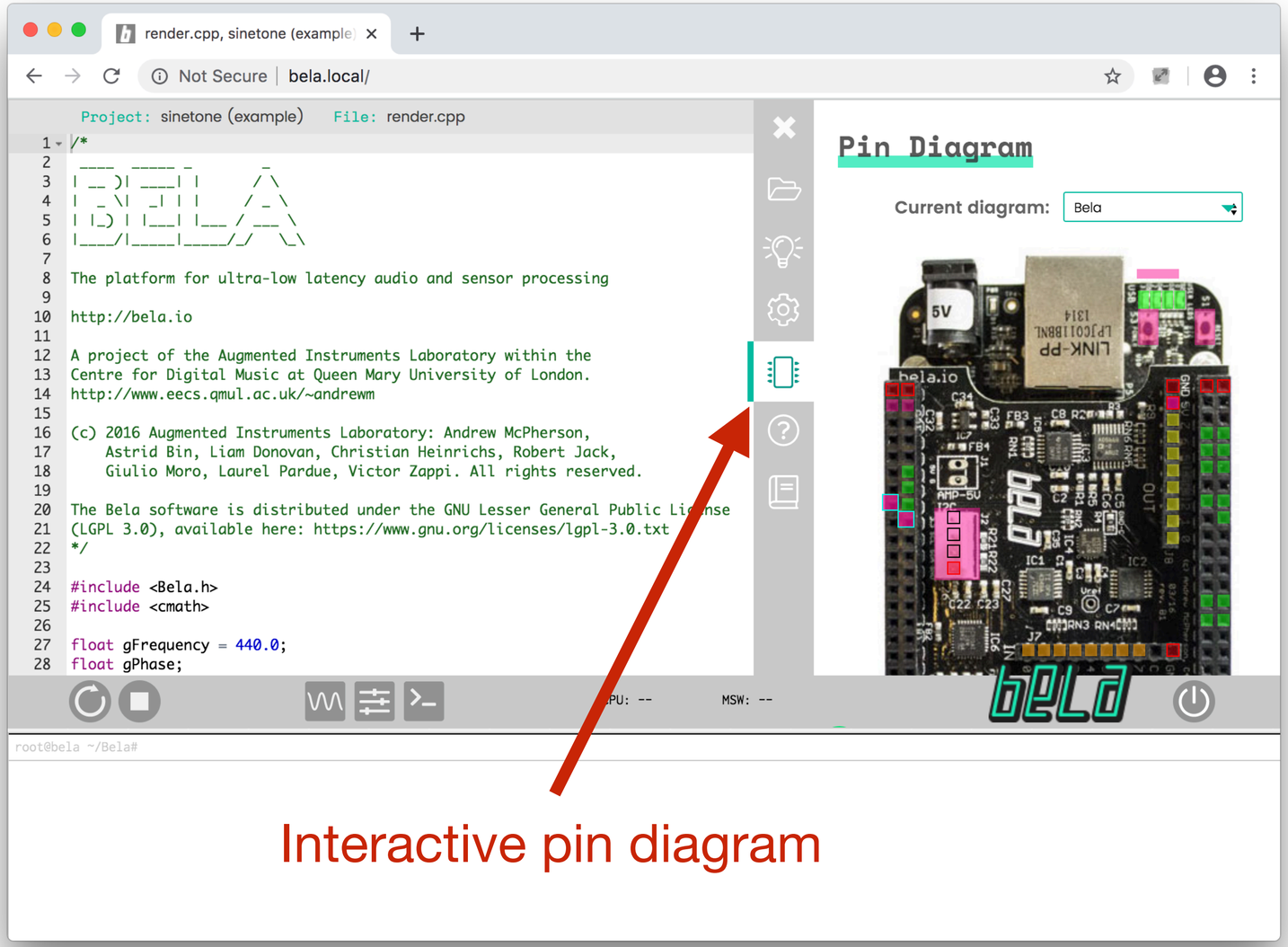
...but not across the break in the middle.

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



Digital pin numbering

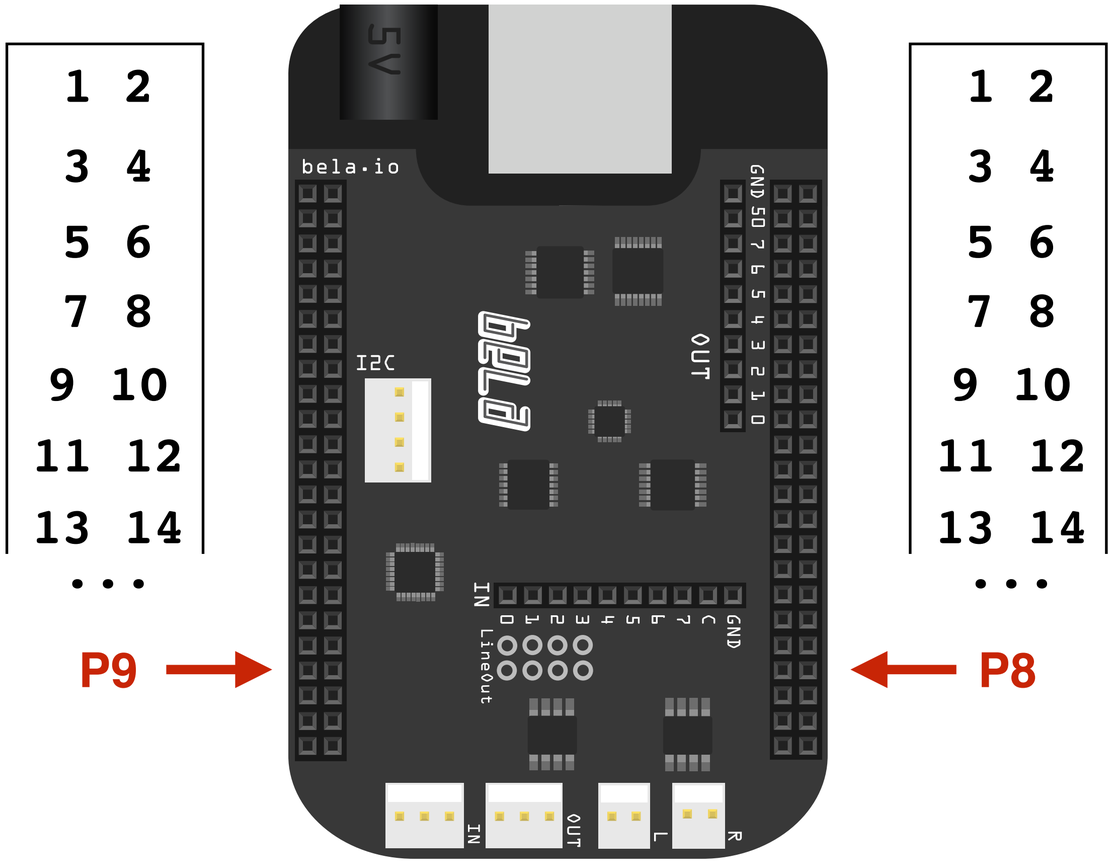
- You can always find the pinout for the digital I/O and other connectors in the IDE's Interactive Pin Diagram



