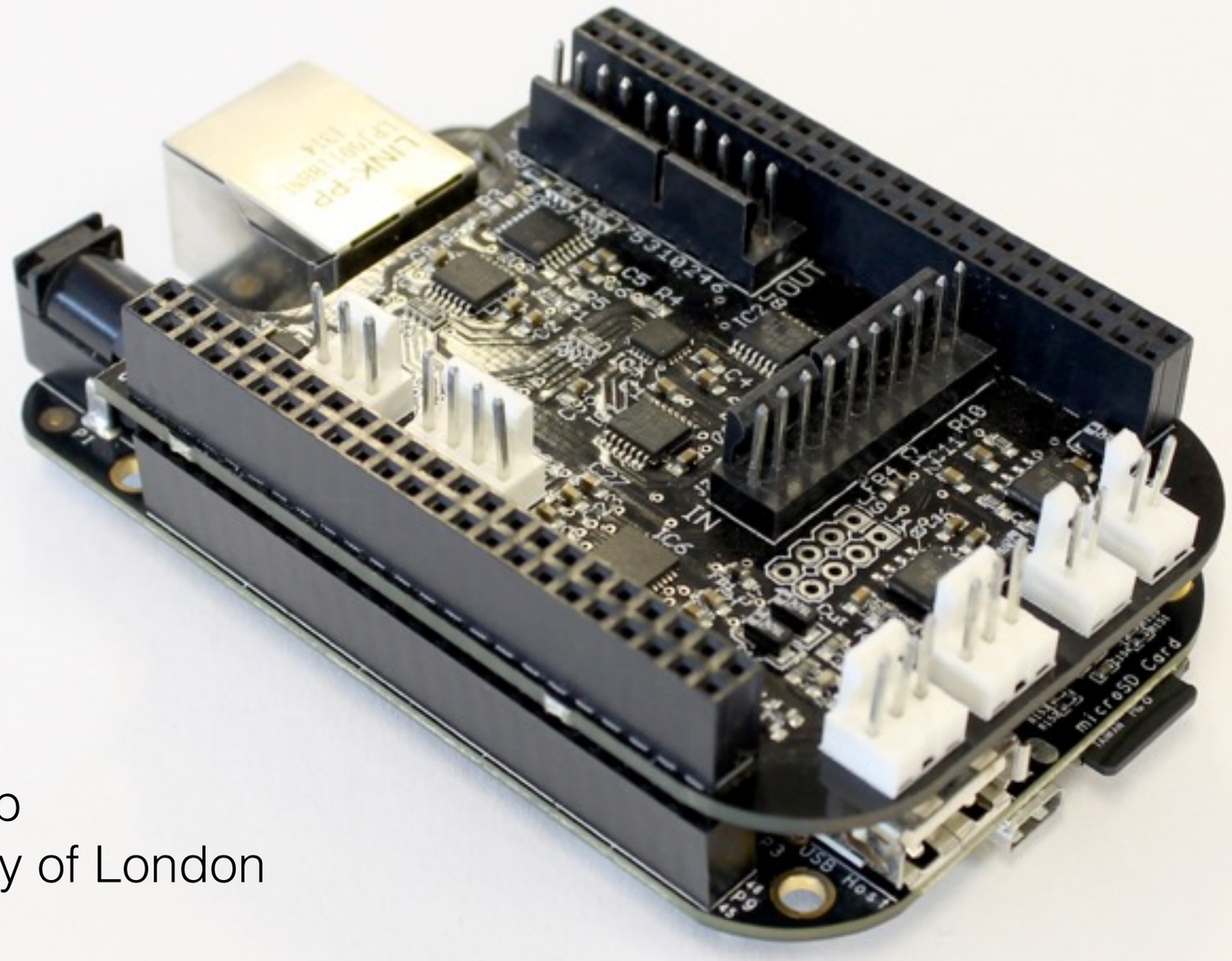
bela

Ultra-low latency audio and sensor processing on the BeagleBone Black



A project by
The Augmented Instruments Lab
at C4DM, Queen Mary University of London

http://bela.io

EPSRC
centre for digital music
Queen Mary University of London

How it started

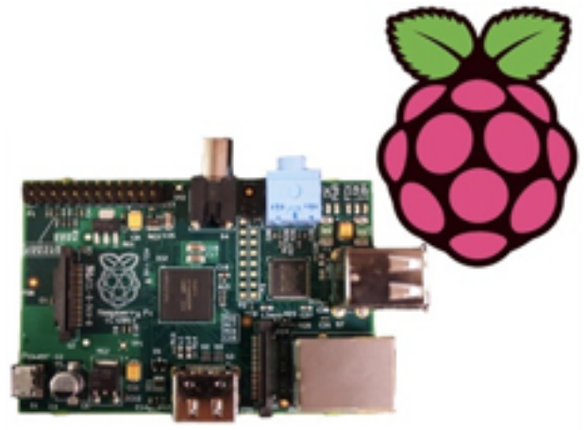
The Goal:

High-performance, self-contained audio and sensor processing

Already available platforms:

ARDUINO

- Easy low-level hardware connectivity
- No OS = precise control of timing
- Very limited CPU (8-bit, 16MHz)
- Not good for audio processing

