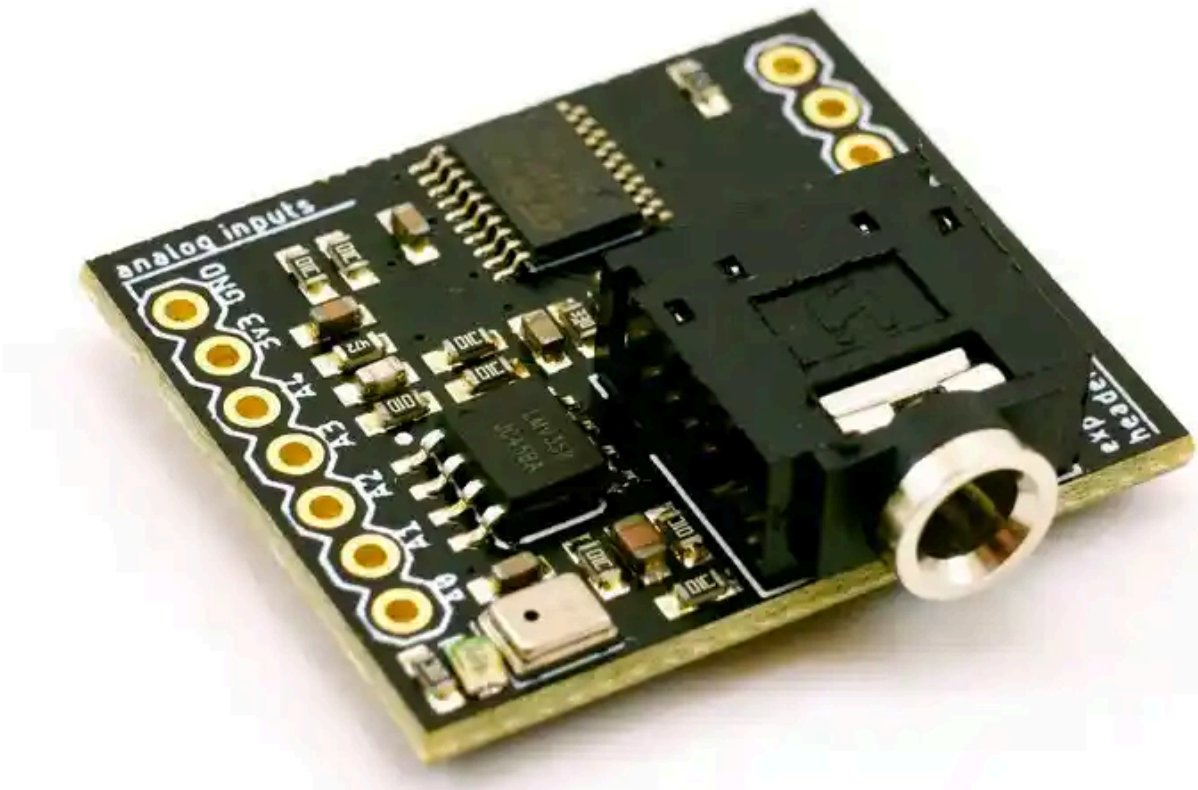
Sensor Expansion Board

With the Sensor Expansion Board (product page), your Pixelblaze (or other microcontroller) can react to sound, ambient light, motion / orientation, and up to 5 other analog sensor signals. Sound from the built-in mic or the line-in jack is transformed into a frequency spectrum.

On this page:

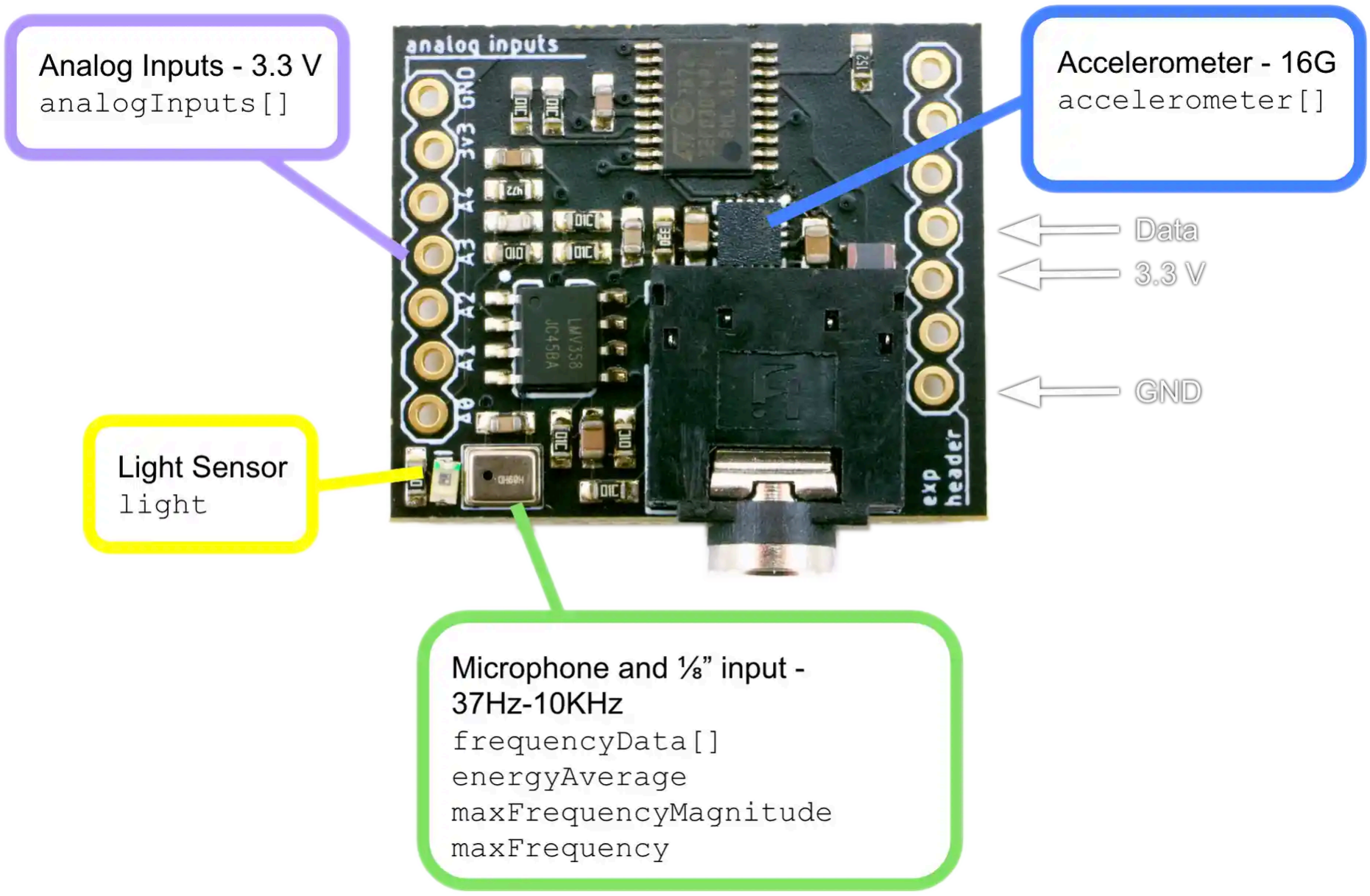


Example applications:

- Sound reactive patterns (for example in this Christmas tree)
- Adjusting brightness to ambient light levels, as in a chandelier
- Objects that react to their orientation (for example the Glow Flow)
- Vehicles that detect braking (for example, LEDxperts Weped)
- Interactive projects detecting people using the five ADC channels via infrared or ultrasonic sensors
- Costumes that react to music or speech
- Connecting up to 5 buttons, switches, or rotary controls (potentiometers).