Bela digital pin numbering

- 16 digital I/O pins are available on the P8 and P9 headers of the Bela cape
  - The names P8, P9 come from the BeagleBone Black
  - You can find what all the other pins do in the BeagleBone Black System Reference Manual:
    - https://github.com/beagleboard/beaglebone-black/wiki/System-Reference-Manual
  - Useful pins:
    - **Ground** on P8_1, P8_2, P9_1, P9_2
    - **3.3V** on P9_3, P9_4
  - Not all the pins are available to us

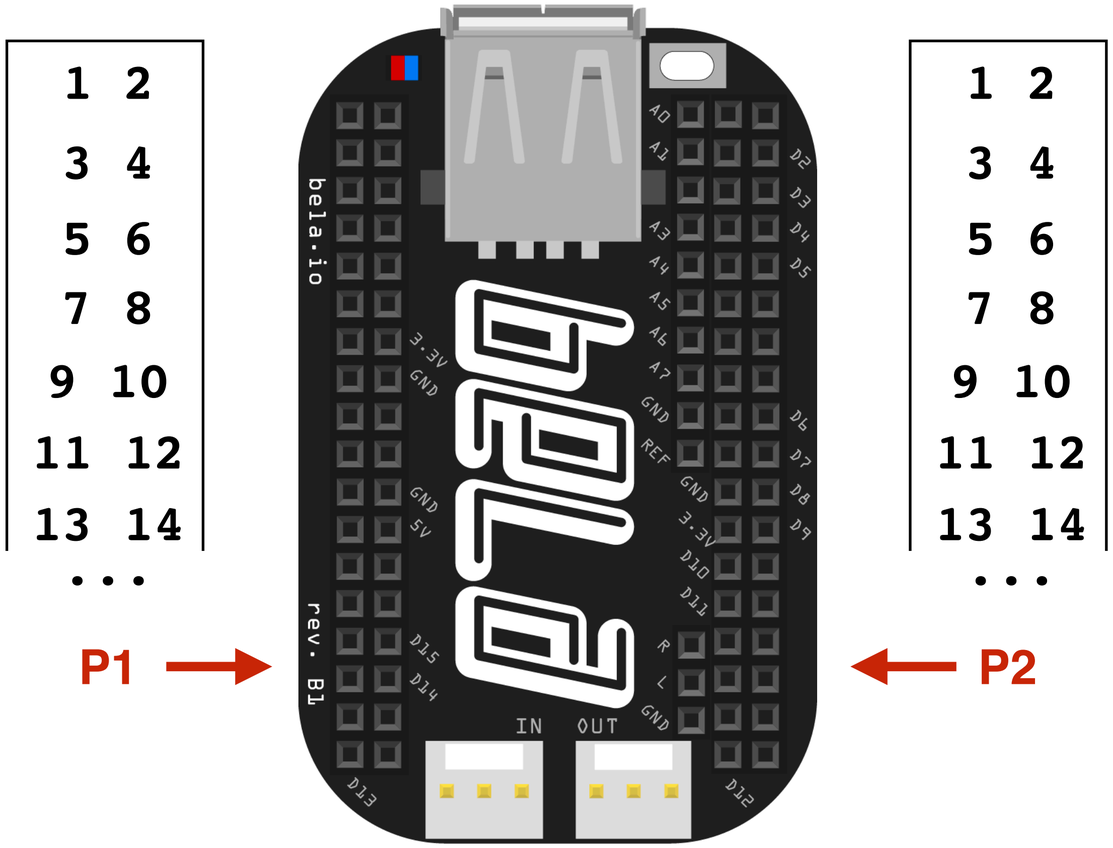
Important: NEVER use 5V with digital I/Os!



