eurotrash (build doc)

basics:

eurotrash is a fairly simple incarnation of the pjrc 'audio adapter' (https://www.pjrc.com/store/teensy3_audio.html) as a eurorack module, fully compatible with the pjrc audio API (https://github.com/PaulStoffregen/Audio).

As far as usage of the API is concerned, the only differences pertain to the (somewhat better) codec being used - PCM5102a not SGTL5000 (or WM8731) – the PCM5102a doesn't need to be initialized, so you can safely ignore any code relating to the SGTL5000 part (as does the eurotrash example firmware); the only difference in pin usage concerns the spi flash IC which can be installed, its chip select is pin 13 on the eurotrash (pin 6 on the official audio shield). If you intend to use it, you'll have to change that number in the spi flash class (though the spi flash currently isn't very well integrated into the API).

The remaining circuitry mainly consists in what's required to turn this into a eurorack module; being a digital module, that's all fairly boring and generic (more info below). pin usage is as folllows:

- digital inputs: pins 0, 2
- CV inputs: pins A2, A3, A4, A5
- digital outputs (eof): pins 20, 21
- encoder left: pins 4, 3 (switch: 5)
- encoder right: pins 8, 6 (switch: 15)

---

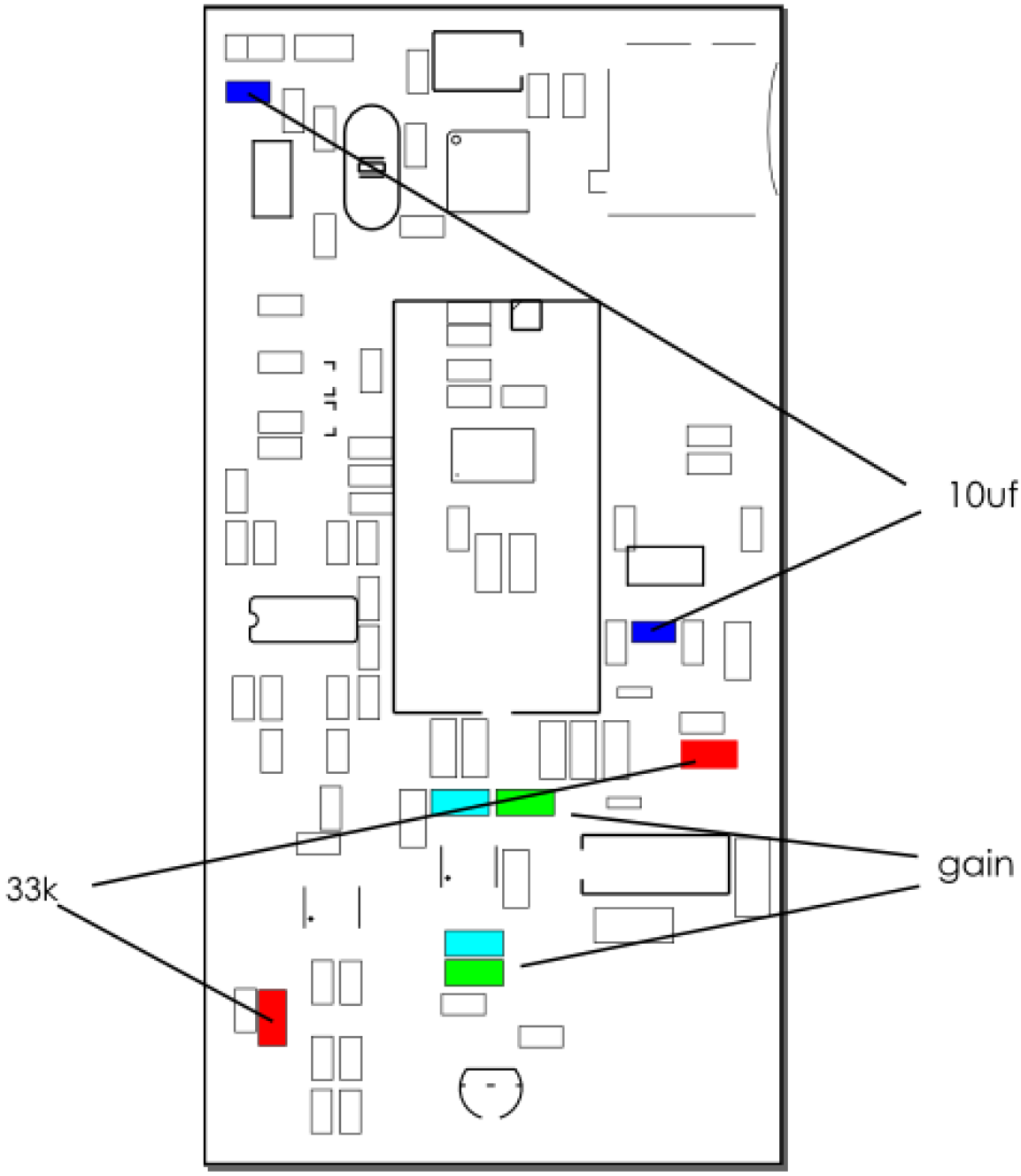
building it:

most of this should be straightforward, but watch out for the 0805 parts. it's easy enough the solder them the wrong way, especially around the op amps; you can follow the values printed on the pcb – **with only a few exceptions**, namely:

- the two 100n caps right next to the regulators should be 10uF each (see pic below)
- the unmarked diodes should be 33k resistors (see pic)
- the 79L05 should really be an LM4040-5; fortunately, that's an easy mod – see below. (it's not a must for the wav player, but for other uses, I'd recommend it)

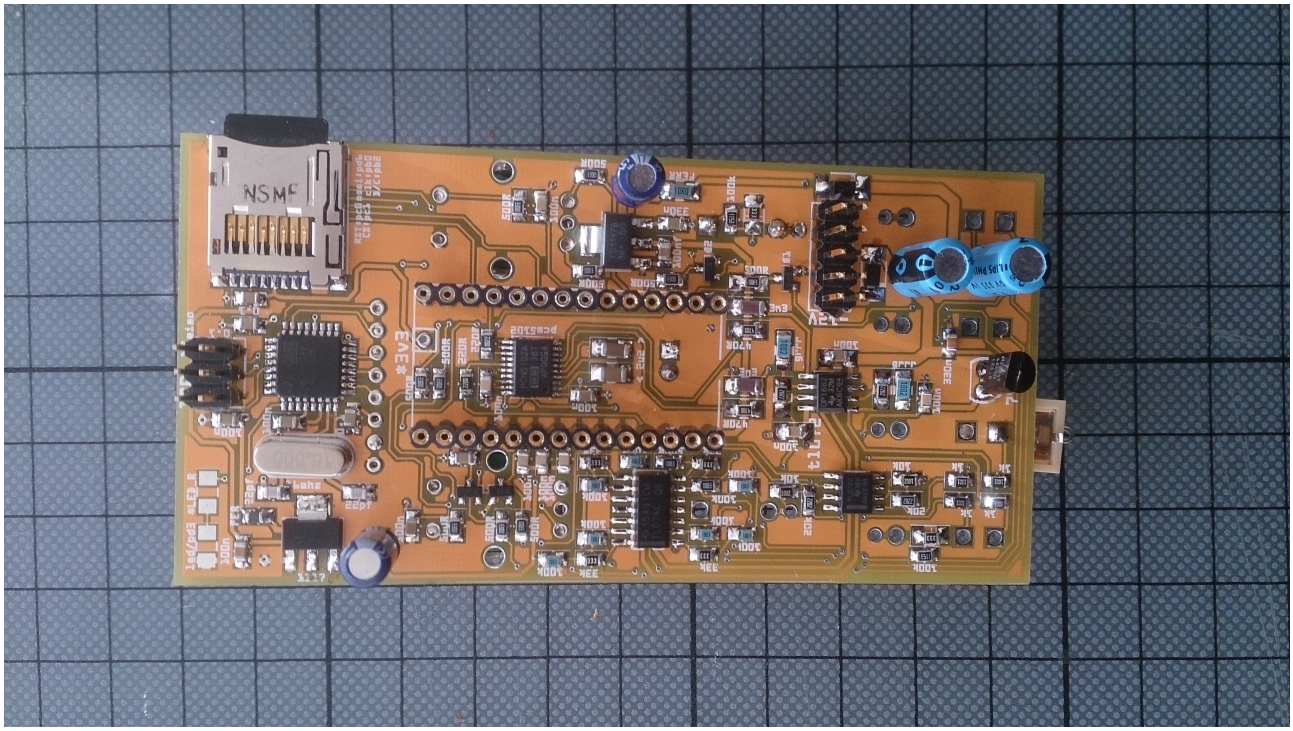
- 'FERR' is for ferrite; you can safely replace these with 10R resistors
- the resistors around the op amp (op1662) labelled 'fb' (feedback) and 'grrr' (ground)-- these determine the gain in the audio output stage. it's a simple non-inverting amplifier, so V_out = V_in * (1 + R_fb / R_gr). The pcm5201a puts out roughly a 6VPP signal (2.1 Vrms), so with ca. 1.7x gain you'll get the usual 10VPP (or