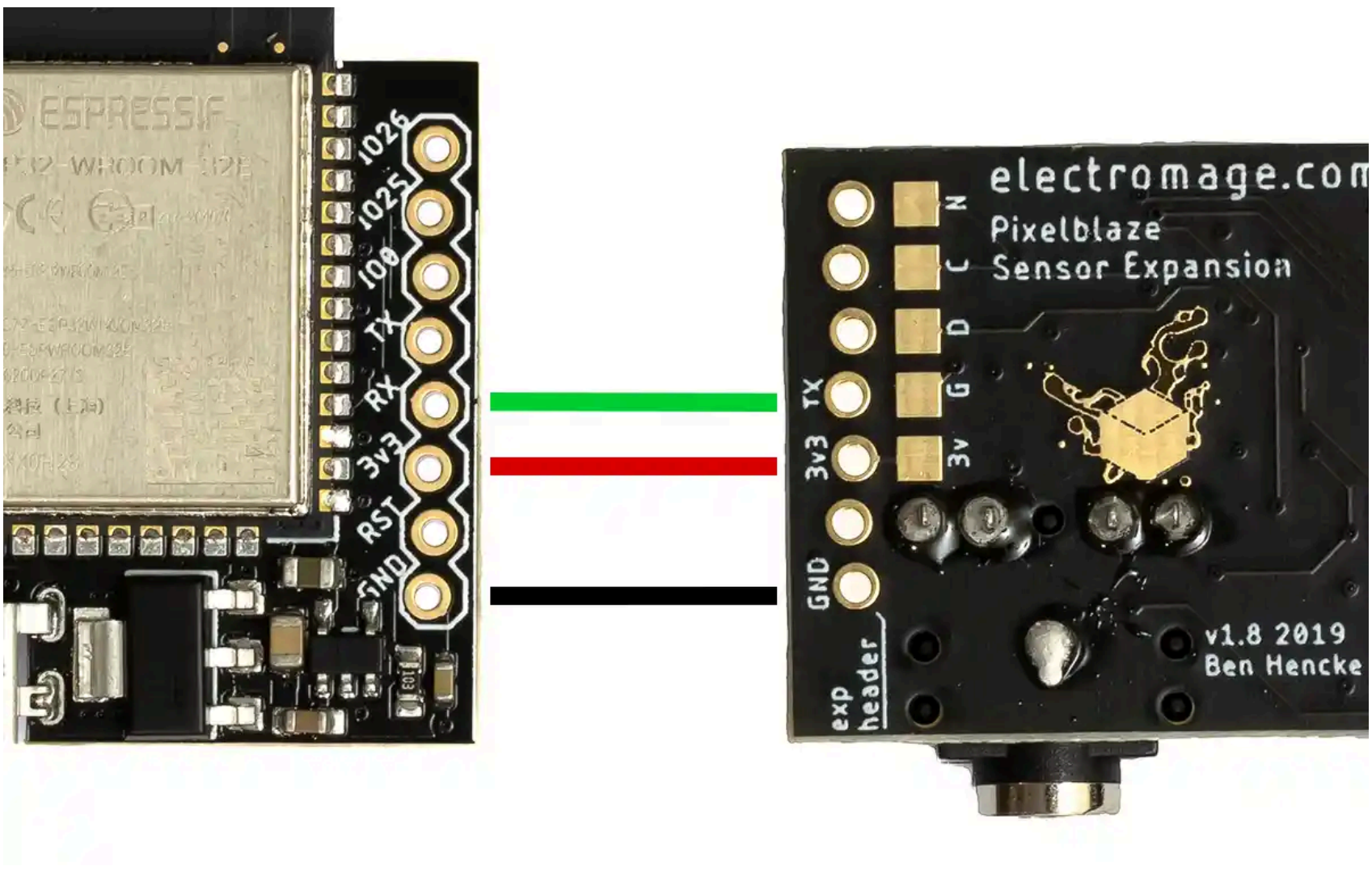
Features

- Sound
  - Built-in microphone and 1/8″ (3.5mm) line-in jack
  - Average loudness value
  - FFT spectrum with 32 frequency bins
  - Strongest detected frequency with 39 Hz accuracy
- 3-axis accelerometer, ±16 G per axis
- Light sensor, from darkness to daylight
- 5 analog inputs, 0-3.3 V 12 bit ADC
- New readings are available in Pixelblaze ~40 times per second
- Readings are normalized to a 0..1 range and made available to your pattern as variables so you can focus on using the data instead of processing or formatting it.
- Open hardware (schematics) with documented serial protocol at 115200 baud
- Library code for use with Arduino, Teensy, or other microcontrollers



