

Pedala de efecte chitara electrica

Autor: Stefan-Dragos Badea - 331CC

Introducere

O pedala pentru chitara este o piesa de hardware care modifica sunetul care vine de la chitara pentru a crea efecte diverse. Existand o gama atat de larga de efecte posibile iar pretul hardware-ului fiind destul de ridicat, am considerat ca o pedala programabila este o solutie lightweight si budget-friendly. Astfel, cred ca ar fi utila pentru orice persoana cu un mini-studio sau doar pasionata de cantat.

Descriere generală



Modul de functionare este urmatorul:

- Conecteaza chitara la intrarea pedalei printr-un cablu jack.
- Conecteaza iesirea pedalei la o interfata audio / amplificator
- Activeaza pedala selectand unul din efectele audio dorite din telecomanda
- Cântă!

Hardware Design

Piese folosite:

- Arduino UNO cu Atmega328p
- DAC PCF8591
- IR sensor si telecomanda
- 2 mufe jack mama
- rezistente, condensatoare

Placuta va prelua semnalul analogic dat de chitara si il va converti in semnal digital folosind ADC-ul integrat. Initial placuta reda sunetul clean, efecte putand fi activate/dezactivate din telecomanda. Semnalul digital modificat se transmite la DAC-ul extern prin I2C, de unde pleaca mai departe la modalitatea de output (amplificator/interfara audio).

Semnalul analogic este filtrat atat la intrare cat si la iesire. Intrarea trece printr-un filtru trece sus care blocheaza frecventele de sub $\sim 66\text{Hz}$. Iesirea trece printr-un filtru trece-jos ce blocheaza frecventele peste $\sim 16\text{kHz}$.



Software Design

Codul este dezvoltat in Arduino IDE.

Librarii:

- <https://github.com/RobTillaart/PCF8591>
- <https://github.com/Arduino-IRremote/Arduino-IRremote>

Intrare:

Intrarea se face prin ADC-ul integrat in placa, cu voltajul de referinta intern de 1.1V. Modulul DAC are o rezolutie de doar 8 biti asa ca primul pas dupa citire este sa transform numerele pe 10 biti in 8 biti, dupa care sa aplic efectele de sunet.

```
unsigned int sensorValue = analogRead(A0);  
sensorValue = (sensorValue + 1) / 4 - 1;
```

Senzorul si telecomanda IR:

Fiecare buton de pe telecomanda este mapat la un numar de 2 cifre. Pentru a selecta/deselecta fiecare efect, folosesc un vector ca un map. Daca valoarea din map pentru un buton este 0, efectul este oprit, altfel este pornit. Butonul '0' dezactiveaza toate efectele.

```
int pressed_key = IrReceiver.decodedIRData.command;  
if (pressed_key != IR_BUTTON_0)  
    activeEffects[pressed_key] = ~activeEffects[pressed_key];  
else  
    memset(activeEffects, 0, sizeof(activeEffects));
```

Efecte:

In spatele efectelor sta o matematica destul de simpla folosita pentru a altera semnalul. Am implementat 4 efecte: distortion, tremolo, flanger si echo.

Distortion se realizeaza prin cresterea/supraincercarea semnalului audio printr-o functie simpla.

```
float apply_distortion(float x) {
```

```

float y = 1;

if (x < 0.33f)
    y = 2.3 * x;

else if (x < 0.66f)
    y = (3 - (2 - 3 * x) * (2 - 3 * x)) / 3;

return y;
}

```

Tremolo presupune combinarea semnalului cu o functie sinusoidala. Am folosit contorul 'index' pentru simularea trecerii timpului.

```

int index = 0;
float apply_tremolo(float x) {
    index++;
    float Fs = TIMER_1HZ / 8;
    float Fx = 5;
    float alpha = 0.35;
    float trem = (1 + alpha * sin(2 * M_PI * index * (Fx / Fs)));
    float y = trem * x;

    return y;
}

```

Pentru celelalte 2 efecte am avut nevoie de valorile anterioare ale semnalului asa ca am pastrat un vector cu acestea. Pentru a economisi memorie si tinand cont ca valorile cu care lucrez sunt pe 8 biti din cauza rezolutiei DAC-ului extern, acest vector este cu elemente tip 'char'.

Flanger este un efect ce presupune redarea in acelasi timp a 2 semnale identice cu un delay variabil intre ele. Pentru simularea trecerii timpului am folosit aceeasi idee ca la tremolo, contorul 'index2'.

```

float flanger_delay = 0.04f;
float flanger_depth = 0.5f;
int index2 = 0;
float apply_flanger(float x) {
    index2++;
    float Fs = TIMER_1HZ;
    float rate = 1.0;
    float max_sample_delay = flanger_delay * MAX_OUTPUT_HISTORY;
    float current_sin = fabs(sin(2 * M_PI * index2 * (rate / Fs)));
    int current_delay = (int)ceil(current_sin * max_sample_delay);

    float out_delayed = (float)output_history[(output_history_last + (
MAX_OUTPUT_HISTORY - 1) - current_delay) % MAX_OUTPUT_HISTORY];
    out_delayed /= SGN_MAX;

    return (1 - flanger_depth) * x + flanger_depth * out_delayed;
}

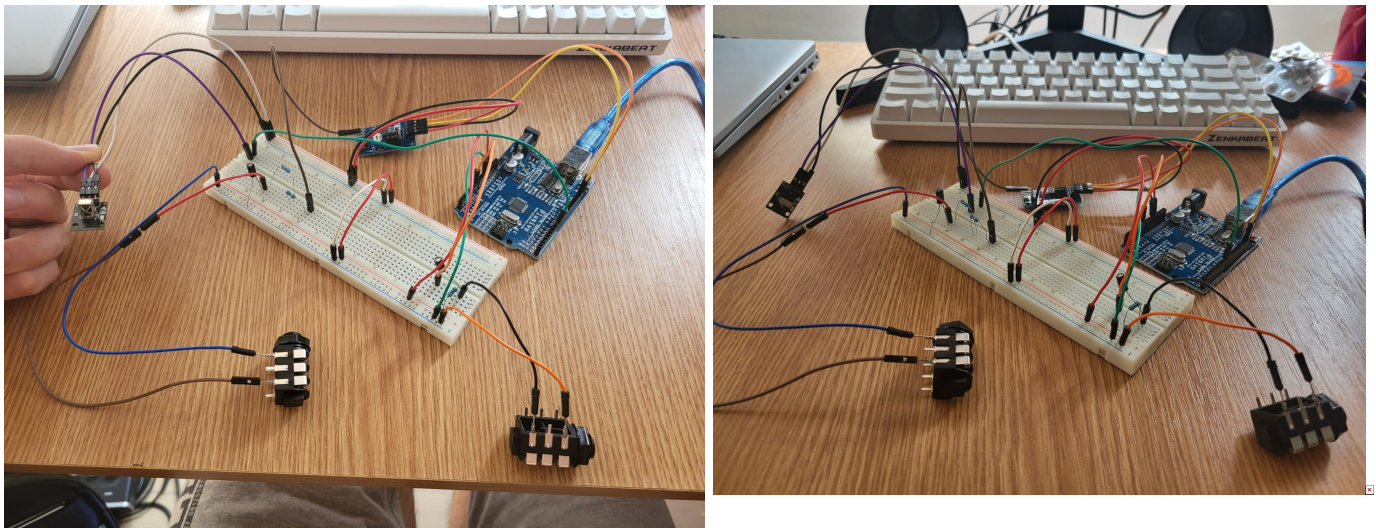
```

In final, echo presupune memorarea semnalului si redarea lui cu un delay.

```
int echo_delay = 350;
float echo_depth = 0.4f;
float apply_echo(float x) {
    float out_delayed = (float)output_history[(output_history_last + (
MAX_OUTPUT_HISTORY - 1) - echo_delay) % MAX_OUTPUT_HISTORY];
    out_delayed /= SGN_MAX;

    return (1 - echo_depth) * x + echo_depth * out_delayed;
}
```

Rezultate Obținute



Concluzii

Am invatat destul de multe si despre arduino si despre procesarea audio real-time si sunt multumit de rezultat.

Download

[proiectpm.zip](#)

Bibliografie/Resurse

Hardware

- https://ocw.cs.pub.ro/courses/_media/pm/doc8272.pdf
- <https://circuitdigest.com/microcontroller-projects/arduino-pcf8591-adc-dac-module-interfacing>
- <https://www.nxp.com/docs/en/data-sheet/PCF8591.pdf>

Software

- <https://github.com/RobTillaart/PCF8591>
- <https://kidpatel.wixsite.com/dspaudioeffects/project-files>
- <https://ccrma.stanford.edu/~orchi/Documents/DAFx.pdf>

Ambele

- <https://www.google.com/>
- <https://ocw.cs.pub.ro/courses/pm>
- <https://chat.openai.com/>
- <https://projecthub.arduino.cc/electrosmash/74978749-a083-47bc-8fde-38fba84737f8>
- <https://roboticsbackend.com/arduino-ir-remote-controller-tutorial-setup-and-map-buttons/>

[Export to PDF](#)

From:

<http://ocw.cs.pub.ro/courses/> - **CS Open CourseWare**

Permanent link:

<http://ocw.cs.pub.ro/courses/pm/prj2023/dene/pedalaefectechitara>



Last update: **2023/05/27 16:00**