Bela Mini digital pin numbering

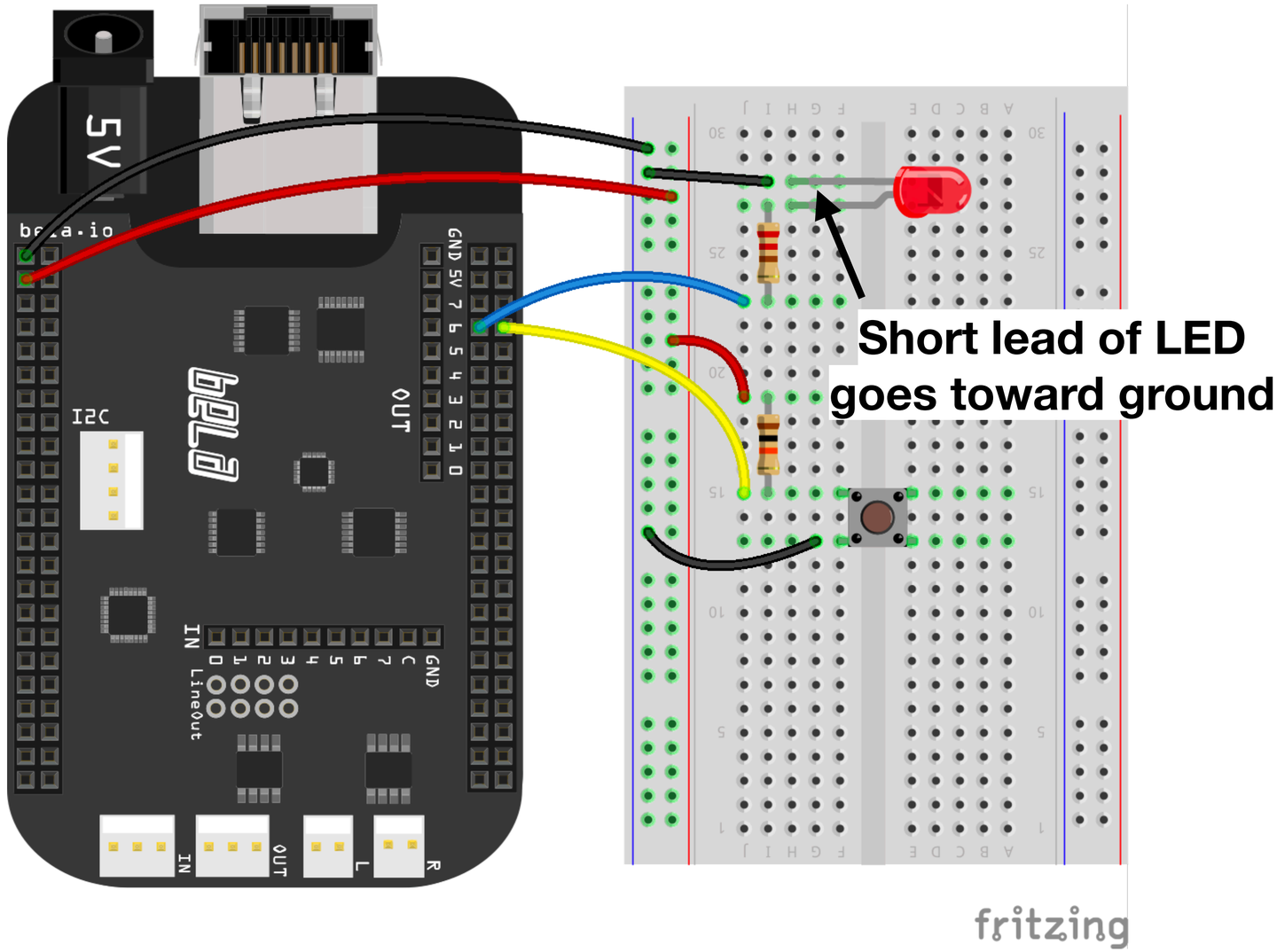
- 16 digital I/O pins are available on the P1 and P2 headers of the Bela Mini cape
  - The names P1, P2 come from the PocketBeagle
  - You can find what all the other pins do in the PocketBeagle System Reference Manual:
    - https://github.com/beagleboard/pocketbeagle/wiki/System-Reference-Manual
  - Digital and power pins are labelled on the Bela Mini cape
  - Not all the pins are available to us

