```
/*
Code for the firmware of the Sonic Mudfish

NOTE: THIS CODE WILL NOT RUN IN SUPERCOLLIDER IDE, AND MUST BE RUN
FROM THE BELA IDE with a Bela attached

See github repo for schematic
*/

s = Server.default;

s.options.numAnalogInChannels = 8; // can only be 2, 4 or 8
s.options.numAnalogOutChannels = 2;
s.options.numDigitalChannels = 16;
s.options.maxLogins = 4;          // set max number of clients

s.options.blockSize = 1024;
s.options.numInputBusChannels = 2;
s.options.numOutputBusChannels = 2;

s.waitForBoot{

// BUS ALLOCATION
--------------------------------------------------------------------------------------------------------------------------

~synBus = Bus.audio(s, 2);     // for modeSwitcher

~derive1 = Bus.audio(s, 1); // ampsig
~derive2 = Bus.audio(s, 1); // pitchSig
~derive3 = Bus.audio(s, 1); // fft flat
~derive4 = Bus.audio(s, 1); // fft bright

~n1out = Bus.audio(s, 1); // output of n1, data
~n1AudioOut = Bus.audio(s, 1); // output of n1, audio

~n2out = Bus.audio(s, 1); // output of n2, data
~n2AudioOut = Bus.audio(s, 1); // output of n2, audio

~n3out = Bus.audio(s, 1); // output of n3, data
```

```
~n3AudioOut = Bus.audio(s, 1);

~n4out = Bus.audio(s, 1); // output of n4, data
~n4AudioOut = Bus.audio(s, 1); // output of n4 audio

~n5out = Bus.audio(s, 1); // output of n5, data
~n5AudioOut = Bus.audio(s, 1); // output of n5 audio

~n6out = Bus.audio(s, 1); // output of n6, data
~n6AudioOut = Bus.audio(s, 1); // output of n6, audio

~updateBus = Bus.audio(s, 1);

s.sync;

// LED DEF
---------------------------------------------------------------------------------------------------------------
-----------------

SynthDef(\LEDon, {
                    DigitalOut.ar(5, 1);
          }).add;

s.sync;

// SYNTHDEFS
---------------------------------------------------------------------------------------------------------------
---------

SynthDef.new(\modeSwitcher, {
arg t_recalibrate=0.0;
var button = DigitalIn.ar(8);
var altButton = button.linlin(0, 1, 1, 0);
var sig;

var i2c_bus = 1;
var i2c_address_1 = 0x20;

var noiseThreshold = 0.02; // float: 0-0.0625, with 0.0625 being the highest noise thresh
var prescalerOpt = 2; // int: 1-8 with 1 being the highest sensitivity
```

```
var touch1, touch2, touch3, touch4, touch5, touchsigs;
var saw, saw1, saw2, saw3, saw4, saw5, lpf;
var centroidsBar, centroidsSquare;
var out, ping;
var range = AnalogIn.ar(1).exprange(100, 2500);

centroidsBar = TrillCentroids.kr(i2c_bus, i2c_address_1, noiseThreshold, prescalerOpt,
t_recalibrate);

touch1 = SinOsc.ar((centroidsBar[1]*range), mul: centroidsBar[2].lag(10));
touch2 = SinOsc.ar((centroidsBar[3]*range), mul: centroidsBar[4].lag(10));
touch3 = SinOsc.ar((centroidsBar[5]*range), mul: centroidsBar[6].lag(10));
touch4 = SinOsc.ar((centroidsBar[7]*range), mul: centroidsBar[8].lag(10));
touch5 = SinOsc.ar((centroidsBar[9]*range), mul: centroidsBar[10].lag(10));

touchsigs = touch1 + touch2 + touch3 + touch4 + touch5 * 0.1;

Out.ar(~synBus, SoundIn.ar(0)!2 * altButton.lag(1));
Out.ar(~synBus,  touchsigs!2 * button.lag(1));

}).add;

SynthDef.new(\listener, { //Test Audio
	var input, ampsig, sound, pitchSig, chain, flat, bright;

//	input = SoundIn.ar(0);
	input = InFeedback.ar(~synBus);

// Amp Derivation
	ampsig = Amplitude.ar(input);
	ampsig = ampsig * 1;
   Out.ar(~derive1, ampsig); //change array when appliacble
   SendReply.kr(Impulse.kr(10), '/port2', [ampsig]);

// Pitch Derivation
   pitchSig = ZeroCrossing.ar(input);
	pitchSig = pitchSig/1000;
	Out.ar(~derive2, pitchSig); //change array when appliacble

// FFT Derivations
```

```
chain = FFT(LocalBuf(512), input);
flat = SpecFlatness.kr(chain); //a power spectrum's geometric mean divided by its arithmetic
mean.
