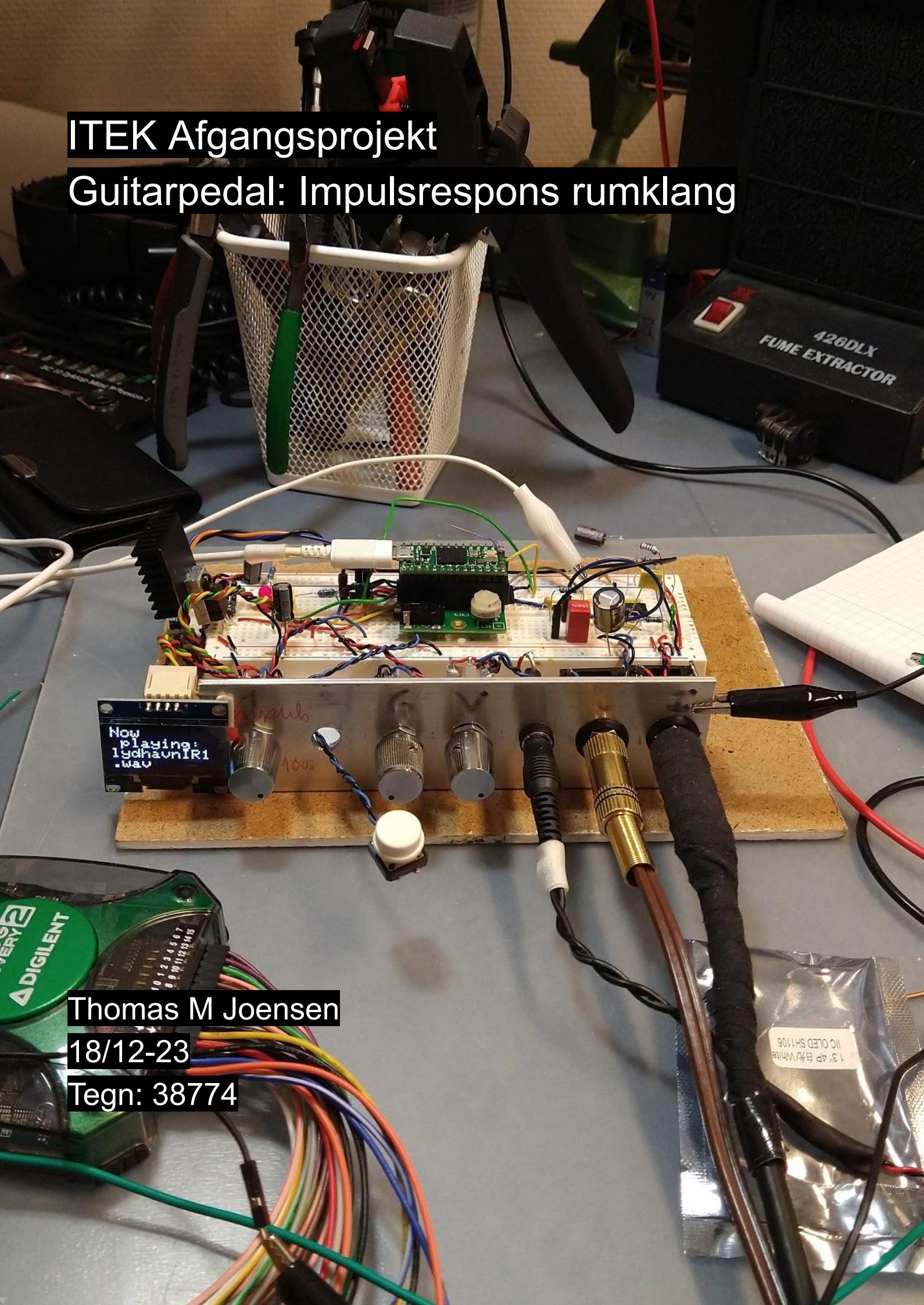


# ITEK Afgangsprøjekt

## Guitarpedal: Impulsrespons rumklang



Thomas M Joensen  
18/12-23  
Tegn: 38774

# Indhold

<b>Introduktion.....</b>	<b>2</b>
<b>Værktøj og metoder.....</b>	<b>3</b>
<b>Designkrav.....</b>	<b>3</b>
Projektbeskrivelse.....	3
Specifikationer.....	3
<b>Designprocess.....</b>	<b>4</b>
<b>Designbeskrivelse.....</b>	<b>5</b>
Hardware.....	5
Generelt om prototypen.....	5
Strømregulering.....	6
Forstærker og inputbeskyttelse.....	7
Forstærkerdel.....	7
Inputbeskyttelse.....	8
Outputbuffer.....	8
Ydre enheder.....	9
Software.....	10
Biblioteker.....	10
Teensy Audio.....	10
SD.....	11
Adafruit OLED Display.....	12
Brian Millier's Teensy-FFT-Convolution-Filter.....	13
Implementation.....	15
Lag 1 - Globalt.....	15
Lag 2 - main.ino.....	17
Lag 3 - menu.h.....	19
Lag 3/4 - utils.h.....	24
Lag 4 - displayHandler objekt.....	27
Lag 4 - SDHander objekt.....	29
Multi-respons funktionalitet (Eksperimentelt).....	36
<b>Test introduktion.....</b>	<b>37</b>
<b>Resultater.....</b>	<b>38</b>
<b>Bevis.....</b>	<b>40</b>
<b>Konklusion.....</b>	<b>40</b>
<b>Bilag.....</b>	<b>41</b>
Adafruit i2c scanner.....	41
main.ino.....	43
menu.h.....	46
loader.h.....	49
dispHandler.h.....	55

utils.h.....	60
convolution.h.....	65

# Introduktion

Effektpedaler er en meget populær form for modulation af lyd fra instrumenter, da de giver mulighed for at modulere lyden med minimal forsinkelse, i forhold til den, der kan opstå når man bruger et lydinterface og laver alle effekterne digitalt på en computer.

Prototypen, der beskrives i denne rapport, er en simulator, der simulerer rum, og i modsætning til algoritmisk rumklang ikke bruger et bestemt antal refleksioner og mængder af henfald for lyden, men derimod at folde en inputbuffer med optagelser af rigtige rum, såkaldte impulsresponser. Såkaldt impulsrespons convolution rumklang.

En impulsrespons optages ved at sætte en mikrofon op i et rum og optage en impuls, der typisk består af et højfrekvent smæld, og i nogle tilfælde et sinus sweep, samt rummets respons, rumklangen.

For at optage en ideel respons, vil man typisk anvende en eller flere højtalere til at afspille impulsen, og en eller to mikrofoner til optagelse, skulle man ønske forskellige responser for højre og venstre.

I mangel af bedre, kan man i mange tilfælde også klare sig med en mere lavpraktisk løsning som en telefon og et klap.

Udover mindre rum, kan pedalen også simulere forstærkerkabinetter og kasser fra akustiske instrumenter, med varierende succes.

Som det er, er beholdningen af af denne type pedal i lavvande på markedet, hvis man ser bort fra havet af kabinetssimulatorer, men disse tilbyder normalt ikke ret lange impulsresponser. TC's kabinetssimulator tillader 200ms impulsresponser, hvilket er meget for en kabinetssimulator, da man helst vil nøjes med 5-30 ms til et forstærkerkabinet.

Denne prototype tillader 400 ms. Dette er stadig ikke meget, men det tillader simulation af mindre rum som værelser og akustisk behandlede lydstudier, hvilket absolut er anvendeligt.

# Værktøj og metoder

Som sædvanligt har loddekolbe og multimeter været anvendt, samt oscilloskop, laboratoriestrømforsyning og funktionsgenerator.

Til teensyen har teensy's økosystem været anvendt.

Nyere teensy modeller er lavet til brug af arduino IDE'en, og har et yderst veludviklet audiobibliotek, der meget enkelt integrerer teeny's audio shield, via I2S og SPI.

Anvendelsen af det, er meget nemt at sætte sig ind i, men samtidigt er det ret svært at lave sine egne moduler, da det hænger sammen på en ret kompliceret vis.

# Designkrav

## Projektbeskrivelse

Digital guitar pedal, der simulerer rumklang fra virkelige rum. Disse rumklange, eller impulsresponser, optages, behandles og lægges på SD kort, der indsættes i pedalen.

Derefter kan effekterne loades via en menu på et display, der styres via en knap.

Effekten virker ved at foretage en FFT af impulsresponsen og en input buffer, hvorefter de foldes (ganges og summeres), og bliver outputtet. Denne proces kaldes også convolution.

Da biblioteker til foldninger allerede eksisterer vil af projektet vil bestå af at integration af de forskellige teknologier. I2S codec, display, SD kort læsning, impulsrespons foldning bibliotek, præprocesering og justeringmuligheder for impulsresponserne, gain og volumen.

# Specifikationer

## NEED TO HAVE:

- \* Throughput med rumklang
- \* Læsning af forskellige impulsresponser fra SD
- \* Basale krav til guitarpedaler som volumenregulering og gain på input.
- \* Optagelser af egne impulser

## NICE TO HAVE

- \* Diverse potentiometre til justering af effekten
  - Effektens styrke
  - Lavpas/højpas filtreret respons

Oprindeligt indeholdt kravene også,

- \* Analog postprocessering
- \* Regulering af effektens længde og hastighed

hvilke har vist sig at være irrevante. ADC'en er af sigma-delta typen og sørger selv for inputfiltrering, og DAC'en er ren. Regulering af længde og hastighed var alt for ambitiøst at lave på mikrocontrolleren selv.

## Designprocess

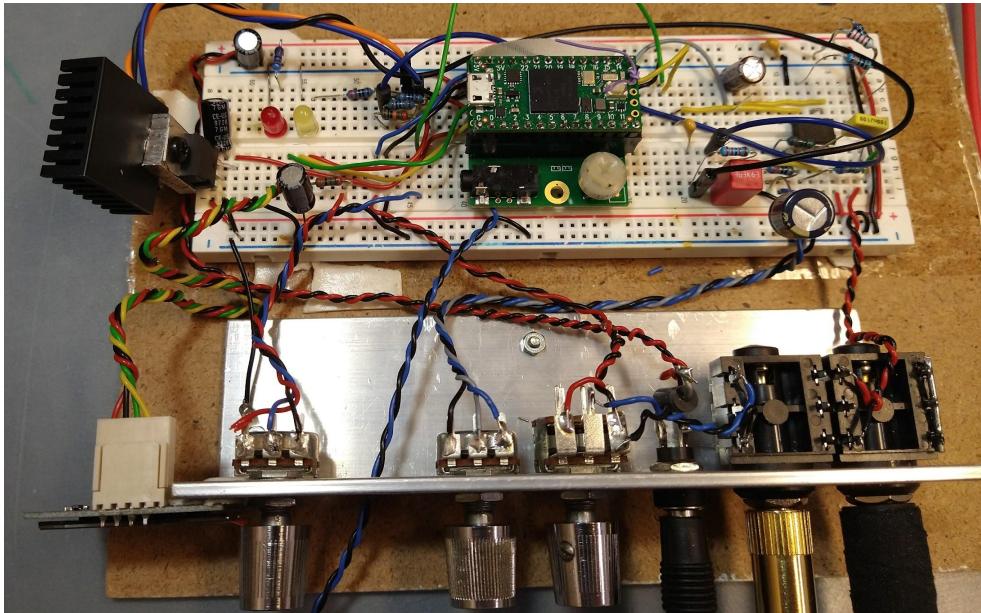
Design af prototypen har foregået iterativt, så der således blev stillet meget simple krav i begyndelsen, for at have sikkerhed om, at de kunne udføres og for at opdele processen i segmenter. Dette er gjort især fordi sværhedsgraden af integrationen af selve rumklangseffekten var meget usikker i begyndelsen. Med tiden er kriterierne selvfølgelig justeret i takt med, at de er blevet opfyldt og forståelsen af materialet og mulighederne for viderebygning er forbedret.

Specifikt er en teensy rimelig nem at designe til, men der er naturligvis aspekter man skal tage højde for. Eksempelvis har audio shieldets ADC en meget inputimpedans<sup>1</sup>, der skal korrigeres for med en buffer, da  $\frac{1}{2}$  til  $\frac{1}{3}$  af volumen ellers ville blive afsat i guitarens pickup, samt en inputspænding 50-100 over hvad guitarer normalt leverer, så forforstærkning er nødvendig. Udover dette er det også relevant at lave en inputbeskyttelse til ADC'en, da forstærkeren ellers potentelt ville kunne beskadige den, da forstærkerens forsyningsspænding vil være forskellig fra codec'ets referencespænding og forsyning.

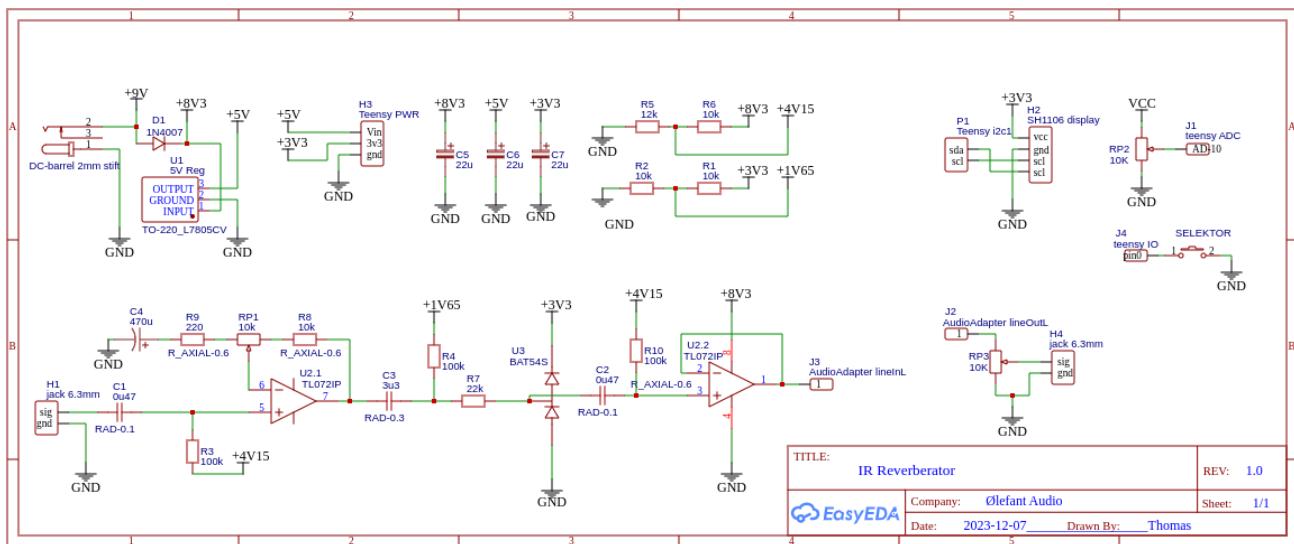
---

<sup>1</sup> SGTL5000 datablad

# Designbeskrivelse



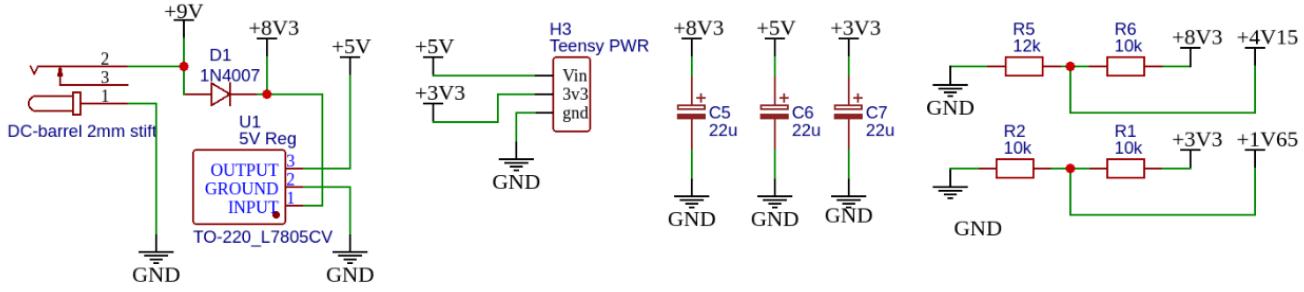
## Hardware



## Generelt om prototypen

Prototypen blev sat op så den nemmeste muligt kunne arbejdes på og udbygges. Derfor foregår alt I/O og interface på et forpanel, mens den elektriske kreds foregår på et breadboard midt på et bræt. En reelt guitarpedal er endnu ikke konstrueret, men der kunne måske være potentiale for det i fremtiden<sup>2</sup>.

<sup>2</sup> I øvrigt er teensier er ikke billige, så der er belæg for at aflevere den tilbage efter eksamen.



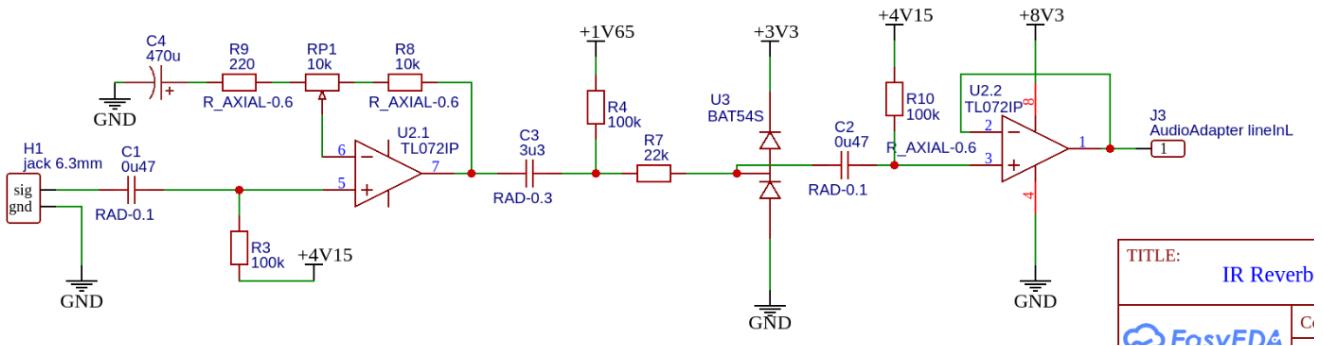
## Strømregulering

Strømforsyninger til guitarpedaler leverer 9V gennem et almindeligt DC jack med 2mm stift, med positiv yderst og nul på stiften.

Derfor bruges en ensretter diode for at beskytte mod omvendt spænding fra en forkert strømforsyning. 9V bruges som forsyningsspænding på forforstærkeren, og da standarden indenfor guitarpedaler er at bruge en TL072, som er en almindelig og meget billig JFET input operationsforstærker, bruges sådan en selvfølgelig. En TL072 er ikke rail-to-rail. Dette vil sige dens output bliver cirka mellem 1.2 og 7,8V så derfor anvendes en 12k modstand og 10k modstand til spændingsdeleren, for at ramme et falskt nulpunkt nogenlunde midt mellem maksimum- og mininumspændingerne.

En teensy kører på 5V, der normalt kommer fra USB, men i dette tilfælde bruges en 5V spændingsregulator. Den indbyggede switchmode adapters støj, ikke har betydning for lydkvaliteten og der er meget større fejlmargin på 5V input end på dens 3V3 rail, så det ville ikke give mening at drive den via en 3.3V forsyning. Nogle ARM microcontrollere starter ikke, hvis spændingen er den mindste smule over 3.3V. Efter egen observation kan microcontrolleren i en Raspberry Pi Pico, eksempelvis ikke tænde hvis forsyningsspændingen er 3.31V.

Fra teensies 3v3 rail laves også en spændingsdeling, for at kunne bruge denne som falskt nulpunkt i det segment af forforstærkeren, der anvendes til inputbeskyttelse for codec'et.

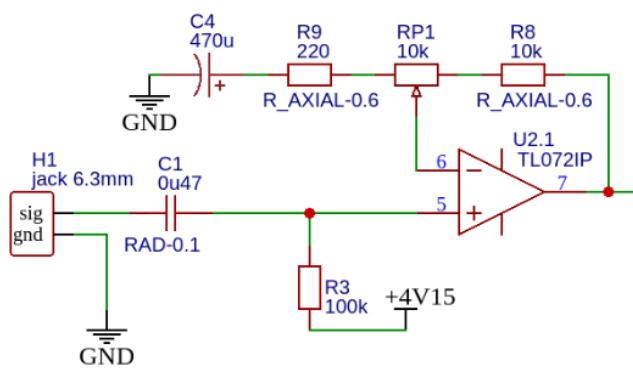


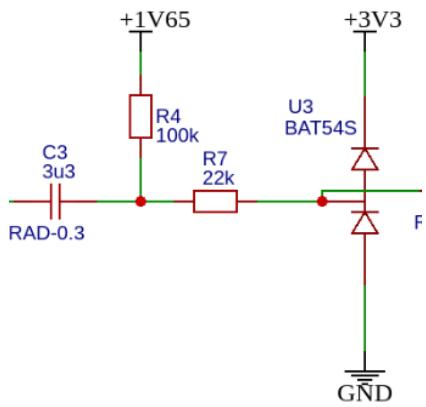
## Forforstærker og inputbeskyttelse

### Forstærkerdel

Forforstærkeren besætår af den ene side af opampen, der anvendes på næsten samme måde som i en distortionpedal, eksempelvis den anden halvdel af gain stadiet i en Boss DS-1, med den undtagelse, at tilbagekoblingskredsen er padded på begge sider, så man får mere kontrol ud af potentiometret, hvilket i forvejen er svært at opnå uden et omvendt logaritmisk potmeter, især når ens kriterie er så specifikt som det er med en 16 bit ADC.

Input koblingskondensatoren er større end den normalt ville være med den enorme inputimpedans på operationsforstærkeren, men da der foregår en del koblinger gennem kæden, der alle fungerer som højpasfiltre, giver det mening at overdimensionere koblingskondensatorerne lidt. Tilbagekoblingen tillader 2 - 100 ganges gain, og koblingskondensatoren i tilbagekoblingen sørger for at blokere jævnstrøm, så offsettet ikke forstærkes. Dette sker fordi kondensatorer yder ekstremt høj modstand, men kan yde meget lav impedans, afhængig af belastningen den denne filter med.





## Inputbeskyttelse

Inputbeskyttelsen begynder med en koblingskondensator, kapacitansen af hvilke er højt, da impedansen i modstanden efter ikke er særligt stor. Før modstanden sættes et nyt offset til signalet. Meningen med modstanden er at afsætte klip i det. Dette er en nødvendighed for at dobbeltdioden, en BAT54s, opfører sig korrekt, da der ikke skal afsættes for meget effekt i dem og for at der kan ske et spændingsfal over modstanden, så der ikke sker mere end de to 0.3V spændingsfald over dioderne fra GND og til 3v3.

Årsagen til modstanden er på 22k Ohm specifikt, er fordi databladet specificerer, at man kan minimere diodernes fremadspænding når strømstyrken er under 0.1mA.

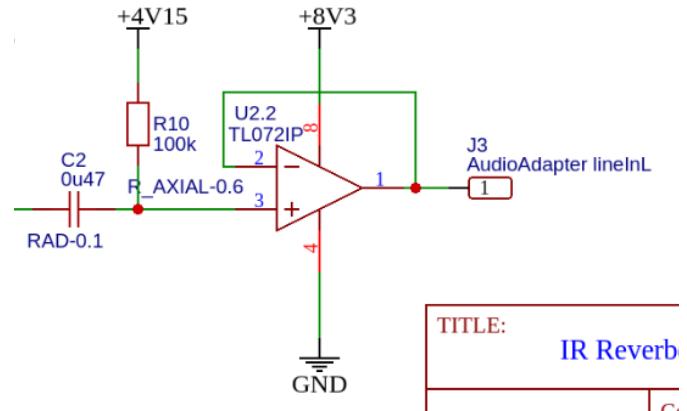
$$0.24V / 0.00001A = 24k\Omega \text{, approksimeres til } 22k\Omega$$

Årsagen til den ikke er endnu større, er for at undgå unødig støj. Det skal dog nævnes at selvom fremadspændingen kan nå ned på 240 mV, er det kun ved peaks over 3333Hz, men det skader ikke alligevel at forsøge at klemme fremadspændingen ned.

## Outputbuffer

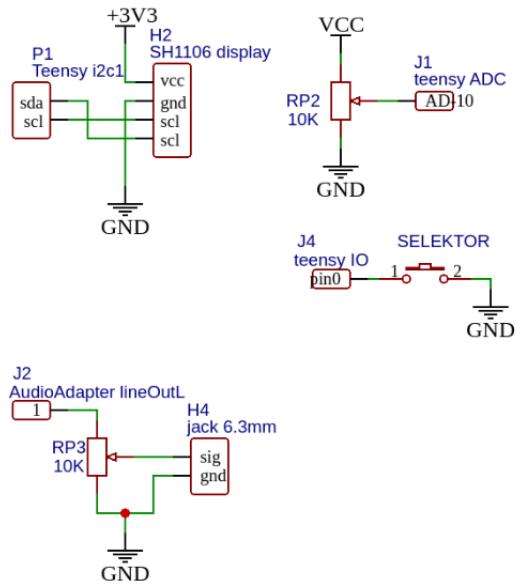
Outputbufferen afkobles og offsettes på samme måde som forstærkningstrinnet. Buffering er en nødvendighed, da codec'et på audio adapteren har en inputimpedans på 29k Ohm, hvilket ville betyde et tab på  $22k/(22k+29k) = 0.43\%$  af signalet i beskyttelseskredsen, hvis bufferen var undladt.

Dette svarer til maksimalt spændingssving på 2.22V frem for de ønskede ~3.1V som codec'et kan aflæse.



## Ydre enheder

Til styring, er der som nævnt et potentiometer, der regulerer forstærkningen i forstærkeren. Udover dette, er der et potentiometer til udgangsvolumen, hvilket er lidt redundant når der ikke er en bypass funktion i prototypen. Ydermere, er der et potentiometer til regulering af effektens styrke, der er sat op mellem 3.3V og gnd med output til teensiens adc10 ben, og en knap til interaktion med menuen, der renderes på en OLED skærm.



# Software

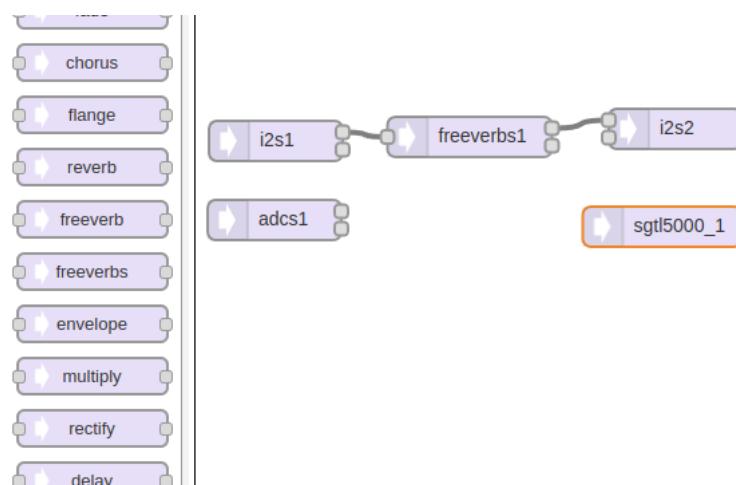
## Biblioteker

### Teensy Audio

Lydbibliotek er bygget op omkring nogle objekter, der tillader input, output, processering, og interfacing mellem dem. Såsom I2S input og output (AudioInputI2S, AudioOutputI2S), som bruges i dette projekt, afspilning fra SD, og et hav af effekter som eksempelvis bitcrush, delay, ensretter og samt nogle tredjepartseffekter som freeverb. Effekten der bruges i dette projekt er en tredjepartseffekt ved navnet Teensy-FFT-Convolution-Filter af Brian Millier, fra Teensy forummet. Til I/O og effekter til at hænge sammen, er der et modul, der gør netop dette, AudioConnection, der oftes vil initialiseres under navnet PatchCord, efterfulgt af et nummer, da der ofte vil være flere end et.

```
// GUItool: begin automatically generated code
AudioInputI2S           i2s1;
AudioFilterConvolutionUP convolution;    // Uniformly-partitioned
                                         convolution filter
AudioOutputI2S           i2s2;
AudioConnection           patchCord1(i2s1, 0, convolution, 0);
AudioConnection           patchCord2(i2s1, 1, convolution, 1);
AudioConnection           patchCord3(convolution, 0, i2s2, 0);
//AudioConnection          patchCord4(convolution, 1, i2s2, 1);
AudioControlSGTL5000     codec;
// GUItool: end automatically generated code
```

Disse opsætninger kan nemt genereres med teensy's audio design tool



depends on Teensy to provide all I2S clocks. This object controls how the SGTL5000 will use those I2S audio streams.

## Functions

These are the most commonly used SGTL5000 functions.

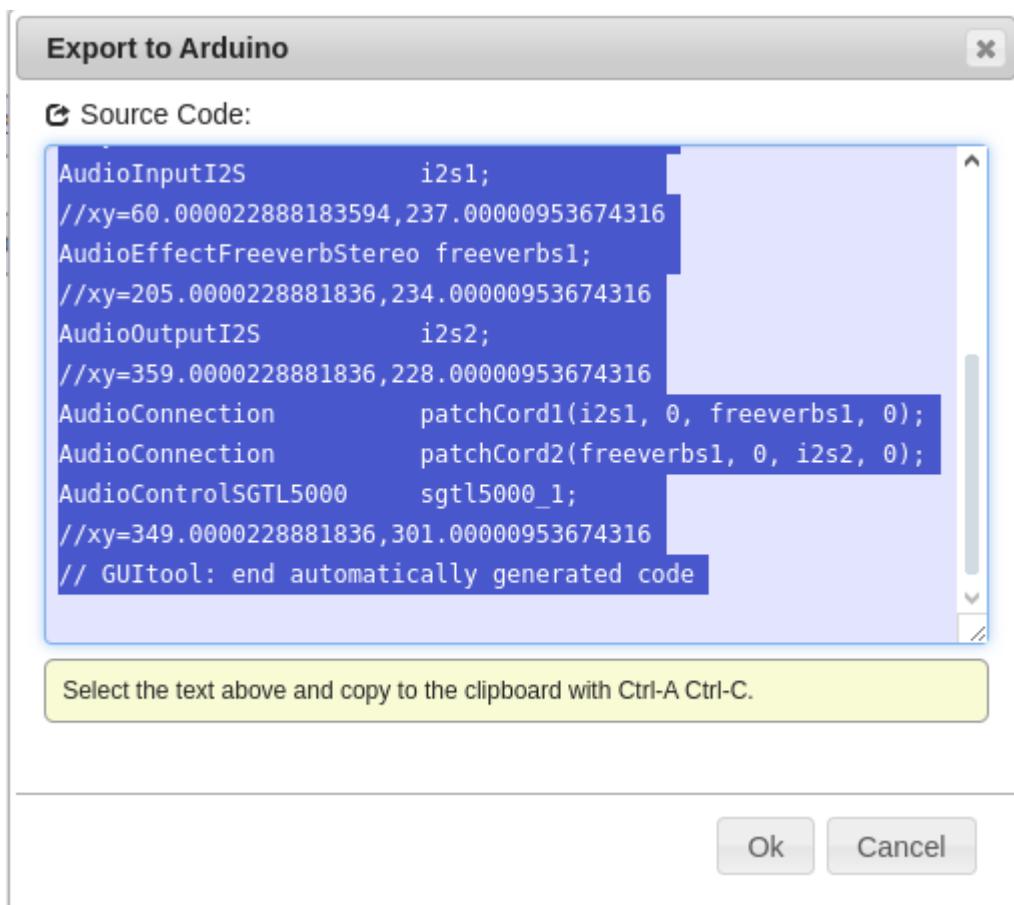
**enable();**

Start the SGTL5000. This function should be called first.

**volume(level);**

Set the headphone volume level. Range is 0 to 1.0, but 0.8 corresponds to the maximum undistorted output for a full scale signal. Usually 0.5 is a comfortable listening level. The line level outputs are *not* changed by this function.

**inputSelect(input);**



I dette projekt anvendes kun den venstre kanal, så der gøres brug af PatchCord objekter, objekt for convolutioneffekten, SGTL5000 og i2s.

Al opsætning af teensy biblioteker er taget fra eksemplet fra convolutionbibliotek.

## SD

Til loading af filer fra SD-kortet anvendes to File objekter. Et der huser root mappen på SD kortet, som filerne ligger i, og et der huser en lydfil ad gangen.

Vigtige metoder for SD objektet er:

`SD.open("/sti/til/mappe/eller/fil");` der åbner en mappe eller fil i et objekt,  
`File.openNextFile();` der åbner næste i et objekt i et nyt objekt,  
`File.name();` der returnerer filnavn, som bruges som String for nemmer at kunne parse det i funktioner og metoder,  
`File.rewindDirectory;` der starter forfra med første fil i mappen.  
`datafile.seek(0);` der flytter cursor til bestemt byte i fil  
`datafile.read(arrayName, length);` Der loader data fra til over i array, bemærk mængden af bytes der læses per trin kommer an på datatypen i arrayet. 16bit array kræver to bytes.  
`Length` er antal bytes der læses i alt, ikke korrigeret efter typen i arrayet.

## Adafruit OLED Display

For at have en menu, er et monokromt OLED display anvendt. Displayet har en oplosning på 128x64, hvilket tillader 8 linjer i højden og 32 karakterer i længden ved det mindste skriftsstørrelse og halvdelen ved størrelse 2.

For at interface med displayet, der bruger en SH1106 styringschip, over i2c, bruges adafruits SH110X driver. Men for overhovedet at kunne interface med displayet, skal man kende dets i2c adresse, hvilket kan findes med adafruits egen i2c scanner.

```
for(address = 1; address < 127; address++)
{
    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    WIRE.beginTransmission(address);
    error = WIRE.endTransmission();
    if (error == 0)
    {
        Serial.print("I2C device found at address 0x");
        if (address<16) Serial.print("0");
        Serial.print(address,HEX);
        Serial.println(" !");
        nDevices++;
    }
    else if (error==4)
    {
        Serial.print("Unknown error at address 0x");
        if (address<16) Serial.print("0");
        Serial.println(address,HEX);
    }
}
```

Som kan ses i udklippen, cykler scanneren reelt bare gennem de 128 mulige adresser, og melder tilbage om de er optagede. Resten af programmet findes i bilag.

Displayets adresse viste sig at være 0x3C. Udover dette skal man også sætte

De vigtigste funktioner i biblioteket er

Adafruit\_SH1106G display = Adafruit\_SH1106G(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire,  
OLED\_RESET);, der initialiserer objektet

display.begin(0x3C, true);, der starter i2c med displayet

display.clearDisplay(); der nulstiller bufferen, der kan vises på displayet med

display.Display(); der printer bufferen på displayed,

display.setTextSize(3); sætter teksstørrelse

display.setTextColor(SH110X\_WHITE, SH110X\_BLACK); tekstfarve og baggrund

display.println(F("SELECT")); printer tekst til bufferen, F() er en formateringsfunktion

display.setCursor(64,40); flytter cursoren

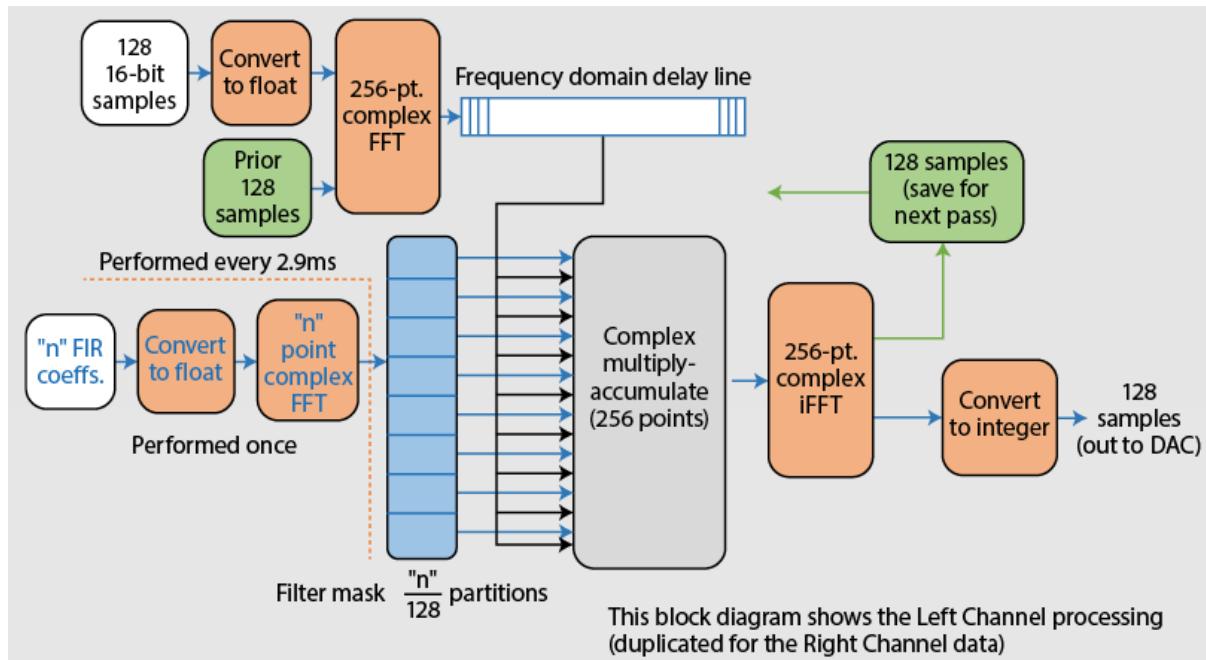
## Brian Millier's Teensy-FFT-Convolution-Filter

Hjertet af dette projekt består af et bibliotek skrevet af Brian Millier, der har udgivet det på teensy forummet og github. Biblioteket er baseret på et lignende et, af brugeren Frank DD4WH, som er lavet ud fra en artikel af Warren Pratt. Selvom Frank's program egentlig er det program på teensy, der tillader de længste rumklange, er koden er skrevet inline, så det gav meget mere mening at anvende Brians i stedet, da hans bibliotek er wrappet til at virke som audio objekt i teensys udviklingsmiljø.

Den måde effekten virker på i grove træk, er ved først og fremmest at lave FFT analyser af en rumklangsbuffer, for at have det i frekvensdomænet, da dette hjælper med at skære kraftigt ned på antallet af operationer. Hvis man foretager direkte foldninger i tidsdomænet, sample for sample, stiger antallet af operationer lineært, hvorimod det kun stiger logaritmisk, hvis man bruger frekvensdomænet. Dette effektiviseres yderligere ved at partitionere éns buffere.

Når rumklangsbufferen er konverteret til frekvensdomænet, gemmes en 128 sample buffer fra i2s codec'et, der er teensys standardstørrelse, indtil en anden 128 sample buffer er optaget. Herefter foretages også FFT på den samlede inputbuffer. Derefter foretages partitionering, hvor bidderne af frekvensspektrerne foldes og akkumuleres. Til slut foretages en invers FFT og halvdelen af 256 sample outputbufferen outputtes og den anden halvdel gemmes til næste omgang.

Som det skarpe øje bemærkede i afsnittet om teensy audiobibliotek, er det faktisk et bibliotek, der kører "stereo" med højre og venstre kanal, der processeres hver for sig, men mono i impulsresponsfilerne. Det er dog uden betydning for dette projekt. Ifølge forfatteren kører det dog lige effektiv i stereo og mono, takket være complex FFT fra CMSIS<sup>3</sup>.



<sup>3</sup><https://github.com/bmillier/FFT-Convolution-Filter-Uniformly-partitioned?fbclid=IwAR0l8WxR3EegE8hAD9x6FzjWdg5u8CErnvXQyW4ZTmzWQz2H6dPqLmBLR6I>

Der er dog ingen gevist ved at forsøge at køre foldningerne i mono, da objektet ikke starter, når der kun er et input. Det kører dog ubeklagelig med kun et output.

Da teensien's mikrocontrollers SRAM er begrænset til 1MB partitioneret i to 512 kB blokke, DTCM som er generel arbejdshukommelse, og OCRAM, som primært er til større objekter, sætter dette begrænsingen for længden af impulsresponser til 400ms.

I modsætning til Franks inline-program, lykkedes det ikke Brian at fordele dataen fuldstændigt ligeligt mellem de to blokke, grundet fejl under kompilering.<sup>4</sup>

Som nævnt, er Brians bibliotek wrappet i et modul, der foretager alt relateret til foldninger, hvilket gør det muligt at arbejde med. I praksis skal man bare specificere et array med det korrekte navn og definere et antal 'taps' for impulsresponsen, og derfra kan effekten opnås, ved at lade resten af hans eksempelprogram køre.

En dybdegående beskrivelse findes på hans github<sup>5</sup> og hans artikel hos Circuit Cellar<sup>6</sup>.

Udover at fungere som rumklang og kabinetsimulator, kan denne effekt også bruges til såkaldte FIR filtre, hvilket står for Finite Impulse Response, der tillader ekstremt skarpe cutoffs, og meget simpel betjening da filtret opstår ved at folde en input buffer med et filtreret frekvensspektrum. Udskift FIR filen, og man har en ny filterrespons.

Selvom dette er meget interessant, ligger det dog uden for projektets spændvidde.

---

<sup>4</sup><https://github.com/bmillier/FFT-Convolution-Filter-Uniformly-partitioned?fbclid=IwAR0l8WxR3EegE8hAD9x6FzjWdg5u8CErvXQyW4ZTmzWQz2H6dPqLmBLR6I>

<sup>5</sup><https://github.com/bmillier/FFT-Convolution-Filter-Uniformly-partitioned?fbclid=IwAR0l8WxR3EegE8hAD9x6FzjWdg5u8CErvXQyW4ZTmzWQz2H6dPqLmBLR6I>

<sup>6</sup> <https://circuitcellar.com/research-design-hub/projects/cabinet-simulator-stomp-box-for-guitarists/>

# Implementation

## Lag 1 - Globalt

Først og fremmest er der en hulens masse imports og nogle globale typer, der skal til for at det hele kører som det skal.

De inkluderer:

```
#include <string>
#include <cstring>
```

String-bibliotekerne anvendes til Stringoperationer, der er nødvendige for genkendelse af filtyper i brugerinterfacet, for at sikre at der ikke kan loades forkerte filtyper.

```
#include <Audio.h>
#include <Wire.h>
#include <SPI.h>
```

Disse tre er standard for teensy audio, da Audio.h indeholder alle effekterne, som Convolutionbiblioteket's header og cpp filer er tilføjet til. Wire består af i2c og i2s, som begge er anvendt i programmet. SPI er muligvis til opsætning af codec, eller andet på audio adapteren.

```
#include <SD.h>
```

SD anvendes til loading af SD kort mapper, filer fra SD, filnavne, og aflæsning af filer på SD kort.

```
#include <Adafruit_GFX.h>
#include <Adafruit_SH110X.h>
#define i2c_Address 0x3c
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino reset pin)
```

Til displayed anvendes to adafruit-biblioteker, opløsningen defineres, og reset pin specificeres.

```
#include <EEPROM.h>
uint8_t eepAddr; //EEPROM address for reset check
EEPROM anvendes til at vide om intro skal afspilles.
```

```
#define selektor 0 //button pin
#define mixPot 24 //pot for response strength
```

Selektor er knappen på prototypen, mixPot bruges til regulering af styrken af impulsresponsen

```
#include "loader.h"
#include "dispHandler.h"
#include "utils.h"
#include "menu.h"
```

Disse fire er skrevet specifikt til projektet og håndterer SD relaterede operationer, displayfunktionalitet, debounce og andre funktioner, samt brugerinterfacet, der er wrappet i en funktion.

```
const int16_t arrLen = 18001; //array length with padding
float impulse_response[arrLen]; //defining impulse array and int array for 16 bit
signedint file
const int nc = 18000; // number of taps for the FIR filter
#include "convolution.h"
```

Disse fire, er til convolutionbiblioteket. Arraylængden er en variabel, der bruges i loaderobjektet.

```
// GUItool: begin automatically generated code
AudioInputI2S           i2s1;
AudioFilterConvolutionUP convolution; // Uniformly-partitioned convolution filter
AudioOutputI2S            i2s2;
AudioConnection           patchCord1(i2s1, 0, convolution, 0);
AudioConnection           patchCord2(i2s1, 1, convolution, 1);
AudioConnection           patchCord3(convolution, 0, i2s2,0);
//AudioConnection          patchCord4(convolution, 1, i2s2,1);
AudioControlSGTL5000      codec;
// GUItool: end automatically generated code
```

“Kablerne” der får delene til at spille sammen. Se evt. eurorack for reference.

Udover disse, er der også en række utilities, der skal til for convolutionbiblioteket virker, som opsætning af en det variabler og arrays i specifikke ram partitioner, og funktioner til at sætte i2s frekvens og en funktion, der printer info om flexRAM når setup til convolutionbiblioteket er fuldført. Se convolution.h i bilag for mere info.

## Lag 2 - main.ino

Alt kode herfra vil være kogt ind, og overflødige linjer skåret fra. De uredigerede filer kan findes i bilag.

```
void setup()
{
    Serial.begin(115200);
    pinMode(selektor, INPUT_PULLUP);
    while(!(digitalRead(selektor))) ;
```

Denne linje bruges som beskyttelse mod at knappen stadig er holt inde efter reset. Den blokerer indtil knappen er sluppet.

```
    menu();
```

Menuen er hvor magien sker, mere om det senere

```
setupSGTL5000();
AudioMemory(10);
set_arm_clock(600000000);
setI2SFreq(SAMPLE_RATE);
convolution.begin(0,0.20,*fftout,nfor); // turn off update routine until after filter
mask is generated, set Audio_gain=1.00 , point to fftout array, specify number of
partitions
convolution.impulse(impulse_response, maskgen, nc);
flexRamInfo();
Serial.println();
Serial.print("AUDIO_BLOCK_SAMPLES: ");      Serial.println(AUDIO_BLOCK_SAMPLES);
```

Alle disse linjer er fra convolutionbiblioteket

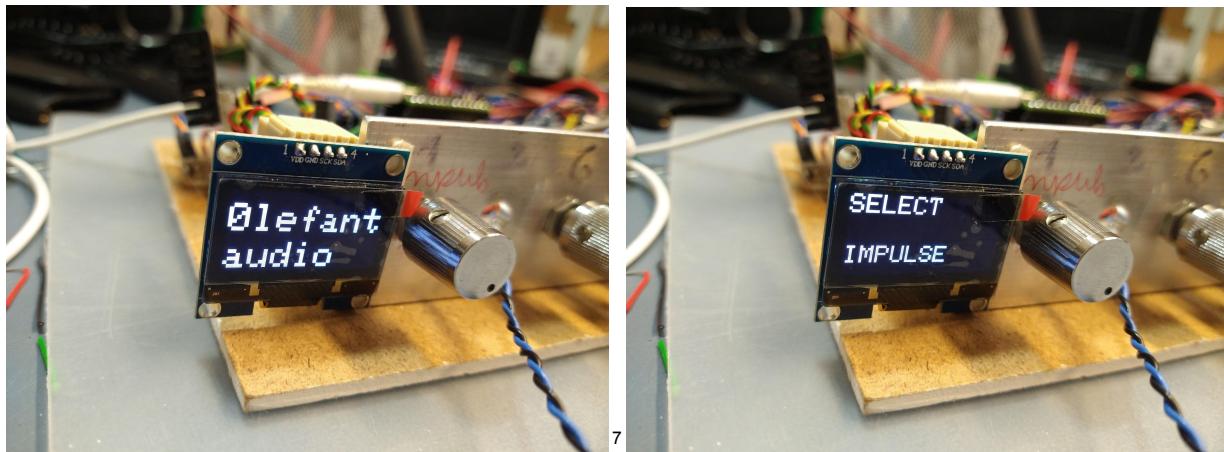
```
}
```

```
void loop() {  
    yield();  
yield funktionen er den gentagende funktion fra convolutionbiblioteket  
    if(!(digitalRead(selektor))) //reset function  
    {  
        EEPROM.update(eepAddr, 0); //Sets value to 0 to prevent intro playing next cycle  
        SCB_AIRCR = 0x05FA0004; //reset teensy  
    }  
}
```

Reset funktionen tjekker om selektor-benet skulle være trukket ned, som sker når knappen bliver aktiveret, efter hvilke værdien 0 bliver skrevet i en byte i EEPROM, der fortæller at introen ikke skal gentages i næste cyklus, mere om dette senere.

Det følgende kald sætter et registers værdi således at mikrocontrolleren resetter.

## Lag 3 - menu.h



For brugeren vil interfacet virke todelt, da menufunktionen indeholder en intro og selve menuen.

```
void menu()
{
    DispHandler disp = DispHandler();
    Loader load = Loader();
    Debouncer debounce = Debouncer(500);
```

Først og fremmest initialiseres tre objekter, en wrapper for display funktionalitet og håndterer alt, der har med skærmen at gøre. Loader Objektet, hvor File objekter parses som parametre i dets metoder, der bruges til at returnere navne, og loade responser. Sidst, er der et debounce objekt, der håndterer tryk og slip og tryk og hold funktionalitet.

```
byte eepVal = EEPROM.read(eepAddr); //loads value from EEPROM address
if (eepVal)      disp.printIntro(); //If value is set, intro will play
EEPROM.update(eepAddr, 255);        //Sets value so intro plays if teensy is reset
via power cut, but not via button
```

Som det næste aflæses en byte fra EEPROM, for at tjekke om introen skal køre. Hvis værdien er over 0, vil introen køre. Derefter opdateres værdien til 255, hvis den ikke allerede er sat, for at sikre at introen spilles, hvis pedalen bliver resat ved at slukke for strømmen. Tilfældet hvor introen ikke spiller, er fremført i det forrige afsnit, netop når pedal bliver resat via knappen, fordi det ville være spild af tid at vente på en intro, hvis man vil prøve en anden rumklang eller lave en ændring i dens styrke.

```
if(SDchk()) disp.error(0);
```

Tjekker om SD kort er sat ind, og er læsebart. Ellers gives fejl 0, SD fejl på display.

```
File root = SD.open("/");
File datafile = load.fileFetch(root);
```

Efter et succesfuldt SD tjek, vil to File objekter initialiseres, det første som root mappen på SD kortet, for at tillade at kunne åbne en fil derfra, hvilket sker i den næste linje ved hjælp af loaderobjektet.

<sup>7</sup> Absolut ikke inspireret af en bestemt person



```

String filename;
uint8_t state = 1;
bool nextFile = 0;
bool next = 0;
bool multi = 0;
bool clen = 0;
uint8_t debbie;      //Bruges i switch sætning
                     //Bruges i første case for at indikere, at næste fil skal loades
                     //Bruges i multirespons undermenu til at styre cursor
                     //Bruges til fejlmeddeelse, hvis tom respons vælges i multirespons
                     //Bruges til styring af now playing:
                     //Return variabel fra debounce objekt

```

debbie = 0: ingenting handling

debbie = 1: trykket, holdt og sluppet, holdt under 500 ms

debbie = 2: trykket, holdt og sluppet, holdt over 500 ms

```

while(state != 0)
{
    switch (state)
    {
        case 1: //fetching name from sd
            if(nextFile) datafile = load.fileFetch(root); //if tjek for at undgå at skippe første fil
            if(datafile)
            {
                filename = datafile.name();
                disp.printFilename(filename); //displays datafile
            }
            if(!datafile) disp.printFilename("No\n selection");
            break;
    }
}

```

Funktionen af den første case, er at loade et filnavn og vise det på skærmen



```

case 2: // button unpressed mode
    debbie = debounce.debounceChk();
    if (debbie == 1)
    {
        if(!nextFile) nextFile = 1; // click
        state = 1;
    }
    else if (debbie == 2) // holds
    {
        if(!datafile)
        {
            if(multi) disp.theMessage(); // simple error message, don't look into it
            load.emptyResponse();
            clen = 1;
            state = 0;
        }
        else
        {
            uint8_t floatReturn = load.floater(datafile);
            if(floatReturn > 0) disp.error(floatReturn);
            state = 3;
        }
    }
    break;
}

```

Funktionen af den anden case, er at tjekke status af knappen. Hvis .debounceChk metoden returnerer 1, bliver nextFile sat, og state = 1. Således overskriver .fileFetch datafile med den næste fil i root, for at den næste fil er i fokus i menuen. Hvis knappen holdes nede længe nok, vil debounce returnere 2. Derefter tjekkes der, om en fil er loadet. Hvis datafile objektet ikke er defineret, vil en tom respons blive brugt og while løkken lukket. Hvis et objekt er defineret, bliver dens indhold loaded ind i impulse\_response arrayet gennem load.floater() metoden. floatReturn bruges til at håndtere fejltyper fra .floater(), som er den, der loader filer til arrayet. Der går derefter til næste case.



```

case 3:
    disp.next(next);
    state = 4;
    break;

```

Denne case får display objektet til at rendere undermenuen. booleannen next er som udgangspunkt 0, så den valgte mulighed er nej, når man bliver spurgt om, om man vil loade endnu en respons. I dette segment antager vi at det ikke er ønsket. Men det at skifte mellem dem, sker ved at trykke på knappen.



```

case 4: // load another impulse
    debbie = debounce.debounceChk();
    if (debbie == 1)
    {
        next = !next;
        state = 3;
    }
    else if (debbie == 2) //hold
    {
        if (next) // if yes is selected
        {
            multi = 1;
            next = 0;
            state = 1;
        }
        else //if no selected
        {
            state = 0;
        }
    }
    break;
}

```

Case 4 virker nogenlunde på samme måde som case 2, ved at tjekke returværdier fra debounceChk metoden. Hvis knappen er holdt under 500ms, bliver next omvendt, og den valgte mulighed mellem yes/no skifter. Hvis den er holdt længere end 500 ms, slutter while løkken, hvis der er valgt nej, ellers fortsætter den. Mere om det i afsnit om multirespons.

```

}
}
```

```

Serial.println("while loop broken");
if(clean) disp.nowPlaying("clean/dry"); //if clean variable is set, it will always
display clean, as selecting clean response overwrites any previous loaded responses
else
{
    if(multi)disp.nowPlaying("multiple"); //if multiple responses are in use
    else disp.nowPlaying(filename); //if only one response is loaded
}
if(datafile) datafile.close(); //DATAfile is close if it exists
root.close(); //same with root
}

```

Efter while løkken er brudt, bliver der tjekket hvilket typer respons der er valgt, hvis bruger har valgt no selection vil en tom respons loades, og while løkken bliver tvunget til at lukke uden brugerinput, og displayet vil skifte til now playing: clean. Dette sker også hvis man vælger no selection efter at have loaded andre.

Hvis en respons er valgt, vil filnavnet vises i now playing:, og hvis der er tale om multirespons, vil der stå multiple.



Efterfølgende bliver SD objekterne lukket, menu funktionen slutter, med hukommelsen frigivet, og setup forstætter, hvor det modificerede impulse\_response array bruges.

## Lag 3/4 - utils.h

Debounce objektet er kritisk for programmet fungerer, da dette ikke kun håndterer debouncing, men også at tjekke tiden knappen har være holdt.

```
class Debouncer
{
private:
    uint8_t debTime = 50;    //50ms er måske en lang debounce, men det er med henblik på
                            //en fodkontakt
    uint16_t timeChk = 500;  //Det samme kan sites om holde tiden
    bool state1 = 0;
    long currTime = 0;       //current, opdateres løbende
    long lastTime = 0;       //Opdateres når deboucne kriterie er opfyldt
    long downTime = 0;       //Tidspunkt, kontakten er trykket ned
    long holdTime = 0;       //Tid holdt nede, opdateres ved slip
    bool but_state_curr = 1;
    bool but_state_last = 1;
public:
    Debouncer(uint16_t compTime)
    {
        timeChk = compTime; //kunne blive til if(compTime) timeChk = compTime; for at
                            //tillade void parameter
    }
}
```

Init opdaterer compTime, da det kunne være oplagt at have den mulighed, hvis der skulle være flere knapper, eller hvis objektet skulle genbruges i andet projekt.

```

uint8_t debounceChk()
{
    currTime = millis();           //current time
    but_state_curr = digitalRead(selektor); //checks whether button is up or down
    uint8_t returnable = 0;
    if(but_state_last == but_state_curr) lastTime = currTime; //resets timer if
button states are the same
    switch(state1)      //sub states
    {
        case 0:
            if(but_state_last && !but_state_curr && currTime - lastTime > debTime)
//checks if but down and prev is up and if debounce time is exceeded
            {
                but_state_last = but_state_curr;      //updates prev state
                downTime = millis();     //starts timer for press and hold
                state1 = 1;
            }
            break;
}

```

currTime opdateres hver gang metoden kaldes, det samme med læsning af status på kontakten, returnvariablen bliver sat til 0 og der tjekkes om knap statusserne er ens. Hvis de er, resettes lastTime. Bid mærke i at den ikke resettes, hvis de er forskellige.

Hvis knappen skulle træffe at være nede, og det forrige stadie skulle være oppe, samt forskellen mellem currTime og lastTime er stor nok, bliver trykket på knappen anerkendt, og status1 skifter.

```

    case 1:
        if(!but_state_last && but_state_curr && currTime - lastTime > debTime)
//if last down and current up and debounce time is exceeded
        {
            but_state_last = but_state_curr;
            holdTime = millis();

```

Her sker det samme, bare omvendt, også debounced

```

            if (holdTime - downTime < timeChk) //if
the downtime is shorter than .5s, the file is shifted to the next one
            {
                returnable = 1;
                state1 = 0;
            }

```

Skulle knappen være holdt i mindre tid end timeChk, vil returnable sættes til 1 og status1 til 0.  
Dette svarer til knappen er trykket, men ikke holdt.

```

        else //if held long
enough, else is broken
        {
            returnable = 2;
            state1 = 0;

```

Ellers sættes returnable til 2 og status 0.  
Dette svarer til knappen er trykket og holdt.

```

        }
    }
    break;
}
return returnable;
}
};
```

Udover debounce objektet, er der også en setupfunktion til codec'et, der slår det til, sætter inputs, høretelefonudgangsniveaueret samt input og output levels til 3.12V pkpk, for at være sikker på at makse det dynamiske område ud, så man kan få et optimalt signal/støj forhold.

Spændingssvinget er omkring 1V pkpk fra standard, og efter test viser det sig at der er en betydeligt mængde støj på den indstilling, men heldigvis næsten ingen ved 3.12V.

Se bilag for koden.

## Lag 4 - displayHandler objekt

DispHandler objektet er sat op således, at alt, der vedrør displayet, er defineret inde i objektet, ligesom med debounce objektet.

```
private:  
    Adafruit_SH1106G display = Adafruit_SH1106G(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire1,  
    OLED_RESET);  
  
public:  
    DispHandler()  
{  
        if (!display.begin(i2c_Address, 0x3C))  
        { // Address for 128x64  
            Serial.println(F("SH1106 allocation failed"));  
            for (;;) ; // Don't proceed, loop forever  
        }  
    }  
}
```

Metoderne i objekt er også ret ligetil.

```
void printIntro(void)  
void printFilename(String name)  
void next(bool yesnt)  
void nowPlaying(String name)  
void error(uint8_t errType)
```

De er alle bygget op omkring metoderne nævnt i afsnittet om SH1106 driveren, og udover introen, der afspiller en sekvens af nogle strenge, er printFilename() og nowPlaying() bygget op omkring en parset streng, og next() og error() omkring en værdi, der er parset.

```
void printFilename(String name) //prints "filename" and a string  
of the file name  
{  
    display.clearDisplay();  
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text  
    display.setTextSize(2); // Normal 1:1 pixel scale  
    display.setCursor(0, 0); // Start at top-left corner  
    display.println(F("filename: "));  
  
    display.setCursor(0,32); // Start at top-left corner  
    display.println(name);  
    display.display();  
    display.clearDisplay();  
}
```

Funktionaliteten af next() ser således ud

```
void next(bool yesnt) //prints "filename" and a selection
{
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw with white text on black background
    display.setTextSize(2); // Normal 1:1 pixel scale
    display.setCursor(0, 0); // Start at top-left corner
    display.println(F("Stack more\nResponses?"));

    display.setCursor(0,40); // Start at top-left corner
    if(yesnt) display.setTextColor(SH110X_WHITE, SH110X_BLACK);
    else display.setTextColor(SH110X_BLACK, SH110X_WHITE); // Different colors for yes/no
    display.println(F("NO"));

    display.setCursor(64,40); // Start at top-left corner
    if(!yesnt) display.setTextColor(SH110X_WHITE, SH110X_BLACK);
    else display.setTextColor(SH110X_BLACK, SH110X_WHITE); // Different colors for yes/no
    display.println(F("YES"));

    display.display();
    display.clearDisplay();
}
```

Hvor valget af forgrund og baggrundsfarven indikerer hvilken mulighed, der er valgt, og hvor muligheden er styret af next booleannen fra menu().

Beskeden i fejlhåndtering er styret af fejlkoden, der bliver parset i metoden.

Fejlkode 0 kommer fra starten af menu() hvis SD kortet ikke er tilgængeligt, ellers kommer de andre fejlkoder fra floater metoden i loader objektet.

```
display.setCursor(0, 16); // Start at top-left corner
if (errType == 0) display.println(F("Error #0:\nSD inaccessible"));
if (errType == 1) display.println(F("Error #1\nIn header bits 8-11:\nUnknown filetype"));
if (errType == 2) display.println(F("Error #2\nIn header bit20/34:\nfile not 16bit PCM"));
if (errType == 3) display.println(F("Error #3\nIn header bit22:\nfile not mono"));
display.display();
EEPROM.update(eepAddr, 0); //Sets value to 0 to prevent intro playing next cycle
while(1) if(!(digitalRead(selektor))) SCB_AIRCR = 0x05FA0004; //reset teensy
```

EEPROM opdateres naturligvis også efter en fejl, så brugeren ikke behøver at vente på introen spiller igen.  
Hele objektet kan findes i bilag.



## Lag 4 - SDHandler objekt

SDhandleren er nok den del af projektet, der har været der længst og set mest udvikling. Det startede som en ydmig funktion ved navn floater(), der kunne loade data fra .RAW filer ind i impulse\_response arrayet, konverterer fra 16bit int til 32 bit float, og kun loadede fra filer med navnet "0.raw". Efterhånden er den blevet wrappet og udvidet til at loade filer både af typen .RAW og .WAV, hvilket gør den meget mere praktisk, især fordi der ikke er noget kontrol over hvad .RAW filer egentlig indeholder. Ud af denne voksede den private metode typeFetch, der kun bruges i floater(), som bruges til at melde tilbage om filtype, .raw eller .wav, og i tilfælde af .wav tjekker den også at dataen i filen er korrekt, ved at læse segmenter af headeren. Det er også her fejlkoderne bliver genereret.

```

int floater(File datafile)//, bool first)      //floates int16 wav PCM/raw file to
normalised float array
{
    if (datafile)    // if file is open
    {
        int len; //impulse length
        float mix = analogRead(mixPot); //Recording mix
        mix = map(mix, 0.0,1023.0,0.0,1.0);           //remap from int10 to 0 - 1 Float32
        std::string name = datafile.name();
        int16_t *SD_response = new int16_t[arrLen]; //dynamic array that can be deleted
        uint8_t type = typeFetch(name, datafile); //0: unknown, 1: Raw int16, 2: wave
        int16, 3: wave int16 stereo, 4: wave F32, 5 wave F32 stereo

```

I den første del bliver eksistensen af filen tjekket, hvorefter en int bliver defineret til at holde længden af file til senere. Derefter bliver en float defineret til at holde multiplikationsfaktoren, der styrer styrken af effekten. Den kommer fra adc'en som integer, der er trukket fra 12 bit til 10, som standard i arduino, hvorefter den remappes til en værdi mellem 0 og 1. Derefter defineres en speciel type streng, der kan søges i, samt et sletbart dynamisk array. Dette er et levn fra før floater() blev modulariseret. Til slut hentes typen gennem typeFetch.

```

if(type == 10) return(1);
if(type == 11) return(2);
if(type == 12) return(3);

```

Hvis svaret fra typefetch er et af disse, slutter floater() med fejlkode 1,2 eller 3.

```

if (type == 1)      len = (datafile.size()-44)/2;      // int16 wav
else if (type == 0) len =  datafile.size()/2;          // int16 raw

```

Hvis filen er af typen .wav bliver længden af wave's standarheader trukket fra den fulde længde. Bemærk at længden deles med 2 fordi .size metoden returnerer længden i en-byte størrelser, hvor vi ønsker længden i to-byte størrelser, 16 bit integers.

```

if(len > arrLen) //checks if file is too large for memory to handle
{
    Serial.print("len too large: "); //error msg
    Serial.println(len);
    Serial.print("will be truncated to 0.4s");
    len = arrLen;      //truncates file to 18000 samples
} else
{
    Serial.print("len: ");
    Serial.println(len);
}
if(type == 1) datafile.seek(44);
if(type == 0) datafile.seek(0);
datafile.read(SD_response, len*2); //reads appropriate amount of data from file
in bytes(not ints)
datafile.seek(0); //reset datafile, may be redundant

```

Herefter korrigeres for filer, der er længere end den maksimale arraylængde ved at trukke læsning til den maksimale længde. Dernæst sættes cursoren på wavefiler til 44, hvilket normalt er den første databit, mere om det i afsnit om .typeFetch(). Derefter aflæses datafil til 16bit int arrayet.

```

float temp;
if(!multi)
{
    for(int x = 3; x < len; x++)
    {
        temp = SD_response[x];
        temp *= mix;
        temp /= 32768;
        impulse_response[x] = temp;
        oldLen = len;
        multi = 1;
    }
    Serial.println("first impulse loaded");
}

```

Her laves en float til midlertidig data og der tjekkes om der tidligere er loadet responser, se multirespons for dette. En for løkke cykler gennem længden af impulsresponsen, hvor temp sættes lig den tilsvarende artikel i int16 arrayet SD\_response, mix værdien ganges på, for at svække styrken af rumklangen, og der deles derefter med en signed int16's maksværdi, for at normalisere den til en 32bit floats -1 til 1 spænd. Derefter sættes den gamle længde, der som udgangspunkt er 1 til den nuværende længde. Denne bruges i multirespons. Og multi sættes til 1 for at indikere, at der i hvert fald er loadet en tidligere respons.

```
delete [] SD_response; //del int array to clear space
```

Her slettes int16 arrayet, som det blev før metoden blev modulariseret.

```

datafile.close();
return 0;
}

```

I tilfælde af, at brugeren skulle vælge no selection i menuen, vil et tom respons blive loadet.

```
int emptyResponse()
{
    for (int x = 1; x < arrLen; x++) impulse_response[x] = 0.0; //zero
    Serial.print("using impulse:");
    //prints whole impulse
    Serial.println("clean");
    for (int x = 0; x < arrLen;
x+=100)Serial.println(impulse_response[x]);
    impulse_response[0] = 1;
    Serial.println("");
    return 0;
}
```



**typeFetch** tjekker om filtypen er i orden, og om hvorvidt wave filer er i orden. Rigtig wave header, 16 bit PCM og mono.

Derfor søges der først efter .raw og .wav i filnavnet. Hvis det er en wavefil loades headeren ind i et char array, for at tjekke bit 8-11 efter "WAVE" strengen, bit 20 for PCM, bit 34 for 16 bit, da 8 bit er irrelevant og 24bit ints ikke findes på arm uden en masse krumspring. Til sidst tjekkes bit 22 for mono. 32bit IEEE float wav ville have været fedt at kunne bruge, men de kan ikke loades direkte fra med arduino SDlib, måske de er little endian, eller komprimeret. Bit 20 har værdien 3 i dem, så de er i hvert fald ikke gemt på samme måde som PCM. typeFetch kunne godt udvides med korrektion for metadata.<sup>8</sup>

endian	File offset (bytes)	field name	Field Size (bytes)
big	0	ChunkID	4
little	4	ChunkSize	4
big	8	Format	4
big	12	Subchunk1ID	4
little	16	Subchunk1 Size	4
little	20	AudioFormat	2
little	22	NumChannels	2
little	24	SampleRate	4
little	28	ByteRate	4
little	32	BlockAlign	2
little	34	BitsPerSample	2
big	36	Subchunk2ID	4
little	40	Subchunk2 Size	4
little	44	data	
		Subchunk2Size	4

The "RIFF" chunk descriptor  
The Format of concern here is "WAVE", which requires two sub-chunks: "fmt " and "data"  
The "fmt " sub-chunk  
describes the format of the sound information in the data sub-chunk  
The "data" sub-chunk  
Indicates the size of the sound information and contains the raw sound data

<sup>8</sup> billede fra <http://soundfile.sapp.org/doc/WaveFormat/>

```

uint8_t typeFetch(std::string name, File datafile)
{
    std::string wave0 = ".wav";
    std::string wave1 = ".WAV";
    std::string raw0 = ".raw";
    std::string raw1 = ".RAW";           //Definition af filtyper
    char header [45];
    uint8_t type = 20;

    if (name.find(wave0) != std::string::npos || name.find(wave1) != std::string::npos) //c++ magic      //Tjekker om .wav eller .WAV er i Strengen

    {
        datafile.read(header, 44); //reads header in chars
        if (header[8] == 0x57 && header[9] == 0x41 &&
            header[10] == 0x56 && header[11] == 0x45)      //Tjekker efter "WAVE"
        {
            if(header[20] == 0x1 && header[34] == 16)      //Tjekker efter 16bit PCM
            {
                if(header[22] == 0x1)                      //Tjekker efter mono (1 kanal)
                {
                    Serial.println("File is mono");
                    type = 1;
                } else type = 12;                         //Flere kanaler
            } else type = 11;                          //Ikke 16bit PCM
        }
    }
    else if (name.find(raw0) != std::string::npos || name.find(raw1) != std::string::npos) //RAW      //Tjekker om .raw eller .RAW er i Strengen
    else type = 10; //Unknown filetype
    return type;
}

```



Message (Enter to send me)

18:07:38.510 -> R	18:16:58.364 -> file
18:07:38.510 -> I	18:16:58.364 -> 52
18:07:38.510 -> F	18:16:58.364 -> 49
18:07:38.510 -> F	18:16:58.364 -> 46
18:07:38.510 -> \$	18:16:58.364 -> 46
18:07:38.510 -> □	18:16:58.364 -> E6
18:07:38.510 -> □	18:16:58.364 -> DA
18:07:38.510 -> □	18:16:58.364 -> A
18:07:38.510 -> □	18:16:58.364 -> 0
18:07:38.510 -> W	18:16:58.364 -> 57
18:07:38.510 -> A	18:16:58.364 -> 41
18:07:38.510 -> V	18:16:58.364 -> 56
18:07:38.510 -> E	18:16:58.364 -> 45
18:07:38.510 -> f	18:16:58.364 -> 66
18:07:38.510 -> m	18:16:58.364 -> 6D
18:07:38.510 -> t	18:16:58.364 -> 74
18:07:38.510 ->	18:16:58.364 -> 20
18:07:38.510 -> □	18:16:58.364 -> 10
18:07:38.510 -> □	18:16:58.364 -> 0
18:07:38.510 -> □	18:16:58.364 -> 0
18:07:38.510 -> □	18:16:58.364 -> 0
18:07:38.510 -> □	18:16:58.364 -> 1
18:07:38.510 -> □	18:16:58.364 -> 0
18:07:38.510 -> □	18:16:58.364 -> 1
18:07:38.510 -> □	18:16:58.364 -> 0
18:07:38.510 -> D	18:16:58.364 -> 44
18:07:38.510 -> ♀	18:16:58.364 -> AC
18:07:38.510 -> □	18:16:58.364 -> 0
18:07:38.510 -> □	18:16:58.364 -> 0
18:07:38.510 -> ♀	18:16:58.364 -> 88
18:07:38.510 -> □	18:16:58.364 -> 58
18:07:38.510 -> □	18:16:58.364 -> 1
18:07:38.510 -> □	18:16:58.364 -> 0
18:07:38.510 -> □	18:16:58.364 -> 2
18:07:38.510 -> □	18:16:58.364 -> 0
18:07:38.510 -> □	18:16:58.364 -> 10
18:07:38.510 -> □	18:16:58.364 -> 0
18:07:38.510 -> d	18:16:58.364 -> 4C
18:07:38.510 -> a	18:16:58.364 -> 49
18:07:38.510 -> t	18:16:58.364 -> 53
18:07:38.510 -> a	18:16:58.364 -> 54
18:07:38.510 -> □	18:16:58.364 -> 18
18:07:38.510 -> □	18:16:58.364 -> 0
18:07:38.510 -> □	18:16:58.364 -> 0
18:07:38.510 -> □	18:16:58.364 -> 0

Og nu til den vi alle har ventet på, fileFetch

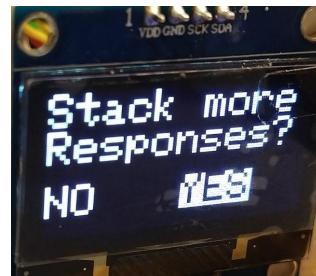
```
File fileFetch (File root) //function that returns an object how cool
{
  if (root.available())
  {
    File datafile = root.openNextFile();
    if(!datafile)
    {
      root.rewindDirectory();
    }
    Serial.println(datafile.name());
    return datafile;
  }
}
```

Denne funktion åbner en ny fil ud fra det parsede root objekt og i tilfælde af at enden af mappen er nået, starter mappen forfra.



## Multi-respons funktionalitet (Eksperimentelt)

Hvis vi denne gang antager at man siger ja til at stable responser, starter menuen forfra, hvor man kan vælge en ny respons. Dette kan gøres så mange gange man ønsker det, på egen risiko, selvfølgelig, for meget rod lyder ikke godt.



Processen med multirespons er sådan set den samme som med den første respons, bortset fra at et gennemsnit mellem responserne tages, når der er tilpas meget af dem begge to. Hvis der ikke er det, vil de blive adderet. Dette kan potentielt introducere værdier over 1, men det er stadig på et meget eksperimentelt niveau

Hvis den nye respons er længere, vil den først stables med den tidligere på den beskrevne måde, og efter den gamles længde er overskredet, erstatter de nye værdier tidligere værdi i arrayet. Derefter bliver den gamle længde sat lig den nye længde hver gang.

Hvis den nye længde er kortere, vil de blive stablet på samme måde.

Det nævnte med at der skal være nok af dem begge to, er det der sker i de lange if-sætninger. De summererer tallet indexværdien peger på i arrayene med deres foregående og efterfølgende værdier, og tjekker hvorvidt lydniveauet er over -60 dB eller 0.0001% af mætning i et af arrayene. Et tilfældes hvor dette ønskes er når vi har en kabinetrespons og en rumklang med et delay, hvor de ikke overlapper, eller ikke overlapper ret meget. Der kan det give mening at gøre det på den måde.

Hvis det skal være helt rigtigt, bør man egentlig afsplille filen for kabinetresponsen gennem rumklangen og optage den, men dette er stadig bare en eksperimentiel feature.

Det har tidligere været forsøgt at gange dem sammen, men dette kvæler værdierne alt for meget. De kunne måske foldes.



```
for (int x = 3; x < len; x++)
{
    temp = SD_response[x];
    temp *= mix;
    temp /= 32768;
    if (    impulse_response[x-1] + impulse_response[x] + impulse_response[x+1] < 0.0001
        && impulse_response[x-1] + impulse_response[x] + impulse_response[x+1] > -0.0001
        && SD_response[x-1] + SD_response[x] + SD_response[x+1] < 32
        && SD_response[x-1] + SD_response[x] + SD_response[x+1] > -32)
        impulse_response[x] += temp;
    else impulse_response[x] = (impulse_response[x] + temp)/2;      //response
}
Serial.print("secondary impulse loaded"); }
```

# Test introduktion

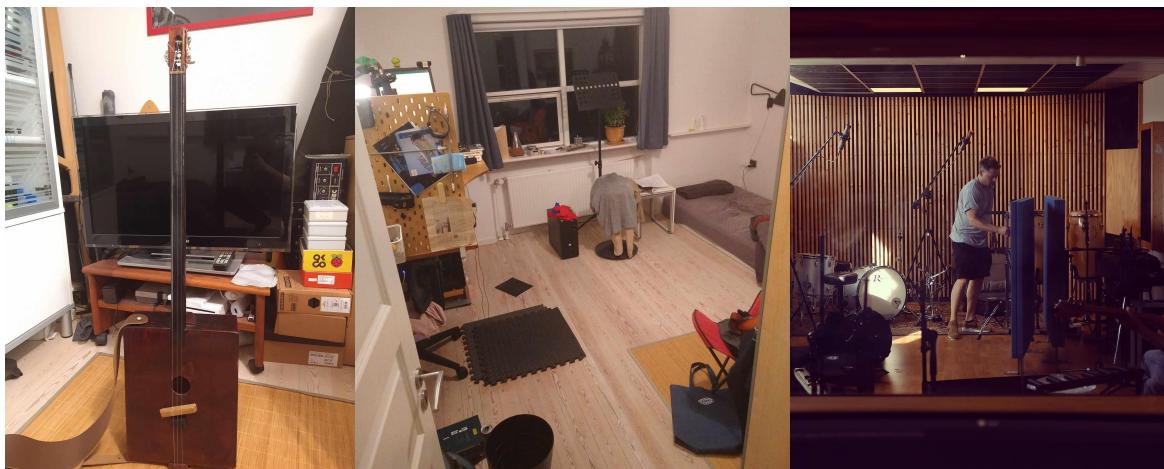
For at teste projekter som dette, vil man løbende foretage tests, næsten for hver udvidelse af prototypen, for at teste implementationen og integrationen af nye funktioner. Dette gøres for ikke at gå for hurtigt til værks, for på et tidspunkt at opdage, at der er en fejl, eller tre, blandt de tyve nyligt introducerede funktioner, der skaber mere eller måske mindre katastrofale problemer. Dette kan nemt resultere i en meget lang proces med at trække i land. Uover at teste efter fejl, er det også vigtigt at teste tesbarheden af produktet og danne erfaringer med nye områder, der kan testes, for at kunne gå mere grundigt til værks med at finde produktets begrænsninger, og senere hen garantere færre løse ender i funktionalitetens yderområder.

Testfaserne for rumklangpedalen har foregået således

1. Test af teensy audio adapter.
2. Test af impulsrespons bibliotek
3. Test og implementation af loading fra SD kort
4. Test og implementation af menu
5. Test og implementation af modularisering og multirespons

Sideløbende med tests af selve funktionaliteten af guitarpedalen, har der også foregået tests af forskellige impulsresponser for bedre at kunne lave dem, og se hvordan de ter sig. Disse har oprindeligt bestået af kabinetresponser og responser fra en cigarkasse guitar, for at teste hvorvidt det kunne lade sigøre at simulere et akustisk instrument med en elektrisk guitar. Disse er optaget med en clip-on piezo pickup, hvor en stålklos og solid glasflaske er slået mod stolen<sup>9</sup> på instrumentet for at simulere den måde strengenes vibrationer vil rejse ned i dækket og resonere.

Derefter er mere seriøse tests foregået med forskellige måder at optage responser fra rum på. Optagelse af et værelse med kondensatormikrofon gennem en focusrite scarlett preamp, og optagelse af lydhavnens lydstudie tapetown, gennem en telefon, men begge med et klap som impuls.

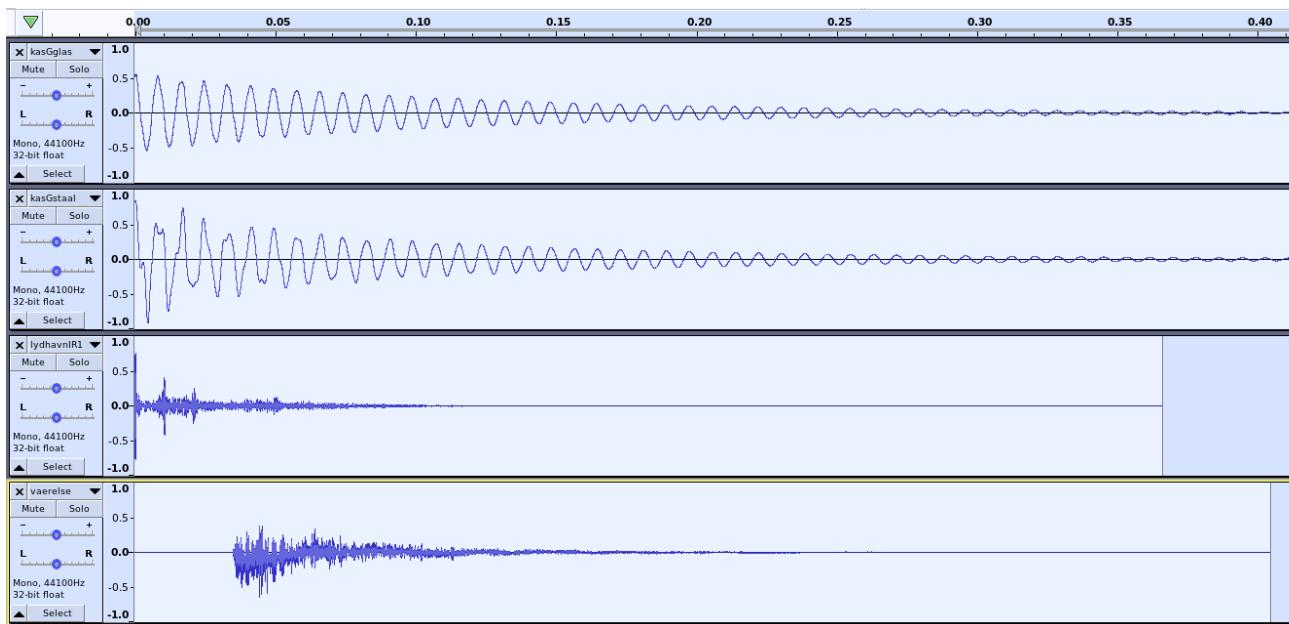


<sup>9</sup> Fagterm for klosen mellem strengene og kroppen

# Resultater

Efter mange uger med mange omgange af tests, er det lykkedes at have en lækker rumklang med mindre rum som et værelse og lydhavnens optagestudie, tapetown. Kasseguitaren, er ikke nær så vellykket, da der var en generelt misforståelse om hvordan impulsresponser bør optages. Forståelsen på det tidspunkt var, at impulsens volumen var underordnet og derfor gerne måtte klippe helt afsindigt, da responsen fra rummet var det vigtigste og derfor skulle have rigeligt volumen.

Impulsresponserne optaget i værelset og dem fra lydhavnen er optaget på en måde hvor impulsen ikke klipper, men stadig er ret tæt på mætning, så mikron forstærkeren ikke bliver overstyrer og dermed ikke opfører sig mærkeligt efter klippet.

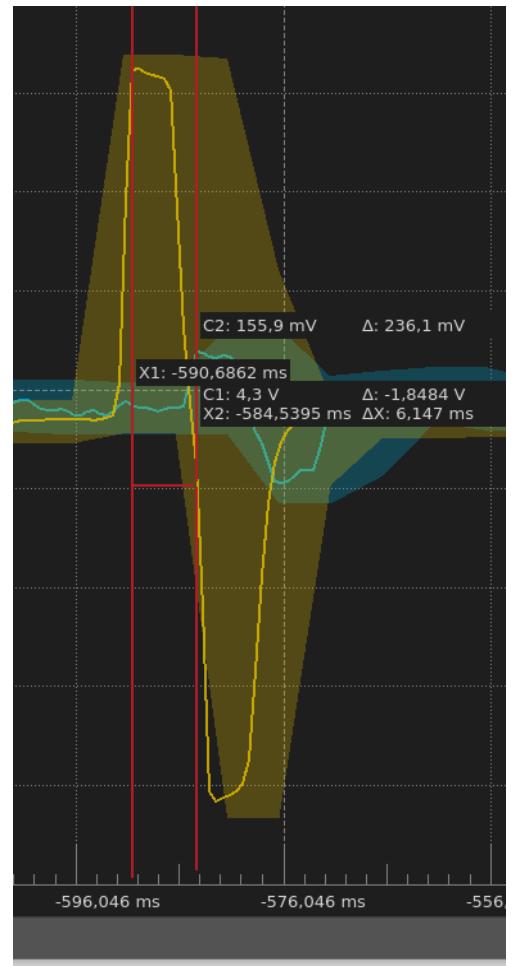
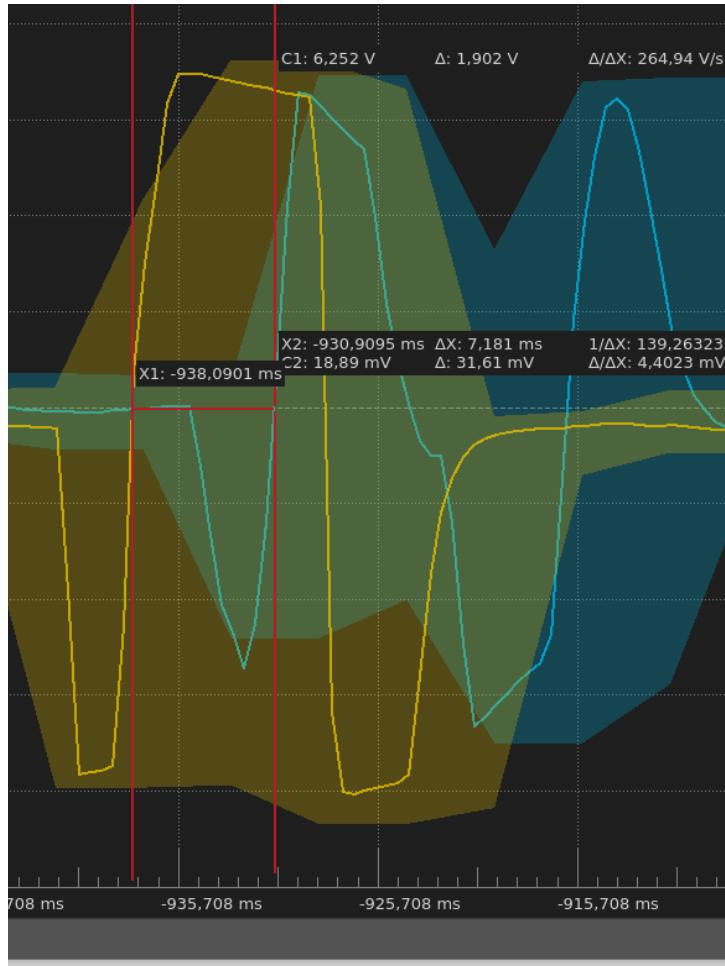


Som det kan ses på de første to kurver er kassen meget mere lavfrekvent end rummene, hvilket skaber nogle problemer når en guitar spilles gennem den. Specifikt frekvensområdet 110-120 Hz er slemt. Her går effekten i selvsving hvis disse toner spilles gennem den. Ellers virker alle de højere toner fint på den. Det lyder ikke som det virkelige instrument, men det er en sjov approksimation.

Dette er der dog potentielle for at udbedre i fremtiden, hvis en FFT bliver inkorporeret i floater() funktionen, og det frekvensområde dæmpes, hvorefter en invers FFT ville returnere det til et array, der kan bruges med convolution biblioteket. Dette kunne også ødelægge responsen, i cigarkassens tilfælde. Det er ikke til at sige, om den skal virke sådan. Det bliver interessant at høre den, når den tone spilles på den, efter den bliver skiftet til tykke strenge.

Nummer 3 og 4, er tapetown og værelse, hvor der er meget pæne, højfrekvente kurver, og en smule kunstig delay på værelset. Disse to er meget vellykkede.

Test af responstid var også yderst tilfredsstillende.



6.15-7.18 ms. Det skulle være godt nok til rockmusik, som man siger i branchen.

# Bevis

Til projektet er en youtube playliste lavet. Bemærk, kun de første fire videoer er uploadet før eksamensfristen. Der vil der dog blive uploaded tests med ordentlig lyd umiddelbart efter.

<https://www.youtube.com/playlist?list=PL1sbzfk5jrqY0XgC5ed2O-zZGCqtP8Gmu>

# Konklusion

Det kunne godt lade sig gøre at lave en impulsrespons rumklangspedal.

Multiresponsfunktionaliteten kræver lidt ekstra arbejde, og der er også plads til at udbygge en evt. filtrering af responserne, selvom dette også kan gøres når de laves på computeren. En opdatering af typeFetch til at understøtte metadata, ved at søge efter "data"-strengen, i stedet for at antage den er ved bit 40, ville også give mening, hvis den ikke kan findes i den loadede header.