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



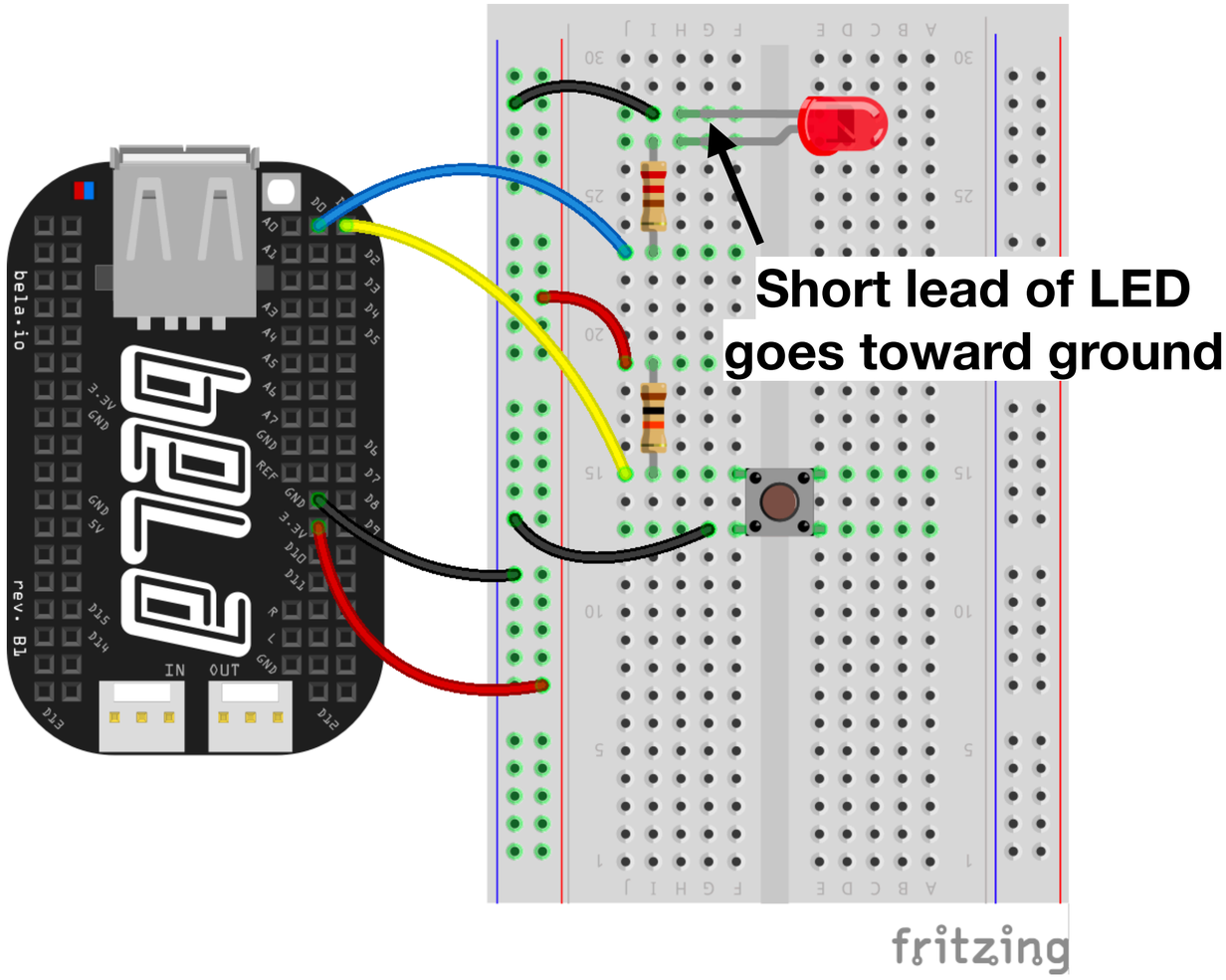
Connect LED and button

- Connect the LED to digital pin 0
  - P8_7 on Bela
  - P2_1 on Bela Mini
- Connect the button to digital pin 1
  - P8_8 on Bela
  - P2_2 on Bela Mini
- Important: the button requires 3 wires and a resistor!
- Run the digital-io example from the companion materials



Connect LED and button

- Connect the LED to digital pin 0
  - P8_7 on Bela
  - P2_1 on Bela Mini
- Connect the button to digital pin 1
  - P8_8 on Bela
  - P2_2 on Bela Mini
- Important: the button requires 3 wires and a resistor!
- Run the digital-io example from the companion materials



Digital I/O

- Digital input and output is sometimes called GPIO
  - General-Purpose Input and Output
- Digital I/Os have only two levels (“logic levels”)
  - High or Low, 1 or 0 - which on Bela corresponds to 3.3V and 0V
- Digital inputs and outputs on Bela are sampled signals
  - 44.1kHz sample rate, just like audio
  - Unlike audio, the possible values are **only** 0 and 1 (nowhere in between)

```
int digitalRead(BelaContext *context, int frame, int channel);
void digitalWrite(BelaContext *context, int frame, int channel, int 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, ..., 15) to write

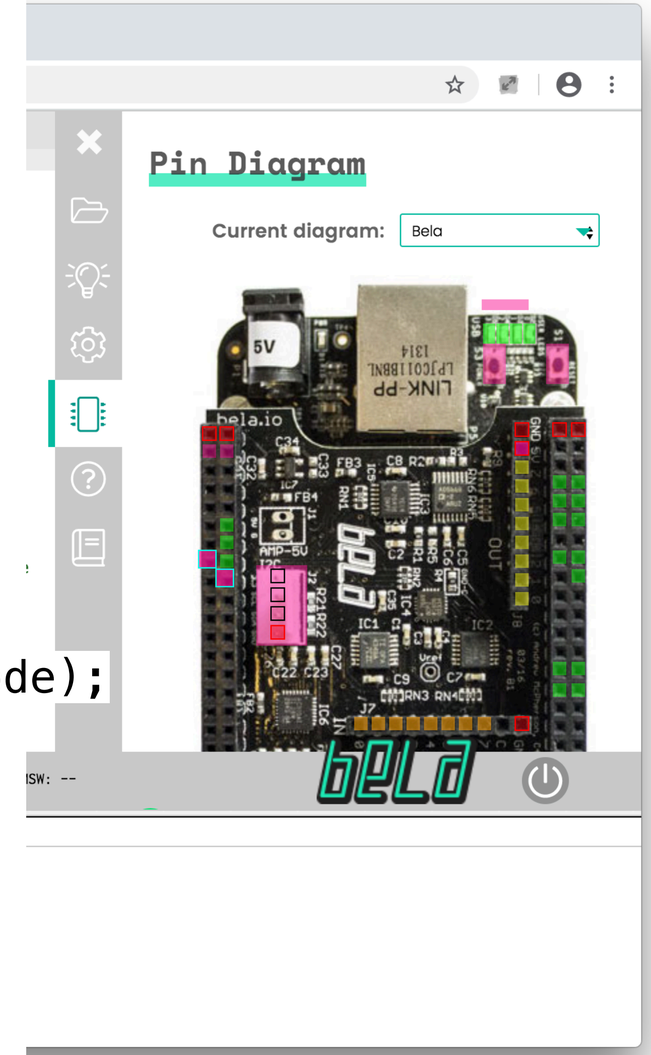
What value to write

Digital I/O

- Bela has only 16 digital pins
  - See the interactive pin diagram for where to find them
- How do we know which are inputs vs. outputs?
  - Any digital pin could be either!
  - We have to tell the system which way to set it
- All digital pins start as inputs by default
  - To change, we call a function:

```
void pinMode(BelaContext *context, int frame, int channel, int mode);
```

  - As with `digitalRead()` and `digitalWrite()`, this is set to take place at a particular frame
    - Note: like analog output, digital output has the functions `digitalWriteOnce()` and `pinModeOnce()`



Digital I/O code

```
// ***** Inside setup():
// Set the LED pin to be an output
pinMode(context, 0, kLedPin, OUTPUT);
// Set the button pin as an input. This isn't strictly needed but is recommended
pinMode(context, 0, kButtonPin, INPUT);
```

Set the pin directions at the beginning

Predefined constants

```
// ***** Inside render(), within the for() loop:
// Read the button
int status = digitalRead(context, n, kButtonPin);

// Check if the value was low
if(status == LOW) {
    // Low input means the button was pushed. Turn on the light and play the sound.
    // Generate white noise: random values between -1 and 1
    float noise = 2.0 * (float)rand() / (float)RAND_MAX - 1.0;
    out = noise * 0.1;
    digitalWriteOnce(context, n, kLEDPin, HIGH);
}
else {
    // Button wasn't pushed. Turn the LED off, no sound
    digitalWriteOnce(context, n, kLedPin, LOW);
}
```

Read the input and check its value

Write the output (3.3V)

Write the output (0V)

White noise

- White noise is a random signal with constant power spectral density
  - Equal energy at all frequencies
  - Compare to pink noise, red noise which have more power at lower frequencies
- We can make a digital white noise signal with a random number generator
  - Each sample is independent of the previous one, with a uniform distribution