Out.ar(~derive3, K2A.ar(flat));


bright = SpecCentroid.kr(chain); //the perceptual brightness of a signal
Out.ar(~derive4, K2A.ar(bright));
}).add;

// PERCEPTRON SynthDEFS
-----------------------------------------------------------------------------------------------------------------------
---------

// Input nodes
-----------------------------------------------------------------------------------------------------------------------
-------------------

SynthDef.new(\n1, {
	arg in1 = 1, in2 = 1, w1 = 0.1, w2 = 0.1, b = 1, bW = 0.1;
	var y, x, update, sig, n6AudioFeedback, scaler;

	sig = InFeedback.ar(~synBus, 2);
   update = InFeedback.ar(~updateBus, 1);
   scaler = AnalogIn.ar(0).linexp(0.0, 1.0, 0.1, 50.0);
	n6AudioFeedback = InFeedback.ar(~n6AudioOut, 1);
	n6AudioFeedback = n6AudioFeedback * scaler;
	in1 = In.ar(~derive1, 1);
	in2 = In.ar(~derive2, 1);


	w1 = w1 + update * in1; // delta rule update
	w2 = w2 + update * in2; // delta rule update

	y = (in1 * w1) + (in2 * w2) + (b * bW); // weighted sums
	x = 1 / (1 + y.neg.exp); // sigmoid computation

	sig = CombL.ar(sig + n6AudioFeedback * 0.5, 2, AnalogIn.ar(3).range(0.1, 1.7).lag(1) +
x.lag(1), 5);
```

```
        Out.ar(0, sig);
        Out.ar(~n1AudioOut, sig);
        Out.ar(~n1out, x);
}).add;

SynthDef.new(\n2, {
        arg in1 = 1, in2 = 1, w1 = 0.1, w2 = 0.1, b = 1, bW = 0.1;
        var y, x, update, sig, n6AudioFeedback, scaler;

        sig = In.ar(~synBus, 2);
   update = InFeedback.ar(~updateBus, 1);
   scaler = AnalogIn.ar(0).linexp(0.0, 1.0, 0.1, 50.0);
        n6AudioFeedback = InFeedback.ar(~n6AudioOut, 1);
        n6AudioFeedback = n6AudioFeedback * scaler;

        in1 = In.ar(~derive3, 1);
        in2 = In.ar(~derive4, 1) / 2000;
        n6AudioFeedback = In.ar(~n6AudioOut, 1);

        w1 = w1 + update * in1; // delta rule update
        w2 = w2 + update * in2; // delta rule update

        y = (in1 * w1) + (in2 * w2) + (b * bW); // weighted sums
        x = 1 / (1 + y.neg.exp); // sigmoid computation

        sig = CombL.ar(sig + n6AudioFeedback * 0.5, 2, AnalogIn.ar(3).range(0.1, 1.7).lag(1) +
x.lag(1), 5);

        Out.ar(~n2AudioOut, sig);
        Out.ar(0, sig);
        Out.ar(~n2out, x);
}).add;

// hidden layers
-----------------------------------------------------------------------------------------------------------------
-----------

SynthDef.new(\n3, {
        arg in1 = 1, in2 = 1, w1 = 0.3, w2 = 0.3, b = 1, bW = 0.3;
```

```
        var y, x, update, sig, sig1, sig2;
        var presVol = AnalogIn.ar(2).exprange(0.0001, 1.0);

        sig = In.ar(~n1AudioOut, 1) + In.ar(~n2AudioOut, 1) * 0.5;

    update = InFeedback.ar(~updateBus, 1);
        in1 = In.ar(~n1out, 1);
        in2 = In.ar(~n2out, 1) / 2000;

        w1 = w1 + update * in1; // delta rule update
        w2 = w2 + update * in2; // delta rule update

        y = (in1 * w1) + (in2 * w2) + (b * bW); // weighted sums
        x = 1 / (1 + y.neg.exp); // sigmoid computation

        sig = CombL.ar(sig, 2, AnalogIn.ar(3).range(0.1, 1.7).lag(1) + x.lag(1), 5);
        sig = sig*presVol;
        Out.ar(~n3AudioOut, sig);
        Out.ar(0, sig);
        Out.ar(~n3out, x);
