Electric guitar effect AMP - DRÎNCIU Cristina

Introducere

Proiectul constă într-un procesor de efecte audio digital pentru chitară electrică, realizat cu ajutorul unui Arduino UNO (ATmega328), care aplică efecte audio (ex: distorsiune, tremolo, delay) semnalului de la chitară.

Scopul proiectului este de a oferi o soluție accesibilă și personalizabilă pentru chitariști, care le permite să adauge diverse efecte audio chitarii lor folosind un procesor digital realizat cu Arduino. Proiectul își propune să înlocuiască echipamentele de efecte tradiționale costisitoare, oferind un control total asupra parametrilor fiecărui efect.

În plus, proiectul permite extinderea prin adăugarea de noi efecte sau îmbunătățiri, ceea ce îl face o platformă ideală pentru învățare și experimentare continuă.

Flow

Chitara se va conecta la procesorul de efecte printr-o mufa Jack 6.3 mm. Procesorul are 3 butoane si 3 LEDs care reprezinta cate un efect:

- Rosu - Distortion
- Verde - Tremollo
- Albastru - Delay

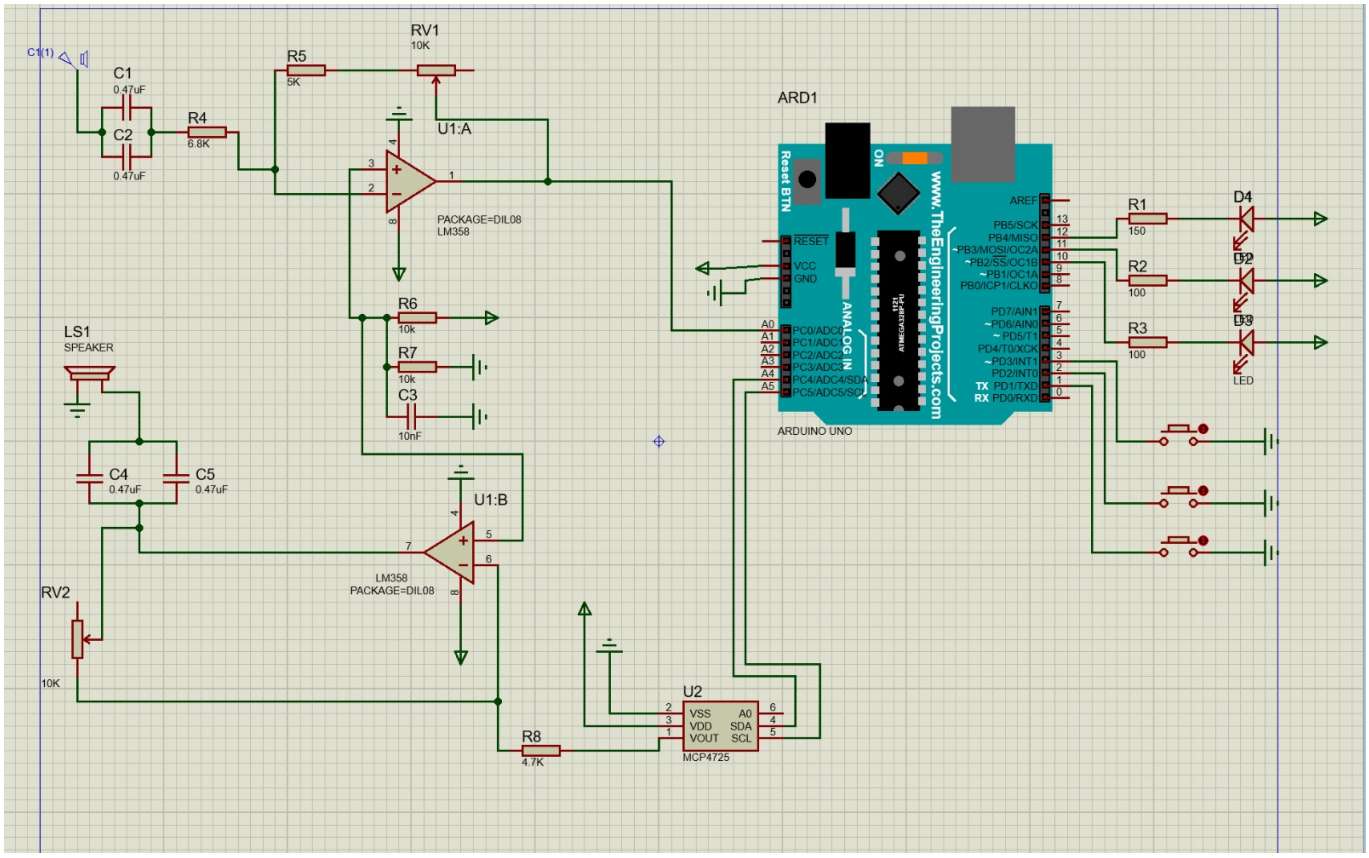
Deoarece semnalul de la chitara este unul slab, acesta va trece printr-un amplificator LM358. Apoi va trece in convertorul ADC din Arduino, procesat, iar apoi trimis la convertorul DAC MCP4725, apoi se poate alege volumul care trece ca output prin a doua mufa jack 6.3 mm unde va fi conectata o boxa.

