EchoWreck

A PT2399-Based Work-Alike to the Classic Echo Machine

Overview

In a discussion of classic delay/echo effects, the ones that most frequently get mentioned are the EchoPlex EP-3, Roland Space Echo, and the Binson Echorec. The EP3 and Space Echo are tape delays, while the Echorec is based around a magnetic disk that spins past 4 playback heads.

There have been some DIY projects in the past that have attempted to replicate the sound of the Echorec with varying amounts of success. The awesome Multiplex from 1776 Effects cops the general vibe of the Echorec, but uses only two delays instead of 4. The FV-1 chip also makes it possible to get a high level of accuracy and detail, but I'm a cheapskate and wanted to see what I could do with the old standby, PT2399.

The Echorec has a fixed delay time of approximately 300 ms, but I wanted to give a little more flexibility, so I added adjustable (but synchronized) delay times and decided to add tap tempo for good measure. Additionally, there is filtering on both the playback and feedback paths.

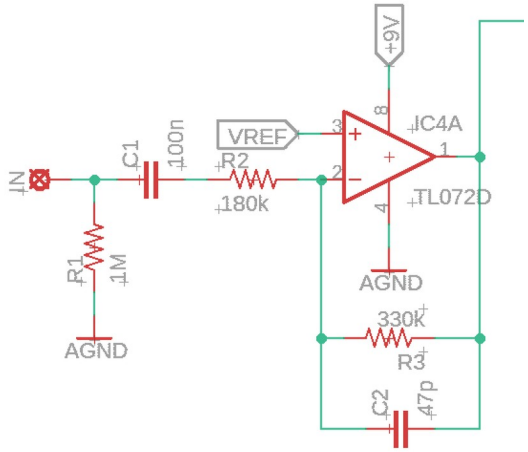
How it Works

From a very high level, EchoWreck is very similar to other popular PT2399 delays. It has an input buffer, PT2399 for delay, adjustment for mix level, and output buffer. However, instead of a single delay line, there are four in parallel. This could also be done with the delay lines in serial, but it makes the switching logic, particularly for feedback, more difficult and can result in more noise in the later repeats, which is not characteristic of the Echorec.

Toggle switches are used to control which “heads” are active and for which function. Toggles are provided for all four heads for both playback (initial echo) and feedback (subsequent repeats). There is also a toggle for “swell” mode, which really just activates all four heads for playback which, when used in conjunction with feedback, can give that big, cavernous sound the Echorec is known for. Finally, there is a toggle for the “drift” function, which currently is programmed to change the time division of the second and fourth repeats to give a bit of a “wobble” characteristic. I have coded it so that they are

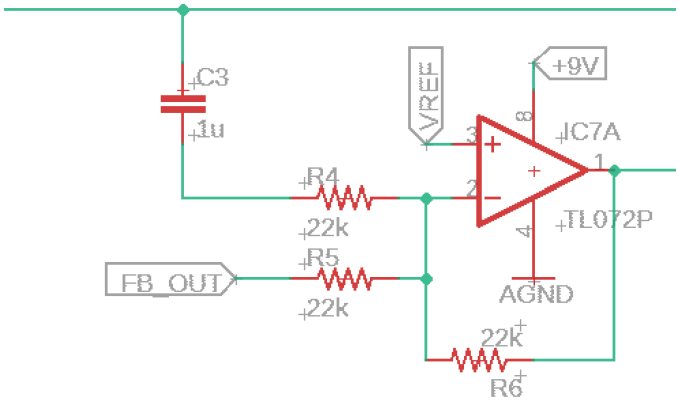
still rhythmically related, but feel free to change the code to make it be whatever you want.

The input buffer is a run-of-the-mill opamp buffer to prevent loading and present a high input impedance. Nothing special here.