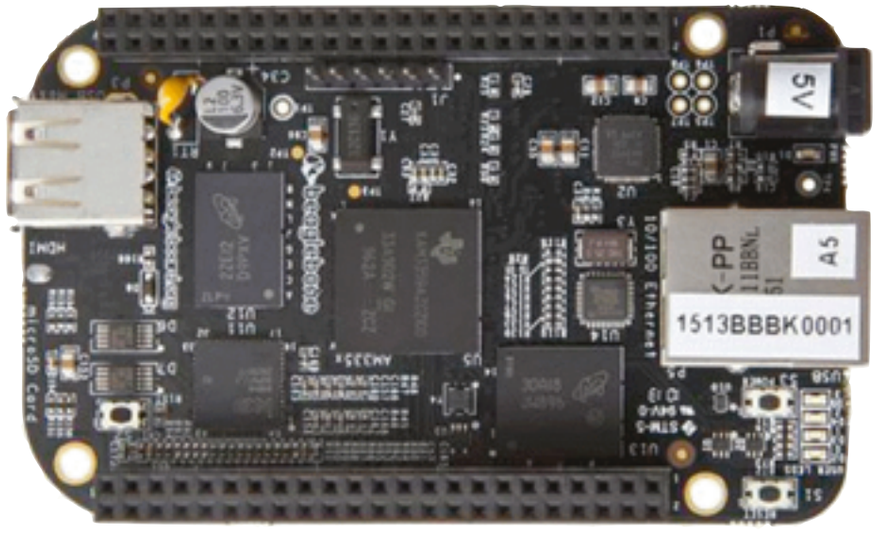


- Reasonable CPU (up to 1GHz ARM)
- High-level hardware (USB, network etc.)
- Limited low-level hardware
- Linux OS = high-latency / underruns

- Fast CPU
- High-level hardware (USB, network etc.)
- Arduino for low-level
- USB connection = high-latency, jitter
- Bulky, not self-contained

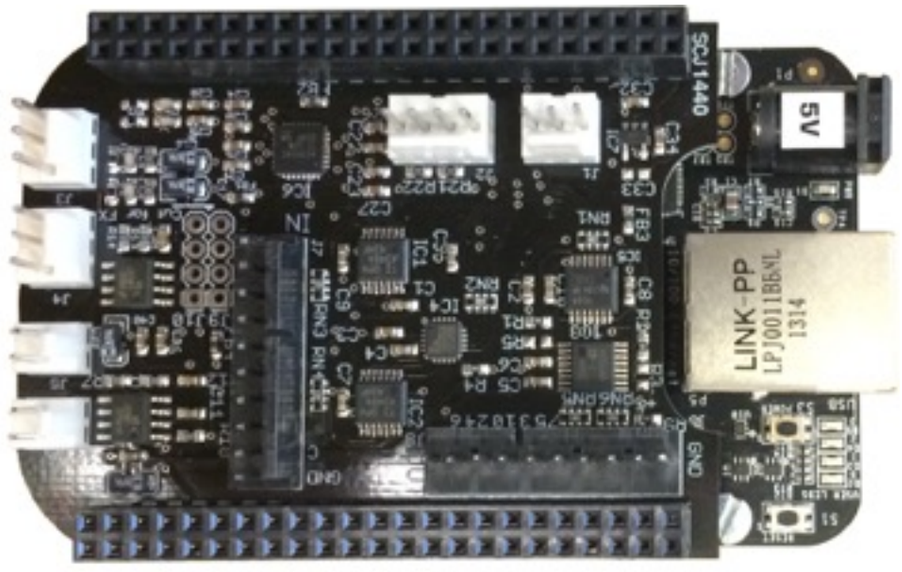
bela

Hardware



BeagleBone Black

1GHz ARM Cortex-A8
NEON vector floating point
PRU real-time microcontrollers
512MB RAM