Schema bloc

Hardware Design

Componente principale:

- Arduino UNO (cu ATmega328) – 1 buc
  - Utilizat pentru controlul efectelor și procesarea semnalului audio.
- LM358 (Op-Amp dual, DIP8) – 1 buc
  - Înlocuiește TL072 pentru preamplificarea audio, funcționează la 5V.
  - Link aici
- MCP4725 (DAC 12-bit I2C) – 1 buc
  - Convertor digital-analog pentru ieșirea semnalului procesat.
  - Link aici
- Mufă jack mamă 6.3mm mono – 2 buc
  - Una pentru intrarea audio (chitară) și una pentru ieșirea audio către amplificator.
- Potențiometru 10kΩ – 2 buc
  - Control volum audio (pentru semnal analogic).
  - Control gain ca feedback pentur intrarea negativa de la U1:A din LM358
- Butoane colorate – 3 buc
  - (rosu, verde si albastru) pentru selectarea efectului
- Rezistente si condensatoare



Functionalitate

1. Intrarea semnalului de chitară (jack intrare)

- Semnalul audio de la chitară este un semnal analogic cu amplitudine mică, care trebuie amplificat și prelucrat.
- Semnalul intră în filtrul de intrare format din C1 (0.47µF) și C2 (0.47µF) împreună cu rezistența R4 (6.8 kΩ).
- Condensatoarelor C1 și C2 acționează ca filtre de blocare a componentelor continue (DC offset), lăsând să treacă doar semnalul AC (audio) si ajută la eliminarea zgomotelor de frecvență joasă (filtru high-pass pasiv).
- Rezistenței R4 creează împreună cu condensatoarele un filtru RC ce determină frecvența de tăiere a filtrului high-pass si oferă o impedanță de intrare potrivită pentru semnal audio.

2. Etajul de amplificare cu LM358 (U1:A și U1:B)

LM358 este un amplificator operațional dual (două canale într-un singur cip). Primul canal este folosit pentru amplificarea semnalului de intrare.