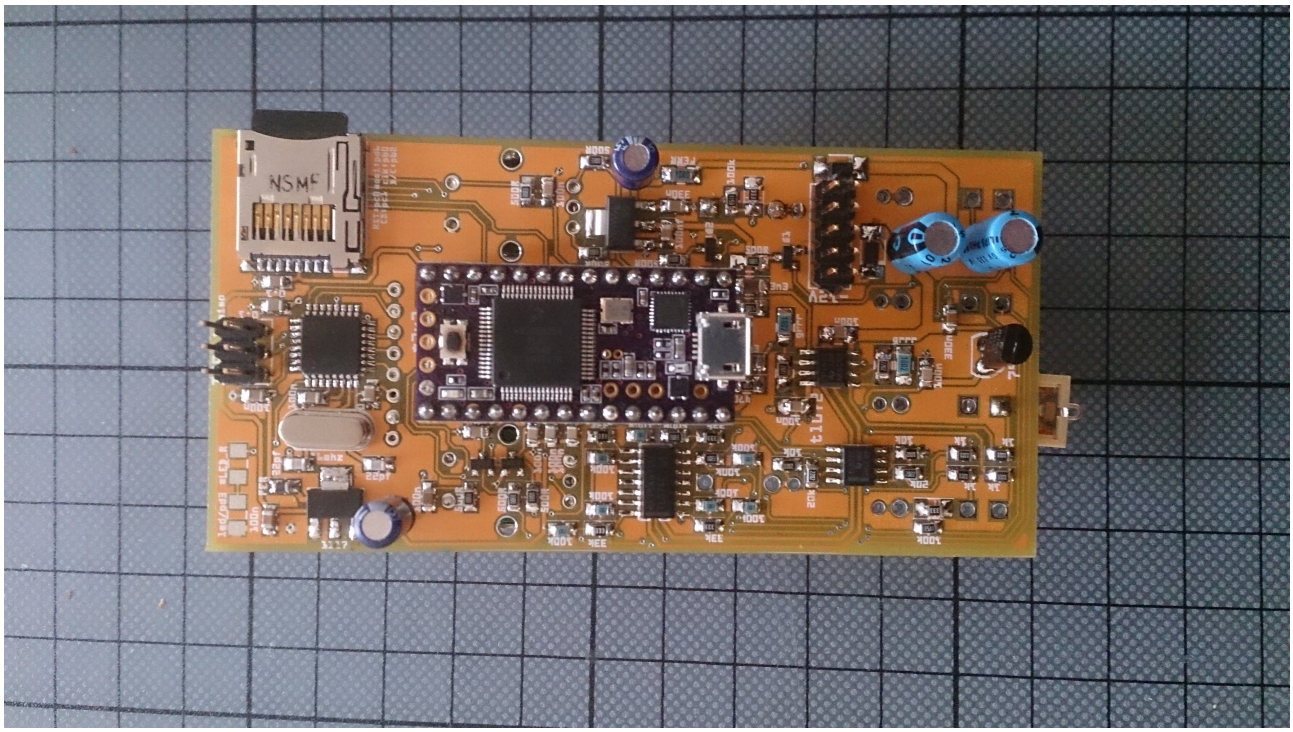
as per the BOM: 22k = fb , 33k = grrr).
- the sot-32 parts labelled ‘q1’, ‘q2’ are the npn resistors; the unmarked ones the bat54s diodes.
- 100n or 0.1 == 100n caps
- finally, the electrolytic caps should all go onto the bottom side, of course (other than indicated); sometimes fritzing acts a little strange, so i had to move them onto the top. Unless indicated otherwise, they should be 10uF. The only exception to is the 10uf cap underneath the teensy; this can’t be tall in either direction, so best to use a low profile one, or bend it so that it’ll fit underneath the panel.
- you can ignore the parts labeled “led/pd3” and “LED_R” in the upper left corner.