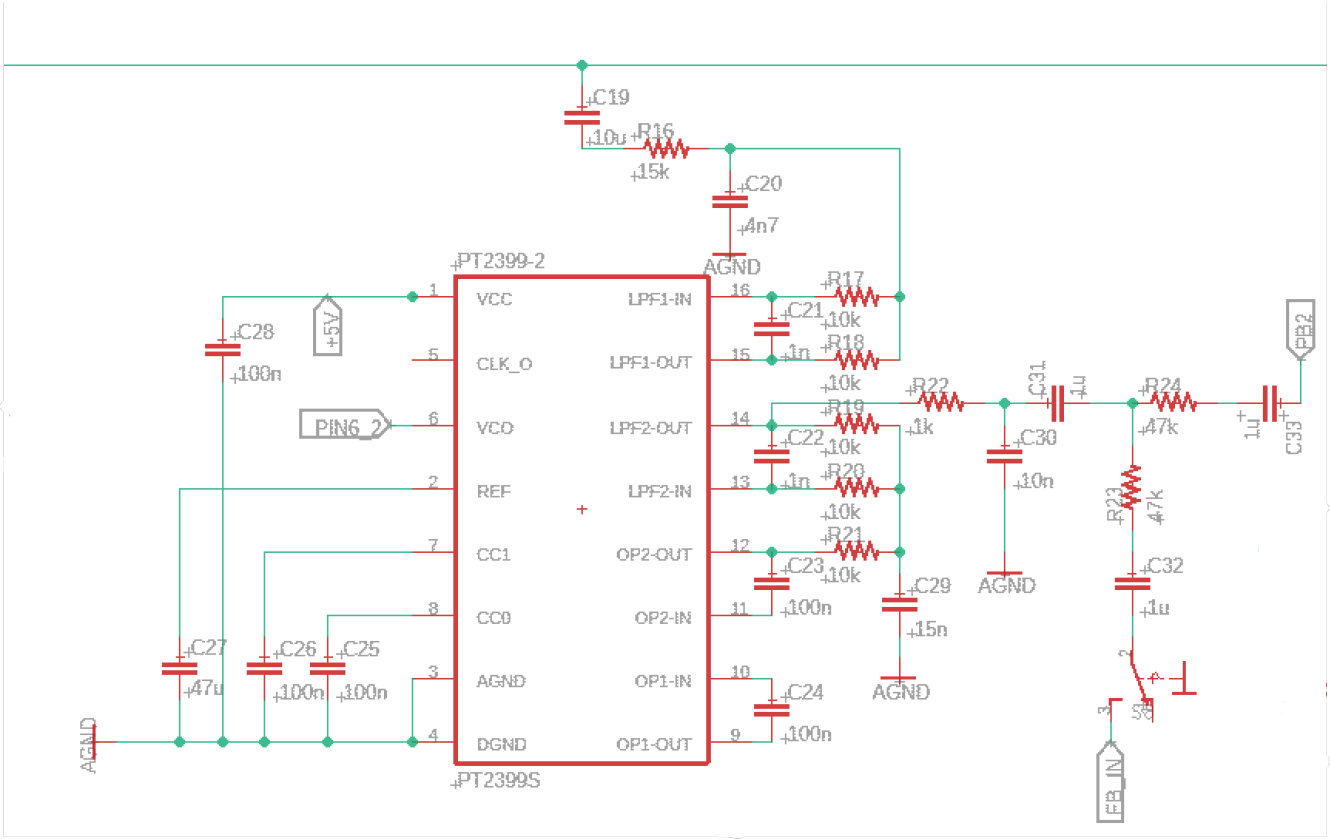
Input Buffer

The output from the input buffer is sent to another opamp, this time configured as a summing amplifier. This sums the dry input signal with the feedback signal prior to sending the signal to the input of each delay stage. This prevents loading between the two signals so that the feedback can be dialed down to a low level without causing the overall delayed signal output level to also drop. Without it, things go bad quickly. Ask me how I know...



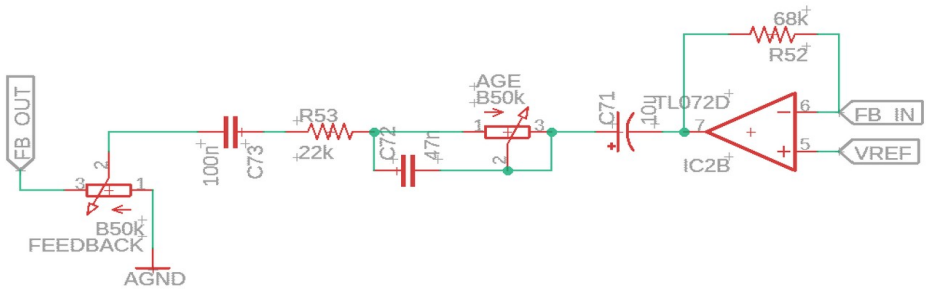
Delay Stage Input

The actual delay stages are pretty straightforward. For a detailed breakdown of what each component is doing here, visit ElectroSmash's most excellent PT2399 circuit analysis page. Just a couple of things to note are that I chose 1 nF caps between pins 15/16 and 13/14 to keep the delays bright. With the tone control, I can always remove some of that brightness, but you can't recoup it once it's gone.



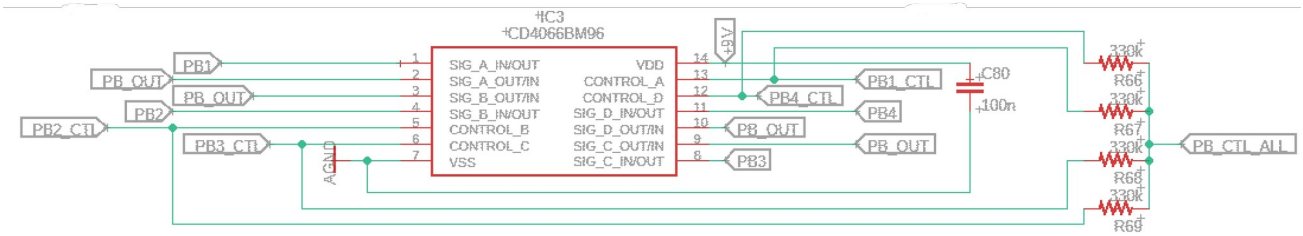
Delay Stage, or "Head"

The feedback path (shown in the figure below) utilizes a summing amplifier to prevent bleed between channels. This is then fed through the Age control, which controls the amount of bass in the feedback signal. With full bass, the repeats have more low end content, like an older disk/head. With the age backed off, the feedback repeats are brighter, like on a brand new disk/head. The feedback level control is configured as it is to prevent pulling the FB_OUT signal to ground when turned to 0, which impacts the input to the delay lines.



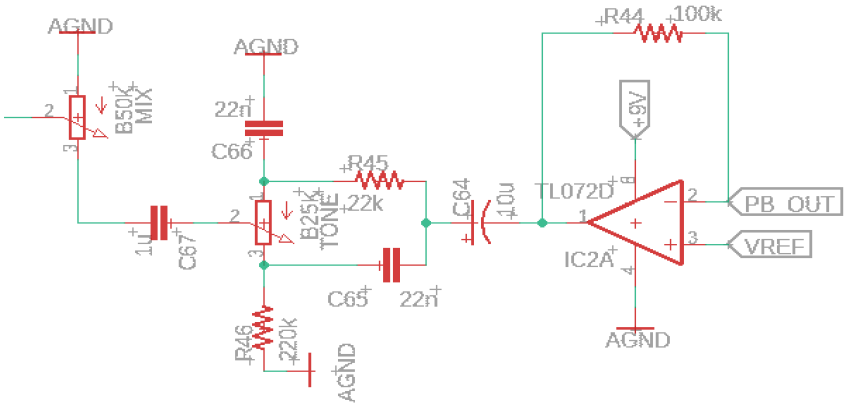
EchoWreck Feedback Path

The output from each PT2399 is sent to an input of a CD4066 CMOS quad switch. Each switch is an SPST, controlled by the voltage on a dedicated control pin. The playback toggle switches control the voltage for the 4066 switch for each "head". The Swell control applies the control voltage to each of the four switches, activating all four "heads", regardless of the other toggle switch settings.



