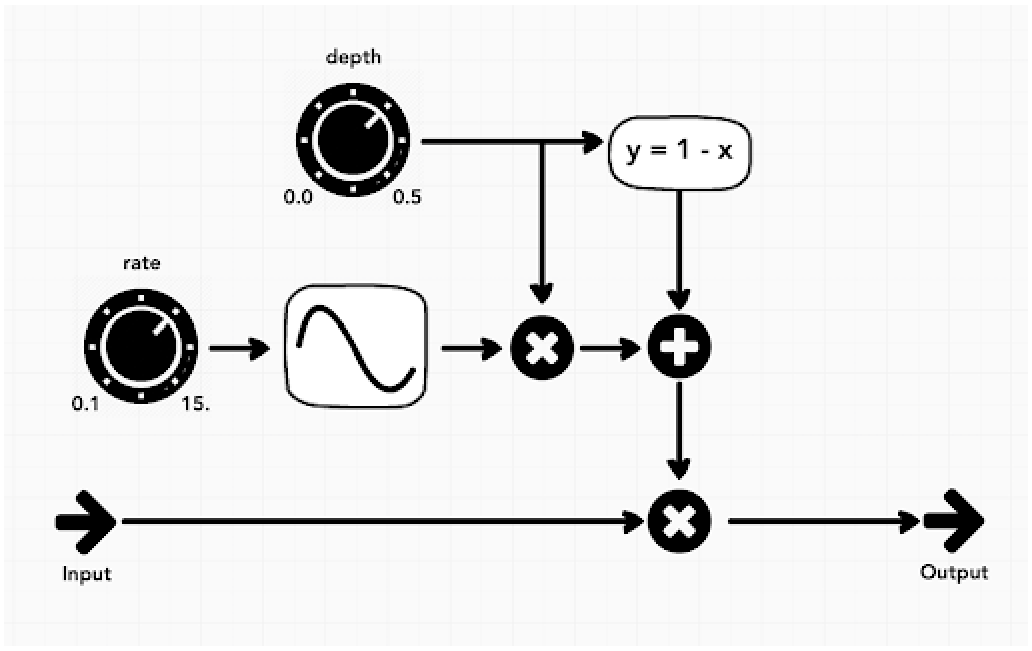
Building a Tremolo Effect in the FX Testing Rig

View the video version of this tutorial at http://electricguitarinnovationlab.org/pedal

V.J. Manzo
vjmanzo@wpi.edu

There are many books that explains digital signal processing (DSP) concepts using figures that look like this:



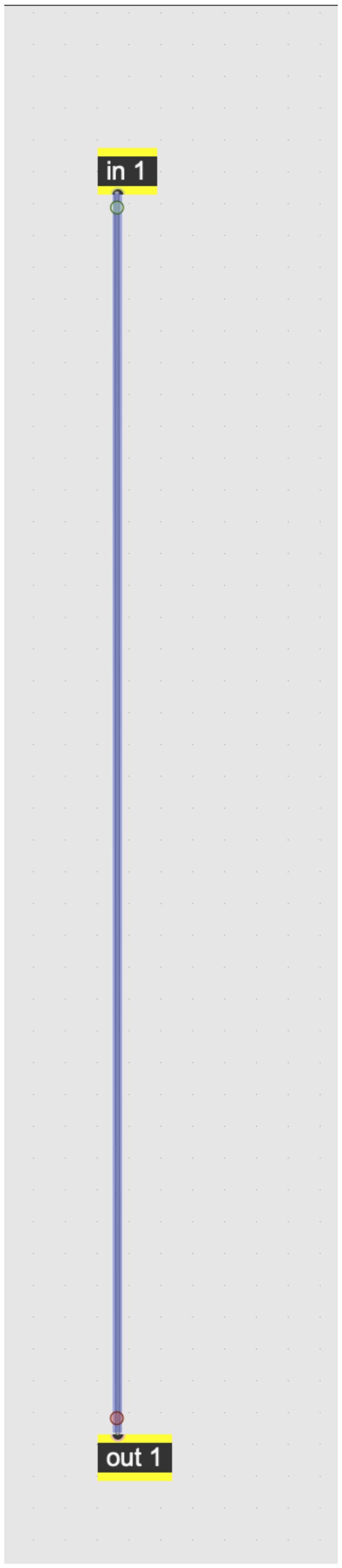
In this tutorial, we will turn the DSP concept in this figure into code that can run on our pedal hardware.

Step 0

We have opened the FX Testing Rig. We'll click the **Default Setup** button from the presets menu in the "main patch"; you should hear audio playback at this point. If you do not, scroll down to check the **Audio Settings** area at the bottom. You can also lower the **Master Output** level here if desired. While we develop our effect, you can click the **Play** and **Stop** buttons to test the effect with audio files from this menu.

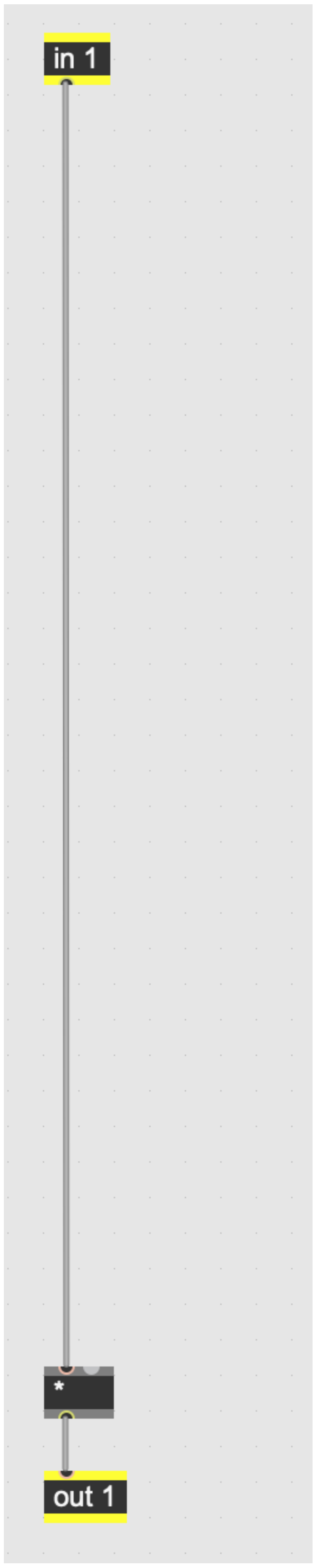
Step 1

We will click the **Open Contents** button to reveal the area of the effect where we'll be building our device; this code will be pushed to our microcontroller later, and we'll refer to this section of code as the "Gen patch". Let's unlock the patch and remove everything else except for the *in* and *out* objects; we want the input signal coming from the *in* object connected directly to the output through the *out* object. If you've accidentally removed all of the objects and no longer hear audio playback, click the *Fix Broken Connections...* button in the main patch and then re-open the contents of the Gen patch. Our effect will be mono, so click the *Mono* button to map the *out 1* object to both the left and right stereo output channels.