}).add;

SynthDef.new(\n4, {
        arg in1 = 1, in2 = 1, w1 = 0.4, w2 = 0.4, b = 1, bW = 0.4;
        var y, x, update, sig;


    update = InFeedback.ar(~updateBus, 1);
        in1 = In.ar(~n1out, 1);
        in2 = In.ar(~n2out, 1) / 2000;

        sig = In.ar(~n1AudioOut, 1) + In.ar(~n2AudioOut, 1) * 0.5;

        w1 = w1 + update * in1; // delta rule update
        w2 = w2 + update * in2; // delta rule update
        y = (in1 * w1) + (in2 * w2) + (b * bW); // weighted sums
        x = 1 / (1 + y.neg.exp); // sigmoid computation

        sig = PitchShift.ar(sig, 0.2, 1.50 * x);
```

```
        Out.ar(~n4AudioOut, sig);
        Out.ar(~n4out, x);
}).add;

SynthDef.new(\n5, {
        arg in1 = 1, in2 = 1, w1 = 0.5, w2 = 0.5, b = 1, bW = 0.5;
        var y, x, update, sig;
        var presVol = AnalogIn.ar(2).exprange(0.0001, 1.0);

    update = InFeedback.ar(~updateBus, 1);
        in1 = In.ar(~n1out, 1);
        in2 = In.ar(~n2out, 1) / 2000;
        w1 = w1 + update * in1; // delta rule update
        w2 = w2 + update * in2; // delta rule update
        y = (in1 * w1) + (in2 * w2) + (b * bW); // weighted sums
        x = 1 / (1 + y.neg.exp); // sigmoid computation

        sig = In.ar(~n1AudioOut, 1) + In.ar(~n2AudioOut, 1) * 0.5;
        sig = CombL.ar(sig, 2, AnalogIn.ar(3).range(0.1, 1.7).lag(1) + x.lag(1), 5);

        sig = sig * presVol;

        Out.ar(~n5AudioOut, sig);
        Out.ar(0, sig);
        Out.ar(~n5out, x);
}).add;

// output layer
--------------------------------------------------------------------------------------------------------------------
---------------------------

SynthDef.new(\n6, { // output node
        arg in1 = 1, in2 = 1, in3 = 1, w1 = 0.6, w2 = 0.6, w3 = 0.6, b = 1, bW = 0.6;
        var y, x, dOut, update, rate, sig1, sig2, sig3, sigM, dOffset;

        in1 = In.ar(~n3out, 1);
        in2 = In.ar(~n4out, 1) / 2000;
        in3 = In.ar(~n5out, 1);

        sig1 = In.ar(~n3AudioOut, 1);
```

```
	sig2 = In.ar(~n4AudioOut, 1);
	sig3 = In.ar(~n5AudioOut, 1);

	sigM = sig1 + sig2 + sig3 * 0.2;

	y = (in1 * w1) + (in2 * w2) + (in3 * w3) + (b * bW); // weighted sums
	x = 1 / (1 + y.neg.exp); // sigmoid computation

	dOut = LFNoise0.ar(AnalogIn.ar(5)).range(0.1, 10.0);
	rate = AnalogIn.ar(5).range(0.1, 10.0);
	update = rate * (dOut - x);
	update = update * (SinOsc.ar(AnalogIn(4).range(50, 500), AnalogIn.ar(4).range(0.1,
100)));

	sigM = CombL.ar(sigM, 2, AnalogIn.ar(3).range(0.1, 1.7).lag(1) + x.lag(1), 5);
	Out.ar(~n6AudioOut, sigM); //audio
	Out.ar(0, sigM * 2);
	Out.ar(~n6out, x); //data
	Out.ar(~updateBus, update);
}).add;

// Generative SynthDefs
-------------------------------------------------------------------------------------------------------------------------
---------

SynthDef.new(\sympWinds {
	arg freq = 100, atk = 0.5, seg2 = 0.3, rel = 0.5, cf = 500, rq = 0.001;
	var sig, env, derive1, derive2, sigWave, contr4, dig1, mul, dig2, ana1;
	dig1 = DigitalIn.ar(1);
	dig2 = DigitalIn.ar(6);

	derive1 = In.ar(~derive1) * 30;
	derive2 = In.ar(~derive2);

	sig = PinkNoise.ar(5) * dig2.lag(1);
	sigWave = Saw.ar(1, mul: LFNoise0.kr(1.5).range(5, 10)) * dig1.lag(10) * 0.5;

