Cosmin-Ionuţ TITILIUC (78571) - ATMega Guitar Effects

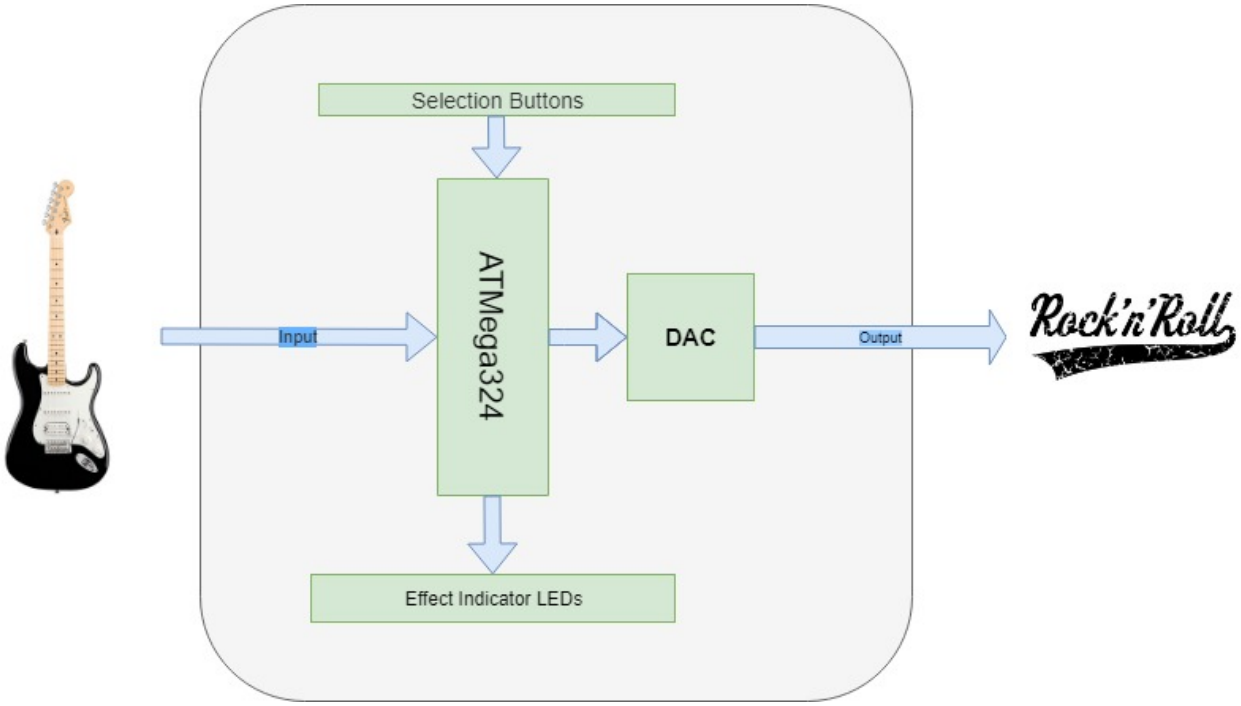
Autorul poate fi contactat la adresa: **Login pentru adresa**

Introducere

Proiectul are ca scop realizarea unui procesor de efecte pentru chitara electrica. El primeste ca input semnalul de la chitara, pe care il modifica pentru a crea diverse efecte. Am o chitara si un amplificator pentru aceasta, insa nu am un procesor de efecte, asa ca, am decis ca este o oportunitate buna sa-mi construiesc unul (“Win-win situation”).

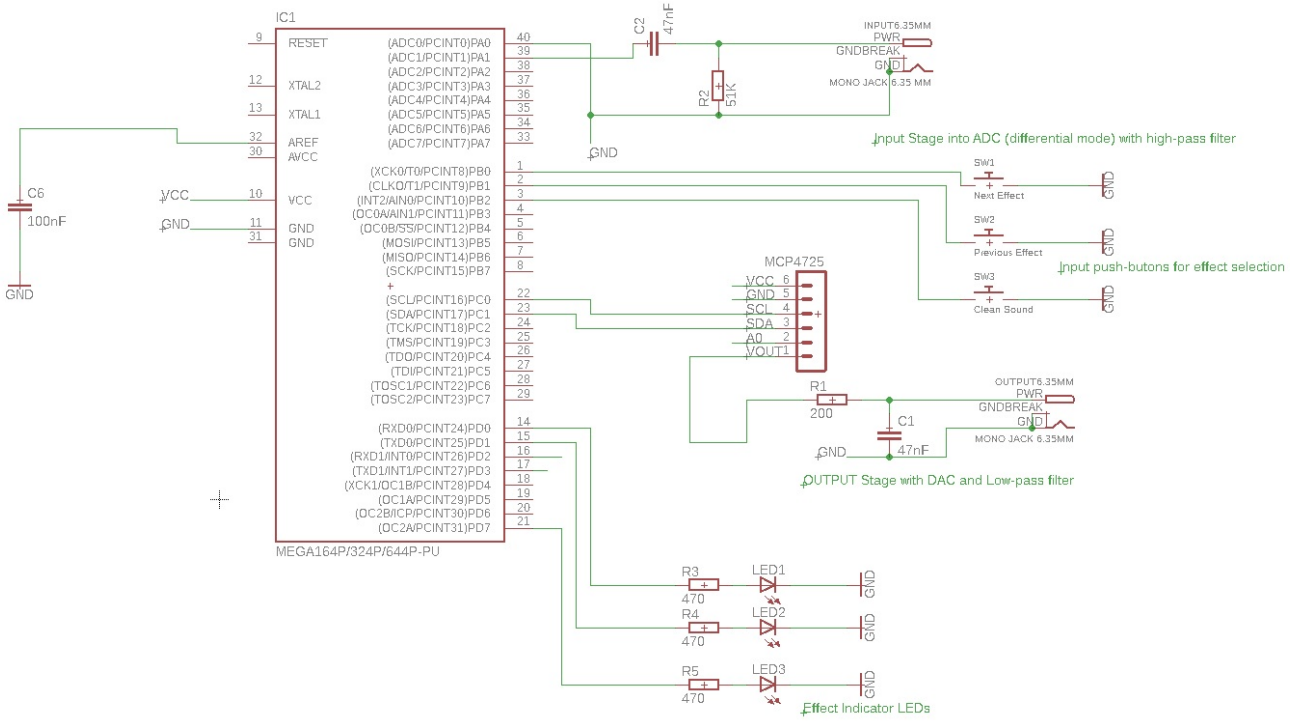
Descriere generală

Schema bloc a proiectului

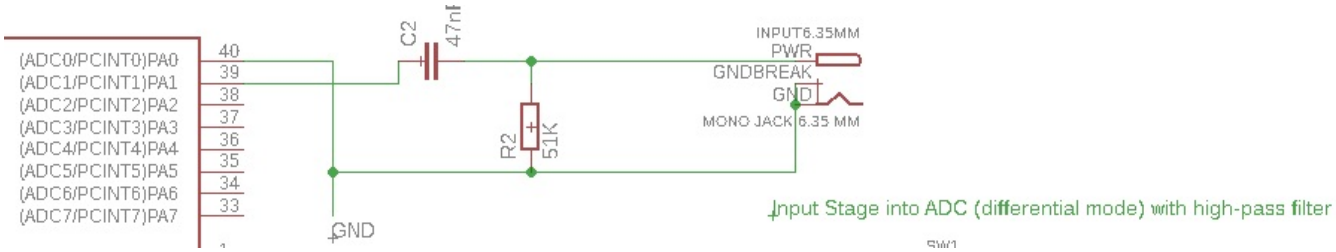


Hardware Design

Schema electrica a proiectului (ansamblu)

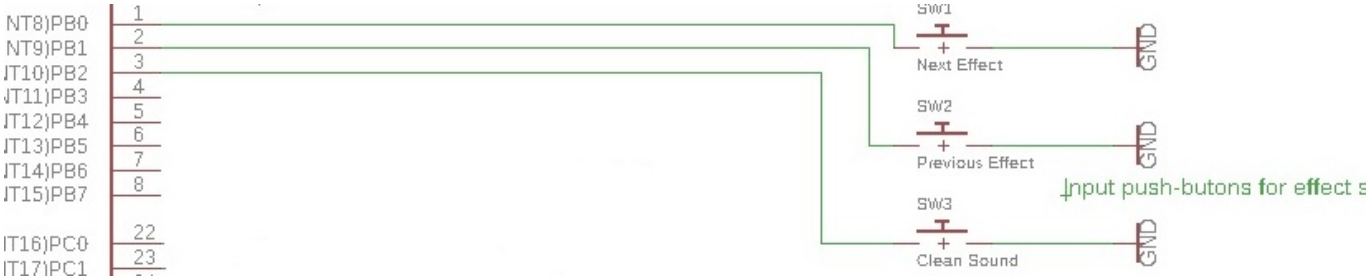


Input Stage



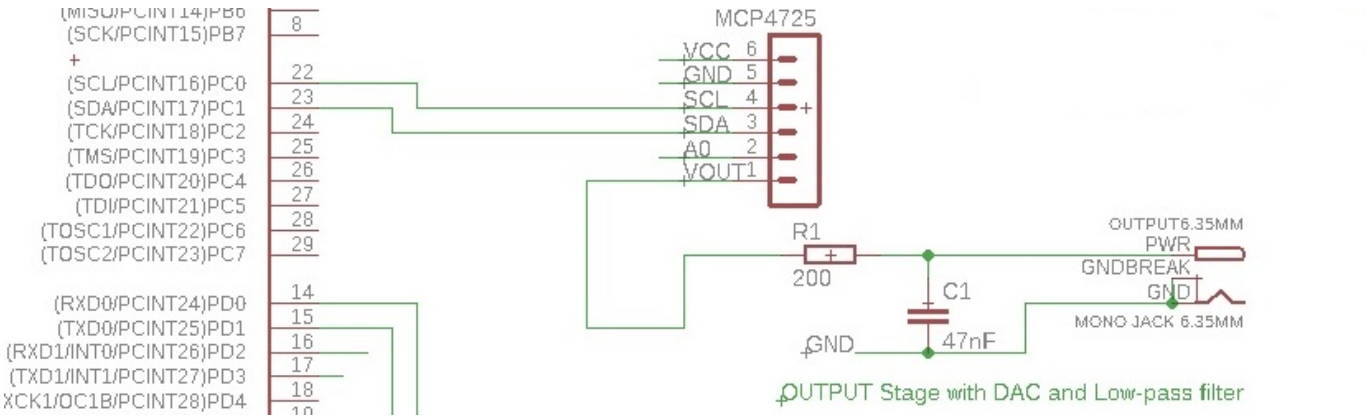
Filtrul trece sus va ignora frecventele sub 70Hz. Cea mai joasa frecventa pe care o chitara electrica acordata in Standard Tuning o poatre produce este de aproximativ 80 de Hz. Semnalul va intra in ADC-ul microcontroller-ului in mod diferential cu un gain de 10x intre PA1 si PA0. Atat pentru intrare cat si pentru iesire avem jack-uri mono de 6.35mm. Valorile rezistentelor si condensatoarelor pentru filtrele de intrare si iesire au fost calculate folosind calculatoare online.

Butoane selectie



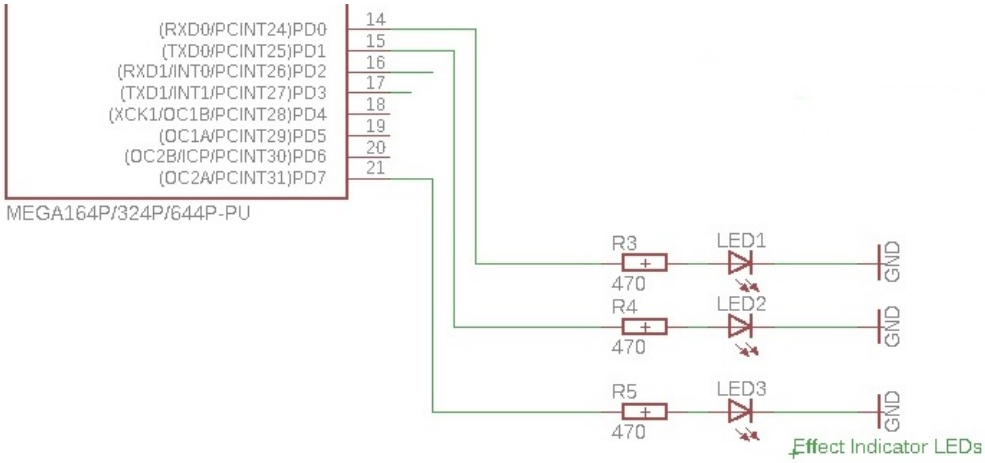
Acestea vor fi butoanele pentru selectia efectului dorit. Doua dintre butoane cicleaza intre cele 4 efecte posibile, iar al treilea buton porneste efectul.

Output stage



Se foloseste un DAC Adafruit MCP4725. VCC si GND vor fi conectate la microcontroller, la fel si SCL si SDA pentru comunicatia de tip TWI I2C. Microcontroller-ul va fi master, iar DAC-ul va fi slave. PIN-ul A0 este folosit pentru a selecta adresa I2C a DAC-ului. Default, daca el nu este conectat la nimic, A0 va avea valoarea logica 0. La iesire vom avea un filtru trece jos (nu am decis inca ce frecvente va trebui sa ignore si deci ce rezistenta si condensator va fi acolo, voi adauga asta la final).

LED-uri indicatoare efect



Simple LED-uri pentru a codifica efectul curent. LED-urile de la PD0 si PD1 vor codifica numarul efectului in binar: 00 gain, 01 delay, 10 reverb, 11 tremolo, iar LED-ul conectat la PD7 va indica daca efectul curent este pornit sau oprit.

Lista pieselor (excluzand placa de baza):

Numar piese	Nume piese	Observatii
1	Placa de baza cu ATMega324a	null
1	Rezistenta 51k Ohmi	filtru trece-sus
1	Rezistenta 200 Ohmi	filtru trece-jos
2	Condensatoare 47nF	ambele filtre
3	Rezistente 470 Ohmi	LED-uri
1	Placa de test	null
2	Mufe Jack 6.3mm mama	Folosite pentru intrare si iesire
1	Adafruit DAC MP4725	https://www.optimusdigital.ro/ro/interfata-altele/3363-modul-adafruit-dac-mcp4725-cu-rezoluie-de-12-bii-i-interfaa-i2c.html?search_query=DAC&results=336

Software Design

Mediul de dezvoltare Dezvoltarea a fost facuta in Windows. Am folosit Programmers Notepad(WinAVR) ca mediu de dezvoltare si HIDBootFlashv1.0 pentru a scrie codul pe placuta.

Biblioteci folosite

Am pornit de la codul oferit aici http://www.electroons.com/blog/interfacing-mcp4725-12-bit-dac-with-avr-microcontroller-avrprayog/ pentru a realiza comunicarea TWI I2C cu DAC-ul MCP4725. Varianta mea de breakoutboard de la Adafruit pentru acest DAC insa a avut o particularitate ciudata. In comunicarea TWI datele se trimit conform schemei:

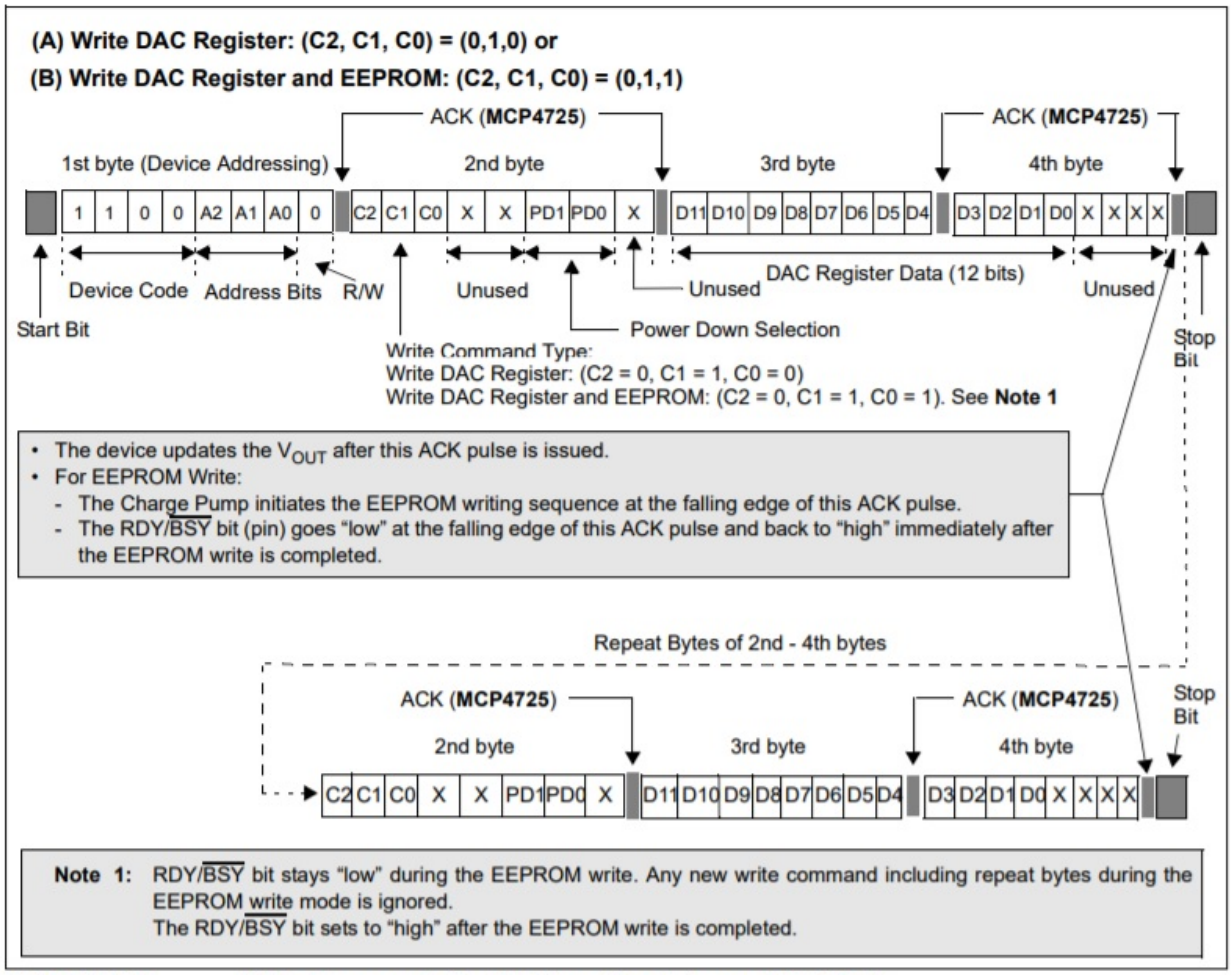


FIGURE 6-2: Write Commands for DAC Input Register and EEPROM.

Se poate observa ca primul byte este folosit pentru device addressing. Pentru MCP4725, in general, bitii A2 si A1 sunt 0. In biblioteca mentionata mai sus, adresa era considerata cu acesti biti setati pe 0 si mi-a luat ceva timp sa ma prind de ce nu functioneaza comunicarea TWI. Acest lucru era datorat faptului ca DAC-ul meu, avea A1 setat pe 1. Astfel, cu A0 la valoarea sa default de 0, byte-ul de adresare corect este: 0b11000100.

Software Design

Primul lucru pe care il face device-ul este partea de initializare. Se initializeaza io-ul (port-urile de

intrare/iesire, rezistente de pull-up pentru butoane), interfata TWI si apoi ADC-ul. ADC-ul este initializat in functia adc_init. Aici el este setat sa foloseasca o referinta interna de 2.56V cu condensator extern conectat la pin-ul AREF pentru reducerea zgomotului. Acest condensator este prezentat in schema hardware. El face parte din placa de baza. Apoi ADC-ul este setat sa functioneze in mod diferential cu gain de 10x intre PA1(pozitiv) si PA0(negativ). Acest lucru va amplifica semnalul de la chitara de 10 ori. De asemenea, prescaler-ul este setat la 64 (ADCul va functiona la 250kHZ), in free running mode cu auto-trigger (conversiile se vor efectua singure una dupa alta). De asemenea, dezactivam digital buffers pentru PA0 si PA1 deoarece nu sunt folosite. La finalul initializarii pornim intreruperile pentru ADC si pornim conversia.

Logica principala a programului nu va avea loc in main, ci va avea loc in handler-ul de intrerupere pentru ADC. Acest lucru este datorat faptului ca ADC-ul va citi la o frecventa de 250kHz semnalul de la chitara, care va fi procesat. In acest handler de intreruperi se va citi valoarea de la ADC, se face normalizarea valorii si aplicarea efectelor asupra acesteia, salvarea valorii rezultat intr-un buffer de delay (ce va fi folosit pentru efectele de delay si reverb), trimiterea rezultatului catre DAC folosind functia dac_send din twi.h, apoi verificarea pentru input nou de la butoane si update-ul LED-urilor.

Controlul efectelor si starea LED-urilor Pentru butoanele PB0, PB1 si PB2 sunt active intreruperile ce vor seta variabila globala button_change pe 1. Starea butoanelor este apoi verificata in handler-ul pentru intreruperile ADC astfel: if (button_change == 1) {

```
button_change = 0;
if ((PINB & (1 << PB0)) == 0) {
    current_effect = (EFFECTS_NO + current_effect - 1) % EFFECTS_NO;
}
if ((PINB & (1 << PB1)) == 0) {
    current_effect = (EFFECTS_NO + current_effect + 1) % EFFECTS_NO;
}
if ((PINB & (1 << PB2)) == 0) {
    if ( effect_on[current_effect] == (int8_t)TRUE) {
  effect_on[current_effect] = (int8_t)FALSE;
    } else {
        effect_on[current_effect] = (int8_t)TRUE;
    }
}
```

} PB0 si PB1 vor schimba in mod ciclic variabila current_effect ce ia valori intregi in intervalul [0, EFFECTS_NO). Vom avea de asemenea vectorul global effect_on, folosit pentru a salva starea efectelor (ON sau OFF). Butonul PB2 va schimba starea efectului selectat curent. LED-urile sunt setate in functie de starea programului cu ajutorul functiei: void toggle_leds(void).

Procesare efecte

Partea de procesare a efectelor are loc in functia: float process(float signal). Aici se verifica pe rand elementele din vectorul effect_on pentru a vedea ce efecte avem active si le adaugam pe rand cu ajutorul a 4 functii:

1. float add_distortion(float signal);
2. float add_delay(float signal);
3. float add_reverb(float signal);
4. float add_tremolo(float signal);

Implementarea efectelor: -add_distortion: se foloseste de functia atan si parametrii “gain” si “depth” pentru a distorsiona efectul;

```
  float depth = 0.5f;
  float gain = 20.0f;
  return (1-depth) * signal + depth * atan(gain * signal);
```

-add_delay(float signal): cum spuneam pentru a realiza efectul de delay, am salvat valorile anterioare ale semnalului intr-un vector. Efectul de delay va suprapune semnalul de acum echo_delay “cicli” peste semnalul curent cu o anumite adancime. Adancimea sugereaza cat % din rezultat va fi reprezentat de semnalul curent si cat % de semnalul din trecut. Eu am setat echo_delay (distanta de la care vine ecoul) la 300 si echo_depth la 0.5f.

-add_reverb(float signal): Reverb este un efect ce imita efectul pe care il are propagarea sunetului intr-o sala mai mare. Astfel, aceleasi unde sonore se vor intoarce la urechea ascultatorului la momente de timp diferite, in functie de distanta peretelui/obiectului de care s-au reflectat. Efectul se aseamana cu cel all delay-ului insa vom culege 3 sunete de la 3 momente de timp anterioare diferite din vectorul de delay si le vom suprapune peste semnalul curent cu ponderi diferite. Exemplu cod pentru functia reverb:

float add_reverb(float signal) {

```
float reverb;
int d1 = 50;
int d2= 100;
int d3 = 150;
float gain1 = 0.2f;
float gain2 = 0.1f;
float gain3 = 0.1f;
reverb = 0.6f * signal;
//add reverb from 3 different locations
reverb += gain1 * ((float)delay_buffer[(DelayCounter + MAX_DELAY - d1) %
MAX_DELAY] / SGN_MAX);
reverb += gain2 * ((float)delay_buffer[(DelayCounter + MAX_DELAY - d2) %
MAX_DELAY] / SGN_MAX);
reverb += gain3 * ((float)delay_buffer[(DelayCounter + MAX_DELAY - d3) %
MAX_DELAY] / SGN_MAX);
return reverb;
```

}

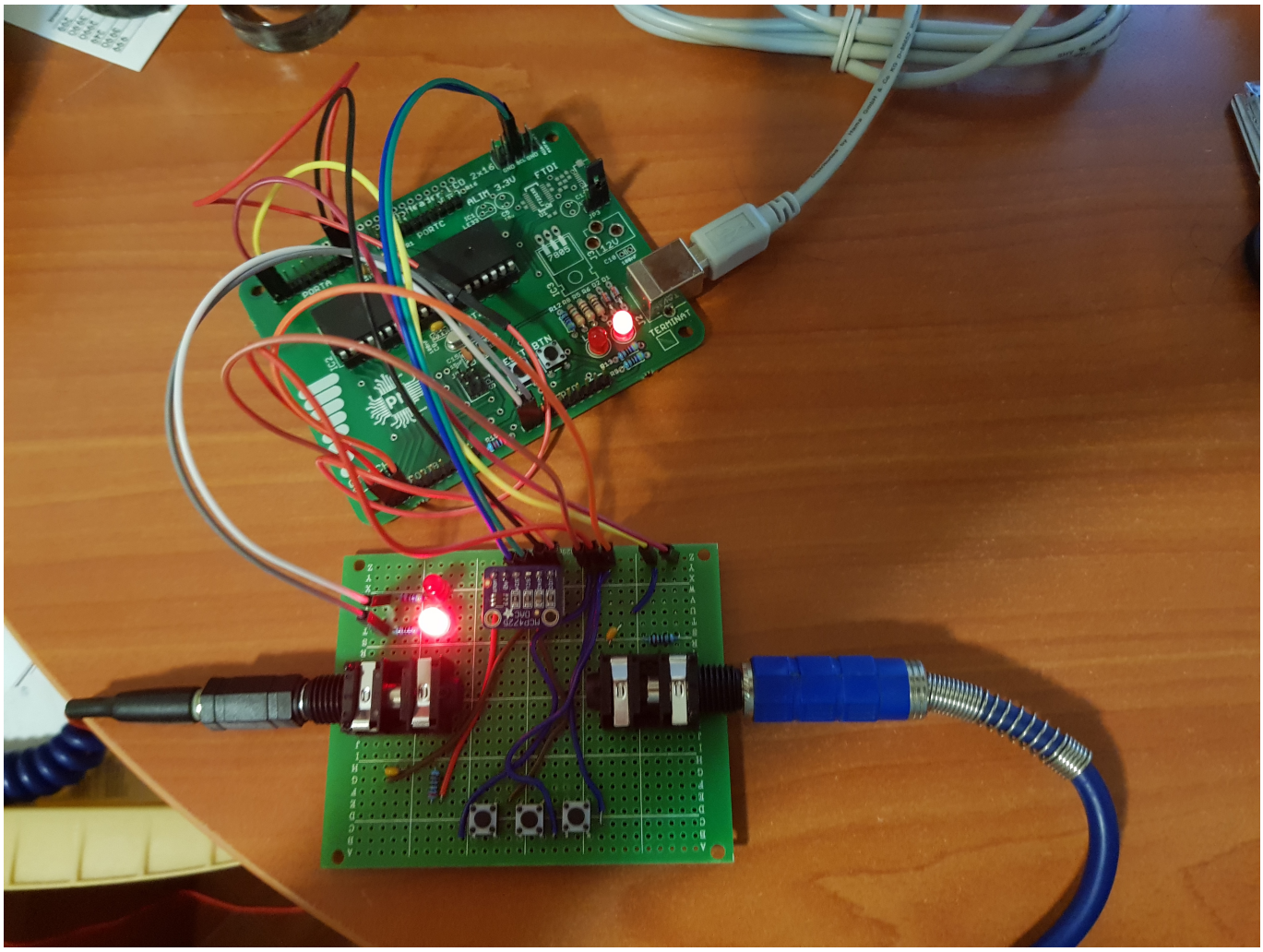
-float add_tremolo(float signal): a fost ultimul efect implementat. El consta in principal in aplicarea unei sinusoide, numita modulatie, asupra semnalului initial. Frecventa oscilatiei sinusoidei o controlez prin #define MAX_SAMPLE 150 si variabila globala int16_t current_sample = 0. current_sample ia valori in intervalul [0, MAX_SAMPLE) si este incrementata la finalul fiecarei conversii ADC. Efectul asupra sunetului chitarii este acela al unui tremurat al amplitudinii acestuia. Formula folosita este:

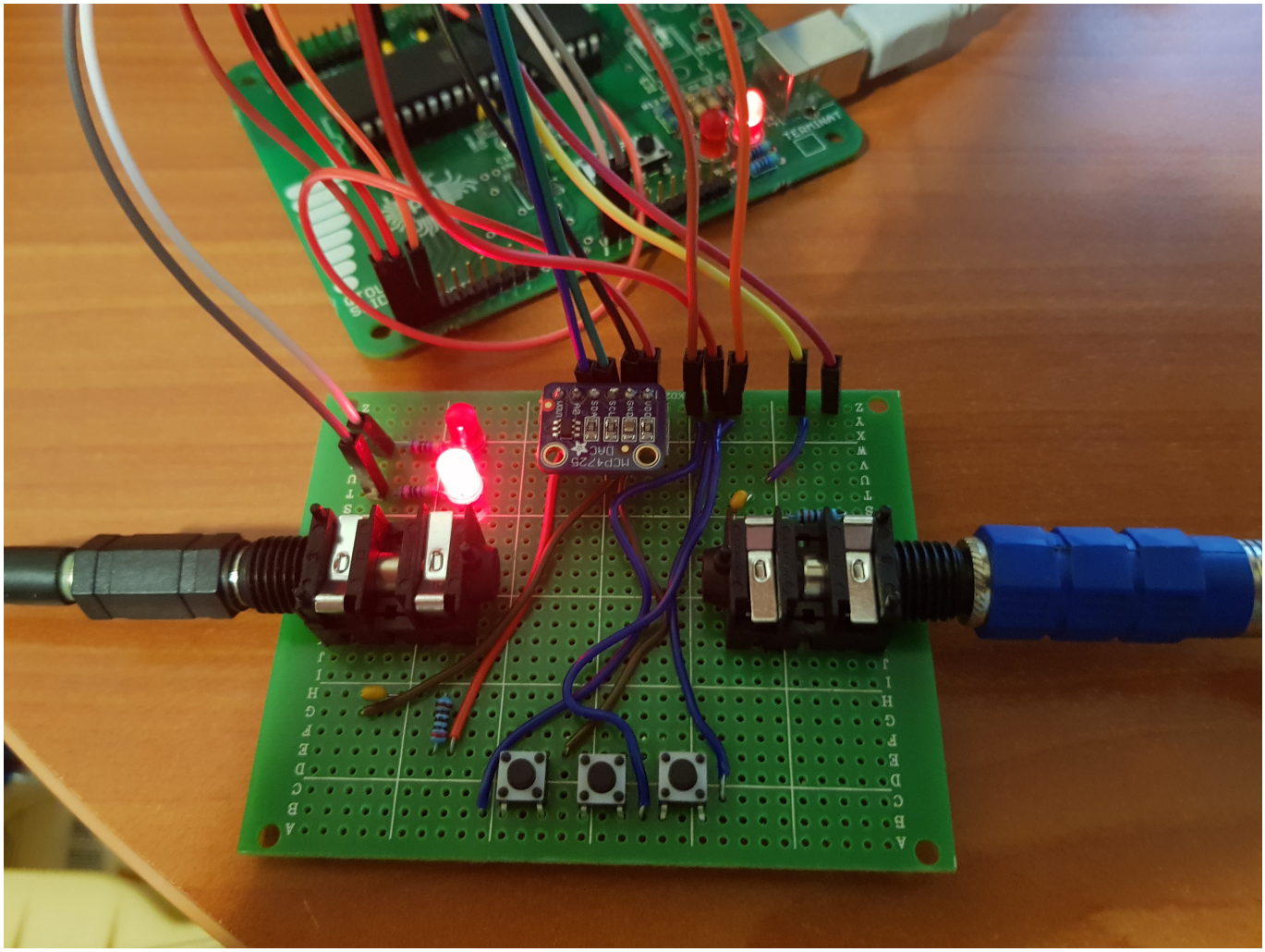
float tremolo_depth = 0.5f; float modulation = sin((2 * M_PI / MAX_SAMPLE) * current_sample); modulation = (1 - tremolo_depth) + tremolo_depth * modulation; return modulation * signal;

(Inspirata de aici: https://christianfloisand.wordpress.com/2012/04/18/coding-some-tremolo/).

Rezultate Obţinute

Forma finala hardware:





Sunet obtinut Un sunet suficient de bun pentru cat hardware am implementat si avand in vedere rezolutia destul de slaba atat a ADC-ului cat si a DAC-ului. Insa, daca suprapun mai multe efecte, sunetul devine destul de distorsionat si nu se mai disting notele prea bine.

Concluzii

Am vazut acest proiect in Hall of Fame, realizat de Constantin Mihalache si am vrut sa realizez un proiect similar, deoarece si eu cant la chitara si mi-am dorit de ceva timp un procesor de efecte. Avand in vedere ca este primul meu proiect ce implica hardware, am avut indoieli destul de mari asupra reusitei pe parcursul proiectului, in special cand, intr-o forma finala, acesta nu functiona din cauza problemei cu adresa de I2C a DAC-ului. Satisfactia a fost insa destul de mare in momentul in care sunetul de chitara s-a auzit in casti, trecand prin procesor. Ceea ce m-a uimit este ca in momentul in care nu ating corzile chitarii, zgomotul in casti este foarte mic, cel mai probabil datorita filtrelor pe intrare si iesire (EEA flashbacks).

Download

332cb_titiliuc_cosmin_proiect.zip

Bibliografie/Resurse

Inspiratia pentru realizarea acestui proiect:
http://cs.curs.pub.ro/wiki/pm/prj2017/ddragomir/cmihalache

Datasheet pentru DAC: https://cdn-shop.adafruit.com/datasheets/mcp4725.pdf

Un mic tutorial pentru DAC:
https://cdn-learn.adafruit.com/downloads/pdf/mcp4725-12-bit-dac-tutorial.pdf

Efecte:

http://atmega32-avr.com/guitar-special-effects-using-atmega32/

https://christianfloisand.wordpress.com/2012/04/18/coding-some-tremolo/

Interfata TWI:

http://www.electroons.com/blog/interfacing-mcp4725-12-bit-dac-with-avr-microcontroller-avrprayog/

- Documentația în format PDF