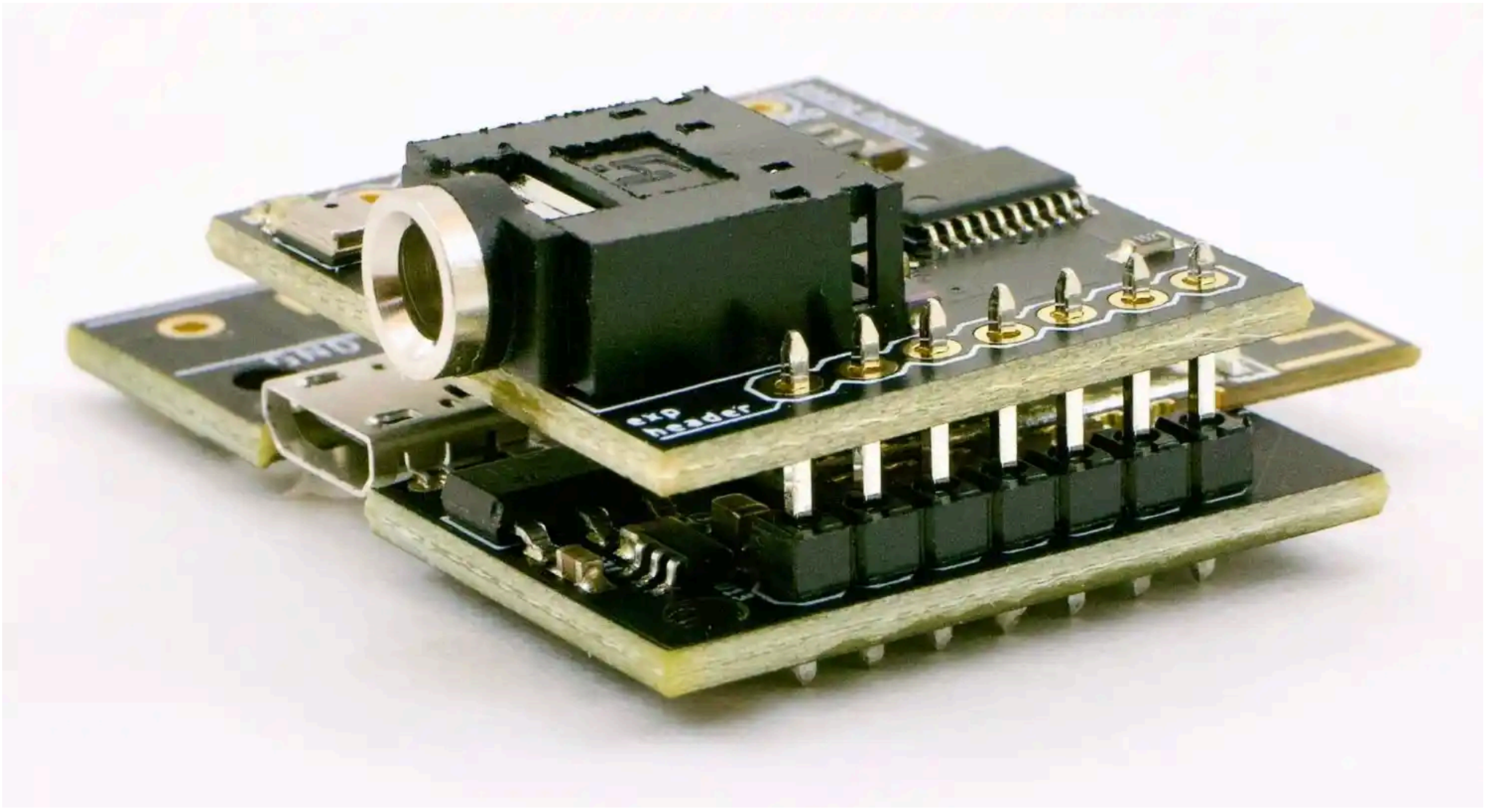
Getting started

To connect the sensor expansion board, you can either solder pluggable pin headers, solder it directly, or run some wires. Only three of the Pixelblaze expansion port pins are actually used with the sensor board: GND, 3v3, and RX (sensor TX).