	sig = sig + sigWave * 0.5;
	sig = BPF.ar(sig, cf, rq);
	env = EnvGen.kr(Env(
```

```
            [0, 0.8, 0.5, 0],
            [atk, seg2, rel],
        ), doneAction: 2);
        sig = sig * env * 3;
        sig = sig * env * derive1;
        Out.ar(0, sig!2);
        Out.ar(~synBus, sig);
}).add;

SynthDef.new(\blipBox, {
        arg freq = 220;
        var sig, env, button;
        button = DigitalIn.ar(0);
        sig = Pulse.ar(freq);
        sig = LPF.ar(sig, 400);
            env = EnvGen.kr(Env(
            [0, 0.5, 0, 0.5, 0],
            [0.05, 0.05, 0.05, 0.05],
        ), doneAction: 2);
        sig = sig * env;
        sig = sig * button.lag(10);
        Out.ar(~synBus, sig);
}).add;

// OSC CONTROL
---------------------------------------------------------------------------------------------------------------------------------------

~ampDerive = 1;

OSCdef('controller2', {
        arg msg;
        ~ampDerive = msg[3];
}, '/port2');

// Generative Def pt 2
--------------------------------------------------------------------------------------------------------------------------

SynthDef.new(\cDrum, { // chaos drums
```

```
		arg out = 0, freq = 35, amp = 0.5, atk_tim = 0.01, dkay_tim = 0.2;
		var sig, env, pitch_env, in1, in2, mod, in3, in4;
		var button2 = DigitalIn.ar(7);

		in1 = InFeedback.ar(~n3out, 1);
		in2 = InFeedback.ar(~n5out, 1);
		in3 = InFeedback.ar(~derive4);
		in4 = InFeedback.ar(~derive2);

		env = EnvGen.kr(Env.perc(atk_tim * in1, dkay_tim * in2), doneAction: 2);
		pitch_env = EnvGen.kr(Env([0.5, 1.5 - in2, 0.2], [0.01, 0.2]));

		mod = Pulse.kr(0.1 * in3.range(0.1, 5), mul: in4.range(0.1, 5));
		sig = SinOscFB.ar(freq * pitch_env, mod, mul:5)!2;
		sig = LPF.ar(sig, 2500);

		sig = sig * env;
		sig = sig * 0.01;
		sig = sig * button2;
		Out.ar(~synBus, sig * 1.5);
}).add;

s.sync;

// PBINDS
-----------------------------------------------------------------------------------------------------------
--------------------------

Pbind(
	\instrument, \cDrum,
			\dur, Pwhite(0.5, 5, inf),
		\freq, Pwhite(1, 10000, inf) * Pfunc({~ampDerive})
).play;

s.sync;

Pbind(
	\instrument, \blipBox,
			\dur, Pexprand(0.1, 0.5, inf),
		\freq, Pwhite(5, 50, inf),
```

```
).play;

s.sync;

Pbind(
	\instrument, \sympWinds,
	\dur, Pwhite(0.5, 5, inf),
	\rq, Pwhite(0.001, 0.005, inf),
   \cf, (PdegreeToKey(Prand([0, 1, 2, 3, 4, 5, 6], inf), Array.rand(7, 0, 10), 5) + 60).midicps *
Prand([0.5, 1, 2], inf),
	\atk, 3,
	\sus, 1,
	\rel, 5,
	\addAction, 'addToTail'
).play;

// SYNTH CALLS
------------------------------------------------------------------------------------------------------------------
---------------------------

s.sync;

Synth.new(\n6);

s.sync;

Synth.new(\n3);
Synth.new(\n4);
Synth.new(\n5);

s.sync;

Synth.new(\n1);
Synth.new(\n2);

s.sync;

Synth.new(\listener);

s.sync;
```

```
Synth.new(\modeSwitcher);

s.sync;

a = Synth(\LEDon);

s.sync;

};

// END OF CODE
-----------------------------------------------------------------------------------------------------------------
---------------------------

ServerQuit.add({ 0.exit }); // quit if the button is pressed
```