Custom Bela Cape

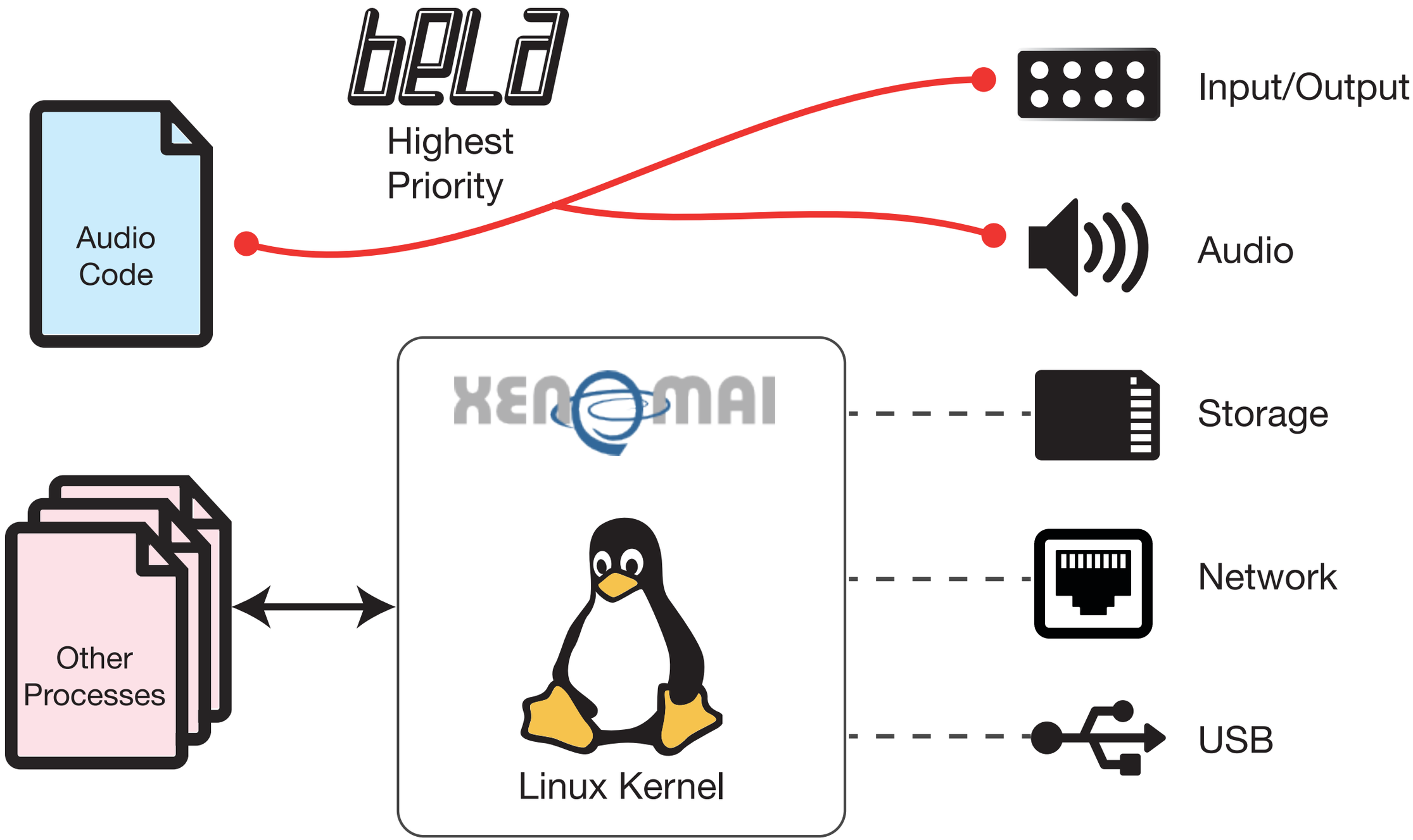
Stereo audio in + out
Stereo 1.1W speaker amps
8x 16-bit analog in + out
16x digital in/out