```
float noise = 2.0 * (float)rand() / (float)RAND_MAX - 1.0;
```

Scale range to 0-2

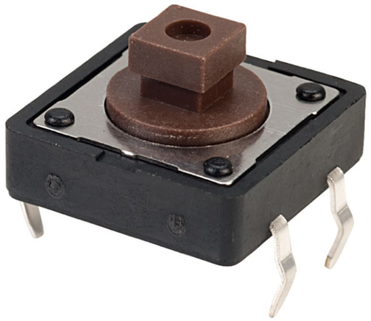
Random integer between 0 and `RAND_MAX` (see `stdlib.h`)

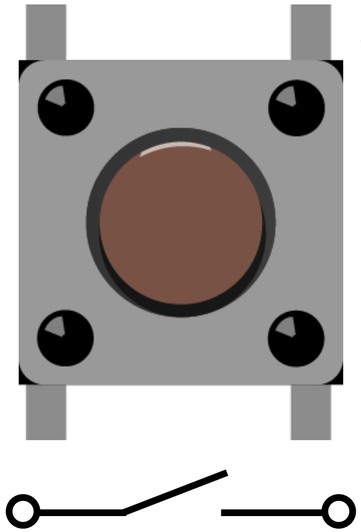
Maximum possible value for the random number: dividing by it normalises to 0-1

Rescale range to -1 to 1

How does the button work?

- Breadboard-style buttons are known as tactile switches
  - Extremely common in electronics, in both through hole and surface mount forms
  - Typically a normally open switch (connects only when button is pushed)

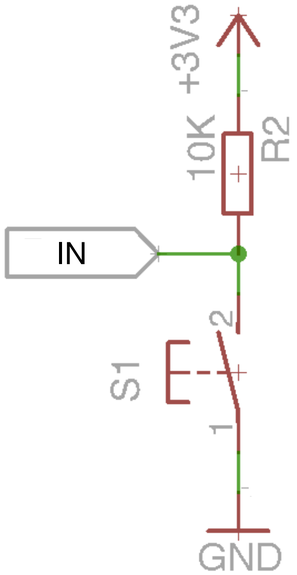




Why does it have 4 leads?

Just for mechanical stability

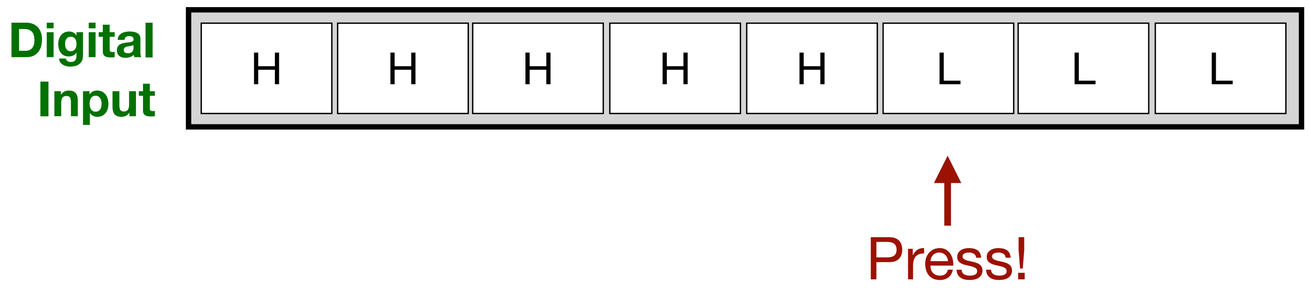
Important: the switch is between the **two adjacent pins on the same side**

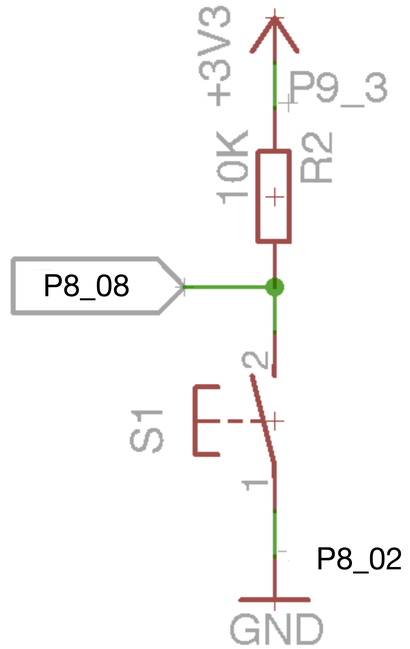


- To use the button, we need 3 wires a resistor:
  - When pushed, digital input connected to ground
  - When not pushed, resistor pulls input up to 3.3V

Detecting button presses

- How do we know if the button was held on any given frame?
  - `digitalRead()` will return... LOW
  - Be sure to compare to the wiring of your specific circuit!
- How do we know if the button was just now pressed?
  - i.e., how do we identify when a button press first begins?
  - Let's look at each successive frame of digital input:

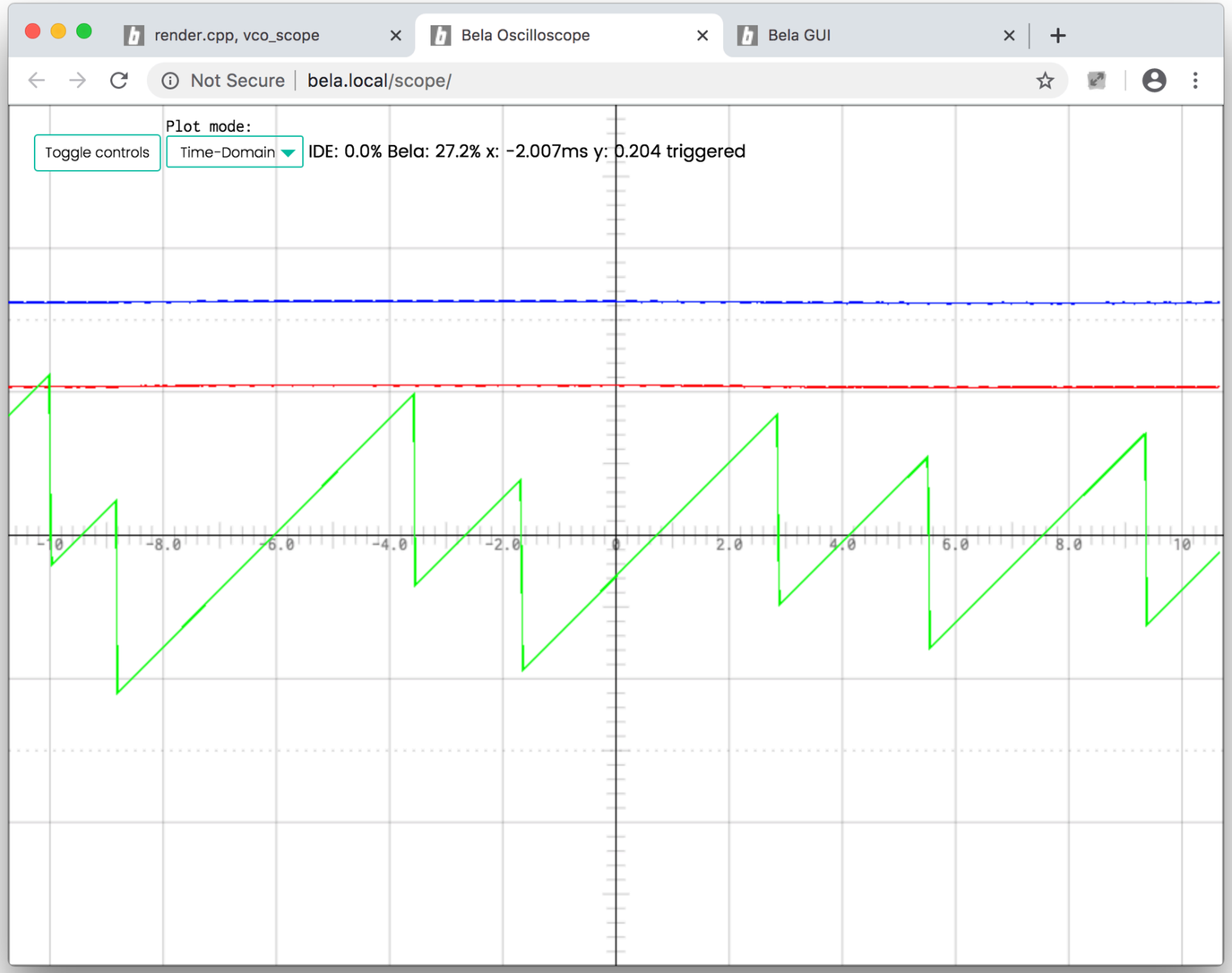




  - Condition is: previous value was high (= 1) and current value is low (= 0)
- This is known as edge detection

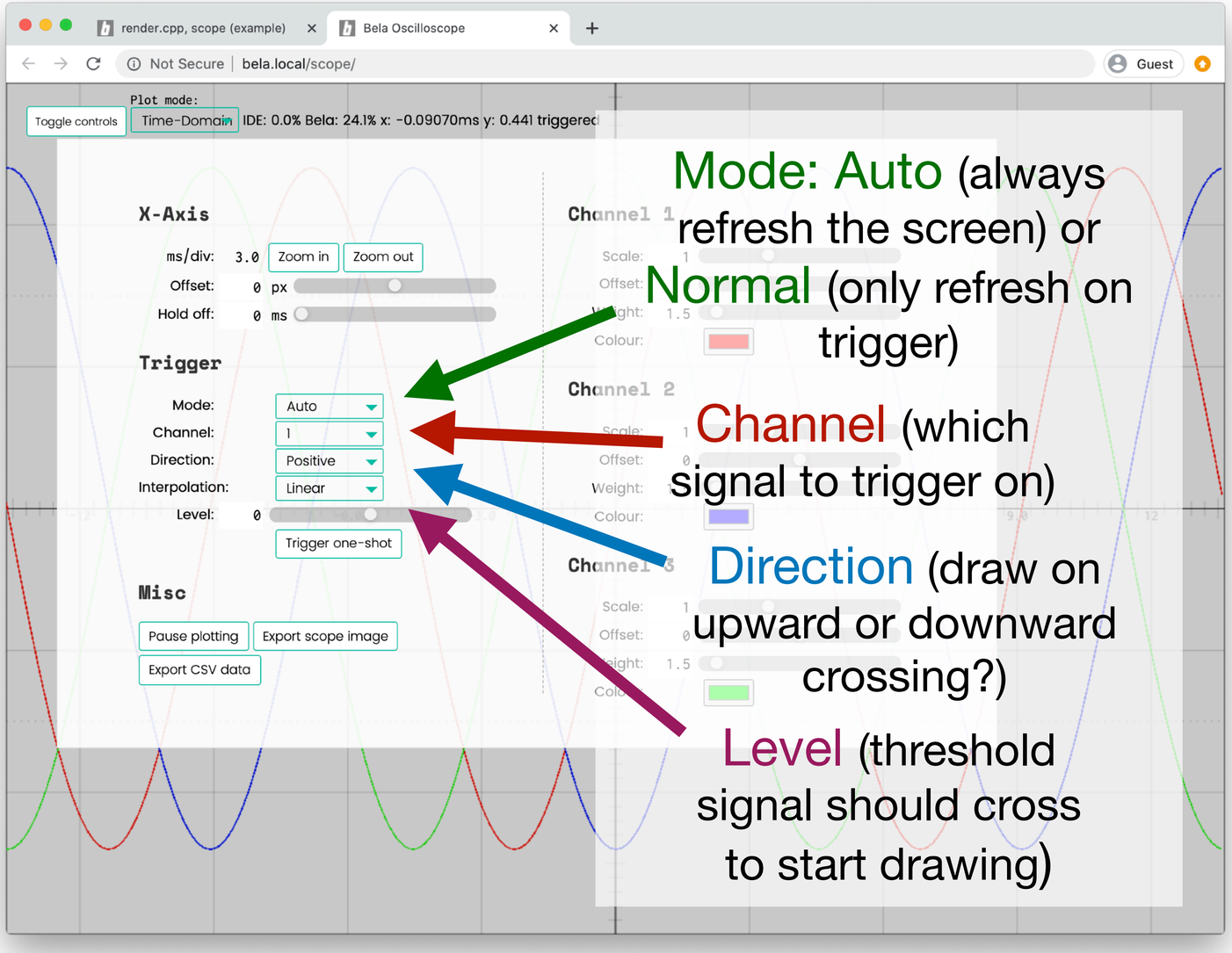
Oscilloscope

- Bela features an oscilloscope that runs in the browser
  - As the name "*oscillo-scope*" implies, a way of viewing oscillations (signals)
- Typical use of oscilloscope is an X-Y plot
  - Time on the X-axis
  - Amplitude on the Y-axis
- The scope can plot any signal made or used by your program
  - Not just I/O signals!



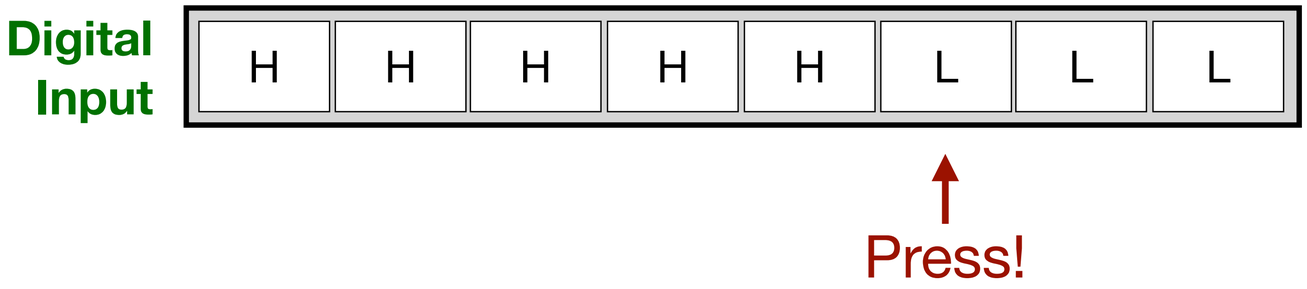
Oscilloscope controls: trigger

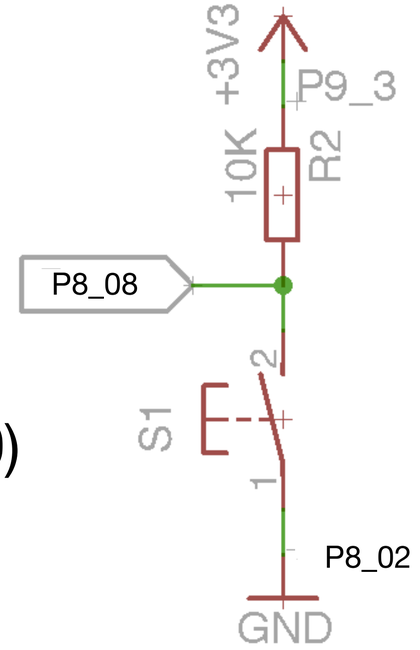
- Oscilloscopes show the signal in brief snapshots
  - One screenful corresponds to a limited window of time
  - To show the signal in real time, the scope regularly refreshes the screen with new time windows
- To keep the picture still, what do we need to do?
  - Start drawing at the same phase in the waveform each time
  - This is what the trigger does: sets where and how the screen starts drawing



Button as trigger

- Once again: how do we know if the button was just now pressed?