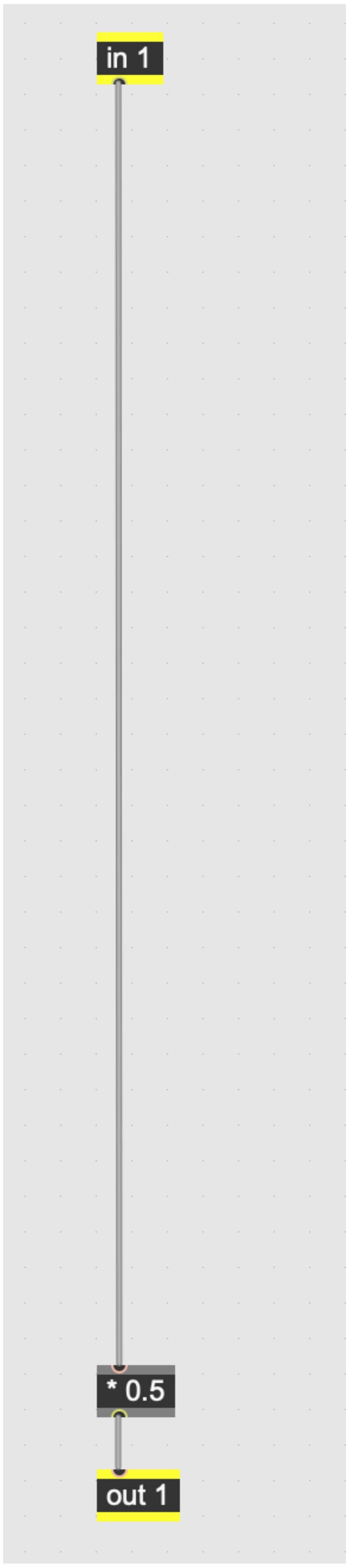
Step 2

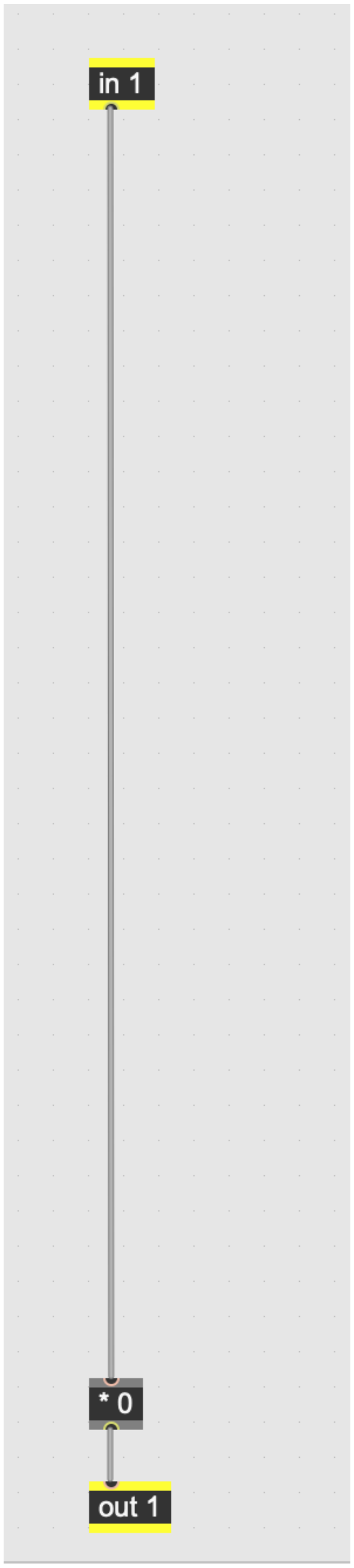
We will add a multiplication object, an asterisk, to change the amplitude of the signal; the maximum amplitude coming from the input will be a value between 0 and 1, so we can think about this multiplier object as a volume adjustment.

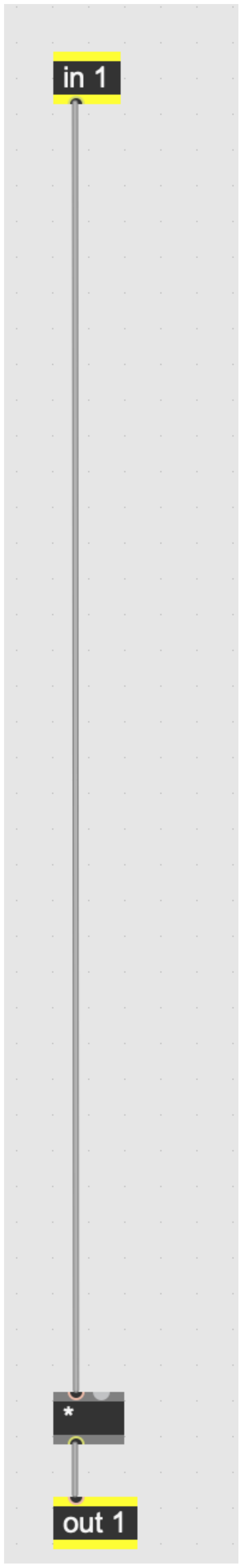


Step 3

If we gave the multiplier the argument .5, it would cut the amplitude of the signal it received in half; if we gave it the argument 0, all signals received will be multiplied by 0 and we wouldn't hear any signal at all! When you add an argument, the option to have that number changed by a control through an additional inlet disappears, so let's remove the argument for the multiplication object.