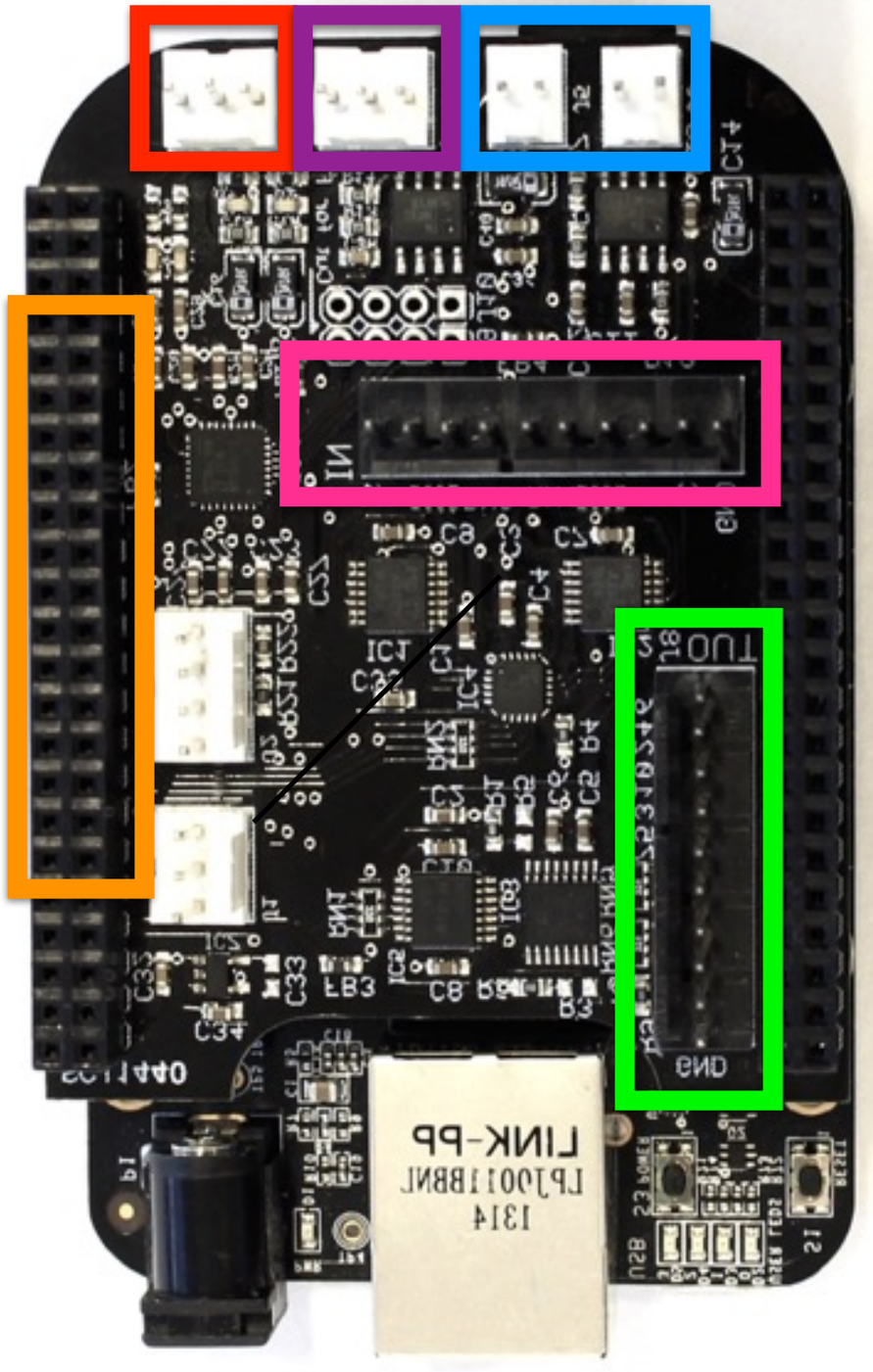
Bela

Features

- **1ms round-trip audio latency** without underruns
- **High sensor bandwidth:** digital I/Os sampled at 44.1kHz; analog I/Os sampled at 22.05kHz
- **Jitter-free alignment** between audio and sensors
- **Hard real-time audio+sensor performance**, but full Linux APIs still available
- Programmable using **C/C++, Pd and Faust**
- Designed for **musical instruments and live audio**

How it works


bELa
Highest
Priority
Audio
Code
Input/Output
Audio
Storage
Network
USB
XENOMAI
Other
Processes
Linux Kernel



- **Speakers** with on-board amps
- **Audio In**
- **Audio Out**
- **16x digital I/O**
- **8x 16-bit analogue in (22.05kHz)**
- **8x 16-bit analogue out (22.05kHz)**

Find an interactive pin out diagram at http://bela.io/belaDiagram

Connected.

Embedded.

Fast.

Easy to get started.

Getting Started

bela.io/code/wiki

Materials

1. **BeagleBone Black** (BBB)
2. **Bela Cape**
3. **SD card** with Bela image
4. 3.5mm headphone jack **adapter cable**
5. **Mini-USB cable** (to attach BBB to computer)
6. Also useful for hardware hacking: **breadboard**, **jumper wires**, etc.

Step 1

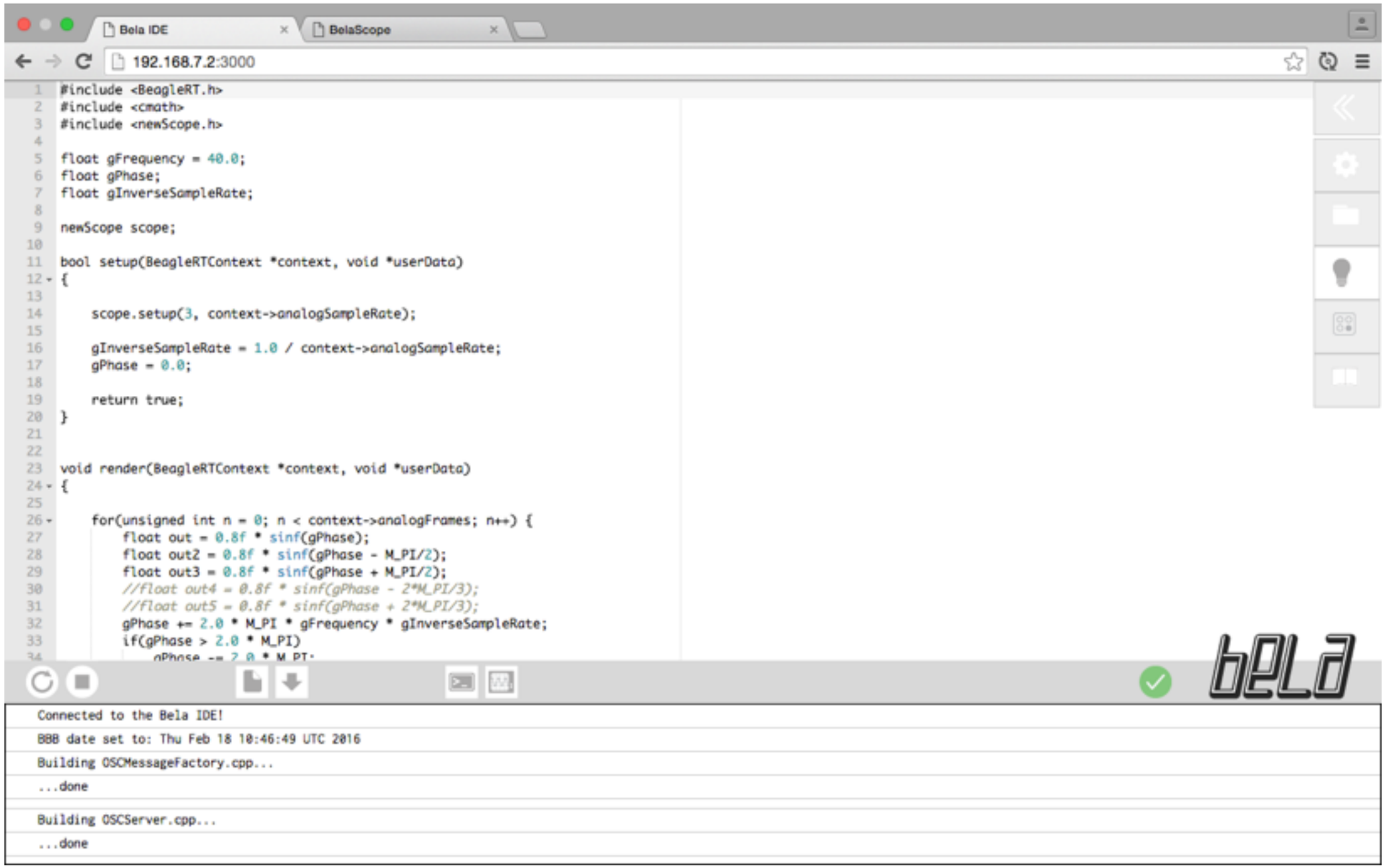
Install BBB drivers

Get and install the BeagleBone Black drivers for your operating system:

http://bela.io/code/wiki --> Getting Started

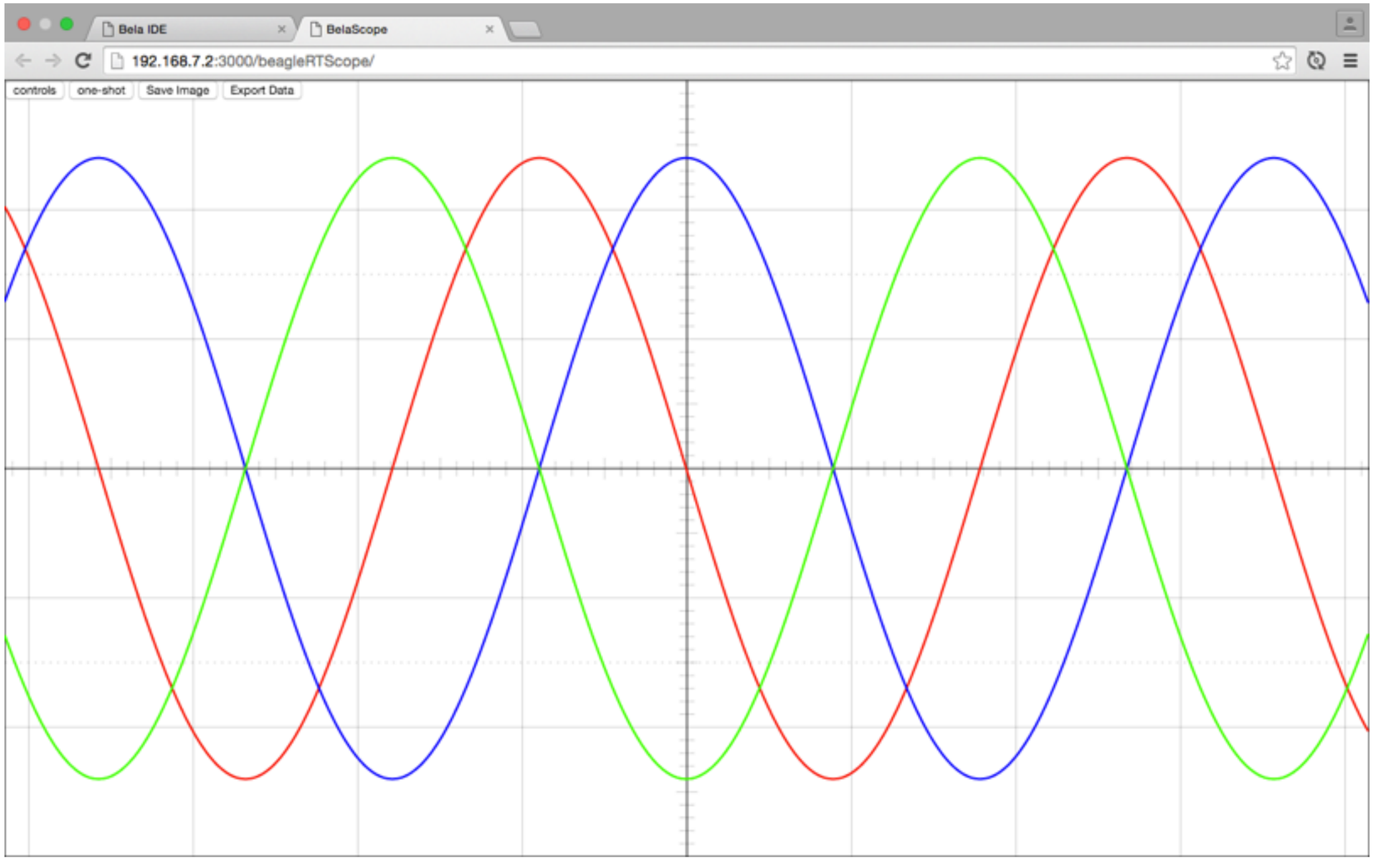
Access the IDE:

http://192.168.7.2:3000



Access the IDE:

http://192.168.7.2:3000



API introduction

- 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
- bela.io/code/embedded *Code docs*

Step 2

How to build other projects

1. **Web interface**: http://192.168.7.2:3000
   Edit and compile code on the board
2. **Building scripts:**
   1. **Heavy Pd-to-C compiler** (https://enzienaudio.com)
      Make audio patches in Pd-vanilla, translate to C and compile on board
   2. **Libpd**
      Compile Pd patches without Heavy - access to more objects but not as fast, but good for prototyping
   3. **Faust**
      Build online, export to C++, run on Bela

Bela and PureData

Heavy	**libpd**
Proprietary compiler, cloud-based, MIT non-commercial code	Free
Targets a variety of platforms (C, js, Unity,VST2)	Many ports (ofxpd, webpd …)
Requires internet connection and local compiling (~1minute)	Instantaneous (save the pd patch and restart)
Generates fast, optimized code, uses little CPU	It is just Pd (...)