  - Let's look at each successive frame of digital input:





  - Edge detection: previous value was high (= 1) and current value is low (= 0)
- What do we mean by "previous value", and how do we find it?
  - Value of the digital input on the frame (sample) before this one
  - We need to remember the previous value by storing it in a global variable
  - That way, our next time through the loop in `render()`, we will have the old value alongside the current one

Gates and triggers

- In addition to control voltages (CVs), some synth modules are controlled by gates and triggers
  - Both are effectively digital signals, with only two states (on and off)
- A gate is a signal representing two states: open and closed
  - Usually, by convention, a positive voltage is open and 0V is closed
  - The ongoing value of the gate signal is what matters
- A trigger is a signal representing an instantaneous event
  - Used to begin envelopes, advance sequencers, and many other applications
  - The transition (or edge) is what we care about - usually the edge from low to high
- In the digital-io example, we built a gate with the button
  - Pushing the button opens the gate so we can hear the noise
- Next, let's build a project using the button as trigger!

Step sequencer

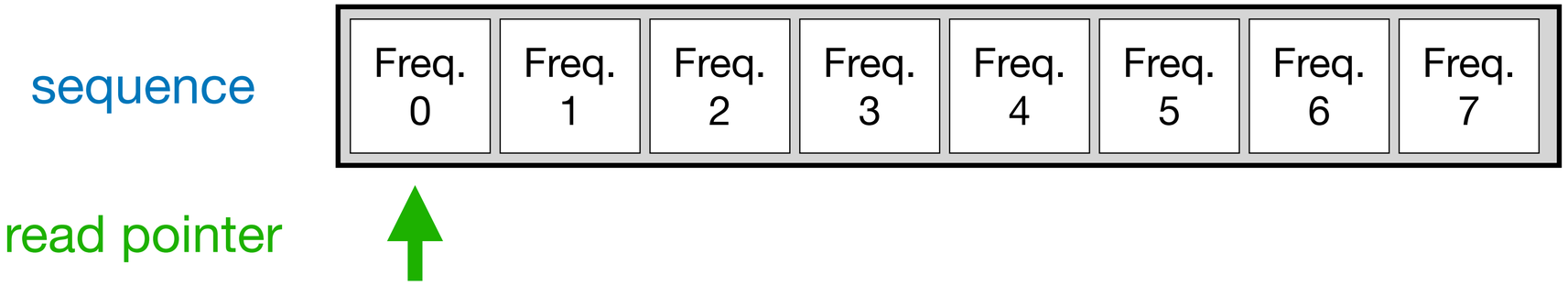
- A step sequencer plays a set pattern of notes in a loop
  - Step sequencers commonly store patterns of 8 or 16 notes
  - Every time it receives a trigger, it advances to the next note
- For now, let's focus just on controlling pitch (frequency)
  - We could implement a step sequencer like this:

  - We store 8 frequencies in a buffer, and the read pointer keeps track of which one we are currently reading. That frequency then controls an oscillator.
  - Does this architecture look familiar?
  - Very similar to playing a wavetable!

Step sequencers and wavetables

- Task: in the step-sequencer example,
  - Store the previous value of the button in a global variable
  - Look for a falling edge (high-to-low transition) on the input, which is the trigger
  - When you receive this trigger, advance the read pointer, looping as needed
  - Once this works, turn the LED on and off for successive steps (e.g. on for even steps)

- For a step sequencer, our buffer holds frequencies (or more generally, CV values)
  - We move the read pointer only when we receive a trigger (i.e. a signal edge)



Edge detection code

- We need the current and previous state of the button:

```
// Last state of the button
int gLastButtonStatus = HIGH;

void render(BelaContext *context, void *userData)
{
    for(unsigned int n = 0; n < context->audioFrames; n++) {
        // [...]
        // Check for a button press to advance the sequence
        int status = digitalRead(context, n, kButtonPin);

        if(status == LOW && gLastButtonStatus == HIGH) {
            // Button was pressed - advance the sequence
            gSequencerLocation++;
            if(gSequencerLocation >= gSequencerBuffer.size())
                gSequencerLocation = 0;
        }

        gLastButtonStatus = status;
        // [...]
    }
}
```

global variable to hold previous state

so that the **next** time it holds last status

set `gLastButtonStatus` to current status

CV out (Bela only)

- As a final step, let's have our step sequencer output a CV
  - We'll use the analog output with a 1V/octave standard
- We already have the sequence stored as MIDI note numbers:

36 (C2)	**48** (C3)	**39** (E♭2)	**51** (E♭3)	**53** (F3)	**41** (F2)	**55** (G3)	**43** (G2)

  - We used this formula to convert to frequency: $f_{out} = 440 \cdot 2^{\frac{N-69}{12}}$
  - Let's convert MIDI notes to octaves with the lowest note (36) as a reference
  - How do we convert note number N to semitones above the reference? N - 36
  - Now how about octaves above the reference? (N - 36) / 12
- Task: add CV out to the step-sequencer example
  - *Hint: given that analog output range is 0-5V, what value do we have to write to get 1V?*

Keep in touch!

Social media:

@BelaPlatform

forum.bela.io

blog.bela.io

More resources and contact info at:

learn.bela.io/resources