Procesor digital de efecte pentru chitară

Introducere

Proiectul este un procesor de efecte pentru chitară, care va prelua semnalul de la chitara ca input, îl va procesa digital aplicând diferite efecte sub forma unor funcții matematice, iar apoi îl va transmite către output. Schimbarea efectelor, și modificarea intensității lor, va fi modificată prin butoane, volumul va fi modificat atat printr-un potentiometru, cât și printr-un fotorezistor prin trecerea mâinii deasupra acestuia. Ecranul este folosit pentru afișarea efectului utiilizat, cât și afișarea intensității acestuia. Efectele implementate sunt:

- **Distorsion** - efect care “saturează” semnalul audio, rezultând într-un sunet mai agresiv și cu mai mult sustain.
- **Tremolo** - efect care modifcă volumul periodic, pe baza unei sinusoide.
- **Bit Crusher** - efect care reduce rezolutia sinutului prin reducerea numărului de biți.

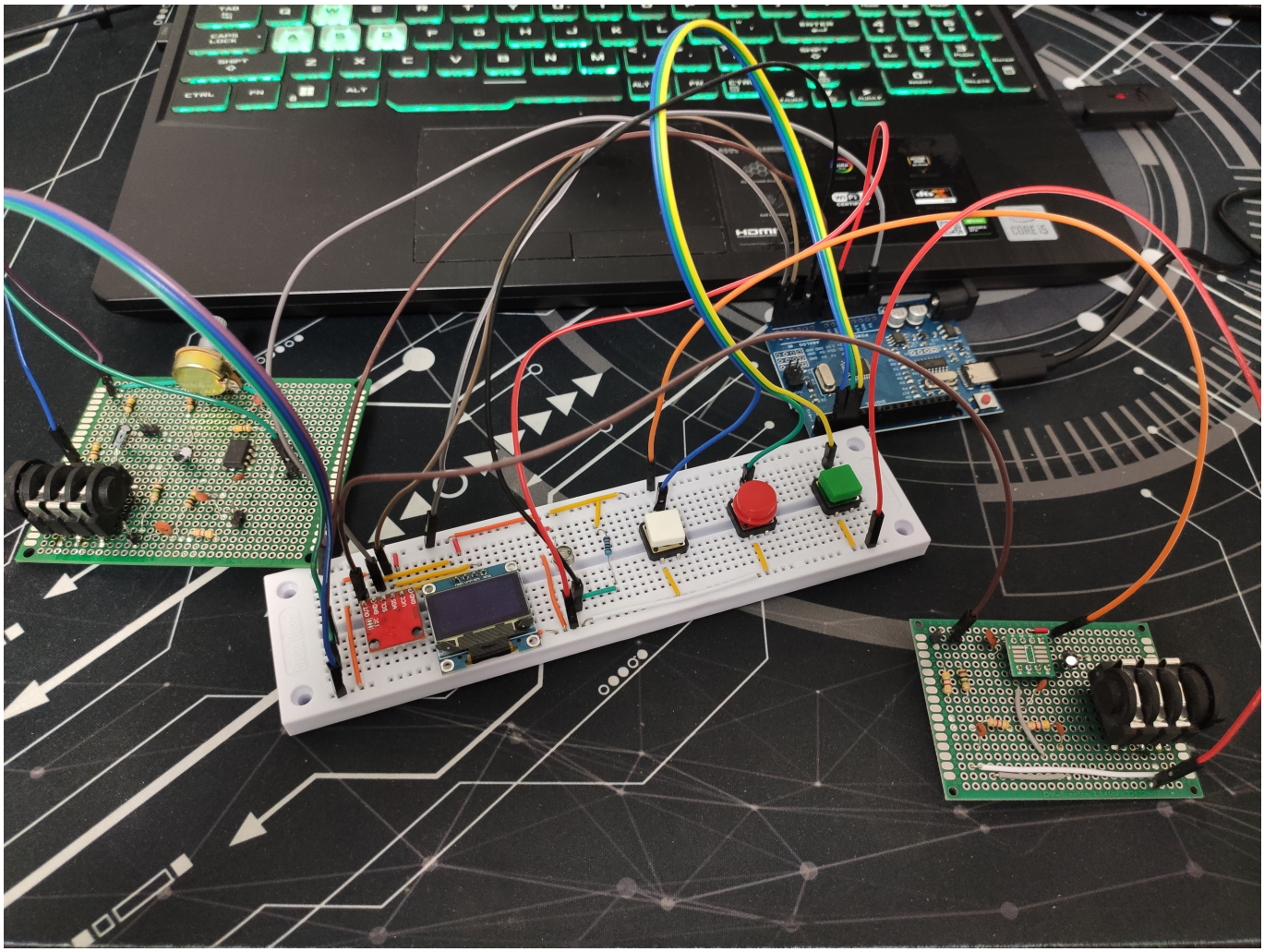
Descriere generală

Schema bloc arată modul general de funcționare al proiectului. Semnalul analogic **filtrat** de la chitară este convertit în analog folosind modulul **ADC de 10 biți** al Arduino-ului, iar apoi se face **procesarea** acestui semnal prin codul scris. Utilizatorul intervine în procesarea semnalului prin **butoane** și prin **fotorezistor**, iar tot el primește informații prin **display-ul OLED**. În final, semnalul este convertit înapoi în analogic folosind DAC-ul extern MCP4725 cu o rezolutie de 12 biți, iar apoi filtrat pentru a fi difuzat cu ajutoul amplificatorului de chitară.

Hardware Design

Nr.	Componentă	Cantitate	Specificații / Comentarii	Link către componentă