U1:A - Amplificator cu feedback

- Semnalul de intrare ajunge la intrarea non-inversoare (+) a amplificatorului U1:A.
- Intrarea inversoare (-) este conectată printr-un divizor de tensiune format din rezistențele R5 (5 kΩ) și RV1 (10 kΩ potențiometru).
- Acest divizor controlează feedback-ul și deci câștigul amplificatorului.
- Ajustarea potențiometrului RV1 schimbă raportul feedback-ului, modificând astfel câștigul și implicit volumul/gradul de amplificare.
- Iesirea acestuia intra in pinul A0 in Arduino

U1:B - Etaj suplimentar de amplificare

- Asigură o impedanță de ieșire joasă pentru semnalul audio generat de DAC-ul MCP4725, astfel încât să nu se suprasolicite ieșirea DAC-ului și să ofere un semnal stabil și puternic către etapa următoare (ieșirea audio către amplificator extern sau difuzor).
- Semnalul amplificat din U1:A ajunge la intrarea non-inversoare (+) a U1:B.
- Feedback-ul la U1:B este format prin R8 (4.7 kΩ) și condensatorii C4 și C5 (0.47 µF) conectați pe intrarea inversoare (-).
- Rezistența R8 împreună cu condensatorii C4 și C5 formează un filtru de frecvență care ajută la stabilizarea semnalului și reduce zgomotele.
- Condensatorii ajută la blocarea componentelor DC și la filtrarea frecvențelor nedorite, contribuind la claritatea semnalului.
- U1:B amplifică și livrează semnalul procesat către ieșire.

3. Potențiometru volum (RV2 - 10 kΩ)

- Legat la ieșirea amplificatorului U1:B și la masa circuitului.
- Permite reglarea nivelului de semnal trimis către DAC și apoi către amplificatorul final.
- Ajustează amplitudinea semnalului ieșit, controlând volumul general al sunetului.

4. MCP4725 - DAC extern

- Conversia semnalului digital prelucrat în analogic, pentru a fi transmis către amplificatorul de putere (extern).
- Conectat prin interfața I2C (SDA, SCL) la microcontrollerul Arduino (pinii A4 - SDA si A5 - SCL)
- Semnalul procesat de Arduino (cu efectele aplicate) este transmis la DAC pentru a genera semnal

analogic la ieșire.

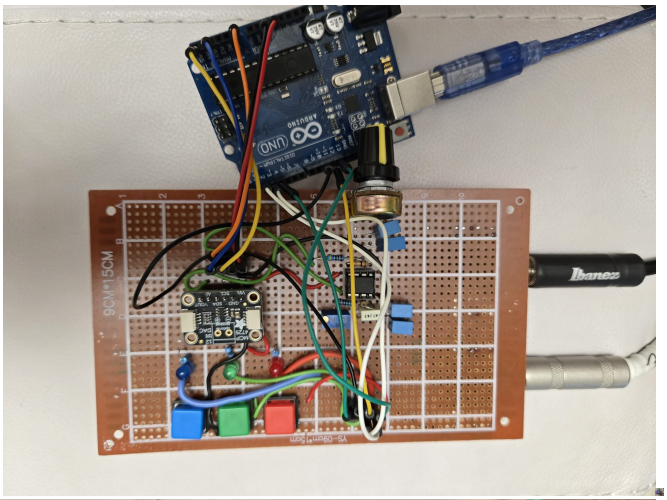
5. Butoanele și LED-urile

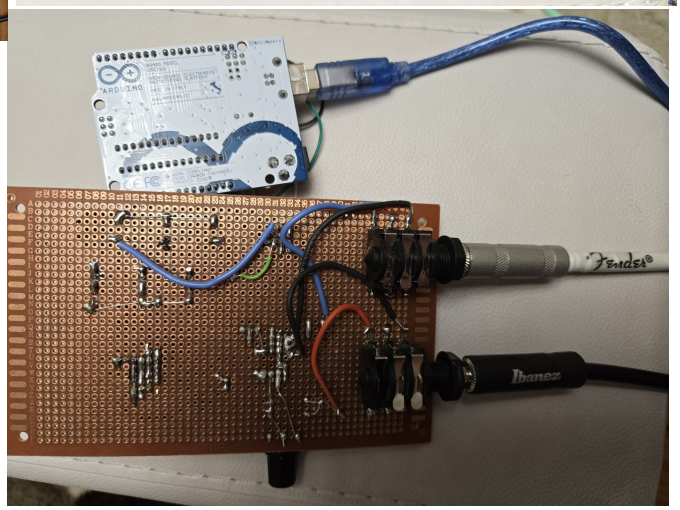
- 3 butoane pentru selectarea efectelor (Distorsiune, Tremolo, Delay), conectati la pini digitali (D2, D3, D4)
- Fiecare buton este legat la un pin digital cu pull-up intern activat (logic HIGH când nu este apăsat).
- La apăsare, microcontroller-ul detectează apăsarea (pin LOW) și comută efectul.
- LED-urile corespondente sunt conectate la pini digitali (D12, D11, D10) și indică starea efectului (pornit/oprit).