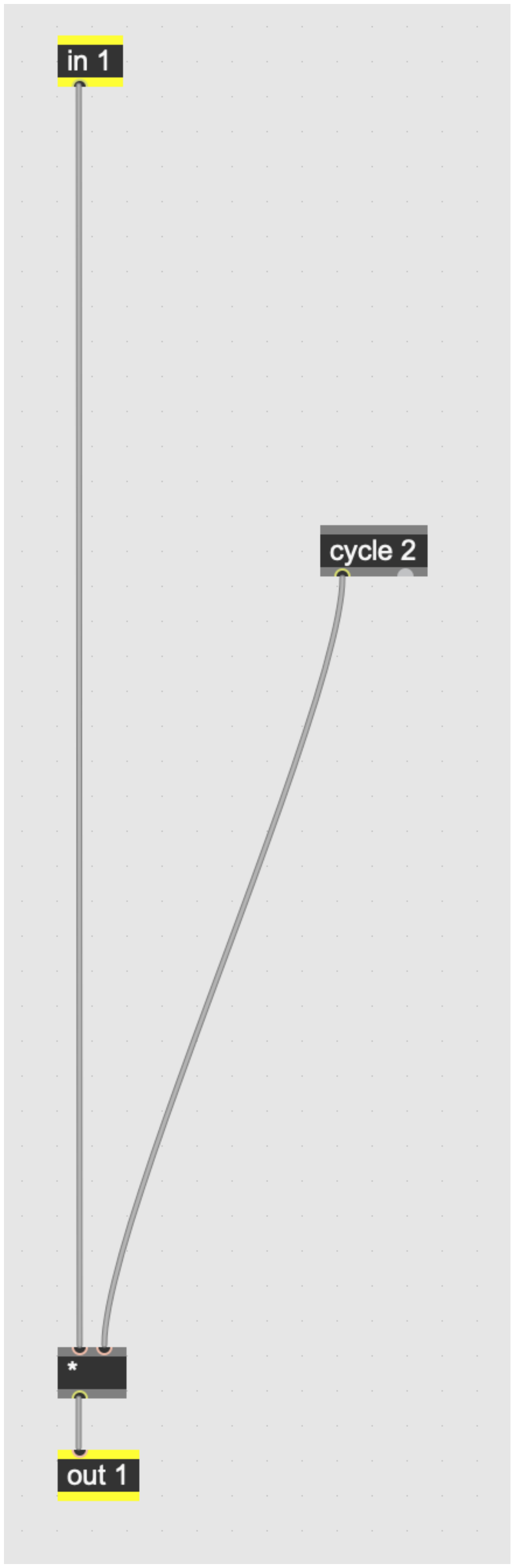






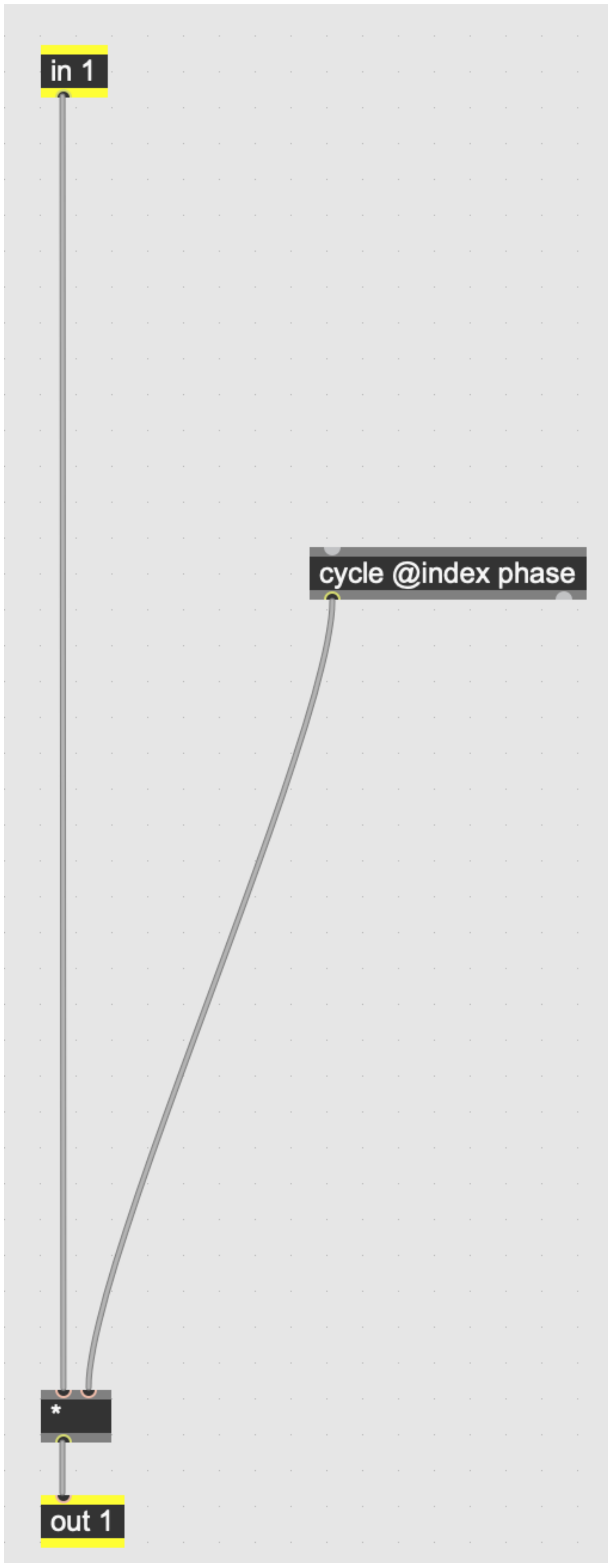
Step 4

To produce a tremolo effect, we want to adjust the amplitude smoothly over time in a sine wave pattern, so we will use a cycle object that transitions between -1 and 1 over some amount of time called the *frequency*. A cycle with the argument 2, for example, will transition from -1 to 1 two times in one second.



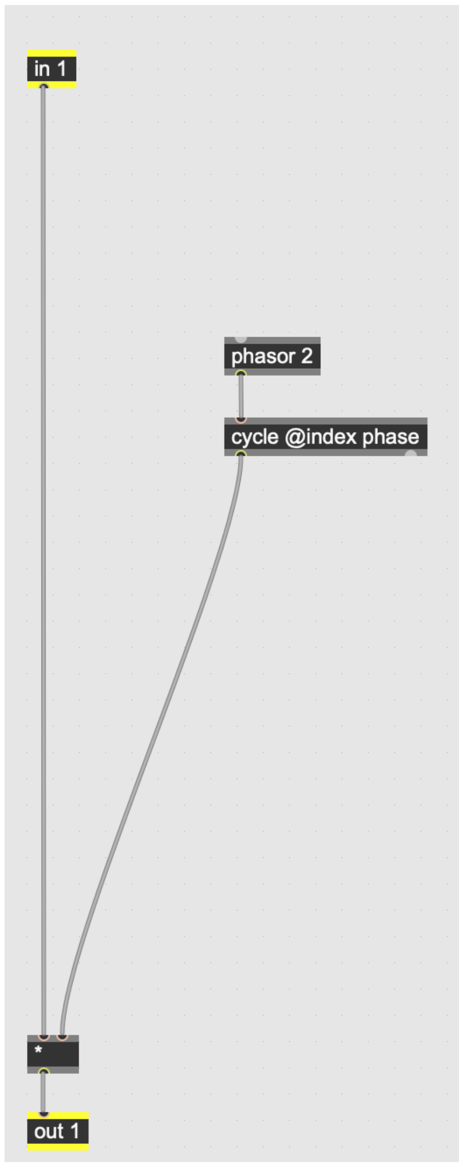
Step 5

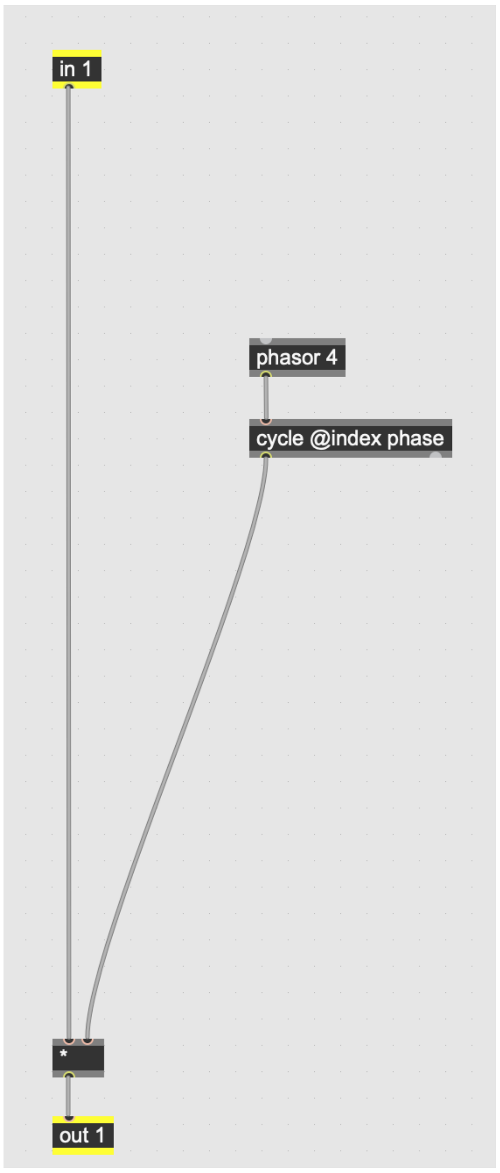
For our needs, however, we don’t want the amplitude to be multiplied by -1 to 1, we only want the signal multiplied by 0 to 1, so we’ll set a mode called *phase* that remaps the input signal range to 0 to 1.

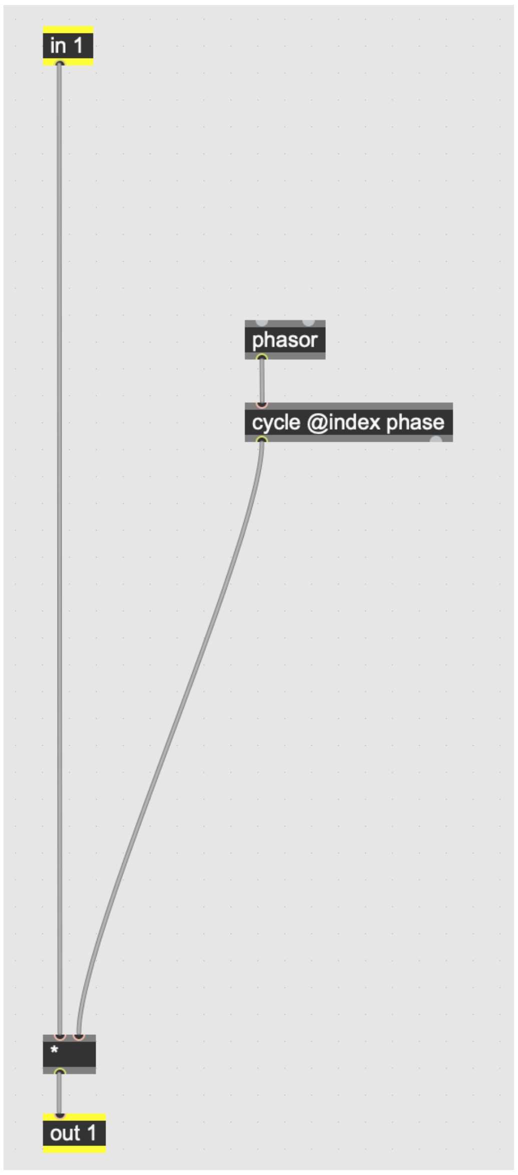


Step 6

The cycle object then needs some information that tells it how frequently it should transition between 0 and 1, so we will use a *phasor* object to output a control signal for the cycle. We can give the *phasor* the argument 2 to set the rate of the *cycle* object back to two, which will get us back to where we were in the previous step. Likewise, we can give *phasor* the argument *4* or *8* or any number we like. Any value above 8 starts to produce an effect beyond what is normally expected of a tremolo. Regardless, it would be useful if we use a hardware potentiometer to control this parameter instead of typing in numbers, so we'll remove the argument for *phasor* for now so we can control that value in another way.

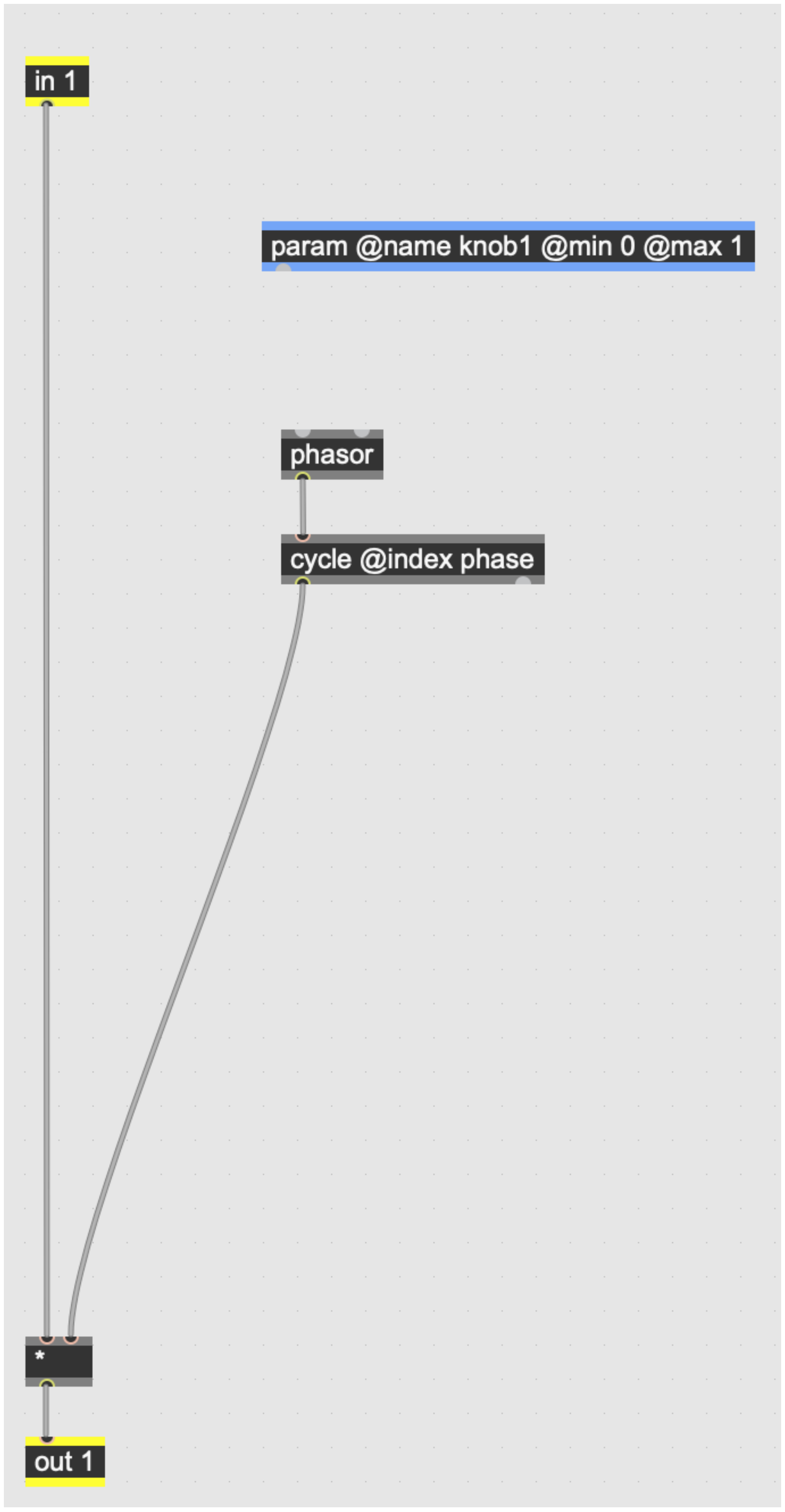






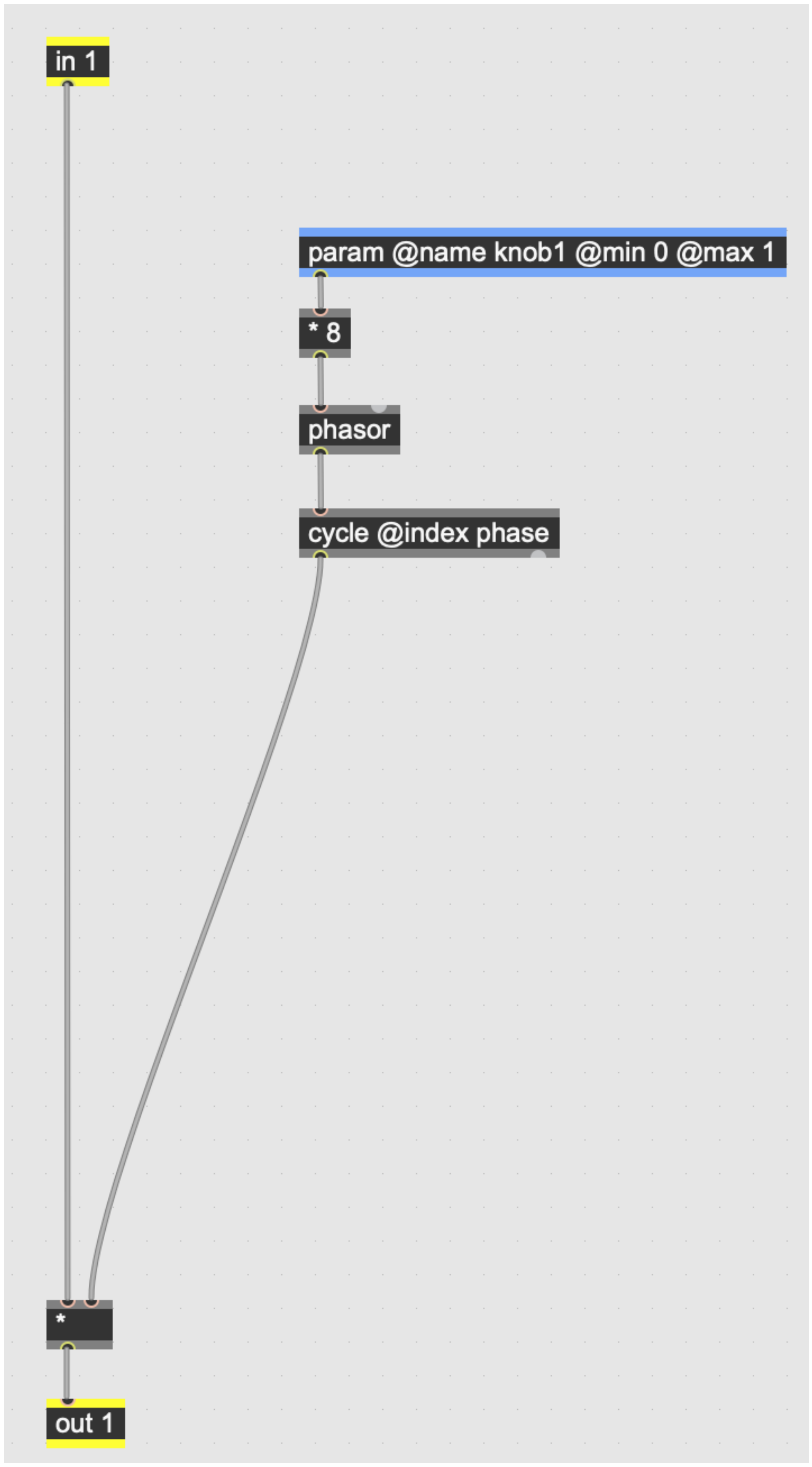
Step 7

We can get data from the knobs and switches connected to our microcontroller by adding a *param* object and specifying some info about that particular control parameter. We'll make a new object called *param* and give it the @name knob1 and specify the @min value we expect to receive and the @max 1 value we expect to receive.



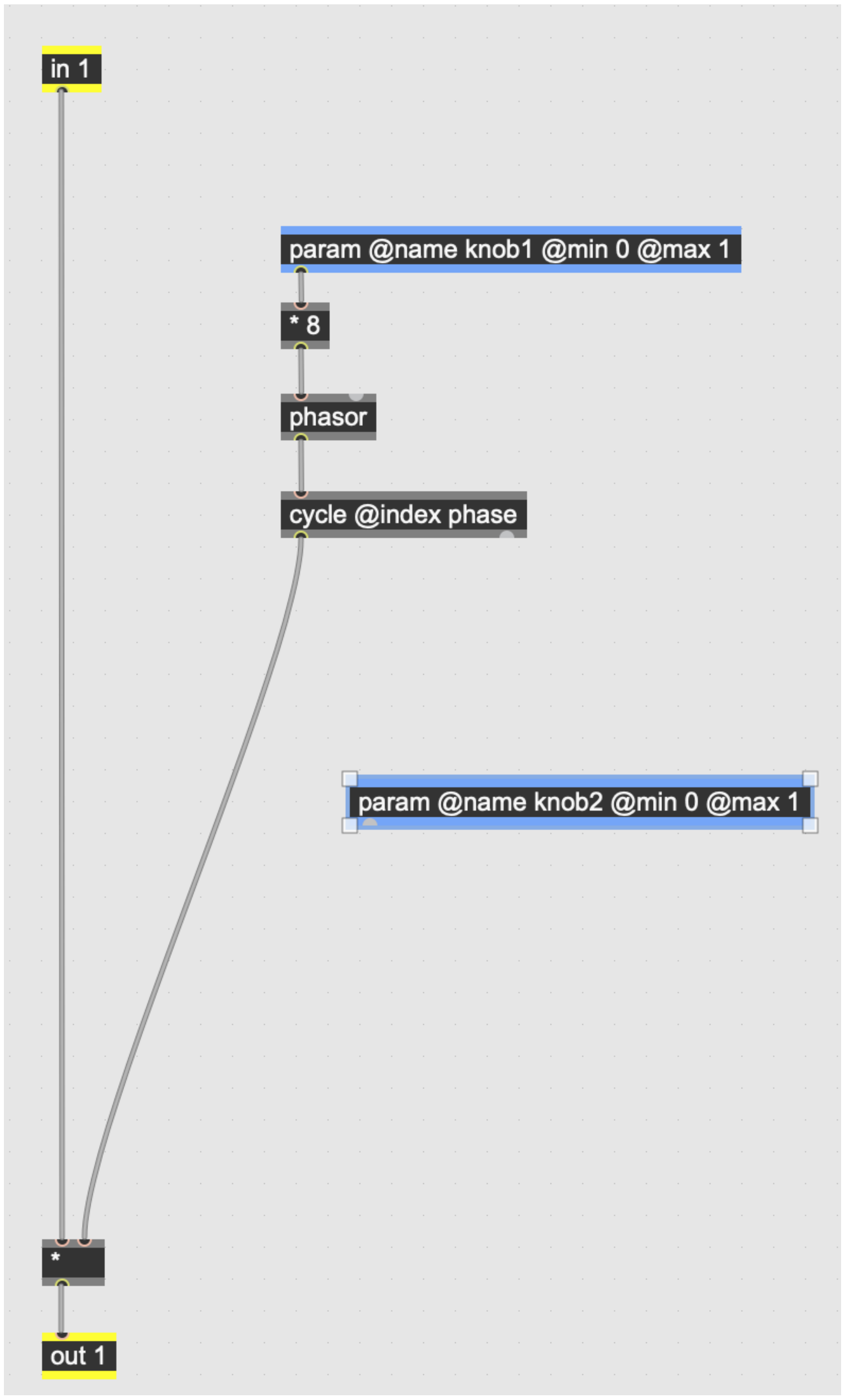
Step 8

In the main patch, there is a knob labeled “Param knob 1”, which will simulate the knob movement we’ll experience later when we load this software onto the microcontroller. Turning the knob in the main patch, and ultimately on our pedal, will output values between 0 and 1, so we’ll multiply those numbers and tell the *phasor* object to run the *cycle* object at that rate.



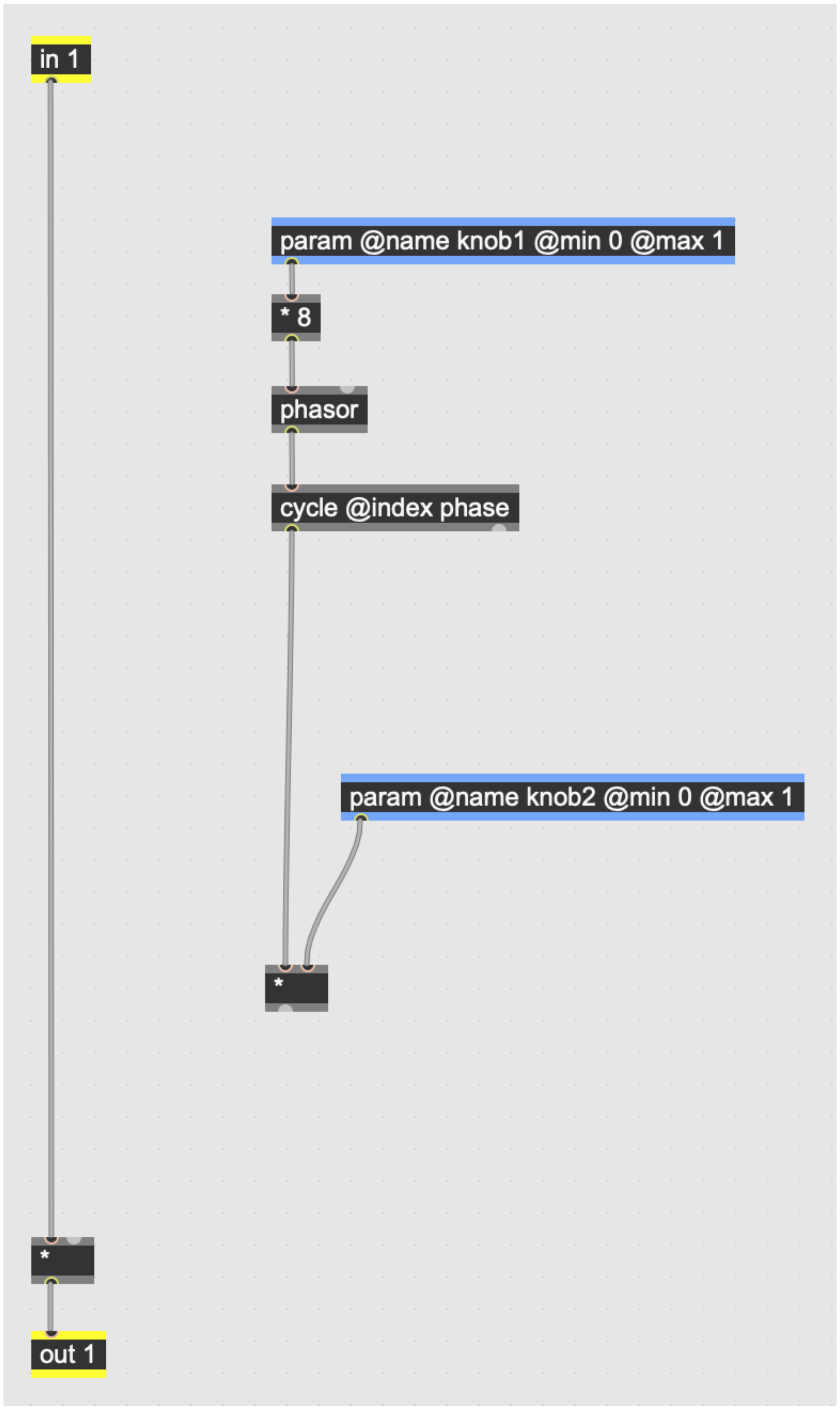
Step 9

As you move the knob in the main patch, floating point numbers between 0 and 1 are sent to the multiplication object, which tells the phasor to tell the cycle how to oscillate! At the rate set by the *phasor*, the amplitude of the input signal will go from 0, where there is no sound heard, to 1, where the original level of the signal is not reduced at all. Right now, we've achieved the tremolo effect, in principle, but it would be useful if we can "shrink" the depth of the tremolo so that the amplitude doesn't have to dip all the way down to nothing (0) and then go all the way up to the full level (1). So we will create another *param* object for knob2 to control the tremolo's "depth".