Ellers er det en meget velfungerende prototype. Det er ærgerligt at der ikke blev til tid at lave en rigtig pedal, men det hænder. Nogle af de optagede rumklange endte endda også med at være emminente.

# Bilag

## Adafruit i2c scanner

```
// SPDX-FileCopyrightText: 2023 Carter Nelson for Adafruit Industries
//
// SPDX-License-Identifier: MIT
// -----
// i2c_scanner
//
// Modified from https://playground.arduino.cc/Main/I2cScanner/
// -----



#include <Wire.h>

// Set I2C bus to use: Wire, Wire1, etc.
#define WIRE Wire

void setup() {
    WIRE.begin();

    Serial.begin(9600);
    while (!Serial)
        delay(10);
    Serial.println("\nI2C Scanner");
}

void loop() {
    byte error, address;
    int nDevices;

    Serial.println("Scanning...");

    nDevices = 0;
    for(address = 1; address < 127; address++ )
    {
        // The i2c_scanner uses the return value of
        // the Write.endTransmisstion to see if
```

```

// a device did acknowledge to the address.
WIRE.beginTransmission(address);
error = WIRE.endTransmission();

if (error == 0)
{
    Serial.print("I2C device found at address 0x");
    if (address<16)
        Serial.print("0");
    Serial.print(address,HEX);
    Serial.println(" !");
}

nDevices++;
}

else if (error==4)
{
    Serial.print("Unknown error at address 0x");
    if (address<16)
        Serial.print("0");
    Serial.println(address,HEX);
}

}

if (nDevices == 0)
Serial.println("No I2C devices found\n");
else
Serial.println("done\n");

delay(5000); // wait 5 seconds for next scan
}

```

## main.ino

```
#include <string>
#include <cstring>

#include <Audio.h>
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <arm_math.h>
#include <arm_const_structs.h> // in the Teensy 4.0 audio library, the ARM CMSIS
DSP lib is already a newer version
#include <utility/imxrt_hw.h> // necessary for setting the sample rate, thanks
FrankB !
#include <EEPROM.h>

uint8_t eepAddr;      //EEPROM address for reset check
#define selektor 0    //button pin
#define mixPot 24     //pot for response strength

#include "loader.h"
#include "dispHandler.h"

const int nc = 18000; // number of taps for the FIR filter
#include "convolution.h"

// GUItool: begin automatically generated code
AudioInputI2S          i2s1;
AudioFilterConvolutionUP convolution;    // Uniformly-partitioned convolution
filter
AudioOutputI2S           i2s2;
```

```

AudioConnection           patchCord1(i2s1, 0, convolution, 0);
AudioConnection           patchCord2(i2s1, 1, convolution, 1);
AudioConnection           patchCord3(convolution, 0, i2s2,0);
AudioConnection           patchCord4(convolution, 1, i2s2,1);
AudioControlSGTL5000     codec;

// GUItool: end automatically generated code

#include "utils.h"
#include "menu.h"

void setup()
{
    Serial.begin(115200);
    Serial.println("setup has begun");
    pinMode(selektor, INPUT_PULLUP);
    while(!(digitalRead(selektor))) ;
    menu();
    Serial.println("floated");
    Serial.println();
    Serial.println();

    //while(!Serial); // must patch out for Visual Micro
    pinMode(14, OUTPUT); // digital signal used to measure routine's execution time
on 'scope
    setupSGTL5000();
    AudioMemory(10);

    set_arm_clock(600000000);
    setI2SFreq(SAMPLE_RATE);

    convolution.begin(0,0.20,*fftout,nfor); // turn off update routine until after
filter mask is generated, set Audio_gain=1.00 , point to fftout array, specify
number of partitions

*****
```

Calculate the FFT of the FIR filter coefficients once to produce the FIR filter mask

```
*****
****/
convolution.impulse(impulse_response, maskgen, nc);

flexRamInfo();

Serial.println();
Serial.print("AUDIO_BLOCK_SAMPLES:  ");
Serial.println(AUDIO_BLOCK_SAMPLES);

Serial.println();

}

void loop() {
yield();

if(!(digitalRead(selektor))) //reset function
{
    EEPROM.update(eepAddr, 0); //Sets value to 0 to prevent intro playing next
cycle
    SCB_AIRCR = 0x05FA0004; //reset teensy
}
}
```

## menu.h

```
void menu()
{
    DispHandler disp = DispHandler();
    Loader load = Loader();
    Debouncer debounce = Debouncer(500);
    byte eepVal = EEPROM.read(eepAddr); //loads value from EEPROM address
    if (eepVal) disp.printIntro(); //If value is set, intro will play
    EEPROM.update(eepAddr, 255); //Sets value so intro plays if teensy is reset via power cut,
but not via button

    if(SDChk()) disp.error(0);
    File root = SD.open("/");
    File datafile = load.fileFetch(root);

    String filename;
    uint8_t state = 1;
    bool nextFile = 0;
    bool next = 0;
    bool multi = 0;
    bool clen = 0;
    uint8_t debbie;

    while(state != 0)
    {

        delayMicroseconds(50);
        switch (state)
        {

            case 1: //fetching name from sd
                if(nextFile) datafile = load.fileFetch(root); //datafile; //((nextFile, datafile); //checks if
next file is set                                         //DELETE check and make org state == 2
                if(datafile)
                {
                    filename = datafile.name();
                    disp.printFilename(filename); //displays datafile
                }
                if(!datafile) disp.printFilename("No\n selection");
                state = 2;

            break;

            case 2: // button unpressed mode
                debbie = debounce.debounceChk();
                if (debbie == 1)
                {
                    if(!nextFile) nextFile = 1; // clicks
                }
        }
    }
}
```

```

        state = 1;
    }
    else if (debbie == 2) // holds
    {
        if(!datafile)
        {
            if(multi) disp.theMessage(); // simple error message, don't look into it itself. Seems to
return to menu after displaying error, unlike the other two, which work if done the first time

            load.emptyResponse();
            clen = 1;
            state = 0;
        }
        else
        {
            uint8_t floatReturn = load.floater(datafile);
            if(floatReturn > 0) disp.error(floatReturn);

            state = 3;
        }
    }

    break;

case 3:
    disp.next(next);
    state = 4;
    break;

case 4: // load another impulse
    debbie = debounce.debounceChk();
    if (debbie == 1)
    {
        next = !next;
        state = 3;
    }
    else if (debbie == 2) //hold
    {
        if (next) // if yes is selected
        {
            multi = 1;
            next = 0;
            state = 1;
        }
        else //if no selected
        {
            state = 0;
        }
    }
    break;

}

```

```
}

Serial.println("while loop broken");
if(clean) disp.nowPlaying("clean/dry"); //if clean variable is set, it will always display clean, as
selecting clean response overwrites any previous loaded responses
else
{
    if(multi)disp.nowPlaying("multiple");    //if multiple responses are in use
    else disp.nowPlaying(filename);           //if only one response is loaded
}
if(datafile) datafile.close(); //DATAfile is close if it exists
root.close();      //same with root
//delete disp;
//delete load;

Serial.println("end of menu.");
}
```

## loader.h

```
const int16_t arrLen = 18001; //array length with padding
float impulse_response[arrLen];           //defining impulse array and int array for 16 bit signedint file

class Loader
{
private:
    uint16_t oldLen = 1;
    uint8_t multi = 0;

    uint8_t typeFetch(std::string name, File datafile)
    {
        std::string wave0 = ".wav";
        std::string wave1 = ".WAV";
        std::string raw0 = ".raw";
        std::string raw1 = ".RAW";
        char header[45];
        uint8_t type = 20;

        if (name.find(wave0) != std::string::npos || name.find(wave1) != std::string::npos) //c++ magic
        {
            datafile.read(header, 44); //reads header in chars
            for (int x = 0; x < 44; x++)
            {
                Serial.print(x); Serial.print(": ");
                Serial.println(header[x], HEX); //prints header
            }

            if (header[8] == 0x57 && header[9] == 0x41 &&
                header[10] == 0x56 && header[11] == 0x45) //checks for "WAVE" in header) //checks
for PCM and mono
            {
                Serial.println("wave detected");

                if(header[20] == 0x1 && header[34] == 16) //Checks for PCM and 16 bits of depth
                {
                    Serial.println("File is 16bit PCM");

                    if(header[22] == 0x1) //Confirms channel count as single or mono
                    {
                        Serial.println("File is mono");
                        type = 1;
                    } else type = 12;
                } else type = 11;
            }
        }
        else if (name.find(raw0) != std::string::npos || name.find(raw1) != std::string::npos) type = 0;
    //RAW
    }
}
```

```

else type = 10; //Unknown filetype

return type;

}

public:
    Loader()
{
    impulse_response[0] = 1; //Necessity, as empty response and initial delays in room responses
otherwise choke the direct sound
    impulse_response[1] = 1; //Necessity, as empty response and initial delays in room responses
otherwise choke the direct sound
    impulse_response[2] = 1; //Necessity, as empty response and initial delays in room responses
otherwise choke the direct sound
    for (int x = 1; x < arrLen; x++) impulse_response[x] = 0.0; //zero padding
}

File fileFetch (File root)//File datafile)//(bool next, File datafile) //function the returns an
object how cool
{
    if (root.available()) //checks if SD folder is defined
    {
        File datafile = root.openNextFile();           //does what it says
        if(!datafile)
        {
            root.rewindDirectory();                  //starts over when data file can't be defined due
to reaching end
        }
        Serial.println(datafile.name());
        return datafile;
    }
}

*******/

int floater(File datafile)//, bool first)      //"floates int16 wav PCM/raw file to normalised float
array
{
    //Range +/-15bit int converted to -1 to +1 Float

    if (datafile) // if file is open
    {
        int len; //impulse length

        float mix = analogRead(mixPot); //Recording mix
        mix = map(mix, 0.0,1023.0,0.0,1.0);           //remap from int10 to 0 - 1 Float32
        Serial.print("mix: ");
        Serial.println(mix);
    }
}

```

```

Serial.println("SD card OK");
Serial.println("file found");
std::string name = datafile.name();

int16_t *SD_response = new int16_t[arrLen]; //dynamic array that can be deleted

uint8_t type = typeFetch(name, datafile); //0: unknown, 1: Raw int16, 2: wave int16, 3: wave
int16 stereo, 4: wave F32, 5 wave F32 stereo

if(type == 10) return(1);
if(type == 11) return(2);
if(type == 12) return(3);

Serial.print("type detected, type: ");
Serial.println(type);

//while(1) if(!(digitalRead(selektor))) SCB_AIRCR = 0x05FA0004; //reset teensy

if (type == 1)      len = (datafile.size()-44)/2;           // int16 wav
else if (type == 0) len =  datafile.size()/2;                // int16 raw

if(len > arrLen) //checks if file is too large for memory to handle
{
    Serial.print("len too large: "); //error msg
    Serial.println(len);
    Serial.print("will be truncated to 0.4s");
    len = arrLen;      //truncates file to 18000 samples
} else
{
    Serial.print("len: ");
    Serial.println(len);
}

if(type == 1) datafile.seek(44);
if(type == 0) datafile.seek(0);

datafile.read(SD_response, len*2); //reads appropriate amount of data from file in bytes(not
ints)

datafile.seek(0); //reset datafile, may be redundant

```

```

Serial.print("mix: ");
Serial.println(mix);

float temp;
if(!multi)
{
    for(int x = 3; x < len; x++)
    {
        temp = SD_response[x];
        temp *= mix;
        temp /= 32768;
        impulse_response[x] = temp;
        oldLen = len;
        multi = 1;
    }
    Serial.println("first impulse loaded");
}
else
{
    if(len > oldLen)
    {
        Serial.println("oldLen smaller");
        for (int x = 3; x < oldLen; x++)
        {
            temp = SD_response[x];
            temp *= mix;
            temp /= 32768;
            if ( impulse_response[x-1] + impulse_response[x] + impulse_response[x+1] < 0.0001
// checks for changes to counter an initial delay of a room response, that would otherwise choke a cab
response
                && impulse_response[x-1] + impulse_response[x] + impulse_response[x+1] > -0.0001
//the numbers themselves and the shifts x-1 to x+1 are arbitrary
                && SD_response[x-1] + SD_response[x] + SD_response[x+1] < 32
                && SD_response[x-1] + SD_response[x] + SD_response[x+1] > -32) impulse_response[x] +=
temp;
                //if (impulse_response[x] > -0.0001 && impulse_response[x] < 0.0001) impulse_response[x]
+= temp; // if the previous response is too low, the current is added on top. Might create
artificating, but prevents a cab response from being muted by a room
                else impulse_response[x] = (impulse_response[x] + temp)/2; //response is averaged
with prev response
                //else impulse_response[x] *= temp; //response is multiplied with previous response
            }
        for (int x = oldLen; x < len; x++)
        {
            temp = SD_response[x];
            temp *= mix;
            temp /= 32768;
            impulse_response[x] = temp; //Part of response that is further out the reverberation than the
previous response is used as is
        }
        oldLen = len;
    }
}

```

```

        else
    {
        Serial.println("newLen smaller");
        for (int x = 3; x < len; x++)
        {
            temp = SD_response[x];
            temp *= mix;
            temp /= 32768;
            if ( impulse_response[x-1] + impulse_response[x] + impulse_response[x+1] < 0.0001
                && impulse_response[x-1] + impulse_response[x] + impulse_response[x+1] > -0.0001
                && SD_response[x-1] + SD_response[x] + SD_response[x+1] < 32
                && SD_response[x-1] + SD_response[x] + SD_response[x+1] > -32) impulse_response[x] +=
temp;
            //if (impulse_response[x] > -0.0001 && impulse_response[x] < 0.0001) impulse_response[x]
+= temp;
            else impulse_response[x] = (impulse_response[x] + temp)/2;      //response is averaged
with prev response
            //else impulse_response[x] *= temp;    //response is multiplied with previous response
        }
    }
    Serial.print("secondary impulse loaded");
}
//}

Serial.println("");
Serial.print("using impulse:");           //prints whole impulse
Serial.println(datafile.name());
for (int x = 0; x < oldLen; x+=100) Serial.println(impulse_response[x],4);
Serial.println("");

delete [] SD_response;                  //del int array to clear space
/*impulse_response[2] = 1;
impulse_response[3] = 1;
impulse_response[4] = 1;*/

datafile.close();
return 0;
}

}

int emptyResponse()
{
    for (int x = 1; x < arrLen; x++) impulse_response[x] = 0.0; //zero
    Serial.print("using impulse:");           //prints whole impulse
    Serial.println("clean");
    for (int x = 0; x < arrLen; x+=100) Serial.println(impulse_response[x]);
    impulse_response[0] = 1;
    Serial.println("");
    return 0;
}
};


```



## dispHandler.h

```
#include <Adafruit_GFX.h>
#include <Adafruit_SH110X.h>

#define i2c_Address 0x3c //initialize with the I2C addr 0x3C Typically eBay OLED's
//#define i2c_Address 0x3d //initialize with the I2C addr 0x3D Typically Adafruit OLED'
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino reset pin)

class DispHandler
{
private:
    Adafruit_SH1106G display = Adafruit_SH1106G(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire1, OLED_RESET);

public:

    DispHandler()
    {
        if (!display.begin(i2c_Address, 0x3C))
        { // Address for 128x64
            Serial.println(F("SH1106 allocation failed"));
            for (;;)
                ; // Don't proceed, loop forever
        }
    }

void printIntro(void)      //goes through a series of text displays
{
    display.clearDisplay();
    display.setTextSize(3);           // Normal 1:1 pixel scale
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text

    char name[] = "Ølefant";
    char name1[] = "audio";

    display.setCursor(0, 10);         // Start at top-left corner
    display.println(name);
    //display.println(F("ÖLEFANT"));
    display.display();

    delay(1000);
    display.setCursor(0, 40);         // Start at top-left corner
    display.println(name1);
    display.display();

    delay(2500);
    display.clearDisplay();
    display.display();
    display.setTextSize(2);          // Normal 1:1 pixel scale
```

```

display.setCursor(0, 0);           // Start at top-left corner
display.println(F("SELECT"));
display.display();
delay(1000);
display.setCursor(0, 40);         // Start at top-left corner
display.println(F("IMPULSE"));
display.display();

delay(2500);
display.clearDisplay();
display.display();
delay(300);
display.setCursor(0, 0);           // Start at top-left corner
display.println(F("filename: "));
display.display();
//delay(300);
display.clearDisplay();
}

void printFilename(String name)           //prints "filename" and a string of the file name
{
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    display.setTextSize(2);                  // Normal 1:1 pixel scale
    display.setCursor(0, 0);                // Start at top-left corner
    display.println(F("filename: "));

    display.setCursor(0,32); // Start at top-left corner
    display.println(name);
    display.display();
    display.clearDisplay();
}

void next(bool yesnt)           //prints "filename" and a string of the file name
{
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    display.setTextSize(2);                  // Normal 1:1 pixel scale
    display.setCursor(0, 0);                // Start at top-left corner
    display.println(F("Stack more\nResponses?"));

    display.setCursor(0,40); // Start at top-left corner
    if(yesnt) display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    else display.setTextColor(SH110X_BLACK, SH110X_WHITE); // Draw black text
    display.println(F("NO"));

    display.setCursor(64,40); // Start at top-left corner
    if(!yesnt) display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    else display.setTextColor(SH110X_BLACK, SH110X_WHITE); // Draw black text
    display.println(F("YES"));

    display.display();
    display.clearDisplay();
}

```

```

}

void nowPlaying(String name)
{
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    display.setTextSize(2); // Normal 1:1 pixel scale
    display.setCursor(0, 0); // Start at top-left corner
    display.println(F("Now \n playing: "));
    display.setCursor(0,32); // Start at top-left corner
    display.println(name);
    display.display();
    display.clearDisplay();

}

void error(uint8_t errType)
{
    if (errType == 0) Serial.println("SD error");
    if (errType == 1) Serial.println("Unknown filetype");
    if (errType == 2) Serial.println("Error in header pos20, file not 16 bit PCM");
    if (errType == 3) Serial.println("Error in header pos22, file not MONO");
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    display.setTextSize(1); // Normal 1:1 pixel scale
    display.setCursor(0, 16); // Start at top-left corner
    if (errType == 0) display.println(F("Error #0:\nSD inaccessible"));
    if (errType == 1) display.println(F("Error #1\nIn header bits 8-11:\nUnknown filetype"));
    if (errType == 2) display.println(F("Error #2\nIn header bit20/34:\nfile not 16bit PCM"));
    if (errType == 3) display.println(F("Error #3\nIn header bit22:\nfile not mono"));
    display.display();
    EEPROM.update(eepAddr, 0); //Sets value to 0 to prevent intro playing next cycle
    while(1) if(!(digitalRead(selektor))) SCB_AIRCR = 0x05FA0004; //reset teensy
}

```

```

/*********************SUPER DUPER SECRET DON'T
LOOK****************************/
/*********************SUPER DUPER SECRET DON'T
LOOK****************************/
/*********************SUPER DUPER SECRET DON'T
LOOK****************************/
//you were warned
void theMessage()
{
  for(int x = 0; x < 3 ; x++)
  {
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    display.setTextSize(5);
    display.setCursor(0, 10);           // Start at top-left corner
    display.println(F("NICE"));
    display.display();

    delay(700);
    display.clearDisplay();
    display.setCursor(0, 10);           // Start at top-left corner
    display.println(F("TRY"));
    display.display();
}

```

```
    delay(700);

}

display.clearDisplay();
display.setTextSize(1);
display.setCursor(0, 30);           // Start at top-left corner
display.println(F("no impulse for you :")));
display.display();
delay(800);

}
};
```

## utils.h

```
#include <Adafruit_GFX.h>
#include <Adafruit_SH110X.h>

#define i2c_Address 0x3c //initialize with the I2C addr 0x3C Typically eBay OLED's
//#define i2c_Address 0x3d //initialize with the I2C addr 0x3D Typically Adafruit OLED'
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino reset pin)

class DispHandler
{
private:
    Adafruit_SH1106G display = Adafruit_SH1106G(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire1, OLED_RESET);

public:

    DispHandler()
    {
        if (!display.begin(i2c_Address, 0x3C))
        { // Address for 128x64
            Serial.println(F("SH1106 allocation failed"));
            for (;;)
                ; // Don't proceed, loop forever
        }
    }

void printIntro(void)      //goes through a series of text displays
{
    display.clearDisplay();
    display.setTextSize(3);           // Normal 1:1 pixel scale
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text

    char name[] = "Ølefant";
    char name1[] = "audio";

    display.setCursor(0, 10);         // Start at top-left corner
    display.println(name);
    //display.println(F("ÖLEFANT"));
    display.display();

    delay(1000);
    display.setCursor(0, 40);         // Start at top-left corner
    display.println(name1);
    display.display();

    delay(2500);
    display.clearDisplay();
    display.display();
    display.setTextSize(2);          // Normal 1:1 pixel scale
```

```

display.setCursor(0, 0);           // Start at top-left corner
display.println(F("SELECT"));
display.display();
delay(1000);
display.setCursor(0, 40);         // Start at top-left corner
display.println(F("IMPULSE"));
display.display();

delay(2500);
display.clearDisplay();
display.display();
delay(300);
display.setCursor(0, 0);           // Start at top-left corner
display.println(F("filename: "));
display.display();
//delay(300);
display.clearDisplay();
}

void printFilename(String name)           //prints "filename" and a string of the file name
{
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    display.setTextSize(2);                  // Normal 1:1 pixel scale
    display.setCursor(0, 0);                // Start at top-left corner
    display.println(F("filename: "));

    display.setCursor(0,32); // Start at top-left corner
    display.println(name);
    display.display();
    display.clearDisplay();
}

void next(bool yesnt)           //prints "filename" and a string of the file name
{
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    display.setTextSize(2);                  // Normal 1:1 pixel scale
    display.setCursor(0, 0);                // Start at top-left corner
    display.println(F("Stack more\nResponses?"));

    display.setCursor(0,40); // Start at top-left corner
    if(yesnt) display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    else display.setTextColor(SH110X_BLACK, SH110X_WHITE); // Draw black text
    display.println(F("NO"));

    display.setCursor(64,40); // Start at top-left corner
    if(!yesnt) display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    else display.setTextColor(SH110X_BLACK, SH110X_WHITE); // Draw black text
    display.println(F("YES"));

    display.display();
    display.clearDisplay();
}

```

```

}

void nowPlaying(String name)
{
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    display.setTextSize(2); // Normal 1:1 pixel scale
    display.setCursor(0, 0); // Start at top-left corner
    display.println(F("Now \n playing: "));
    display.setCursor(0,32); // Start at top-left corner
    display.println(name);
    display.display();
    display.clearDisplay();

}

void error(uint8_t errType)
{
    if (errType == 0) Serial.println("SD error");
    if (errType == 1) Serial.println("Unknown filetype");
    if (errType == 2) Serial.println("Error in header pos20, file not 16 bit PCM");
    if (errType == 3) Serial.println("Error in header pos22, file not MONO");
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    display.setTextSize(1); // Normal 1:1 pixel scale
    display.setCursor(0, 16); // Start at top-left corner
    if (errType == 0) display.println(F("Error #0:\nSD inaccessible"));
    if (errType == 1) display.println(F("Error #1\nIn header bits 8-11:\nUnknown filetype"));
    if (errType == 2) display.println(F("Error #2\nIn header bit20/34:\nfile not 16bit PCM"));
    if (errType == 3) display.println(F("Error #3\nIn header bit22:\nfile not mono"));
    display.display();
    EEPROM.update(eepAddr, 0); //Sets value to 0 to prevent intro playing next cycle
    while(1) if(!(digitalRead(selektor))) SCB_AIRCR = 0x05FA0004; //reset teensy
}

```

```

/*********************SUPER DUPER SECRET DON'T
LOOK****************************/
/*********************SUPER DUPER SECRET DON'T
LOOK****************************/
/*********************SUPER DUPER SECRET DON'T
LOOK****************************/
//you were warned
void theMessage()
{
  for(int x = 0; x < 3 ; x++)
  {
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    display.setTextSize(5);
    display.setCursor(0, 10);           // Start at top-left corner
    display.println(F("NICE"));
    display.display();

    delay(700);
    display.clearDisplay();
    display.setCursor(0, 10);           // Start at top-left corner
    display.println(F("TRY"));
    display.display();
}

```

```
    delay(700);

}

display.clearDisplay();
display.setTextSize(1);
display.setCursor(0, 30);           // Start at top-left corner
display.println(F("no impulse for you :")));
display.display();
delay(800);

}
};
```

## convolution.h

```
#include <Adafruit_GFX.h>
#include <Adafruit_SH110X.h>

#define i2c_Address 0x3c //initialize with the I2C addr 0x3C Typically eBay OLED's
//#define i2c_Address 0x3d //initialize with the I2C addr 0x3D Typically Adafruit OLED'
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino reset pin)

class DispHandler
{
private:
    Adafruit_SH1106G display = Adafruit_SH1106G(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire1, OLED_RESET);

public:

    DispHandler()
    {
        if (!display.begin(i2c_Address, 0x3C))
        { // Address for 128x64
            Serial.println(F("SH1106 allocation failed"));
            for (;;)
                ; // Don't proceed, loop forever
        }
    }

void printIntro(void)      //goes through a series of text displays
{
    display.clearDisplay();
    display.setTextSize(3);           // Normal 1:1 pixel scale
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text

    char name[] = "Ølefant";
    char name1[] = "audio";

    display.setCursor(0, 10);          // Start at top-left corner
    display.println(name);
    //display.println(F("ÖLEFANT"));
    display.display();

    delay(1000);
    display.setCursor(0, 40);          // Start at top-left corner
    display.println(name1);
    display.display();

    delay(2500);
    display.clearDisplay();
    display.display();
    display.setTextSize(2);           // Normal 1:1 pixel scale
```

```

display.setCursor(0, 0);           // Start at top-left corner
display.println(F("SELECT"));
display.display();
delay(1000);
display.setCursor(0, 40);         // Start at top-left corner
display.println(F("IMPULSE"));
display.display();

delay(2500);
display.clearDisplay();
display.display();
delay(300);
display.setCursor(0, 0);           // Start at top-left corner
display.println(F("filename: "));
display.display();
//delay(300);
display.clearDisplay();
}

void printFilename(String name)           //prints "filename" and a string of the file name
{
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    display.setTextSize(2);                  // Normal 1:1 pixel scale
    display.setCursor(0, 0);                // Start at top-left corner
    display.println(F("filename: "));

    display.setCursor(0,32); // Start at top-left corner
    display.println(name);
    display.display();
    display.clearDisplay();
}

void next(bool yesnt)           //prints "filename" and a string of the file name
{
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    display.setTextSize(2);                  // Normal 1:1 pixel scale
    display.setCursor(0, 0);                // Start at top-left corner
    display.println(F("Stack more\nResponses?"));

    display.setCursor(0,40); // Start at top-left corner
    if(yesnt) display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    else display.setTextColor(SH110X_BLACK, SH110X_WHITE); // Draw black text
    display.println(F("NO"));

    display.setCursor(64,40); // Start at top-left corner
    if(!yesnt) display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    else display.setTextColor(SH110X_BLACK, SH110X_WHITE); // Draw black text
    display.println(F("YES"));

    display.display();
    display.clearDisplay();
}

```

```

}

void nowPlaying(String name)
{
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    display.setTextSize(2); // Normal 1:1 pixel scale
    display.setCursor(0, 0); // Start at top-left corner
    display.println(F("Now \n playing: "));
    display.setCursor(0,32); // Start at top-left corner
    display.println(name);
    display.display();
    display.clearDisplay();

}

void error(uint8_t errType)
{
    if (errType == 0) Serial.println("SD error");
    if (errType == 1) Serial.println("Unknown filetype");
    if (errType == 2) Serial.println("Error in header pos20, file not 16 bit PCM");
    if (errType == 3) Serial.println("Error in header pos22, file not MONO");
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    display.setTextSize(1); // Normal 1:1 pixel scale
    display.setCursor(0, 16); // Start at top-left corner
    if (errType == 0) display.println(F("Error #0:\nSD inaccessible"));
    if (errType == 1) display.println(F("Error #1\nIn header bits 8-11:\nUnknown filetype"));
    if (errType == 2) display.println(F("Error #2\nIn header bit20/34:\nfile not 16bit PCM"));
    if (errType == 3) display.println(F("Error #3\nIn header bit22:\nfile not mono"));
    display.display();
    EEPROM.update(eepAddr, 0); //Sets value to 0 to prevent intro playing next cycle
    while(1) if(!(digitalRead(selektor))) SCB_AIRCR = 0x05FA0004; //reset teensy
}

```

```

/*********************SUPER DUPER SECRET DON'T
LOOK****************************/
/*********************SUPER DUPER SECRET DON'T
LOOK****************************/
/*********************SUPER DUPER SECRET DON'T
LOOK****************************/
//you were warned
void theMessage()
{
  for(int x = 0; x < 3 ; x++)
  {
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE, SH110X_BLACK); // Draw white text
    display.setTextSize(5);
    display.setCursor(0, 10);           // Start at top-left corner
    display.println(F("NICE"));
    display.display();

    delay(700);
    display.clearDisplay();
    display.setCursor(0, 10);           // Start at top-left corner
    display.println(F("TRY"));
    display.display();
}

```

```
    delay(700);
}

display.clearDisplay();
display.setTextSize(1);
display.setCursor(0, 30);           // Start at top-left corner
display.println(F("no impulse for you :")));
display.display();
delay(800);

}
};
```