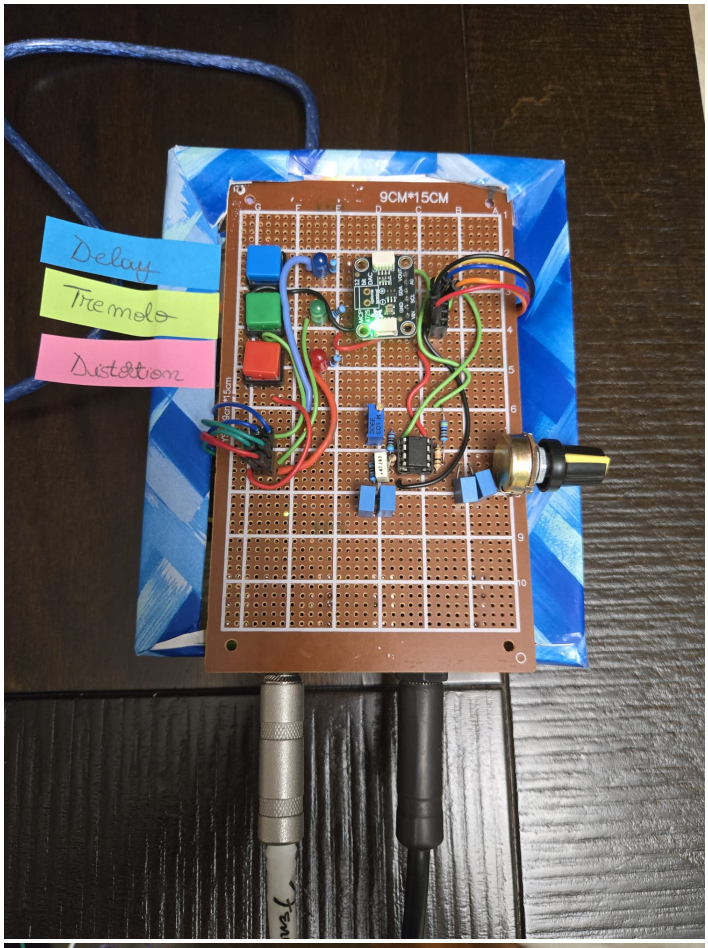
Interfete

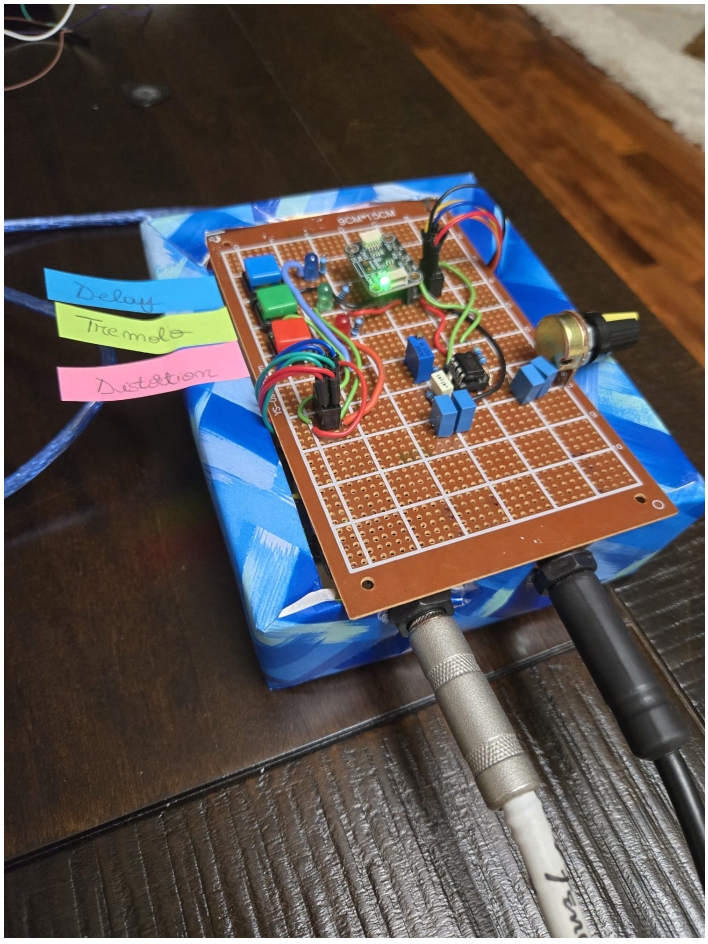
- ADC pe pinul A0 pentru semnal audio.
- I2C pe pinii A4 (SDA) și A5 (SCL) pentru MCP4725.
- GPIO digitali pentru butoane și LED-uri.

Poze









Link catre poze si video demonstrativ: aici

Software Design

Din simplul fapt ca acest proiect lucreaza si proceseaza semnal audio, inseamna ca partea de cod trebuie sa functioneze foarte repede si sa faca masuratori (esantioane) foarte rapid (sample rate). Deci sample rate mare → frecventa mare. Acest lucru nu se pot obtine cu biblioteci Arduino, deci singura solutie a fost de a lucra in PlatformIO in maniera de la laborator.

Notiuni folosite din laborator

1. GPIO

Am folosit pini digitali PD2, PD3, PD4 ca intrări cu pull-up pentru butoane și PB2, PB3, PB4 ca ieșiri pentru LED-uri.

2. ADC (Converto Analog-Digital)

Convertirea semnalului audio, adica analog, in digital pentru prelucrare si aplicarea efectelor selectate, apoi pasarea datelor la DAC. De asemnea, este configurat în Free-Running Mode pentru eșantionare continuă la ~10 kHz.

3. Timer(CTC)

Am folosit Timer0 CTC, folosit in tratarea intreruperilor pentru button debounce

4. I2C (TWI)

MCP4725 (DAC) functioneaza prin protocolul I2C, de asemenea este si rapid.

5. Intreruperi (ISR)

Toate interactiunile externe functioneaza cu intreruperi pentru optimizarea codului si a timpului de executie, pentru a nu afecta frecventa pentru sample rate. Am folosit intreruperi pentru:

- **ISR(ADC_vect)** - finalizarea fiecarei conversii ADC in modul Free-Running
- **ISR(PCINT2_vect)** - cand se declanseaza orice schimabre la nivel pe pinii PCINT18..20 (PD2, PD3, PD4), adică la apăsarea sau eliberarea butoanelor.
- **ISR(TIMER0_COMPA_vect)** - se declanșează la fiecare 1 ms, conform configurației: prescaler 64, OCR0A = 249, pentru gestionarea contoarelor de debounce si toggle pentru LEDs

I2C (TWI)

Rol: Interfața I²C (TWI) generică, folosită pentru a comunica cu DAC-ul MCP4725 la ~400 kHz.

Ce face fiecare functie:

1) `twi_init()`

- `TWSR = 0x00;` → prescaler = 1
- `TWBR = 12;` → bit-rate generator pentru SCL ≈ 400 kHz la 16 MHz
- `TWCR = (1«TWEN);` → activează modul TWI (SDA/SCL)

2) `twi_start(address)`

- `TWCR = (1«TWSTA)|(1«TWEN)|(1«TWINT);` → emite condiția START
- Așteaptă TWINT = 1 → START transmis
- Scrie TWDR = address; și trimiți SLA+R/W

3) `twi_write(data)`

- `TWDR = data;`
- `TWCR = (1«TWEN)|(1«TWINT);` → începe transmiterea
- Așteaptă TWINT = 1

4) `twi_stop()`

- `TWCR = (1«TWSTO)|(1«TWEN);` → emite STOP

DAC

Rol: : Strat de abstracție peste TWI pentru a scrie rapid valori 12-biți în MCP4725.

Ce fac functiile:

1) `dac_init()`

- Apel `twi_init()`

2) `mcp4725_write(val)`

- `twi_start(addr|0)` → trimite pe magistrală condiția START, apoi adresa de 7 biți a DAC-ului (0x62) urmată de bitul R/W=0 (scriere).
- `twi_write(0x40)` → fast-mode command
- `twi_write(val»4)` → cei 8 biți ai codului DAC de 12 biți, adică biții de ordin înalt D11…D4
- `twi_write`[1)] → Trimite următorii 4 biți (D3…D0), puși în nibble-ul superior al octetului:
- astfel DAC va primii bitii in felul urmator:

```
Octet 1 = [ D11 D10 D9 D8 D7 D6 D5 D4 ]
Octet 2 = [ D3 D2 D1 D0 0 0 0 0 ]
```

- `twi_stop()` - trimite conditia de STOP si elibereaza magistrala I2C

MCP4725 în fast-mode aşteaptă un octet de comandă, apoi doi octeți de date ce conțin cei 12 biți de nivel pe DAC. Primul transportă biții cei mai semnificativi (MSB), al doilea transportă cei mai puțin semnificativi (LSB) “aliniati” în nibble-ul de sus, pentru ca DAC-ul să știe exact cum să le reasambleze.

Efecte

Rol: Procesarea semnalului convertit in digital, aplicand efectele stabilite: **distorsiune, tremolo, delay**

Distortion

Creaza o forma de unda distorsionata si ascutita, cu tranzitiii rapide pentru un sunet “heavy”.

- *Pre-emphasis*

Amplifică ușor frecvențele medii și înalte pentru mai mult „bite”

y=x·(0.7+0.3·∣x∣)

- *Soft clipping*

Tranziție netedă la limită.

y=tanh(gain·y)

- *Hard limit*

Taie vârfurile peste ±0.8 pentru senzație agresivă.

y=max(min(y,+0.8),−0.8)

Pentru că DAC-ul MCP4725 lucrează cu rezoluţie de 12 biţi (domeniu numeric 0…4095), trebuie să mapăm semnalul nostru normalizat (−1…+1) pe acest interval.

Astfel, pentru un semnal normalizat x∈[−1,1]x∈[−1,1], codul final este

DAC_code=⌊x·2047⌋+2048

asigurând conversia corectă pe 12 biţi.

Tremolo

Acest efect oscileaza volumul semnalului la o frecvență joasă (LFO) pentru „vibrato de volum”.

Generăm un LFO sinusoidal pe 16 biți de fază si modulam aplitudinea cu factorul 2 pentru a compensa atenuarea la min(LFO).

Vom modifica periodic volumul semnalului audio pentru a crea un efect de „pulsare”.

În cod ținem o variabilă de fază pe 16 biți care se crește constant de fiecare dată când vine un eşantion nou.

La fiecare eşantion, din această fază calculăm o valoare între 0 și 1 (LFO) care oscilează la frecvența tremolo-ului (de exemplu 5 pulsuri pe secundă).