libpd on Bela

How to run PureData patches on Bela with libpd :

1. Go to http://bela.io/code/files and download the bela-ableton-2016-04-12.zip archive
2. Unzip the archive into a convenient location and open a terminal window
3. Navigate into the scripts/ folder and run
   `./run_pd_libpd.sh ../projects/heavy/pd/demo-track/`
4. Type "yes" and you should hear something

Queen Mary University of London

Bela and Faust

- Today: you will have to download the C++ file generated by the http://faust.grame.fr/onlinecompiler/ (after setting the -i flag), save it on your computer and target it with the build_project.sh script, as in:

```
/path/to/bela/repo/scripts/build_project.sh /path/to/faust/file/CppCode.cpp
```

```
freq = hslider("[1]Frequency[BELA:ANALOG_0]", 440,460,1500,1):smooth(0.999);
pressure = hslider("[2]Pressure[style:knob][BELA:ANALOG_4]", 0.96, 0.2, 2.0, 0.01):smooth(0.999):min(0.99):max(0.2);
gate = hslider("[0]ON/OFF (ASR Envelope)[BELA:DIGITAL_0]",0,0,1,1);
```

Queen Mary University of London

Help me with Supercollider

- We got it to work, thank to Dan Stowell at C4DM
- We run 120sinewaves for 55% CPU time.
- Can you make something more useful with it?

Queen Mary
University of London

Interested to pre-order a kit?

65£ for a cape + SD card

Delivery: July

http://bit.ly/1eKffsL

Stay tuned! Join the announcement list at

http://bela.io

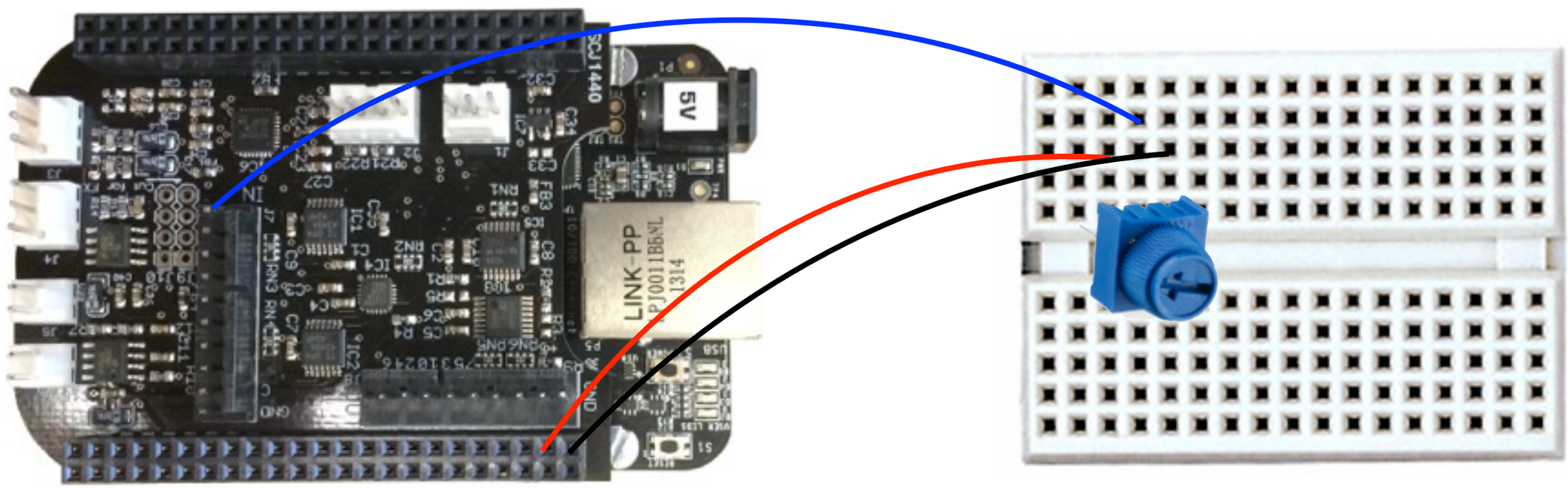
Join the discussion list at
http://lists.bela.io/pipermail/discussion-bela.io/

