Constantin MIHALACHE (67299) - Procesor de efecte pentru chitara

Autorul poate fi contactat la adresa: **Login pentru adresa**

Introducere

Scopul acestui proiect este realizarea unui procesor de efecte pentru chitara. Practic, microcontroller-ul va prelua semnalul generat de chitara, il va modifica si il va da mai departe la iesire. Ideea mi-a venit din necesitatea de a putea folosi efecte cat mai diverse pentru chitara fara a cheltui foarte mult pe pedale de efecte analogice. Acest proiect este util, in primul rand, pentru a intelege cum functioneaza procesoarele digitale de efecte din comert, dar si pentru a crea un procesor de efecte configurabil ce va putea fi folosit pe scena.

Efectele ce vor fi implementate:

- Distortion
- Tremolo
- Flanger
- Echo
- Compression

Descriere generală

Diagrama bloc:

Hardware Design

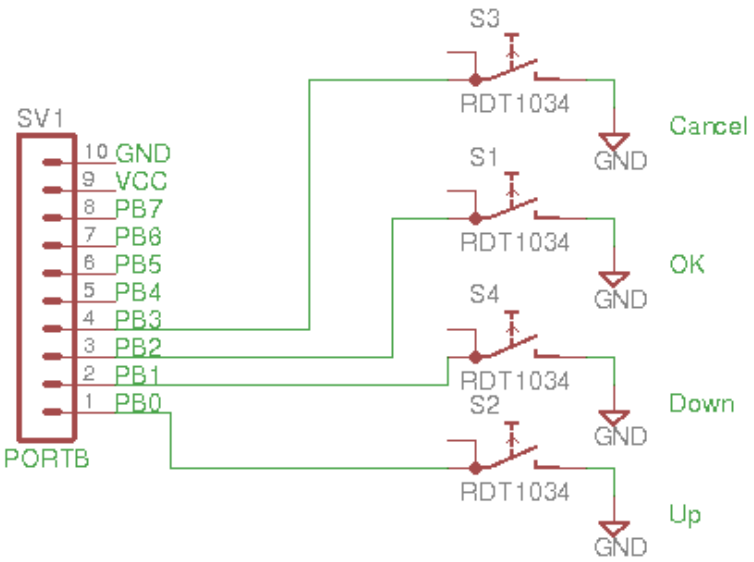
Lista de piese:

Numar piese	Nume piesa	Obs.
1	Placa de bază cu ATMega324A-PU	
1	LCD Text 2×16	https://www.robofun.ro/lcd/lcd-16x2-alb-albastru-3v

2	Mufa Jack 6.3mm Mama	https://electroniclight.ro/mf1401-jack-mama-63mm-mono/2907.htm
2	Rezistente de 1kΩ	
2	Rezistente de 1MΩ	
2	Condensatoare 4.7nF	
4	Butoane pentru control si setari	https://electroniclight.ro/mf4308-push-buton-6x6x7mm/1722.htm
1	Potentiometru 10k	
1	DAC MCP4725 cu interfata I2C (TWI)	https://www.optimusdigital.ro/altele/1327-modul-dac-mcp4725-cu-interfaa-i2c.html

Control

Pentru a naviga prin meniul de setari, vor exista 4 butoane (Up, Down, Ok, Cancel).



Afisaj

Pentru afisaj am folosit un LCD Text 2×16 ce are un controller Hitachi HD44780 similar celui de la laborator.

Intrare

Pentru obtinerea semnalului de la chitara am folosit ADC-ul integrat in AtMega324 in modul diferential, datorita faptului ca semnalul de chitara este in intervalul [-300mV, +300mV]. Pentru a elimina zgomotul datorat masurarii semnalului, am conectat la GND-ul placii de paza pinul la care este conectat GND-ul jack-ului de chitara. Pentru excluderea frecventelor nedorite am adaugat un filtru trece sus format dintr-o rezistenta de 500kΩ si un condensator de 4.7nF ce blocheaza frecventele care sunt sub ~60Hz. Alegerea acestui prag pentru frecventa minima a rezultat in urma cautarii celei mai joase note (celui mai jos sunet) ce poate fi produs de o chitara electrica.

Iesire

Pentru performante bune in generarea unui semnal analogic am ales folosirea unui DAC extern (MCP4725) ce interfateaza cu AtMega324 folosind Two-Wire Interface (TWI), standardul I2C. Fara prea multe batai de cap, acest standard (Fast mode) promite un bitrate de 400Kbps. Exista si High-Speed Mode in acest standard, ce poate atinge 3.2Mbps, insa nu a fost nevoie de el.

Comunicatia folosind TWI se realizeaza intr-o configuratie Master-Slave, unde AtMega324 este Master, iar DAC-ul MCP4725 este slave. Master-ul impune bitrate-ul folosind pin-ul SCL ce nu este altceva decat un semnal de ceas trecut printr-un prescaler, iar datele sunt transmise folosind pin-ul SDA.

In documentatia DAC-ul se gaseste un exemplu foarte intuitiv despre cum functioneaza interfata TWI in Fast-Mode pentru a da la iesire o tensiune in intervalul [0, VCC], in functie de cum sunt setati bitii de date.

Semnalul analogic dat de iesire de DAC este trecut printr-un filtru trece jos, format dintr-un condensator de 4.7nF si o rezistenta de 2kΩ ce exclude frecventele ce trec de pragul de ~16kHz. In plus, apare un potentiometru ce formeaza un divizor rezistiv intre GND si OUT (semnalul de iesire din DAC) pentru a controla volumul.

Software Design

Firmware-ul aplicatiei se imparte in mai multe componente.

LCD

Pentru interfatarea cu LCD-ul, am pornit de la codul de la laboratorul 1, avand in vedere ca LCD-ul folosit in proiect are un controller identic cu cel de la laborator. Singura modificare necesara a fost identificarea port-urilor si pinilor la care se conecteaza LCD-ul pe placa de baza.

Codul pentru aceasta componenta se gaseste in fisierele `lcd.h` si `lcd.c`.

Input butoane

Pentru identificarea input-ului de la utilizator folosesc intreruperile PCINT1, mai precis PCINT8, PCINT9, PCINT10, PCINT11 pentru pinii PB0, PB1, PB2 si PB3. In rutina pentru tratarea intreruperii setez o variabila globala `ui_check` care va determina verificarea starii butoanelor in bucla principala a programului.

```
ISR(PCINT1_vect)
{
    ui_check = 1;
}
```

Daca se determina ca unul dintre butoane este apasat, starea acestuia este directionata catre componenta Meniu.

```
if (ui_check) {
        // check for menu key inputs
        if ((PINB & _BV(PB0)) == 0) {
            // Left (Back)
            MENU_updatestate(MENU_INPUT_BACK);
        } else if ((PINB & _BV(PB2)) == 0) {
            // Down
            MENU_updatestate(MENU_INPUT_DOWN);
        } else if ((PINB & _BV(PB1)) == 0) {
            // Up
            MENU_updatestate(MENU_INPUT_UP);
        } else if ((PINB & _BV(PB3)) == 0) {
            MENU_updatestate(MENU_INPUT_OK);
        }
        ui_check = 0;
    }
```

ADC

ADC-ul este configurat sa masoare in modul diferential, cu un gain de 10x, intre pinii PA0 si PA1 cu tensiune de referinta interna de 2.5V. Prescaler-ul este setat la 64, fapt ce implica o frecventa de masurare de aproximativ 250kHz. Pentru comoditate, ADC-ul este setat pe Auto-Trigger si Free-Running Mode. Practic, se va porni o conversie imediat dupa ce s-a terminat cea anterioara, fara sa fie nevoie setarea bit-ului ADSC.

Am observat ca am avut performantele cele mai bune (zgomot mai mic), atunci cand am folosit rutina de intrerupere pentru finalizarea conversiei ADC-ului. Avand in vedere ca punctul central al proiectului este masurarea semnalului de chitara si apoi modificarea acestuia, aceasta rutina a devenit bucla principala (aici se verifica si starea butoanelor).

Conform documentatiei, ADC-ul in mod diferential are un rezultat in intervalul [-512, 511], insa numerele negative sunt salvate in complement fata de 2.

```
// gather from ADC
read_adc = ADC;

// Twos Complement converter
if (read_adc & _BV(9)) {
    //negative number
    read_adc = read_adc | ~(_BV(10) - 1);
}
```

DAC (TWI)

Implementarea TWI este realizata in fisierele `twi.h` si `twi.c`. In urma implementarii au rezultat urmatoarele:

```
#define TWI_status() (uint8_t)(TWSR & 0xF8)

void TWI_init(void);
void TWI_start(void);
void TWI_stop(void);
void TWI_write(uint8_t data);

// fast mode I2C write
uint8_t DAC_output(uint16_t data);
```

Functia `DAC_output(…)` utilizeaza functiile `TWI_*` pentru a transmite date catre DAC, folosind protocolul din documentatia MCP4725.

Procesare audio

Pentru ca majoritatea documentatiei despre procesarea audio digitala functiona cu operatii in virgula mobila, am ales sa folosesc `float`-uri, cu riscul de a avea o performanta mai slaba a redarii sunetului :). In final, am reusit sa obtin un ton de chitara decent pentru hardware-ul folosit adaugand la compilare flag-urile `-O3 -ffast-math`.

Fiecare efect implementat are o functie `float apply_<effect>(float input)`. Toate functiile considera un sample de semnal in intervalul [-1, 1]. Mai multe efecte pot fi aplicate asupra unui sample de semnal (masurat cu ajutorul ADC-ului) doar prin inlantuirea acestor functii de efecte. Spre exemplu:

```
input = apply_distortion(input);
input = apply_flanger(input);
```

Exista pe deoparte efecte stateless (distortion, tremolo si compression), dar si efecte ce au nevoie de

sample-uri de sunet anterioare pentru a functiona (flanger si echo), de acceea am ales sa stochez toate sample-urile procesate intr-un buffer circular declarat global `output_history`. Numarul de sample-uri ce poate fi stocat este unul nu foarte mare (800), datorita limitarilor hardware.

Primul efect implementat in firmware a fost **Distortion**. Formula pentru obtinerea acestui efect are ca scop modificarea formei semnalului.

```
float apply_distortion(float x)
{
    float depth = 0.64f;
    float gain = 20.0f;

    return (1 - depth) * x + depth * tanh(gain * x);
}
```

Un alt efect stateless este **Tremolo** ce implica combinarea semnalului de la chitara cu semnalul unui oscilator sinusoidal.

```
float apply_tremolo(float x)
{
    float depth = 0.8f;
    float modulation = sin((2 * M_PI / OCR1A) * TCNT1);
    modulation = (1 - depth) + depth * modulation * modulation;

    return x * modulation;
}
```

Pentru a obtine oscilatorul sinusoidal, am folosit timer-ul 1 in modul CTC, cu top la OCR1A. OCR1A este setat in asa fel incat frecventa oscilatorului sa fie intre 2Hz si 20Hz.

Primul efect la care a fost nevoie de sample-uri de semnal anterioare este **Flanger**. Acest efect implica variatia delay-ului. Practic, variaza in timp index-ul semnalului anterior ce este ales pentru a fi interpolat liniar cu semnalul de chitara.

```
float apply_flanger(float x)
{
    float delay = 0.01f;
    float amp = 0.5f;
    float max_sample_delay = delay * MAX_OUTPUT_HISTORY;
    float current_sin = fabs(sin((2 * M_PI / timer0_top) * TCNT0E));
    int16_t current_delay = (int16_t)ceil(current_sin * max_sample_delay);
    int16_t current_delay = (output_history_last + (MAX_OUTPUT_HISTORY - 1)
- current_delay);

    float out_delayed = (float)output_history[current_delay %
MAX_OUTPUT_HISTORY];
    out_delayed /= SGN_MAX;

    return x * amp +
        (1 - amp) * out_delayed;
```

```
}
```

Daca pentru primul tremolo am folosit timer-ul 1 care era pe 16-biti, pentru acest efect am construit o extindere pentru timer 0 pentru a simula un timer pe 16 biti.

```
uint16_t timer0_counter = 0;
uint16_t timer0_top = 0;

#define TCNT0E ((timer0_counter + TCNT0) % (timer0_top + 1))

ISR(TIMER0_COMPA_vect)
{
    timer0_counter = (timer0_counter + 0xFF) % (timer0_top + 1);
}
```

Si aici, timer0_top este setat astfel incat sa se obtina o anumita frecventa a oscilatorului. In acest caz, o frecventa apropiata de 1Hz.

Meniu

Meniul afisat pe LCD este controlat cu butoanele conectate la PB0, PB1, PB2 si PB3 si serveste pentru activarea / dezactivarea efectelor si pentru configurarea particularitatilor fiecarui efect. Implementarea se afla in fisierele `menu.h` si `menu.c`.

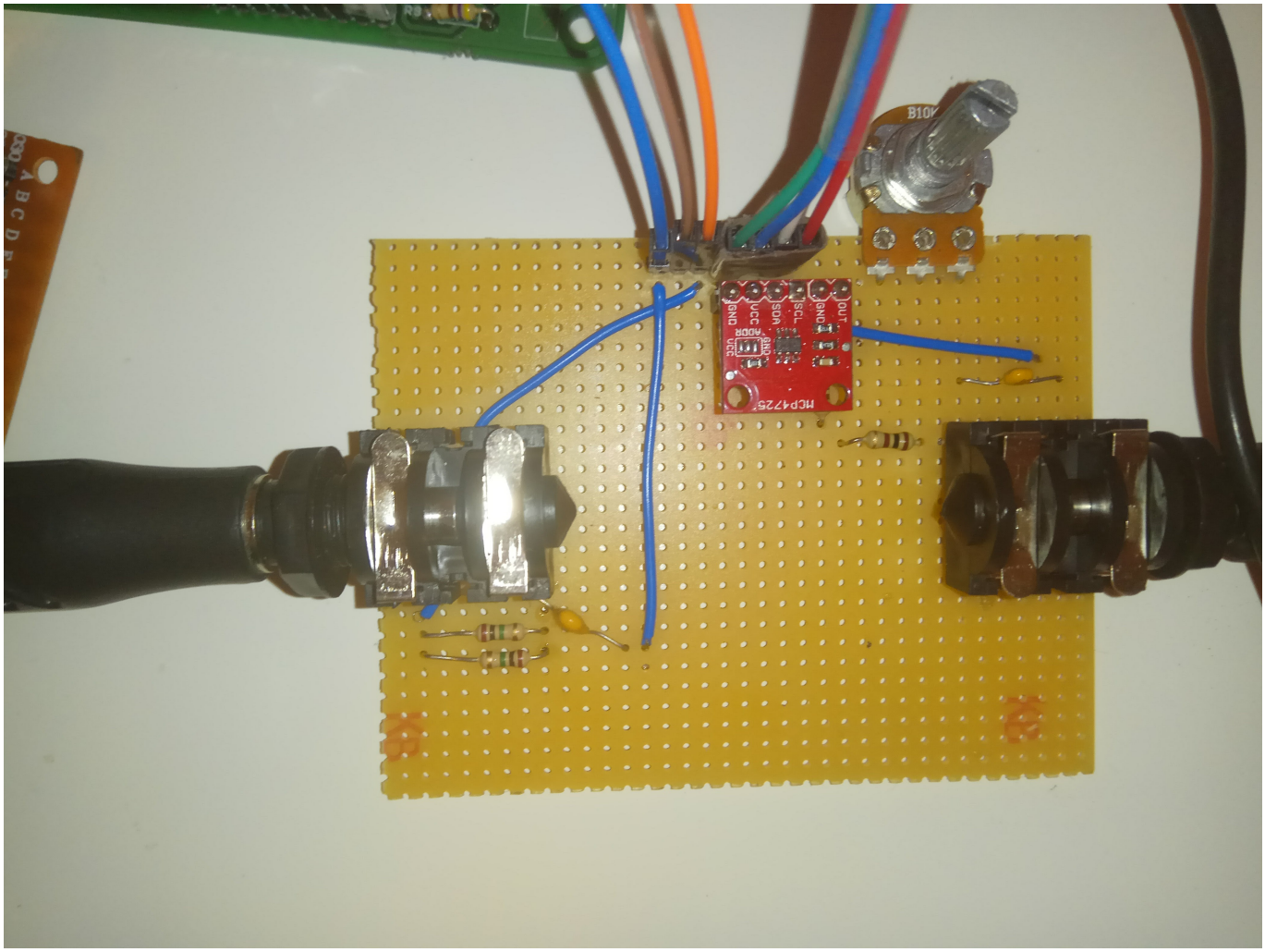
Functia “expusa” este una foarte simpla, ce primeste un enum - butonul ce a fost apasat de utiliztor si modfica starea interna a meniului. Modificarea de stare implica alterarea unei configuratii pentru efecte si totodata modificarea continutului afisat pe LCD.

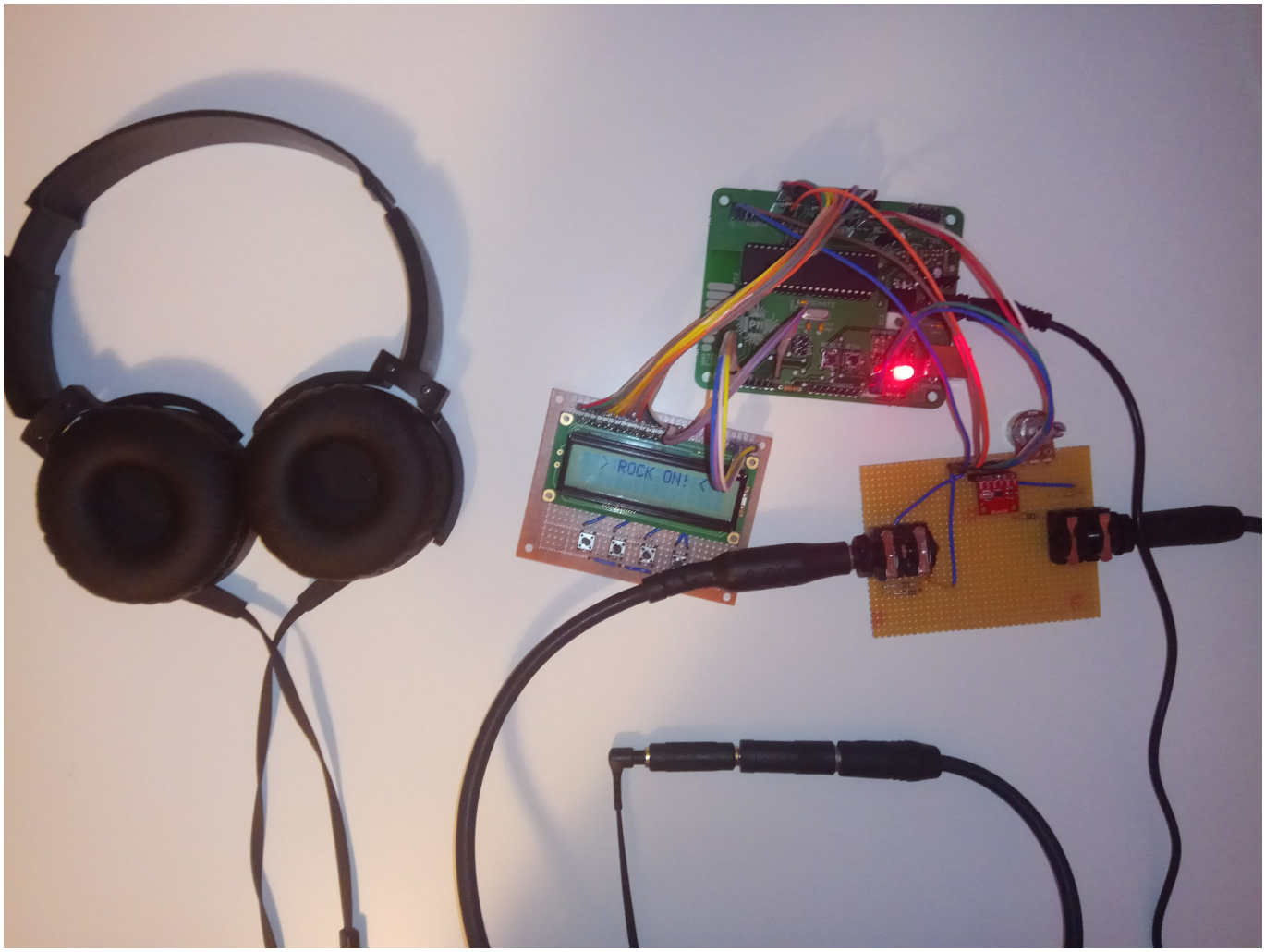
```
void MENU_updatestate(enum MENU_INPUT input);
```

Side note: String-urile folosite in meniu le-am salvat in memoria de program din cauza lipsei de memorie 

Rezultate Obţinute


ROCK ON!


B10K
GND
VCC
SDA
SCL
GND
OUT
ADDR
MCP4725



Concluzii

Comparand acum ce stiam la inceputul proiectului cu ceea ce am descoperit pe parcurs ca nu stiu si trebuie sa inteleg, pot sa spun ca sunt foarte multumit de rezultat :). YOLO.

Download

cmihalache_guitar_effects2.zip

Bibliografie/Resurse

- Google <3 Link
- pedalSheild Link
- Arduino Guitar Pedal Link
- Coding some Tremolo Link
- Digital Audio Effects (Course Slides) Link

- Pyo DSP library (Source Code) Link
- Dynamic processing: Compressor / Limiter Link
- Android library: LibEffects (Source code) Link
- DSP Audio effects Link

- Documentația în format PDF

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

Permanent link:
http://ocw.cs.pub.ro/courses/pm/prj2017/ddragomir/cmihalache

Last update: **2021/04/14 15:07**