in 1
param @name knob1 @min 0 @max 1
* 8
phasor
cycle @index phase
param @name knob2 @min 0 @max 1
*
out 1

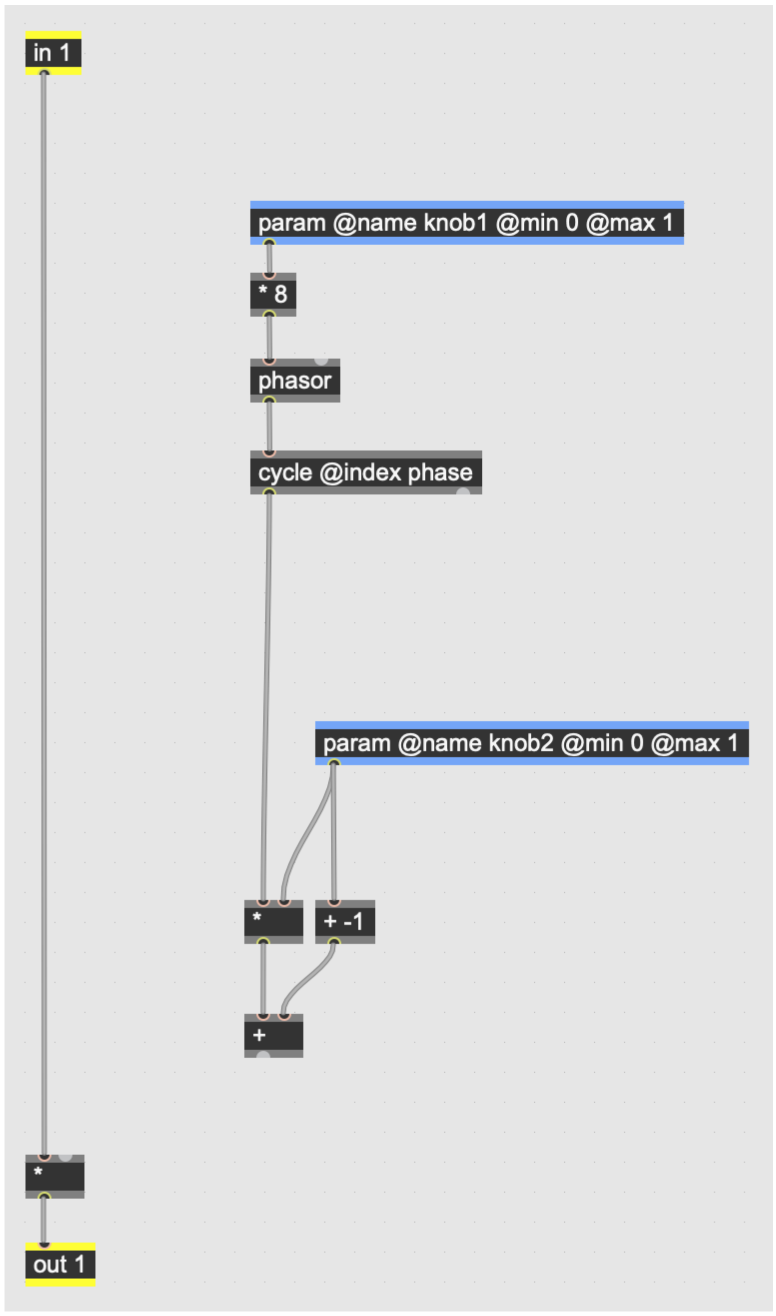
Step 10

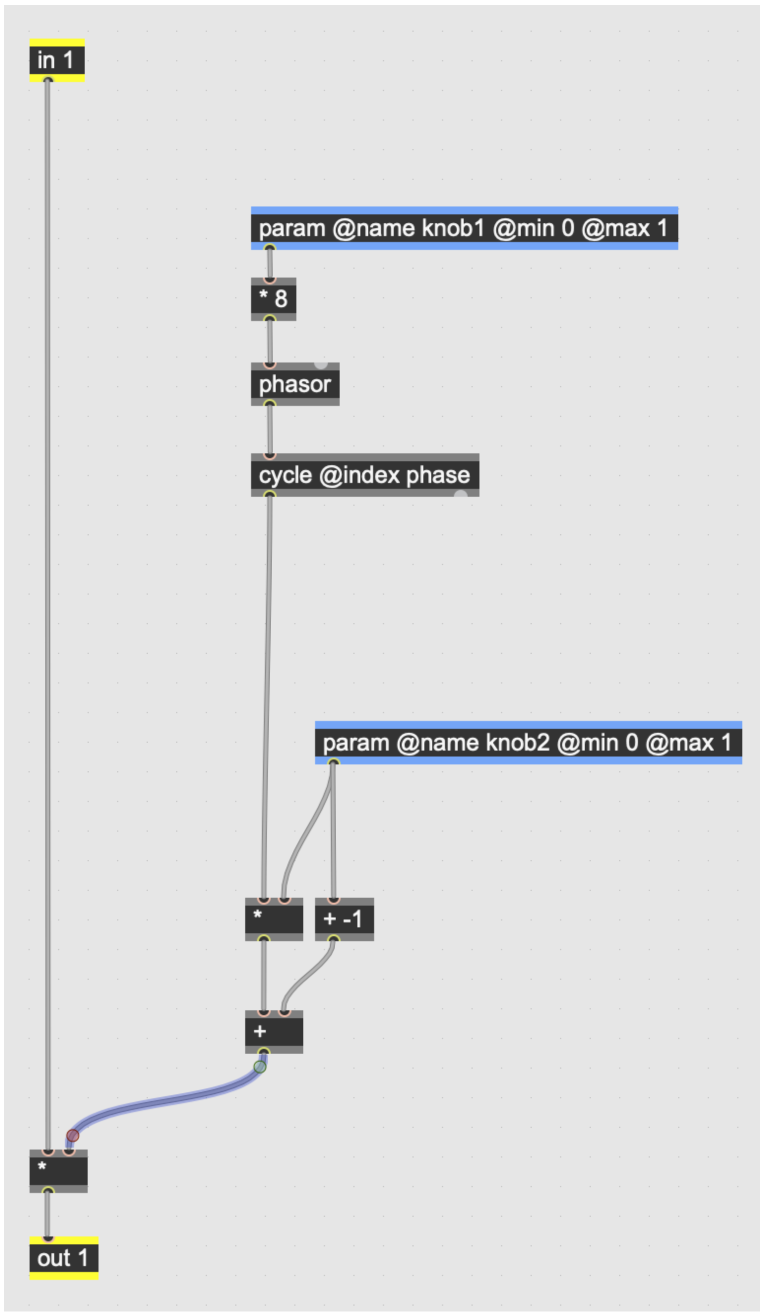
In the same way that we multiplied the input signal by a number to change its level, we will multiply the cycle level so that, instead of the depth going from 0 - 1, we can turn the knob to make it go, for example, from .25 to .75.


in 1
param @name knob1 @min 0 @max 1
* 8
phasor
cycle @index phase
param @name knob2 @min 0 @max 1
*
*
out 1

Step 11

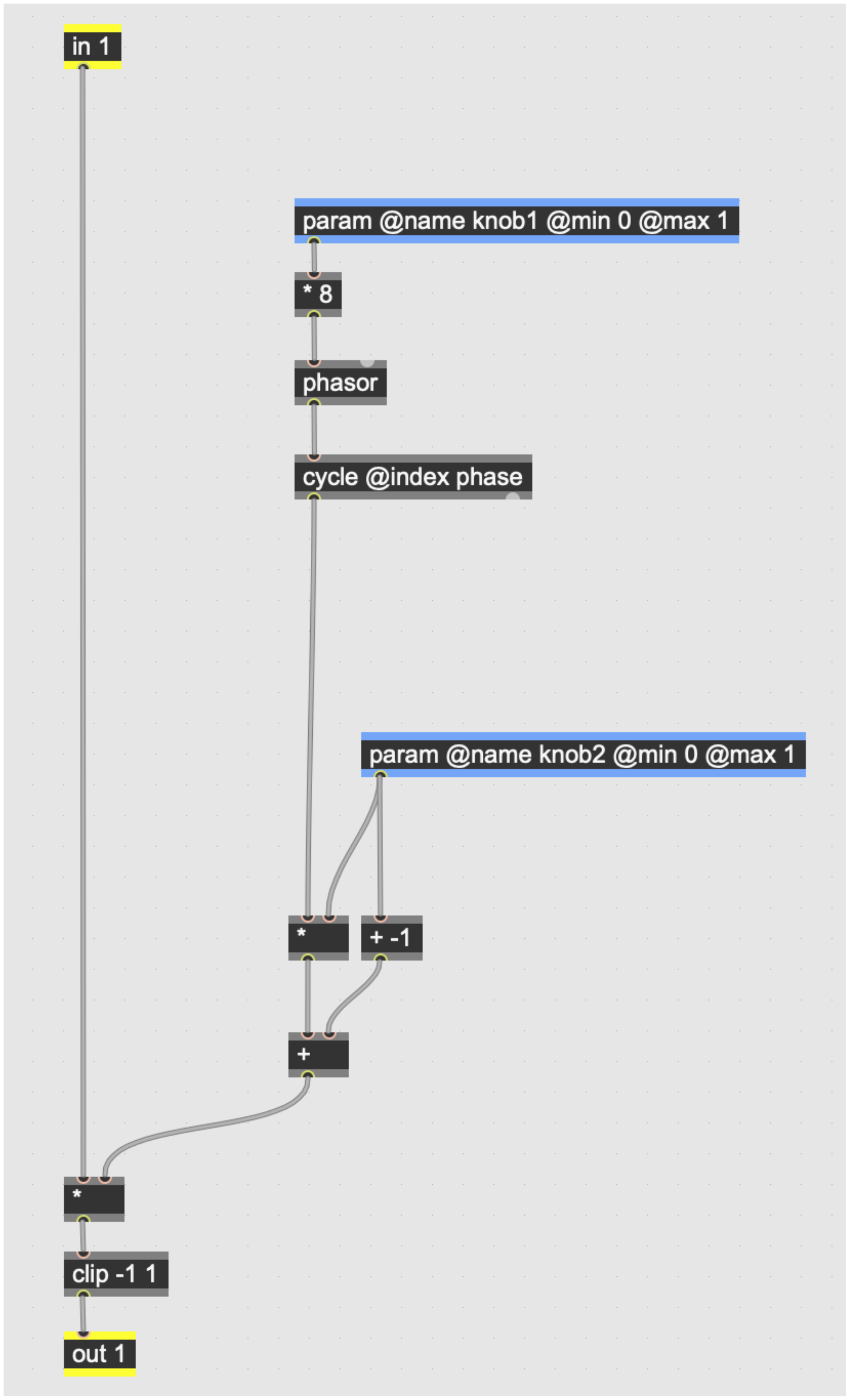
By reducing the depth of the cycle, however, the signal depth we're using to change the input signal's amplitude won't reach all the way up to 1, so there will be a reduction in the output level when the effect is being used. So to compensate for this, we'll add an offset to the signal so that whatever the new max depth number is from *cycle*, we add the difference back to the signal.


in 1
param @name knob1 @min 0 @max 1
* 8
phasor
cycle @index phase
param @name knob2 @min 0 @max 1
*
+ -1
+
*
out 1


in 1
param @name knob1 @min 0 @max 1
* 8
phasor
cycle @index phase
param @name knob2 @min 0 @max 1
*
+ -1
+
*
out 1

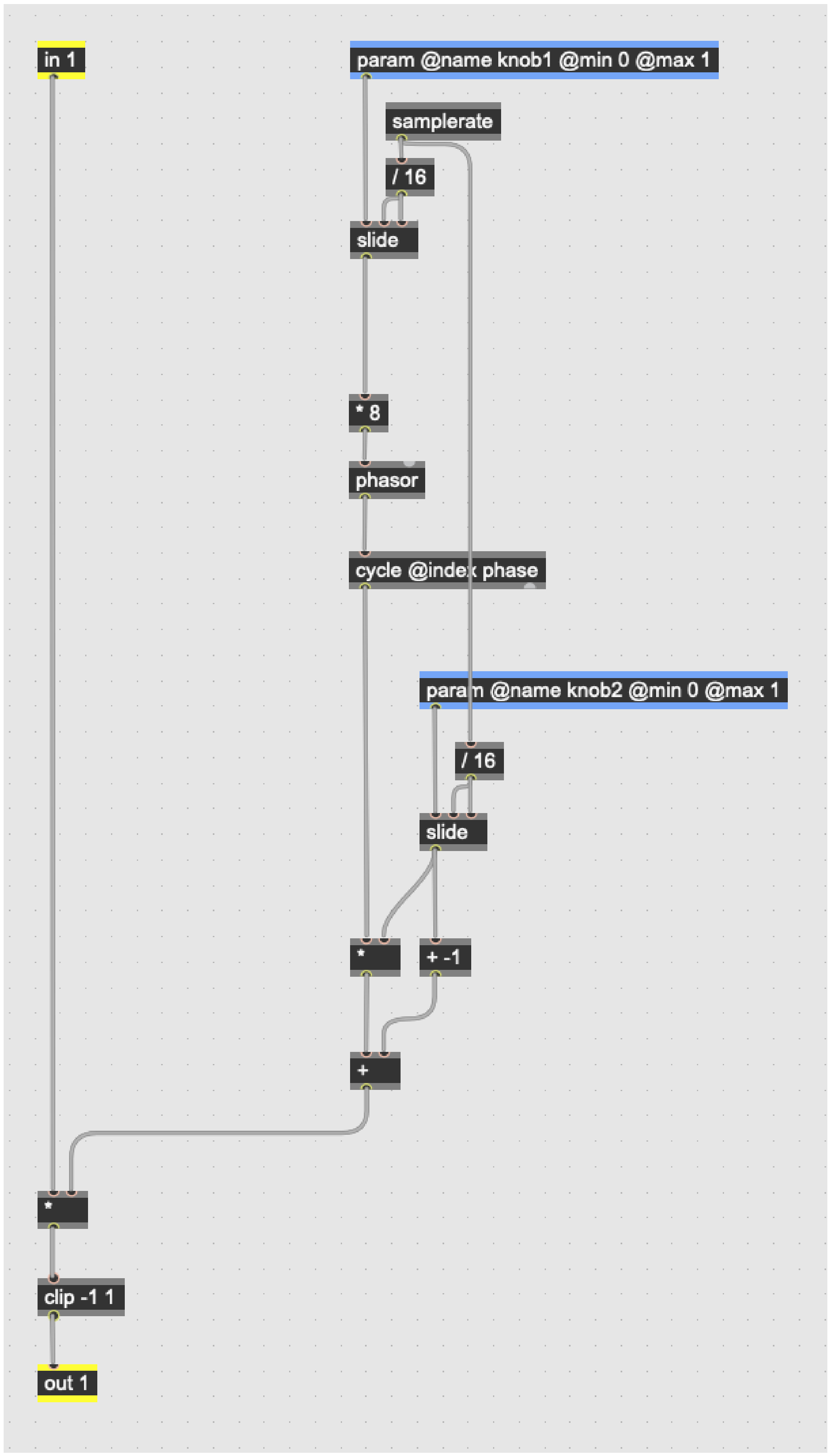
Step 12

Now we've got the effect working with two useful controls. The example patch in our kit has a few minor tweaks added to avoid clipping at the output level by adding a clip object that hard limits the output level of the audio so that signals louder than that are reduced instead of being passed on to the output jack.


in 1
param @name knob1 @min 0 @max 1
* 8
phasor
cycle @index phase
param @name knob2 @min 0 @max 1
*
+ -1
+
*
clip -1 1
out 1

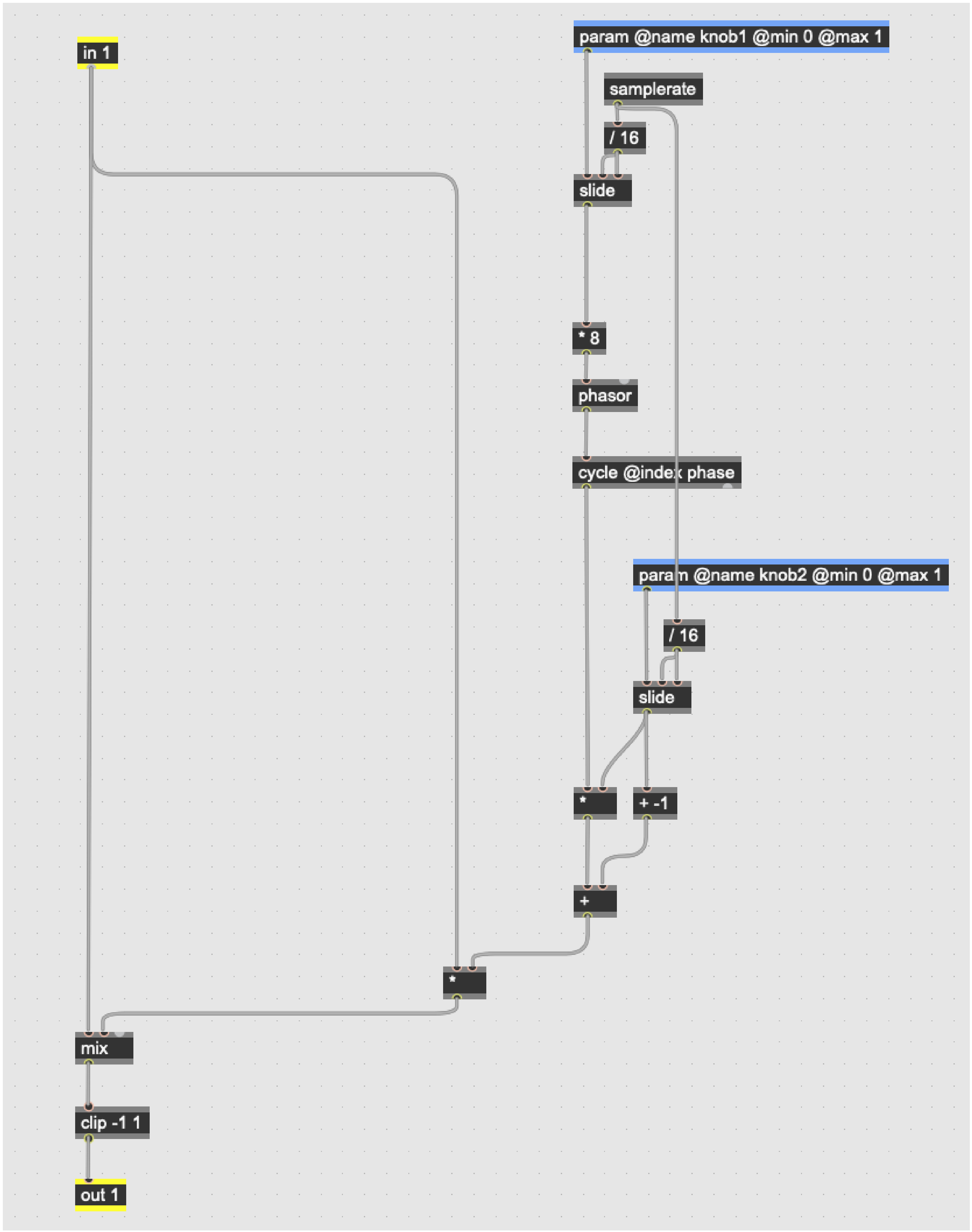
Step 13

We also added a tweak to avoid clicks and other noise from the potentiometers; when these knobs send numbers, the transition from one number to a new number can cause noise, we've used a slide object so that the numbers count up to new values instead of abruptly jumping there. In this case, we're setting that transition rate to a number of samples per second determined by the microntroller's sample rate.



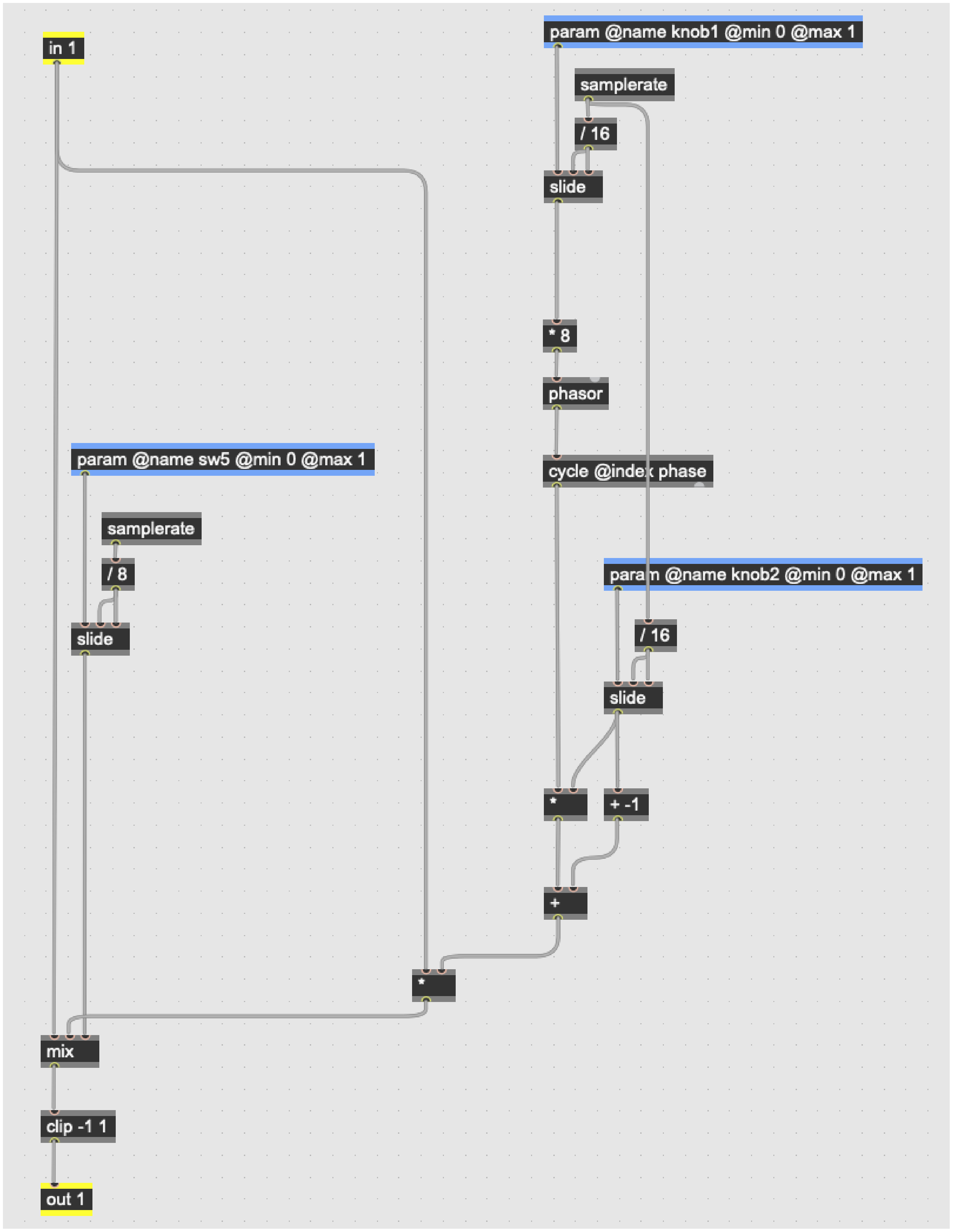
Step 14

Finally, we'll add a bypass switch for this pedal so that the tremolo effect portion can be turned on or off; the mix object will take the unaffected input from *in* its left inlet and the affected tremolo signal in its second inlet.



Step 15

We will then add a push button *param* to tell the *mix* object which signal we'd like to be sent to the output: the dry signal received at inlet 1 or the processed signal received at inlet 2. When you press the UI control labeled switch 5 in the main patch, the *param* named sw5 will send a 0 or a 1 and toggle between the dry version of the signal and the affected version of the signal via the *mix* object.


in 1
param @name knob1 @min 0 @max 1
samplerate
/ 16
slide
* 8
phasor
cycle @index phase
param @name sw5 @min 0 @max 1
samplerate
/ 8
slide
param @name knob2 @min 0 @max 1
/ 16
slide
*
+ -1
+
*
mix
clip -1 1
out 1