Apoi multiplicăm semnalul de intrare cu un amestec între volumul original și valoarea LFO:

Dacă LFO este aproape de 1, semnalul are volum aproape normal.

Dacă LFO este aproape de 0, semnalul este atenuat (mai silențios).

Prin schimbarea „adâncimii” (depth) poți controla cât de puternică este această variație de volum.

Totul se întâmplă foarte rapid (mii de ori pe secundă), astfel încât urechea percepe un „vibrato de volum” constant și sincronizat, fără distorsiuni suplimentare.

Delay

Rol: acest efect functioneaza ca un ecou, deci a fost nevoie de un buffer pentru a memora semnalul, deci memorie RAM.

Stocăm fiecare eşantion de semnal într-un buffer circular de mărime fixă (de exemplu 980 poziții).

Când procesăm un eşantion nou, citim din buffer eşantionul aflat cu un anumit „pas” înapoi (de exemplu, 800 poziții) — acesta este semnalul întârziat.

Combinăm semnalul întârziat cu semnalul curent: o parte din ecou (feedback) se adaugă peste semnalul nou.

Scriem în buffer valoarea rezultatului (semnal direct + ecou) pentru ca, la următoarea trecere, să putem crea ecouri în buclă.

Prin ajustarea mărimii buffer-ului (delay time) și a procentului de feedback, controlăm cât de lung și cât de intens este ecoul.

Practic, buffer-ul reține „urmele” semnalului și le redă cu întârziere, creând senzația de reverberație și ecou.

ADC

Rol: Preluare continuă de semnal analogic și declanșare ISR pentru procesare.

Avem urmatoarele setari pentru registre:

- `ADMUX = (1«REFS0);` - REFS0=1 → referință AVcc, canal ADC0
- `ADCSRA = ADEN|ADIE|ADATE|(ADPS2|ADPS1);` care este responsabil de:

`ADEN = enable ADC`

`ADIE = enable ADC interrupt`

`ADATE = auto-trigger → modul Free-Running  ADPS2:1 = prescaler 64 → ADC clock ~250 kHz`

- `ADCSRB = 0;` → Free-Run
- `DIDR0 = (1«ADC0D);` → dezactivează intrare digitală pe ADC0
- `ADCSRA |= (1«ADSC);` → pornește prima conversie

Intreruperile folosite:

`ISR(ADC_vect)`

- Citește ADC (0–1023), recentrează la ±512
- Normalizează la ±1.0
- Aplică efectele active
- Clamp și convert la 0–4095 → mcp4725_write()

De ce Free-Running? Vrem eșantionare continuă la frecvență fixă, fără să declanșăm singur fiecare conversie. Asa obtinem eficienta mai buna.

Butoane

Rol: *Gestionare butoane cu PCINT + debounce cu Timer0. De asemnea va trata si activarea butoanelor.*

Acestea functioneaza pe modul activer low, deci 0 activate, 1 dezactivate.

ISR(PCINT2_vect) (Pin Change)

- Setat pe PD2..PD4 (PCINT18..20)
- La trecerea HIGH→LOW pornește counter deb once
- eficient pentru ca nu asteptam intr-un loop apasarea butoanelor, doar intra rutina de tratare a intreruperilor pentru a schimba starea butoanelor si a LED-urilor.

ISR(TIMER0_COMPA_vect)

- (apelat in functia main), aceasta rutina va apela procesarea semnalului prin efecte, in functie daca au fost activate sau nu
- Timer0 CTC: prescaler 64, OCR0A=249 → 1 kHz interrupt
- decrementeaza counterele, iar la zero toggle LED și flag efect

De ce PCINT + timer? Detectăm rapid schimbarea de buton și facem debounce fără blocări în bucla

principală.

Main

Inițializează toate modulele și intră în buclă infinită.