Connect a Potentiometer

a.k.a. a “pot” or knob

The pot has 3 pins
5V and GND on the outside
Bela analog in in the middle

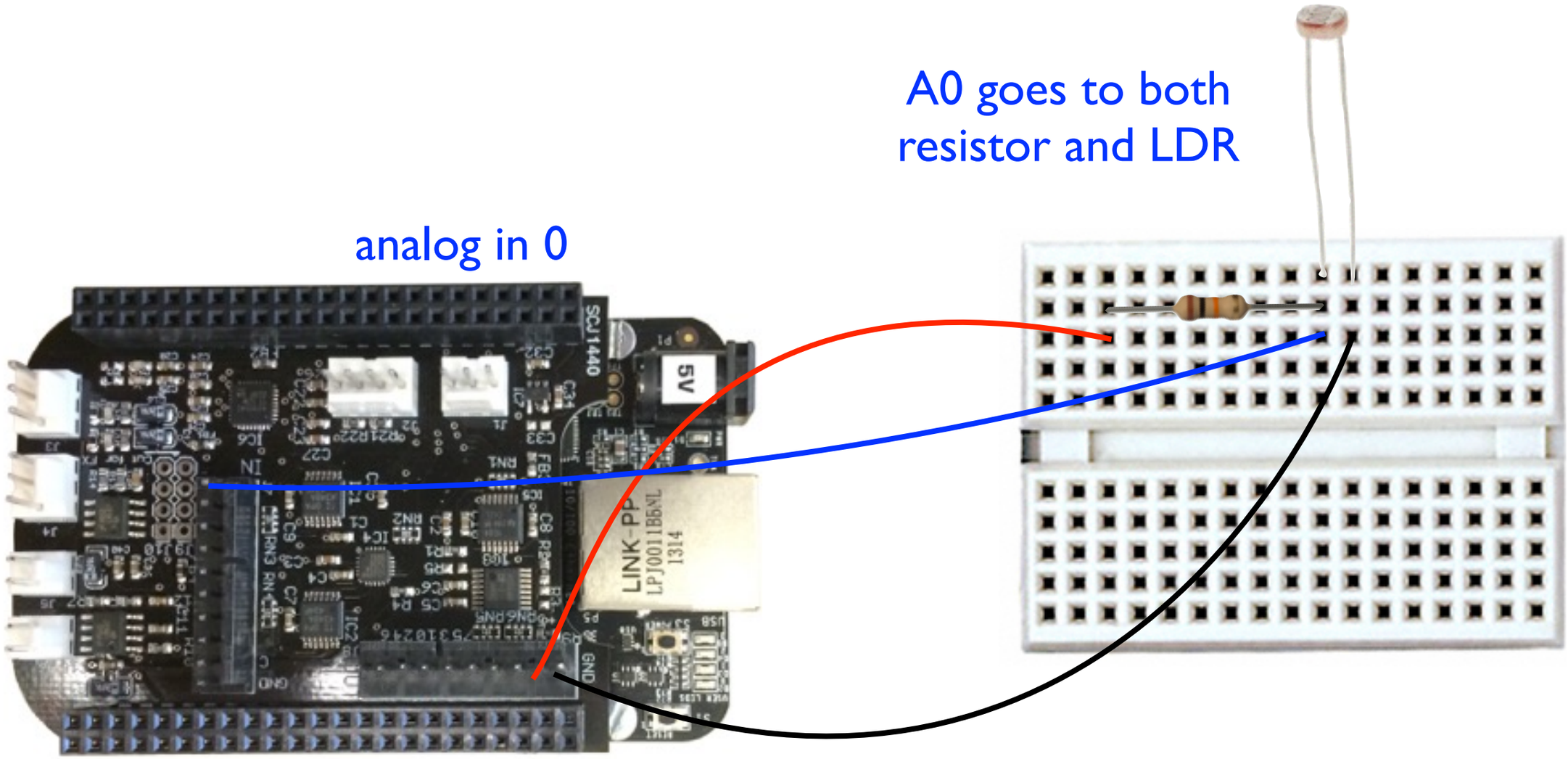
analogue in 0



5V GND (ground)