1	Arduino Uno	1	Microcontroler pentru procesarea semnalului audio	https://sigmanortec.ro/Placa-dezvoltare-UNO-R3-Arduino-Compatibil-ATmega328p-CH340G-cu-bara-pini-p170362384
2	Preamplificator op-amp (ex: TL072)	1	Buffer pentru semnalul de intrare de la chitară	https://www.emag.ro/circuit-integrat-dip8-texas-instruments-tl072cp-t106485/pd/D40M9YMBM/
3	Rezistențe (4.7kΩ, 10kΩ, 100kΩ, etc.)	5-10	Pentru biasare, divizoare, filtre	https://sigmanortec.ro/Kit-Rezistori-si-Potentiometre-A1-p141487466
4	Condensatori (100nF, 10uF, 220nF etc.)	5-10	Pentru filtre de semnal și decuplare	https://sigmanortec.ro/Set-120-condensatori-electrolitici-p125779654
5	Convertor D/A extern (ex: MCP4725)	1	DAC pe 12 biți controlat prin SPI	https://www.emag.ro/modul-convertor-digital-analog-mcp4725-12bit-i2c-pentru-analiza-eeprom-2-7-5-5v-mcp601/pd/D0Z8WSYBM/

6	Potențiometru (100kΩ)	1	Modificarea amplitudinii semnalului de input	https://sigmanortec.ro/Kit-Rezistori-si-Potentiometre-A1-p141487466
7	Amplificator op-amp de ieșire (ex: TL972IPWR)	1	Filtru Sallen-Key 3rd order	https://www.tme.com/ca/en/details/tl072cpwr/smd-operational-amplifiers/texas-instruments/
8	Breadboard și fire de conexiune	1	User Interface	https://sigmanortec.ro/Breadboard-760-puncte-p190992404
9	Jack 6.3mm stereo (intrare/ieșire)	2	Pentru conectarea chitarei și ieșirii audio	https://www.tme.com/ca/en/details/nys215/jack-connectors/rean/
10	Adaptor SSOP8	1	Pentru folosirea op-ampului TL92IPWR	https://www.optimusdigital.ro/ro/accesorii-adaptoare-pcb/198-adaptor-pcb-sop8ssop8tssop8-la-dip.html
11	Display 0.96 inch I2C	1	Afișare informații pentru utilizator	https://sigmanortec.ro/display-oled-096-i2c-iic-alb