Flux general:

- Butoane → PCINT + Timer0 → update flags
- ADC Free-Run → ISR(ADC_vect) → apply_effects → DAC
- DAC → ieșire audio analogic

Rezultate Obţinute si Performanta

Amplificatorul functioneaza destul de bine, putand sa genereze un semnal audio destul de bun. Chiar daca am incercat sa lucrez la o frecventa destul de ridicata pentru a avea un sample rate cat mai mare, notele joase sunt redate foarte clar iar cele mai inalte sunt redate o idee mai prost, avand o frecventa destul de mare si masuratorile nu sunt atat de accurate.

Optimizarile folosite pentru a obtine performante si imbunatiri care pot fi adaugate

1) Free-Running ADC

- Folosim ADC în Free-Running Mode pentru conversii continue la rata stabilită de prescaler, fără a reporni manual conversia la fiecare eşantion.
- Avantaj: latență minimă între eşantioane și utilizare eficientă a CPU în ISR

2) Intreruperi

- toată logica de procesare audio (distorsiune, tremolo, delay) se execută în ISR(ADC_vect), eliminând buclele de polling
- debounce butoane și toggle LED se fac în ISR-uri separate (PCINT2 si TIMER0), evitând blocarea buclei principale.

3) Math Library

- Tremolo folosește sinf() din <math.h>, costisitor în cicluri CPU
- Pentru o versiune și mai rapidă, se poate înlocui cu un tabel de valori pre-calculate (lookup table) pentru LFO.

4) Buffer Circular Fix-Size pentru Delay

- Delay folosește un buffer circular simplu, cu indexare modulară, pentru acces și scriere în timp constant (O(1)).
- Dimensiunea buffer-ului (980) este un compromis între intârziere perceptibilă și consum de memorie SRAM (≈2 KB).

Frecvente ale proiectului

1) Rata de eşantionare ADC: ~10 kHz

Prescaler ADC = 64 la 16 MHz → ADC clock ≈ 250 kHz → 25 cicluri pe conversie → ~10 k conversii/s, fiind suficient pentru frecventele relevante ala chitarii.

2) LFO Tremolo: 5 Hz

Frecvență perceptibilă plăcută de modulare de volum. Phase increment pre-calculatează saltul de fază pe eşantion, fără recalcule de frecvență în timp real.

3) Timer Debounce: 1 kHz (1 ms tick)

Asigură o rata de debounce robustă în maxim 50 ms fără pierderi de evenimente.

4) I2C (TWI): ~400 kHz

Comunicarea cu DAC la fast-mode pentru a corespunde celor 10 k eşantioane/s și overhead-ul I2C.

Estimarea Consumului de Putere

- Microntroller ATmega328P in mod normal (16 MHz, Vcc = 5 V): ~10–20 mA
- DAC MCP4725: ~0.5 mA
- LM358: fiecare canal ~0.7 mA
- LED-uri și Butoane: LED-uri aprinse (3 × 2 mA) = ~6 mA maxim

Total: I_total≈15mA (MCU)+1mA (DAC)+2mA (LM358)+6mA (LED)**≈24mA**

Putere: P=V×I≈5V×24mA=**120mW**

Concluzii

Acest proiect a fost o modalitate perfecta de a aprofunda cunostiintele atat de proiectarea microprocesoarelor, a laboratoarelor de la aceasta materie, cat si mai important, cunostiintele de

electronica. Constructia partii de HARDWARE a fost o provocare. In final, sunt multumita de outcome.

Jurnal

7 Mai: Descriere poriect si adaugarea listei de piese

18 Mai: Descriere detaliata pentru hardware si modificare liste de componente.

25 Mai: Decriere detaliata Software + estimarea puterii consumate si optimizarile facute

Bibliografie/Resurse

Datasheet LM358: LINK

Datasheet MCP4725: LINK

Datasheet ArduinoUno ATmega328p: LINK

Export to PDF

Arhiva aici

[1)] val&0xF)«4