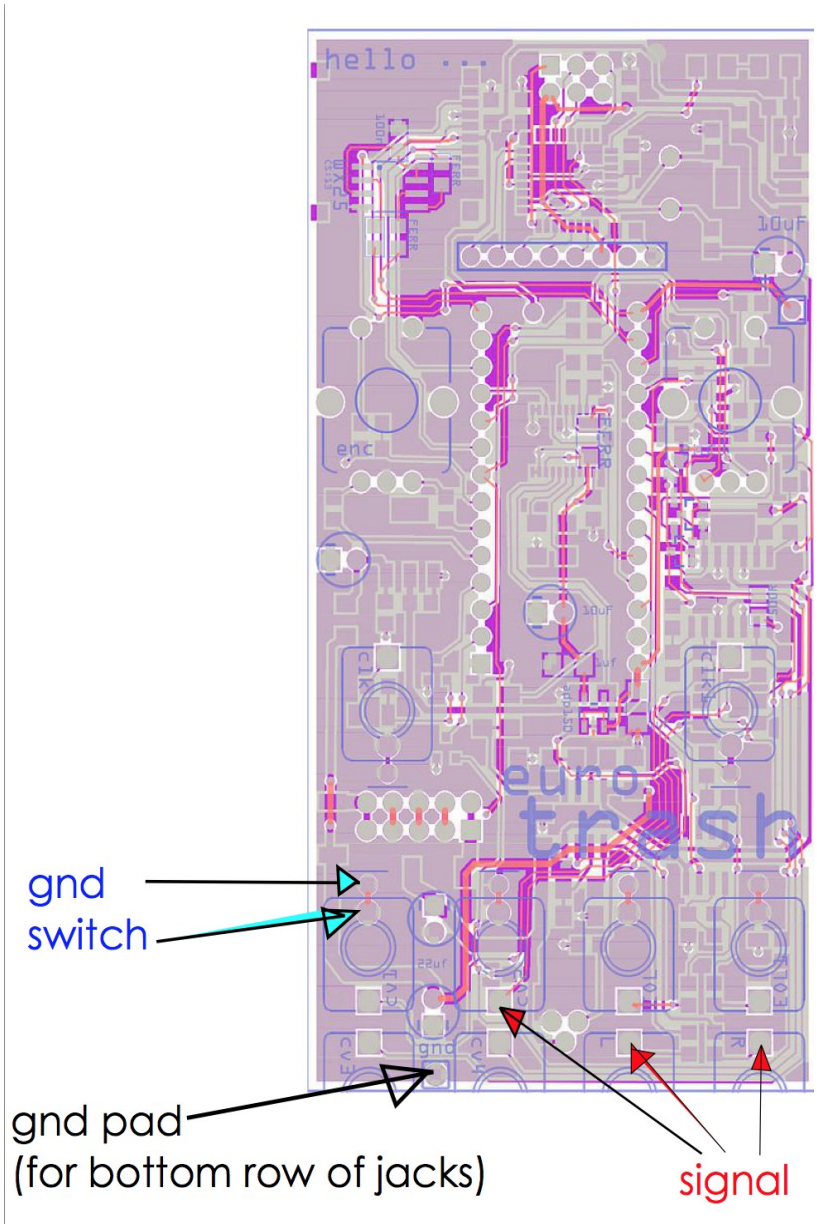
- solder the ICs first, followed by resistors, caps, regulator etc; then the through hole parts. don't attempt to push the ISP header into the holes, they're too small, as i had to route some traces in between; just solder it onto the pads.