Connect a LDR/FSR*

* Light-Dependent Resistor / Force-Sensing Resistor

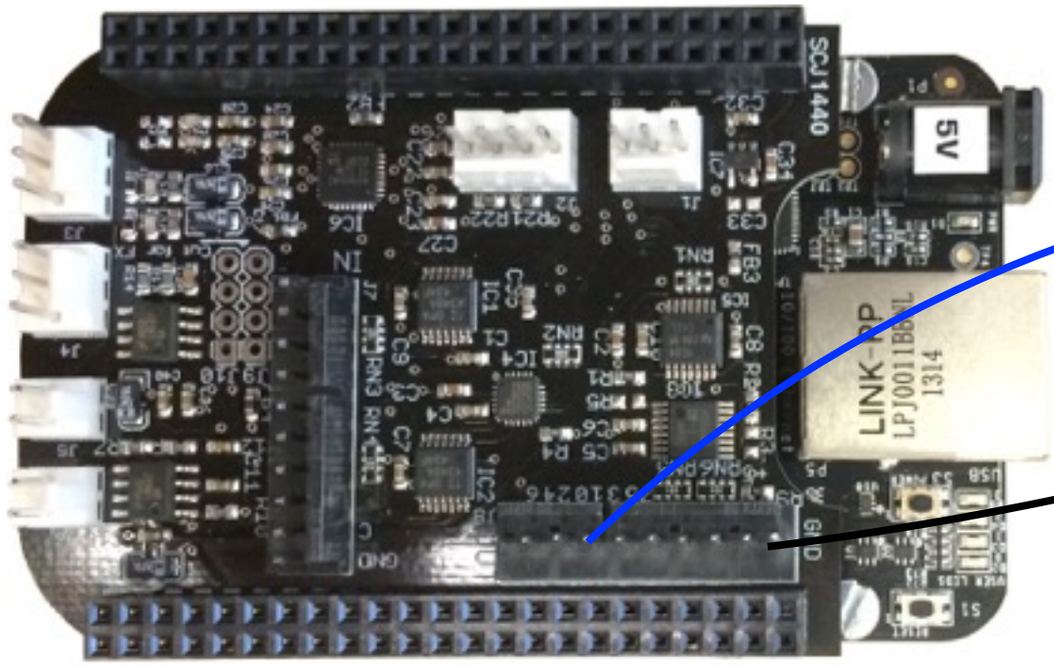


Connect an LED*

* Light-Emitting Diode

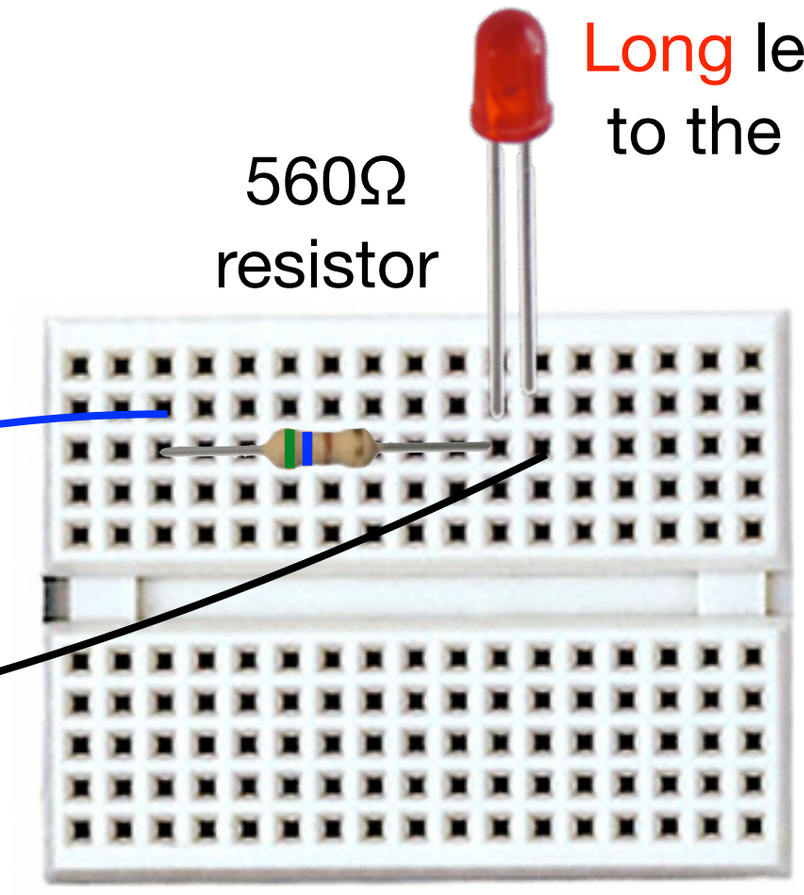
analog out 0

(note pinout:
6 4 2 0 1 3 5 7)



560Ω
resistor

Long lead goes
to the resistor



GND (ground)

Analog input

```
float gPhase;
float gInverseSampleRate;           /* Pre-calculated for convenience */
int gAudioFramesPerAnalogFrame;

extern int gSensorInputFrequency;   /* Which analog pin controls frequency */
extern int gSensorInputAmplitude;   /* Which analog pin controls amplitude */

void render(BeagleRTContext *context, void *userData)
{
    float frequency = 440.0;
    float amplitude = 0.8;

    for(unsigned int n = 0; n < context->audioFrames; n++) {
        /* There are twice as many audio frames as matrix frames since
            audio sample rate is twice as high */
        if(!(n % gAudioFramesPerAnalogFrame)) {
            /* Every other audio sample: update frequency and amplitude */
            frequency = map(analogReadFrame(context,
                                n/gAudioFramesPerAnalogFrame,
                                gSensorInputFrequency),
                            0, 1, 100, 1000);
            amplitude = analogReadFrame(context,
                            n/gAudioFramesPerAnalogFrame,
                            gSensorInputAmplitude);
        }

        float out = amplitude * sinf(gPhase);

        for(unsigned int channel = 0; channel < context->audioChannels; channel++)
            context->audioOut[n * context->audioChannels + channel] = out;

        gPhase += 2.0 * M_PI * frequency * gInverseSampleRate;
        if(gPhase > 2.0 * M_PI)
            gPhase -= 2.0 * M_PI;
    }
}
```

This runs every other sample

Read the analog input at the specified frame

Map the 0-1 input range to a frequency range

Digital I/O

```
void render(BeagleRTContext *context, void *userData)
{
    static int count = 0; // counts elapsed samples
    float interval = 0.5; // how often to toggle the LED (in seconds)
    static int status = GPIO_LOW;

    for(unsigned int n = 0; n < context->digitalFrames; n++) {
        /* Check if enough samples have elapsed that it's time to
           blink to the LED */
        if(count == context->digitalSampleRate * interval) {
            count = 0; // reset the counter
            if(status == GPIO_LOW) {
                /* Toggle the LED */
                digitalWriteFrame(context, n, P8_07, status);
                status = GPIO_HIGH;
            }
            else {
                /* Toggle the LED */
                digitalWriteFrame(context, n, P8_07, status);
                status = GPIO_LOW;
            }
        }

        /* Increment the count once per frame */
        count++;
    }
}
```

This runs once per digital frame

Write the digital output at the specified frame

Bela and Heavy

- https://docs.google.com/presentation/d/1DLCDUgZp0liaQhnO55uhOJ5iymbNMmDqPC_JyfTkAaE/
- Nice URL, uh?

Stay tuned! Join the announcement list at

http://bela.io

Xenomai remarks

- scheduler can preempt non-preemptable kernel operations
- audio-thread can be set at a higher priority than the Kernel
- mode switches into kernel mode need to be avoided in the audio thread:
  - disk I/O
  - socket
  - printf
  - pthread
  - available.notify_one(); triggers a mode switch