Scheme electrice

Partea de hardware este inspirată din proiectul celor de la ElectroSmash, PedalShield UNO și este împărțită în 3 secțiuni, după cum se poate observa și din poză:

- **Input stage**: Constă în partea de preamplificare și de filtrare a semnalului, pentru a fi gata de conversia ADC și de procesarea din Arduino. Amplificatorul folosit este TL092CP, diferit de cel din schema de bază (TL972), un amplificator care nu este rail-to-rail, ceea ce poate duce la distorsionarea semnalului (prin clipping) dacă amplificarea este prea mare. Inițial valoarea rezistenței R_6 trebuia să fie de 4.7k, ceea ce ducea la o amplificare maximă de 21, conform

formulei: $ A = 1 + \frac{R_6 + VR_11}{R_5}$. Deoarece semnalul trecea de pragurile de 1V și 4V, am decis să schimb valoarea acestei rezistențe la 10k, astfel ducând la o amplificare maximă de 10.
- Pentru filtrare, se folosesc 3 filtre trece-jos (R_3 & C_3, R_5 & C_2, R_4 & C_5), care blochează frecvențele de peste aproximativ 5kHz.

- **Arduino Board și UI**: Semnalul filtrat (care vine în pinul A0) este convertit în semnal digital, folosind ADC-ul Arduino-ului, iar apoi semnalul digital este procesat în funcție de starea butoanelor și a fotorezistorului, setată de către utilizator. Pentru a modifica tensiunea transmisă către pinul A1, prin fotorezistor, folosim un simplu divizor de tensiune. Mai multe detalii despre modul în care se face procesarea vor fi la partea software.

- **Output stage**: Semnalul analogic convertit este filtrat folosind un filtru trece-jos Sallen Key de ordinul 3, unde amplificatorul TL972IPWR este setat în mod de repetor. Acesta are rolul de a elimina frecventele de peste 5kHz. La final, este folosit un capacitor care blochează “scurgerile” de curent continuu.

- **Alimentare**: Alimentarea este furnizată de către Arduino, prin pin-ul de 5V, iar pentru a centra semnalul de intrare în jurul valorii de 2.5V, pentru a fi gata de amplificare, folosim un simplu divizor de tensiune.

Software Design

Overview

Partea software a proiectului se bazează pe următoarele etape:

- Citește semnalul audio de la intrarea analogică A0, folosind în spate ADC-ul integrat.
- Aplică unul din cele 4 efecte.
- Scalează semnalul rezultat în funcție de valoarea luminozității ambientale, detectată cu un LDR pe A1.
- Trimite semnalul procesat către DAC-ul extern, scalând mai întâi output-ul pentru a fi pe 12 biți. Comunicarea cu DAC-ul se face protocolul I2C.
- Efectele, cât și intensitatea lor sunt schimbate cu ajutorul butoanelor prin întreruperi

Pentru comunicarea cu DAC-ul am folosit biblioteca Adafruit_MCP4725.

Implementarea Efectelor Audio

- **Clean**: nu se aplică nimic asupra semnalului și este transmis nemodificat.
- **Distorsion**: Inputul este amplificat și apoi limitat la o anumită valoare maximă/minimă

```
int distortionEffect(int input) {
  // Centrare în jurul lui 0
```

```
  int centered = input - 512;

  // Partea de amplificare a semnalului (între 1.5 și 5)
  float gain = map(intensity, minIntensity, maxIntensity, 150, 500) / 100.0;
  int boosted = centered * gain;

  // Partea de tăiere a semnalului după un anumit threshold
  int threshold = map(intensity, minIntensity, maxIntensity, 512, 128);
  if (boosted > threshold) boosted = threshold;
  if (boosted < -threshold) boosted = -threshold;

  // Recentrare in 512
  return constrain(boosted + 512, 0, 1023);
}
```

- **Tremolo**: Modifică volumul în timp folosind o undă sinusoidală. Pentru a face codul mai eficient, nu am folosit funția sin din math.h, ci am salvat valorile sinusoidei într-un fișier separat, într-un vector. Fiecare valoare din acest vector reprezintă un coeficient de volum între 0 și 1, care este aplicat semnalului audio pentru a-l face mai tare sau mai slab în mod ciclic. Dacă parcurgem vectorul pas cu pas, adică câte un eșantion la fiecare buclă, vom avea o frecvență joasă de modulare (efectul tremolo va suna lent).