here's an image for guidance (with and without the teensy)

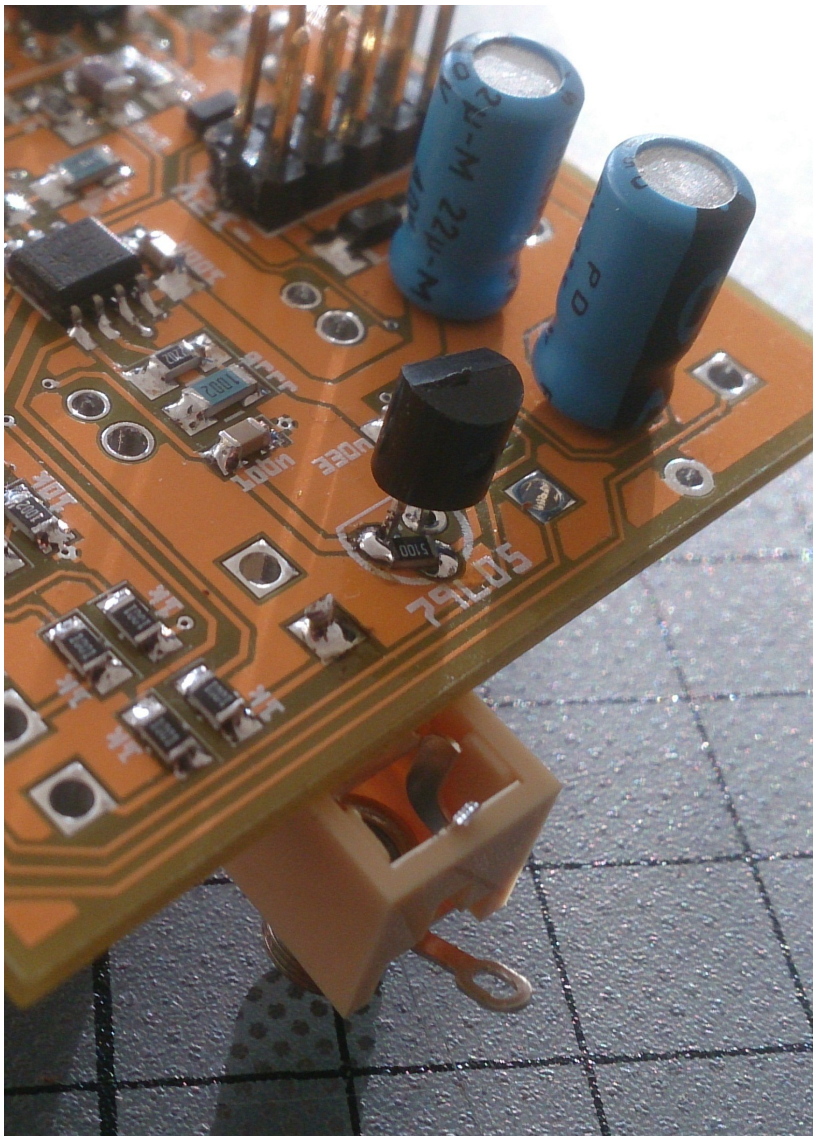




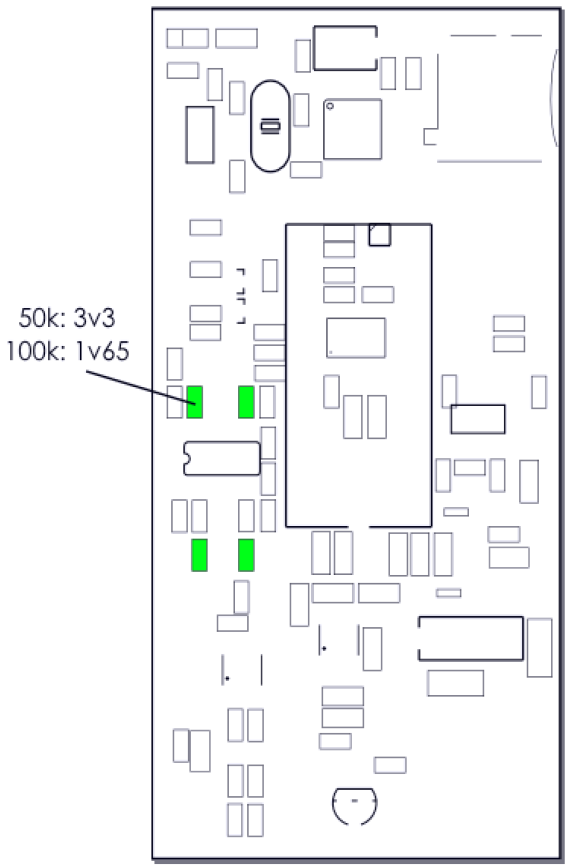
- jacks: the bottom row of jacks has to be mounted semi-off-board. that's slightly annoying, but it was easier that way to stay below 100mm ($$). square pads = signal. there's a gnd pad in between jacks 'cv3' and 'cv4' -- connect it to all four jacks (ground lug), ie cv3, cv4, L and R. For the CV jacks, you also want to connect the switch lug. For the audio outputs, you can snip it off:



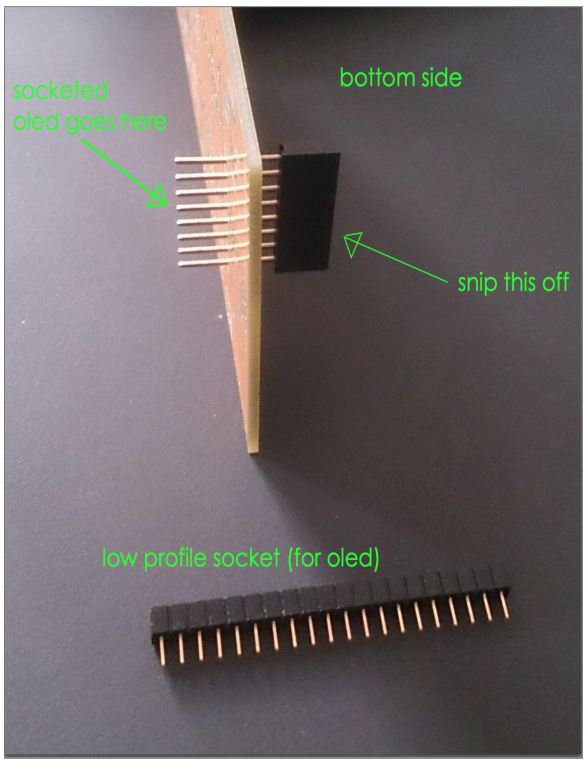
- substituting the 79L05 regulator. This is used as an offset for the CV inputs. A LM4040, being less noisy, will result in better ADC performance. Substituting is easy: first, snip off the "NC" leg of the LM4040. Then connect the two remaining pins as per the picture below (the middle pin (cathode)goes to ground, the other pin (anode) to the trace leading up the 100n cap; that's our - 5v); then solder a 500R (ish) resistor across the remaining pad (-12v) and the anode.