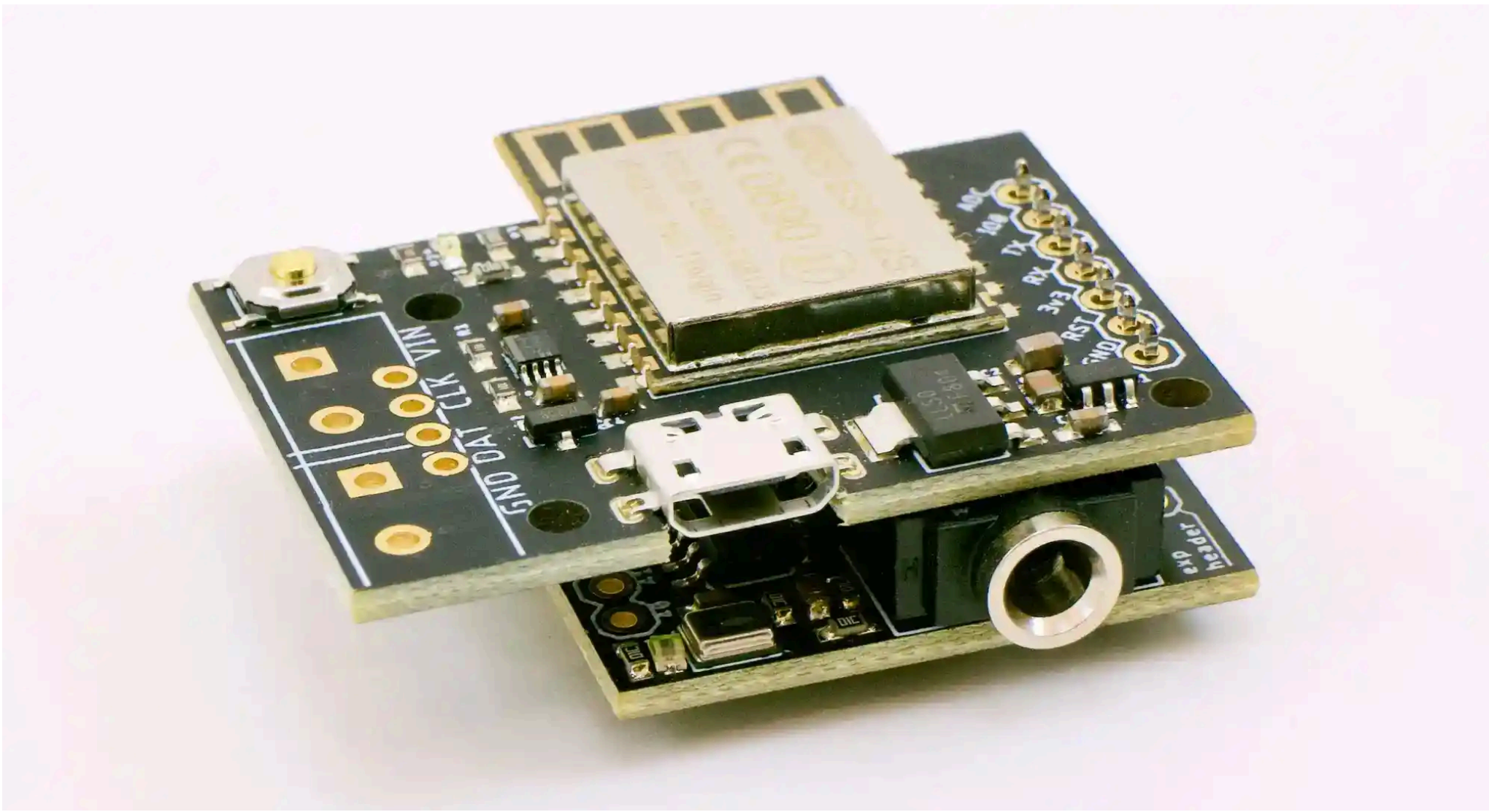


Align the GND pins when connecting a Sensor Expansion Board. On v3s, the new IO26 is left unconnected. Only 3 connections are necessary:GND (Ground, or 0 V), 3v3 (Positive 3.3 V), and TX (Transmit serial data) (expansion) -> RX (Receive serial data) (Pixelblaze). The rest are not used but will not harm anything if connected.

The sensor expansion board can sandwich on top of or below a Pixelblaze. If you solder it on top, make sure there's some clearance between the sensor board and the silver WiFi shield.



Sensor expansion stacked above a Pixelblaze



Sensor expansion stacked below a Pixelblaze

You can tell it's working in the Pixelblaze web app. The status area (top right of the page) will update when a sensor board is detected.

Not detected	Detected
Exp:-	Exp:SB1.0

Using the Pixelblaze sensor board with other microcontrollers

Be sure to power the sensor board with 3.3 V - it's not 5 V tolerant.

Have a look at the schematics and source code, and consider using or referencing the Arduino/Teensy Library code.

Sending output to multiple controllers

The Sensor Expansion Board's TX pin is a serial output that can be connected to any number of Pixelblazes (or other microcontrollers). This way they can all consume the same sensor data. This could be used to have one knob affect patterns on multiple controllers at the same time, or to synchronize sound-reactive patterns across controllers.

This approach requires at least 2 physical wires running to each controller: The sensor board's TX and GND to all controllers' RX and GND.

To distribute sensor data wirelessly using cheap ESP8266s, see the awesome project contributed by community member ZRanger1:

One Sensor Board to Multiple Pixelblazes

Code

Access the sensor data by declaring the following variables when a sensor board is connected:

- `frequencyData[]` - 32 element array with frequency magnitude data
- `energyAverage` - total audio volume
- `maxFrequency` and `maxFrequencyMagnitude` - strongest tones
- `accelerometer[]` - 3 element array with [x, y, z]
- `analogInputs[]` - 5 element array containing 0-1 values for voltages in 0-3.3 V on pins A0-A4
- `light`

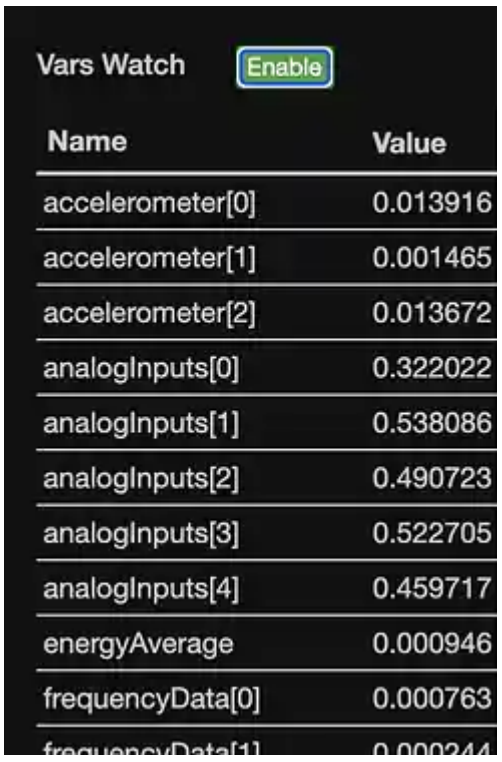
Each of these can be accessed in a pattern by using the `export var` syntax, with optional defaults if the board is not connected. With a sensor expansion board connected, consume an individual reading like this:

```
export var frequencyData

// Read the sound energy level around 237Hz,
//    near middle C
middleCMagnitude = frequencyData[7]
```

Or access all available sensor data with a pattern like this:

```
export var frequencyData
export var energyAverage
export var maxFrequencyMagnitude
export var maxFrequency
export var accelerometer
export var analogInputs
export var light

export function render(index) {}
```

You can inspect the data it's sending by enabling the Vars Watch section of the Edit tab. You should see the values updating live based on what it senses:



Setting defaults and detecting the board

If a pattern can still function without a sensor reading, it's good practice to set a reasonable default for when the sensor board isn't connected, or for sharing patterns with others who don't have one.

```
// Set `light` to 0.5 if the sensr board doesn't exist.
// Otherwise light's 0.5 value is overwritten
export var light = 0.5

// Equivalently:
export var light
light = 0.5
```

To detect the board, you can initialize a specific sensor value to an impossible value. Later in your pattern, you can test for this value and branch appropriately:

```
// It's important to init these reserved-word
// sensor arrays' names to the correct size
export var frequencyData = array(32)
// Start with an impossible value for data from the SB
frequencyData[0] = -1

export function beforeRender(delta) {
  if (frequencyData[0] == -1) {
    // Code for when no sensor board is connected
  } else {
    // Code for when the sensor board *is* connected
  }
}
```

The important and safe path is that if you create an array it should be the right size. E.g. for `frequencyData[]`, it should be of size 32; `accelerometer` should have 3 elements, and `analogInputs` should have 5.

The sensor board variables and `pixelCount` are all written just before each `beforeRender()` call for each animation frame.

Sensors

Sound and frequency spectrum

The built-in microphone is designed to work in loud environments. The 1/8″ (3.5mm) line-in input will disconnect the built-in microphone when something is plugged in. The left and right channels are summed.

In your code:

`frequencyData[]`
: 32 element array with frequency magnitude data ranging from 37.5 Hz to 10 kHz

`energyAverage`
: Total audio volume

`maxFrequency` and **`maxFrequencyMagnitude`**
: Detects the strongest tones with resolution of about 39 Hz

Many sound-reactive patterns look best when using an AGC (automatic gain control) so that patterns remain lively whether reacting to soft conversations or loud music. See the example pattern named "*sound - spectromatrix optim*" for an example implementing an AGC using a PI controller.

Range	12 Hz - 10 KHz
Resolution	`maxFrequency`: 39 Hz `frequencyData[]`: 32 bins, see below
Example values	

Input	`maxFrequencyMagnitude` `frequencyData[bin]`	`energyAverage`
Near silence @ 29 dBA	`0.0002`	`0.0002`
Sine middle C @ 64 dBA	`0.0023`	`0.0007`
White noise @ 80 dBA	-	`0.0021`
Loud music @ 108 dBA	-	`0.06`

Sensor	SPW2430 MEMS microphone

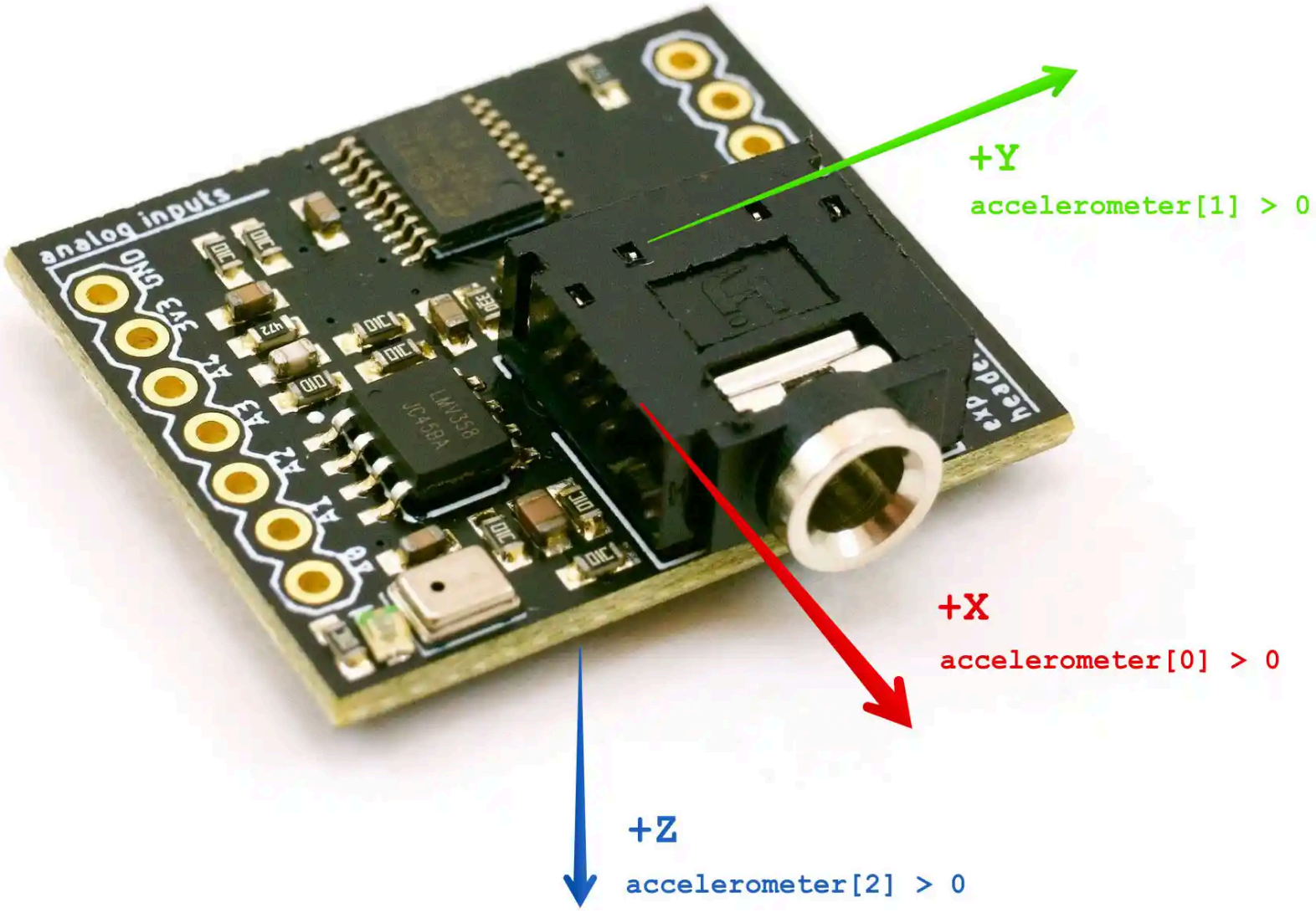
Frequency bins

`frequencyData[index]` - Center frequency (Hz)

0	37.5	8	312	16	1.37K	24	4.10K
1	50	9	391	17	1.56K	25	4.65K
2	75	10	469	18	1.80K	26	5.31K

Frequency bins							
`frequencyData[index]` - Center frequency (Hz)							
3	100	11	586	19	2.07K	27	6.02K
4	125	12	703	20	2.38K	28	6.84K
5	163	13	859	21	2.73K	29	7.77K
6	195	14	976	22	3.12K	30	8.79K
7	234	15	1.17K	23	3.59K	31	9.96K

Accelerometer



Range	±16 g
Resolution	12 mg per digit (sign + 12 bits)

Example values	

Input	`accelerometer[]` `[x:0,y:1,z:2]`
0 g Falling	`0.0`
1 g Gravity	`0.020`
3 g Shaking by hand	`0.057`
12 g Shock by flicking	`0.243`

Sensor	LIS3DH

Example code: Shake to switch modes

```
// Shake to switch modes

export var accelerometer  // Enable the accelerometer
mode = debounce = 0

export function beforeRender(delta) {
  // 3D vector sum of x, y, and z acceleration
  totalAcceleration = sqrt(
    accelerometer[0] * accelerometer[0] +
    accelerometer[1] * accelerometer[1] +
    accelerometer[2] * accelerometer[2]
  )

  debounce = clamp(debounce + delta, 0, 2000) // Prevent overflow

  // Cycle mode if sensor board is shaken, no more than 1x / sec
  if (debounce > 1000 && totalAcceleration > 0.03) {
    mode = (mode + 1) % 3
    debounce = 0
  }
}

h = s = v = 1
modes = array(3)
modes[0] = (index) => { h = 1 - index / pixelCount / 4; v = 1 }
modes[1] = (index) => { h = 0.5; v = index % 2 }
modes[2] = (index) => { h = wave((index+wave(time(0.04))*5)/pixelCount)*6; v = 0.1+random(wave(time(0.03)))
}

export function render(index) {
  modes[mode](index)
  hsv(h, s, v)
}
```

ADCs: Analog-to-Digital inputs (5)

Range	0 - 3.3 V
Resolution	0.8 mV (12 bits)
Example values	**Input** / `analogInputs[n]` Gnd (0 V) / `0` Vdd (3.3 V) / `0.99...`
Sensor	STM32F030's ADC

Light

Range	Readings ~linear 0 - 2500 lux Values still variable to 100 klx
Resolution	0.7 lux in the linear range (12 bits)
Example values	**Input** / `light` 0 lux Total darkness / `0` 300 lux An office / `0.06` 2500 lux Flashlight aimed at sensor / `0.88` 20 klx Clear day, indirect sunlight / `0.99`
Sensor	ALS-PT19

Note that many light sources have imperceptible flicker or pulsed duty cycle dimming that can result in a higher variance on light readings. Darkness and sunlight have lower noise.

A few different light sources tested produced standard deviation of between 5% and 90% of the mean value. For example, one PWM-dimmed LED flashlight set to half brightness produced random light values between 0.01 and 0.2

The implication is that you may wish to average some readings, or use a PI controller as in the sound examples.

Example code: Match the ambient light level

```
// White light in proportion to the ambient light level

export var light        // Sensor board variable
var avgLight            // Light value to use in a pattern

var sampleSize = 50
var samples = array(sampleSize)
var sampleIdx = 0
var samplesTotal = 0

// Circular buffer
function runningAverage(sample) {
  samplesTotal -= samples[sampleIdx] // Remove the oldest value
  samplesTotal += sample             // Add this value
  samples[sampleIdx] = sample        // Store this value
  sampleIdx = (sampleIdx + 1) % sampleSize // Roll the index
  return samplesTotal / sampleSize   // Average = total / count
}

deltaAccumlator = 0
export function beforeRender(delta) {
  deltaAccumlator += delta
  // Sample light readings no faster than every 25ms
  // as the sensor board sends readings at ~40 Hz
  if (deltaAccumlator > 25) {
    deltaAccumlator = 0
    // Stores sample and return average
    avgLight = runningAverage(light)
  }
}

export function render(index) {
  // Minimum inensity in total darkness
  v = clamp(avgLight, 0.01, 1)
  hsv(0, 0, v)
}
```

Buy the Sensor Expansion Board