Dar dacă dorim să creștem frecvența tremolo-ului (adică să facem vibrația volumului mai rapidă), avem două opțiuni:

- Scădem timpul între eșantioane (frecvența cu care se face sample++) (rate)
- Parcurgem mai rapid vectorul – adică sări peste eșantioane: sample += tremoloStepSize

```
int tremoloEffect(int input) {
  // Nr de frame-uri ca valoarea sinusoidei sa ramana constanta.
  int rate = map(intensity, minIntensity, maxIntensity, 16, 1);

  // Nr de step-uri sa fie sarite intre sample-uri
  int tremoloStepSize = map(intensity, minIntensity, maxIntensity, 2, 8);

  step++;
  if (step >= rate) {
    step = 0;
    sample += tremoloStepSize;
    if (sample >= 1024) sample -= 1024;
  }

  // Calculul factorului de scalare pe baza valoriii sinusoidei din fisier
  float tremolo = map(waveform[sample], 0, 255, 0, 1000) / 1000.0;
  return input * tremolo;
}
```

- **BitCrusher**: Efectul bitcrusher constă în scăderea rezoluției semnalului audio, adică în reducerea numărului de biți folosiți pentru a reprezenta fiecare eșantion. În loc de un semnal fluid de 10 biți (0–1023, pe Arduino), semnalul este „zdrobit” în trepte mari, ceea ce duce la un sunet digital, agresiv. În proiect am folosit folsit între 4 și 64 de nivele.

```
int bitcrusherEffect(int input) {
  // Nr de nivele
  int bitDepth = map(intensity, minIntensity, maxIntensity, 1, 4);
  int levels = 1 << bitDepth;

  // Diferenta dintre 2 nivele
  int stepSize = 1024 / levels;

  // Maparea inputului la un nivel din semnal
  int crushed = (input / stepSize) * stepSize;
  return constrain(crushed, 0, 1023);
}
```

Controlul intensității și schimbarea de efecte: Pentru un răspuns rapid și a elimina overhead-ul, am utilizat întreruperi: Trei butoane sunt folosite:

- D4: schimbă efectul (prin întrerupere **PCINT**): Folosim PCINT, deoarce D4 nu are întrerupere INT. Astfel, avem nevoie de o condiție suplimentară pentru a nu se activa întreruperea pe ambele fronturi.

```
// Enable la PCINTs
  PCICR |= (1 << PCIE2);

  // Enable pt D4
  PCMSK2 |= (1 << PCINT20);
```

- D2: scade intensitatea (prin întrerupere **INT0**): Întreruperea este activată pe frontul negativ HIGH to LOW.
- D3 crește intensitatea (prin întrerupere **INT1**) Întreruperea este activată pe frontul negativ HIGH to LOW.

În toate cazurile am folosit o măsură de a evita debounce-ul folosind niște variabile care rețin timestamp-ul ultimei schimbări.

Conversia Analog-Digital: Pentru prelucrarea semnalului de la chitară, am folosit convertorul analog-digital intern al Arduino-ului, configurat manual pentru a controla mai precis performanța și viteza de eșantionare. Inițializarea se face în funcția setupADC() prin care se se setează referința la AVcc, se activează ADC-ul și se se setează prescaler-ul la 64.

```
void setupADC() {
  // AVcc ca referinta si calanalul 0 by default
  ADMUX = (1 << REFS0);

  // Enable la ADC su Prescaler 64
  ADCSRA = (1 << ADEN)
         | (1 << ADPS2) | (1 << ADPS1);
}
```

Pentru citire în bucla de loop, se schimbă canalul între A0 și A1 (folosim A1 pentru fotorezistor), iar apoi se porneste conversia și se asteaptă terminarea.

```
uint16_t readADC(uint8_t pin) {
  // Setam canalul, 0 sau 1.
  ADMUX = (ADMUX & 0xF0) | (pin & 0x0F);

  // Start conversie
  ADCSRA |= (1 << ADSC);

  // Se asteapta pana conversia este terminata
  while (ADCSRA & (1 << ADSC));

  return ADC;
}
```

Conversia Digital-Analog: Semnalele procesate sunt trimise la DAC-ul **MCP4725**, care are implicit adresa 0x60, și care funcționează pe **protocolul I2C**. Valoarea de 10 biți este transformată într-una de 12 biți prin shiftare la stânga.

```
dac.setVoltage(scaled << 2, false);  // false = fără scriere în EEPROM
```

Controlul Volumului prin Fotorezistor: La pinul A1 este conectat potențialul divizorului de tensiune din care face parte fotorezistorului, astfel încât, când fotorezistorul nu este acoperit, potențialul este mic, iar atunci când este acoperit, potențialul este mare. Acest potențial ajustează volumul semnalului.

```
float getVolumeFromLDR() {
  int ldr = analogRead(A1);
  int volumeInt = map(ldr, 300, 700, 1000, 0);
  volumeInt = constrain(volumeInt, 0, 1000);
  return volumeInt / 1000.0;
}
```

Concluzii și Rezultat

Implementarea finală oferă un procesor de efecte simplu și modular, care imită decent cele 3 efecte. Rezultatul este unul satisfăcător, sunetul fiind destul de bun pentru procesarea cu un Arduino Uno. Drept îmbunătățire, aș încerca să fac codul low-level, fără bibliotecile Arduino, astfel putând să obțin atât o procesare mai bună a sunetului, dar și posibilitatea de a adăuga noi efecte. Proiectul a fost unul foarte interesant, care m-a făcut să înțeleg mai bine noțiunile de la laboratorul de PM, și să îmi aprofundez cunoștiințele de electronică. Cel mai mult m-am distrat la partea hardware, la lipit. Nu pot să exprim în cuvinte fericirea pe care am simțit-o atunci când am văzut că trece semnalul de chitară nedistorsionat

Jurnal

- 2.4.2025: Alegerea proiectului, initial procesor de efecte cu pedala

- 10.4.2025: Asamblarea primei parti hardware, input stage. Am testat valorile semnalului de input care vin în Arduino, și am modificat valoarea rezistenței R_4 petru a micșora amplificarea maximă.
- 12.4.2025: Am facut partea de output doar cu un singur filtru trece jos, pentru a testa sunetul. Deoarece sunetul nu era grozav, am decvis să folosesc filtrul Sallen Key.
- 14.4.2025: Am făcut output stage-ul, folosind filtrul Sallen Key. Drept amplificator am folosit TL972, pe care l-am pus pe PCB cu ajutorul unui adaptor SMD, după **multe ore de chin** pentru a-l lipi. Neașteptat, chiar a funcționat!

- 16.4.2025: Am implementat primele versiuni ale distorsion-ului si al tremolo-ului. Observ ca sunetul este destul de “muffled”, probabil din cauza filtrelor de 5kHz.
- 18.4.2025: Adaugare a butoanelor care modifica intensitatea efectului si a fotorezistorului.
- 23.4.2025: Am observat că ceva s-a întâmplat cu procesorul plăcii, deoarece nu mai trimite bine semnalul. cu același cod folosit... A trebuit să renunț l ecran pentru a obține un sunet decent.
- 25.4.2025: Finalizare proiect + pagină de OCW.

Bibliografie/Resurse

Datasheet DAC MCP4725 Pedal Shield Uno

Codul efectelor pentru Pedal Shield

Father of Electronics

Proiect inspiratie

Arhiva cod

Export to PDF