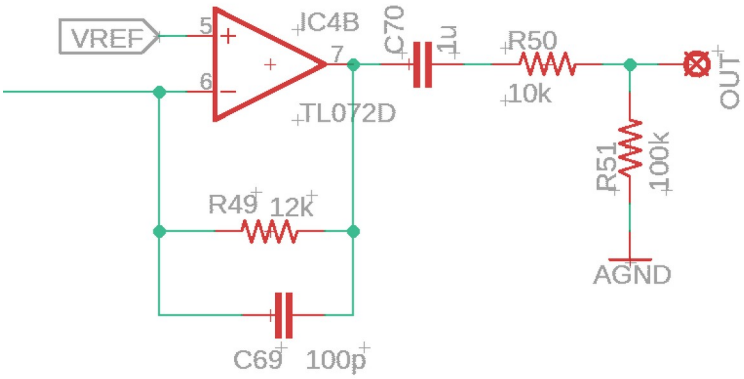
Playback Signal Switching Using CD4066

The outputs of all four CD4066 switches are tied together and sent to an opamp summing amplifier to prevent loading and bleed between “heads”. The playback signal then goes through a tone control that can roll off highs from the playback signal to give it a “softer” feel, or give it a little bit of brightness like the Echorec is renowned for. From there, it goes into the Mix control before being combined with the dry signal at the output buffer. Note that the mix control in the figure below is shown as B50k. It can be increased to B100k to get to slightly greater than unity repeats, which I did on my personal build. It's up to you.



Playback Path

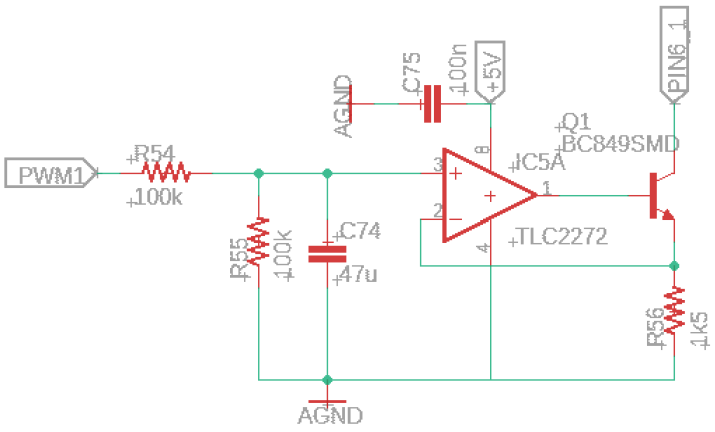
The output buffer is a standard output buffer that doubles as a summing amplifier for the dry and delayed signals.



Output Buffer/Summing Amplifier

As mentioned earlier, the delay times are synchronized. This is accomplished by using the four PWM outputs of an ATTiny84**1** (*NOT* and ATTiny84, which only has two PWM outs). Using a lookup table

that relates delay time to PWM duty cycle, the microcontroller keeps the delay times in sync by keeping track of the longest delay time and calculating the other delay times from it, then using the lookup table to set the appropriate duty cycle for each output. The PWM signal is then smoothed and stabilized using an active current sink, which is connected to pin 6 of the PT2399 to set delay time. The figure below shows what this current sink looks like. Note the use of the TLC2272, which is a rail-to-rail opamp. This current sink will NOT work with something like a TL072, since it cannot swing to the voltage rails.



Current Sink Circuit For Controlling PT2399 Delay Time

The delay time is controlled by either tap tempo, or through turning a rotary encoder. The rotary encoder is used instead of a potentiometer so that a time can be tapped in and then adjusted from there, as opposed to a potentiometer which would override any tap setting when turned.

BOM

The BOM below is the list of parts I used for mine along with quantities. Due to the nature of this project, there are several parts that are not available from Tayda, which is my go-to supplier. For example, the PT2399 chips are the SOIC-W (300 mil wide) package instead of the 150 mil wide package that Tayda offers. This was just a dumb mistake on my part when choosing the footprint, but I had already sent the boards out to fab. You can find them at SmallBear, or I bought mine from LCSC, which also has unbeatable prices on SMD caps, resistors, and other common IC's. I would recommend getting everything you can from there, and then finding other places for the rest (I did use Tayda for all switches and pots, but the encoder came from elsewhere). If you do order from LCSC, they have a cheap shipping option, but it tends to take 3 weeks for me to get my package here in the US, but I'm rarely in a hurry.

Please note that all resistors are 0805 package, as are all ceramic caps. Electrolytic caps are 5.4x6.3mm footprint. All IC's are SMD, including ATTiny841 which is *ONLY* in SMD package.

Part	Qty.	Notes
1k Resistor	4	
1k5 Resistor	4	

4k7 Resistor	1	
10k Resistor	30	
12k Resistor	1	
15k Resistor	4	
22k Resistor	7	
47k Resistor	9	
68k Resistor	1	
100k Resistor	10	
180k Resistor	1	
220k Resistor	3	
330k Resistor	5	
470k Resistor	4	
1M Resistor	1	
RLED_Tap	1	CLR for tap tempo LED
RLED_BP	1	CLR for bypass LED
47pF Capacitor	1	
100pF Capacitor	1	
1nF Capacitor	8	
4.7nF Capacitor	4	
10nF Capacitor	6	
15nF Capacitor	4	
22nF Capacitor	2	
47nF Capacitor	1	
100nF Capacitor	27	
1uF Ceramic Capacitor	16	
10uF Ceramic Capacitor	5	
47uF Ceramic Capacitor	8	
10uF Electrolytic Capacitor	2	Finding these in the large footprint is hard, can be swapped for 47 uF no problem
47uF Electrolytic Capacitor	1	
100uF Electrolytic Capacitor	2	
B25k Potentiometer	1	9mm PCB Mount
B50k Potentiometer	3	9mm PCB mount
EN11 Encoder	1	20mm shaft length

		recommended
SPDT On/On Toggle	9	
SPDT On/Off/On Toggle	1	“Drift” toggle
1N4148W	1	Voltage polarity protection
LED	2	Pads spaced for 3mm LED
BC849	4	
LM78M05	1	
ATTiny841	1	SMD-only!
CD4066	1	
TLC2272	2	
TL072D	3	
PT2399	4	**SOIC-W (300 mil)**
Enclosure	1	1590BBM/1590BBT/Gorva S90
1/4” input jack	2	Lumberg or open frame style
DC power jack	1	
3PDT footswitch	1	
SPST momentary footswitch, NO	1	Tayda #A-2371

Schematic

The schematic for this project is way too big to put on just one or two sheets. Please see the EchoWreck_Schematic.pdf file included in the .zip file package for this project.

Programming the ATTiny841

Programming the ATTiny is most easily done using an Arduino nano or Arduino uno. Unlike some of my past projects, the ATTiny841 requires the ATTinyCore by Spencer Konde, available from his github page. It is really excellent. You still use the Arduino as ISP, but the ATTinyCore has more options. The defaults are fine for most things, though I have the EchoWreck code running at 8 MHz so that the PWM is smooth and so that tap updates are responsive.

One other thing to note is that the ATTiny841 is SMD only. That means that you will need either a spring loaded socket or you will need to use the header pin pads on the EchoWreck Signal rev 1.1 PCB. The pins are clearly labeled with the numbers/designation of the *Arduino* pins. All you have to do is use Dupont wires to connect (ensuring that you are not supplying power to pedal) and then program as normal. I found this to be the most convenient way of programming the chip, and it allows for you to make changes to the code after it's soldered in to place, as opposed to using a socket, which only works before you solder the chip down.

Build Notes

Here are some things I noted from building the EchoWreck using signal PCB rev 1.1 and control PCB rev 1.2. Please read this section to make sure you don't go through excessive frustration.

Enclosure Size

This project is, as I'm sure you've seen by now, pretty intense. It utilizes two PCB's that are stacked and are designed for a 1590BB sized enclosure. However, if you include the programming header pins for the ATTiny841, like I did and like I strongly recommend, you will be *just* over the depth of a standard 1590BB. I decided to go with a Gorva S90 from LoveMySwitches, which is a gorgeous enclsoure. However, it is slightly smaller inside, which means that the two top corners of the control board needed to be every so slightly trimmed using a sander to fit around the posts for the screws in the casing. Also, this requires precise drilling, as there is very little wiggle room inside.

Another option is the 1590BBM or 1590BBT, both of which are deeper than the 1590BB and which have the same footprint.

Jacks

If you are using a 1590BB, you will likely need to use a low profile jack. With a deeper enclosure, it's less important, but I still opted for genuine Lumberg jacks for this build (come on, you're not building something this crazy because you're *only* a cheapskate, right?!)

Switches

The PCB's are notched to use one 3PDT for bypass and one SPST momentary for tap. Note that with the EchoWreck Signal rev 1.1 PCB, the notch for the SPST doesn't quite match the Control rev 1.2 PCB. In my build, this wasn't really an issue due to the height difference between PCB and switch, but it is tight.

Enclosure Drilling

The pots, LED's, and buttons are designed to all go in vertical lines. In the .zip file for this project I have included the artwork/drill template for you to use. Note that the outline is for my Gorva S90, so if you are using a 1590BB-sized box, work off the center line, which is pretty obvious in the artwork.

Changing Microcontroller Parameters

There are several things you can change in the microcontroller code to personalize your EchoWreck. This won't be an exhaustive list, but will point out the ones that are going to be most likely to be useful.

Changing “Drift” Characteristic

The drift switch has three settings. In the middle, the second and fourth repeats are exactly in time with the first and third. For simplicity, I will say that they are 2 and 4 times the first repeat length, respectively. In one direction, they are 1.5 and 4.5 times the first repeat length, and in the second direction, they are 2.5 and 5 times the first repeat length. I chose these so that they are still rhythmically related to the other repeats, so that it doesn't sound really strange. These can be changed to values closer to nominal (e.g., 1.8 and 3.8 times the first repeat length) to give the impression of a slight disk wobble. This can be done in lines 290 and 293.

A change that I have considered but have not implemented is to actually change the code so that each “head” has its own multiplication factor and making it so that the drift settings create unique spacing between them. This is rather easy to do. It would require changing the behavior in the if statements on lines 289 and 292, as well as the multiplication factors applied on lines 497-500.

Changing Tap Tempo Behavior

As it stands, any tap tempo time that is longer than the maximum delay time (650 ms for head 4) is divided by 2 until it falls in the valid delay time range. This means if you tap in a 1000 ms delay, the resulting head 4 delay will be 500 ms. A 1600 ms tapped delay will result in a head 4 delay of 400 ms (1600/2 = 800, 800/2 = 400). This was chosen to keep the delays still in tempo, albeit faster, than what is tapped in, as it can be hard to tap really fast at times.

This can easily be changed in lines 445-446. For example, it could simply be set to make any tapped time greater than maximum to be at maximum. Note that this check does already exist in lines 331-333, so just commenting out lines 445 and 446 would do the trick. Or you can do whatever else you imagine.

Change Debounce Time Requirement

Some people want to be able to tap in very short delays, which can cause the debouncing logic to interfere. Right now the debounce delay is a very short 30 ms, but feel free to make it larger or smaller depending on what you want/need. Note that a very short delay could mess up the tap time because it doesn't adequately debounce it. This is changed on line 30 and is in ms.

Change Encoder Time Step

The encoder is currently set to increase the head 4 delay time by 8 ms each detent (2 ms for the first head). With a 20 detent encoder, that means that to go from the minimum delay time of 184 ms to the

maximum of 650 ms would require 3 full revolutions. This was done to provide a nice, fine resolution on the time, but if you find it annoying, simply change the size of the step on line 45. I do recommend keeping it a multiple of 4 to keep the ratio between the four heads as exact as possible. A factor of 8 is even better, since the lookup table is specified in 2 ms steps.

In Closing

This project was a ton of fun. I spent 6 months of research, design, development, prototyping, building, and documenting this project. I hope you enjoy it as much as I have. However, with the immense amount of time put into this, please remember that this is for personal, non-commercial use. If you would like to use it commercially, please contact me so we can come to an arrangement. Now go get your echo on!