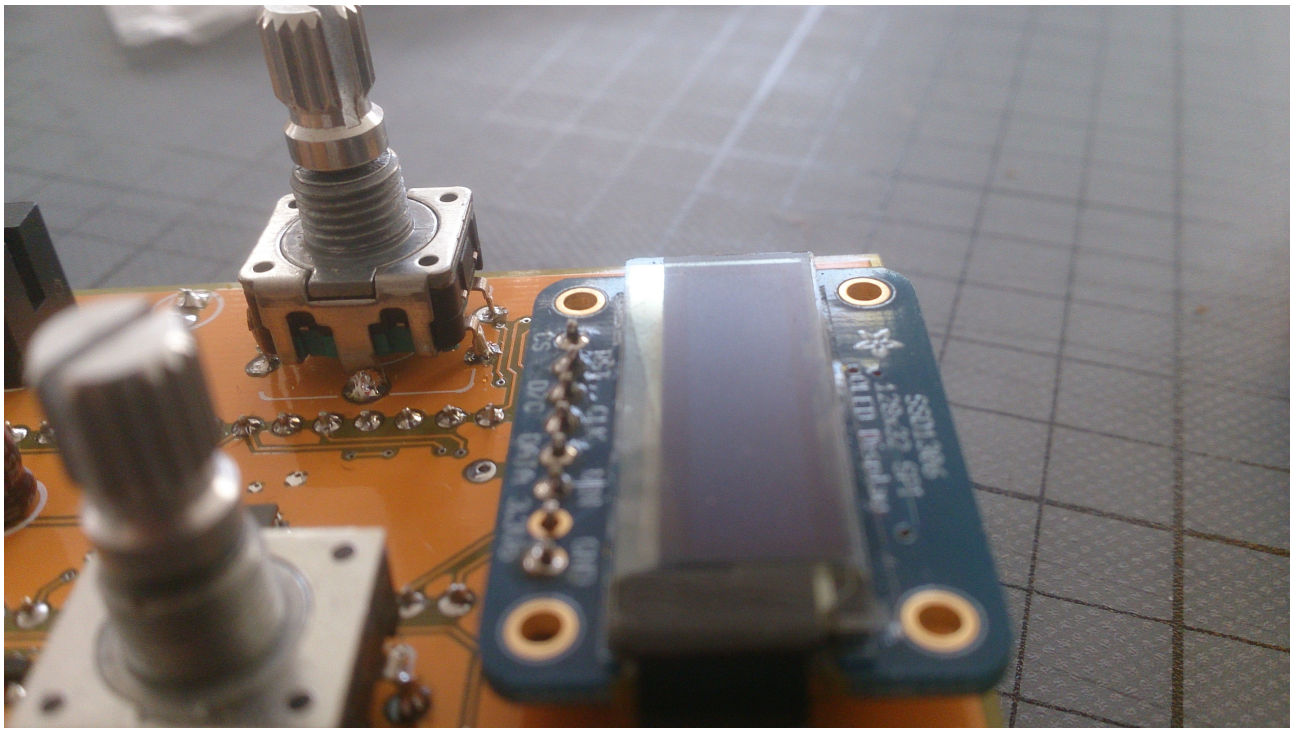
- when using the resistor values as indicated on the pcb, using the LM4040-5v will result in an 1.65v offset, or c. 2048 counts on the ADC; ie it'll work as a bipolar input. If you'd prefer unipolar inputs (might be more suitable for other applications than setting the start pos/end pos in the file player), you have to substitute the four 100k input resistors with 50k as follows (we're dealing with the basic **inverting** op amp configuration):



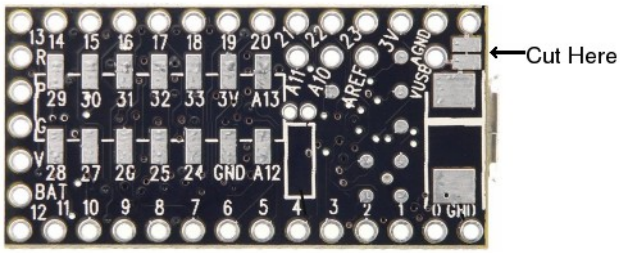
- encoders and oled: these should go on last. There isn't a lot of room in between the panel and the board (10mm or so, ie the height of the jacks), which is why i soldered my oled onto the pads, ie with just the header pins. Obvious downside: it's not removable.
It's doable though with low profile sockets. Someone pointed out this trick:



As for the encoders, the mounting holes are too small for most common encoders; i simply soldered mine onto the topside of the board, mine weren't tall enough anyways (see pic)



- prepare the teensy: you have to cut one trace:



- finally: the spi flash. If you use it, you'll note the footprint is somewhat too small. Fortunately, those winbond ICs have fairly large pins, which can be bent to make things fit. For the parts labelled 'FERR', as mentioned you can use 10R or so resistors.

testing/firmware:

- once having populated the board, plug it in, without the teensy; if nothing blows, we can proceed to flashing the firmware onto the atmega and teensy. If you want to test whether you get the right voltages, here's a few points you want to test:

- right after the ADP150 (at the inductor 'L', for instance), you should see 3v3; at teensy VIN, you should see 5v; at the test point labelled 3v3, you should see 3v3, too; you should also see - 5v coming out of the 79L05 resp. LM4040-5

- first, let's test the codec is working though. If you don't have it already, you need to install arduino/teensyduino (https://www.pjrc.com/teensy/teensyduino.html). Make sure you install the latest version as well as the Audio API (you'll be prompted on install to choose from several available libraries). You can also get it from github (linked above). Download the zip, unzip it and move into your arduino/libraries folder.

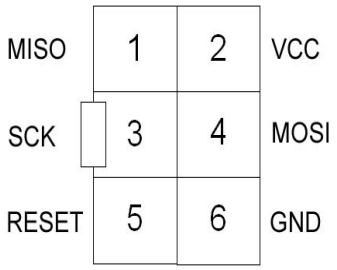
Once installed, open the arduino IDE, and open the **"SamplePlayer"** example; plug in the teensy, and load the sketch onto the teensy. -- you have to select "Teensy 3.1" in the Tools/Boards tab first. After the upload is completed, plug in patch-cables into the L and R outs; press any encoder, you should hear sound. (a snare and hi hats, i think)

in real life, we can't use the official pjrc audio API (yet) unfortunately, because it doesn't support everything we need. From the eurotrash github, you have to replace play_sd_wav.cpp and play_sd_wav.h in the Audio library folder (in your arduino libraries folder); you also need the encoder library called rotaryplus. This goes into the arduino/libraries folder as well.

You should now be able to compile + upload the eurotrash sketch.

- next, the atmega. You either need an ISP programmer or another arduino board; you can also use the teensy 3.1 to flash the firmware. (also to be found on github. The file is called "eurotrash_serial.ino"). If you have an isp programmer, i assume you know what you're doing, so here's how to do it with arduino (in very brief, there's several tutorials on the web). You'll also need to install the "U8glib" library -- https://code.google.com/p/u8glib/ (ie put it into your libraries folder)

The ISP header follows the conventions, and the MISO pin (1) is marked with, well, MISO:



let's assume you have an arduino nano or equivalent (ie something w/ an atmega328). You may also need: a 10uF capacitor and a breadboard (unless you have an uno or the like),

Steps:
- Go into the arduino IDE, select your board, and upload the example sketch "arduinoISP" (in file/examples)

- then run jumper wires from/to the respective pins (ISP < > arduino) (the connections can be found arduinoISP.ino file as well):

MISO < > MISO (pin 12)
SCK < > SCK (pin 13)
RESET < > pin 10
GND < > GND
MOSI < > MOSI (pin 11)

leave VCC unconnected, we'll power up the module from the psu. Next, put the 10uf cap across the GND and RESET pins of your arduino. Power up the module, and you're ready to flash the atmega. We do it two steps:

1). burn a bootloader. To do so, select a suitable board in tools/boards/ : nano will do (atmega328p, 16 MHz external). In tools/programmer/ select "arduino as isp". Then in /tools select "Burn Bootloader". Your arduino should start to flash its LEDs and burn the bootloader. If you get an error message, check your connections and try again.
2). Once the bootloader is there, we can upload the firmware. The procedure is similar. Open the "eurotrash_serial.ino" file; then in files/ click "upload using programmer". that'll take a few seconds.

The display should come to life now. (The procedure using a teensy is similar, but the arduinoISP sketch needs to be modified a tiny bit. See http://forum.pjrc.com/threads/23574-Teensy-3-as-AVR-ISP-programmer?p=32030&viewfull=1#post32030)

Final step: The SD card.

Best to use a class 10 card; upload some wav files. Presently, the restrictions are 16 bit, 44.1KHz stereo; file names should comply to the 8.3 standard, meaning they shouldn't have names containing more than 8 characters (excluding the ".wav' extension).

Put it into the socket. Power up the board. You should see the first file name appear on the display (in the order they were put onto the SD card)

A quick run down of the present I/O: