Cristian-Viorel POPA (78451) - Guitar Hero

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

Introducere

Guitar Hero este un joc in care scopul este apasarea unor butoane, rasplata este o melodie bine-cantata, iar modul de a o obtine este prin reflexe excelente!

Sper ca si proiectul meu se va ridica la standardele acestea pentru distractia neincetata a publicului adorator de muzica de orice fel.

Descriere generală

Flow-ul proiectului va fi urmatorul:

- Playerul va selecta cu ajutorul unui buton o melodie de pe cardul **MicroSD** (titlul melodiilor prin care se trece va fi afisat pe un **LCD**)
- Melodia va incepe sa fie redata pe buzzer si incep o serie de runde in care:
  - Un set de **LED-uri** se vor aprinde
  - Playerul va trebui sa „atinga” notele corespunzatoare, prin **senzori infrarosu**
  - Daca se atinge o nota gresita, melodia se va intrerupe pentru o jumatate de secunda, iar pe buzzer va fi redat un sunet menit sa fie enervant
  - La finalul rundei, daca nu toate notele au fost atinse, atunci va fi redat acelasi sunet

Va exista si un sistem de scoring. La fiecare nota corect atinsa, scorul va creste, iar la fiecare nota incorect atinsa sau runda in care unele note nu au fost atinse deloc, scorul va scadea. Exista si un sistem de combo, cu ajutorul caruia scorul va creste mai repede sau mai incet in functie de cat de bine se descurcs jucatorul.

Notele care trebuie atinse sunt selectate in mod aleator. Cand una din ele este atinsa (este intrerupt laserul de la dioda la fotorezistor), ledul corespunzator notei se stinge. O runda se termina dupa o perioada fixata de timp, chiar daca toate notele din runda respectiva au fost atinse.

Butonul va schimba melodia redata in ordinea lor de pe cardul SD.

Hardware Design

1. Lista cu piese

Pe langa PCB, microcontroller si celelalte componente obligatorii, am avut nevoie de:

Componente	**Pret(RON)**
LCD Hitachi HD44780	10.5
4 diode laser	18
4 fotorezistente	8
4 rezistente de 10K	0.4
4 LED-uri	2
Modul cu buzzer	5.5
Modul slot MicroSD	6
Buton	1
Multi conectori board-to-board	~10
Fire mama-mama si mama-tata	22

Am gasit absolut toate componentele de mai sus la Optimus Digital Apaca.

2. Schema electrica

Observatii:

1. Am conectat manual butonul la GND cu o rezistenta de 100, fara sa iau in calcul rezistenta de pull-up.

2. LCD-ul a fost conectat exact ca in cablajul placii.

3. Nu am gasit un modul de buzzer pentru EAGLE, deci am pus circuitul din laboratorul 0.

4. Nu am un regulator de tensiune la 3.3V, dar acesta se afla in modulul de MicroSD.

Software Design

Am folosit urmatoarele soft-uri de-a lungul proiectului:

- **Schema bloc**: www.draw.io
- **Schema electrica**: EAGLE si biblioteci pentru Atmel (element14), MicroSD si fotorezistori
- **IDE**: Programmer's Notepad (WinAVR)

Nu am folosit nicio biblioteca externa, in afara de cele de la laboratorul de PM.

In continuare, voi descrie implementarea pentru fiecare parte a proiectului.

Selectarea notelor si atingerea lor

Mecanismul notelor cuprinde 3 componente:

- diodele laser
- fotorezistorii
- LED-urile

a) Diodele laser

Acestea sunt conectate intre VCC si GND. Ele sunt aprinse mereu pe parcursul programului, deci nu trebuie programate.

b) Fotorezistorii

Acestia sunt controlati prin pinii de pe PA0, PA1, PA3 si PA4.

Pentru modularizare, am definite o constanta de NUM_NOTES care reprezinta numarul total de note existente si un vector pentru a retine pinii fotorezistorilor.

```
#define NUM_NOTES 4

int notes[NUM_NOTES] = {PA0, PA1, PA3, PA4};
```

Pentru a ma folosi de fotorezistori, activez intreruperile pentru pinii lor, iar atunci cand laserul de la diode este intrerupt, verific care din pini are valoarea 0 si tratez situatia.

```
// set pins for photoresistors as input
for (i = 0; i < NUM_NOTES; i++) {
    DDRA &= ~(1 << notes[i]);
    PORTA |= (1 << notes[i]);
}


// set interrupts for photoresistors
PCMSK0 = 0;
PCMSK0 |= (1 << PCINT0);
PCMSK0 |= (1 << PCINT1);
PCMSK0 |= (1 << PCINT3);
PCMSK0 |= (1 << PCINT4);


// interrupt on photoresistors (note hit)
ISR(PCINT0_vect)
{
    // save PINA, so it doesn't change mid-interrupt
    int PINA_snapshot = PINA;
```

```
    int i;

    for (i = 0; i < NUM_NOTES; i++) {
        if ((PINA_snapshot & (1 << notes[i])) == 0) {
            // treat the hit of a note...
        }
    }
}
```

c) LED-urile

O runda si ce trebuie sa faca playerul este, practic dictata de LED-uri. La inceputul rundei, acestea se aprind aleator, iar acesta trebuie sa intrerupa laserul diodelor catre fotorezistorii corespunzatori.

LED-urile se afla pe pinii PD0, PD1, PD4 si PD7. La fel ca la fotorezistori, am definit un vector cu acesti pini.

```
int leds[NUM_NOTES] = {PD7, PD5, PD1, PD0};
```

Pentru a retine starea unei runde, folosesc 2 vectori de booleene: unul pentru notele care trebuie atinse si unul pentru cele ce au fost deja atinse runda asta:

```
// state of the round
bool should_be_hit[NUM_NOTES];
bool was_hit[NUM_NOTES];
```

Mai multe LED-uri pot fi aprinse simultan. In momentul cand se apeleaza functia de aprindere a acestora, fiecare are aceeasi sansa de a se aprinde, insa cel putin unul va fi aprins la final. Astfel, sunt 2 parti ale functiei:

- se itereaza prin fiecare LED si daca se castiga o sansa, LED-ul se va aprinde
- daca niciunul nu a fost aprins in prima faza, atunci se ia un numar aleator intre 0 si NUM_NOTES si se aprinde LED-ul corespunzator

```
#define NOTE_PROBABILITY_DENOMINATOR 3

// randomly light leds and set the should_be_hit array
void light_leds()
{
    int i;
    int rand_number;
    // check that at least one led was lit up
    bool at_least_one = false;

    // reset the seed
    srand(random_seed);

    // randomly light up leds
    for (i = 0; i < NUM_NOTES; i++) {
        rand_number = rand() % NOTE_PROBABILITY_DENOMINATOR;
```

```
        PORTD &= ~(1 << leds[i]);
        should_be_hit[i] = false;

        if (rand_number == 1) {
            // one was lit successfully!
            at_least_one = true;

            PORTD |= (1 << leds[i]);
            should_be_hit[i] = true;
        }
    }

    if (at_least_one == false) {
        rand_number = rand() % 4;

        PORTD |= (1 << leds[rand_number]);
        should_be_hit[rand_number] = true;
    }
}
```

`random_seed` este o variabila incrementata intr-o bucla while. Nu exista o corelatie intre apelul functiei `light_leds` si incrementarea varibilei, deci rezultatul ar trebui sa fie diferit cu fiecare rulare a programului.

FUN FACT: Pentru `NOTE_PROBABILITY_DENOMINATOR` avand valoarea 3, fiecare LED are o sansa de ~38% sa se aprinda.

Inceperea si terminarea unei runde

O runda dureaza o perioada exacta de timp, indiferent daca toate notele au fost atinse deja sau nu.

Timerele 0 si 1 ale microcontrollerului sunt folositi de functia de play a unei melodii (din laboratorul 4). Pentru masurarea timpului rundei folosesc timerul 2. Totusi, o runda dureaza 4 secunde, deci este nevoie de mai multe intreruperi pentru ca un timer pe 8 biti sa ajunga la o valoare de 62500 cu prescaler de 1024.

Retin numarul de intreruperi ale timerului si, cand ajung la o anumita valoare, resetez counterul, verific finalul rundei si incep una noua.

```
#define TIMER_COUNTER_MAX 125

// timer interrupt counter
unsigned int timer_counter = 0;

void timer2_init()
{
    TCNT2 = 0;

    // CTC mode with top at OCR2A
```

```
    TCCR2A &= ~(1 << WGM20);
    TCCR2B |=  (1 << WGM21);
    TCCR2B &= ~(1 << WGM22);

    // prescaler of 1024
    TCCR2B |=  (7 << CS20);

    // set OCR2A and activate interrupt
    TIMSK2 = (1 << OCIE2A);
    OCR2A = 250;
}

// interrupt after a round expired
ISR(TIMER2_COMPA_vect)
{
    timer_counter++;

    if (timer_counter == TIMER_COUNTER_MAX) {
        timer_counter = 0;
        check_round_end();
    }
}
```

In functia check_new_round, verific starea notelor, daca player-ul a reusit sa le atinga pe toate si acord un scor corespunzator.

```
// check the results of the round
void check_round_end()
{
    int i;
    bool lost = false;

    // check how the player did this round
    for (i = 0; i < NUM_NOTES; i++) {
        if (should_be_hit[i] == true && was_hit[i] == false) {
            lost = true;

            break;
        }
    }

    // if the player missed at least one note
    if (lost) {
        score -= 300;
        combo = 0;

        play_fail();
    }

    // the player score increases after each hit, so no reason to increase
```

```
it here

    // update the LCD and start a new round
    LCD_update();
    new_round();
}
```

Despre functia LCD_update voi vorbi intr-o sectiune ulterioara.

Functia new_round reseteaza vectorii pentru starea rundei si apeleaza light_leds.

```
// start a new round
void new_round()
{
    int i;

    for (i = 0; i < NUM_NOTES; i++) {
        should_be_hit[i] = false;
        was_hit[i] = false;
    }

    // setup new round notes (light_leds)
    light_leds();
}
```

Afisarea pe LCD

Pentru LCD, am folosit scheletul din laboratorul 1, modificand pinii pentru date si control in mod corespunzator.

Functionalitati suplimentare pe care le-am adaugat:

- Am activat backlight-ul prin setarea pinului PC2, conform cablajului placii
- Am adaugat o functie pentru clear-ul display-ului
- Am implementat functia LCD_update

Pentru inceput, functia LCD_clear apeleaza un LCD_writeInstr.

```
void LCD_clear()
{
   LCD_writeInstr(LCD_INSTR_clearDisplay);
}
```

LCD-ul va fi actualizat cand se intampla unul din urmatoarele evenimente:

- player-ul atinge orice nota
- melodia este schimbata
- incepe o noua runda

Pe acesta vor fi afisate doua lucruri, pe cele doua linii ale display-ului:

- numele melodiei care este redata acum
- scorul player-ului

```
// update the display (track name and score)
void LCD_update()
{
    char score_message_string[32] = "Score: ";
    char score_value_string[32];

    // turn the value of the score into a string
    itoa(score, score_value_string, 10);

    // concatenate the message and the value
    strcat(score_message_string, score_value_string);

    // clear the LCD
    LCD_clear();

    // print the name of the file being play on the first line and the score
on the second
    LCD_printAt(0, filename + 9);
    LCD_printAt(LCD_INSTR_nextLine, score_message_string);
}
```

Redarea unei melodii

Melodiile care sunt redate se gasesc pe un MicroSD, cu care microcontrollerul comunica folosind framework-ul din laboratorul 4.

Pinii la care este conectat modulul cardului sunt exact aceiasi ca la laborator, deci nu am facut multe modificari, exceptie facand:

- pornirea buzzerului, in loc de setarea si desetarea pinului de buzzer ca output.
- setarea timerului 1 pe modul FPWM cu top la OCR1A = 0x00FF in loc de FPWM cu top la 0x00FF (acelasi lucru, dar am folosit pentru testare)
- stergerea functiei `continue_play()` si inlocuirea sa cu o variabila `continue_play`, care este `true` cand melodia trebuie sa fie redata si `false` cand se trece la o alta.

Numele melodiei care este redata este pastrat intr-o variabila `filename`. In plus, retin un index al melodiei la care am ramas. Melodia se schimba atunci cand se apasa pe buton, deci folosesc o intrerupere:

```
// set interrupt on the button
PCMSK1 |= (1 << PB2);
PCICR  |= (1 << PCIE1);

// interrupt on button press (track change)
```

```
ISR(PCINT1_vect)
{
    if ((PINB & (1 << PB2)) != 0) {
        change_track();
    }
}

// change the track and do all the management stuff
void change_track()
{
    track_index++;

    // cycle through tracks
    if (track_index > NUM_TRACKS) {
        track_index = 1;
    }

    // get the next track filename
    get_music(track_index, MUSIC, filename);

    continue_play = false;
    // reset round timer counter
    TCNT2 = 0;

    // start a new round
    new_round();
}
```

In final, main-ul arata astfel:

```
int main()
{
    sei();

    LCD_init();
    IO_init();

    for(;;) {
        // mount filesystem
        FRESULT rez = pf_mount(&fs);

        if(rez == FR_OK) {
            break;
        }
    }

    get_music(track_index, MUSIC, filename);

    // do an initial LCD update
    LCD_update();
```

```
    // everything is based on interrupts
    init_intr();
    timer2_init();

    play(filename);

    // this will be executed each time a track has been skipped
    for (;;) {
        get_music(track_index, MUSIC, filename);
        LCD_update();

        play(filename);
    }

    return 0;
}
```

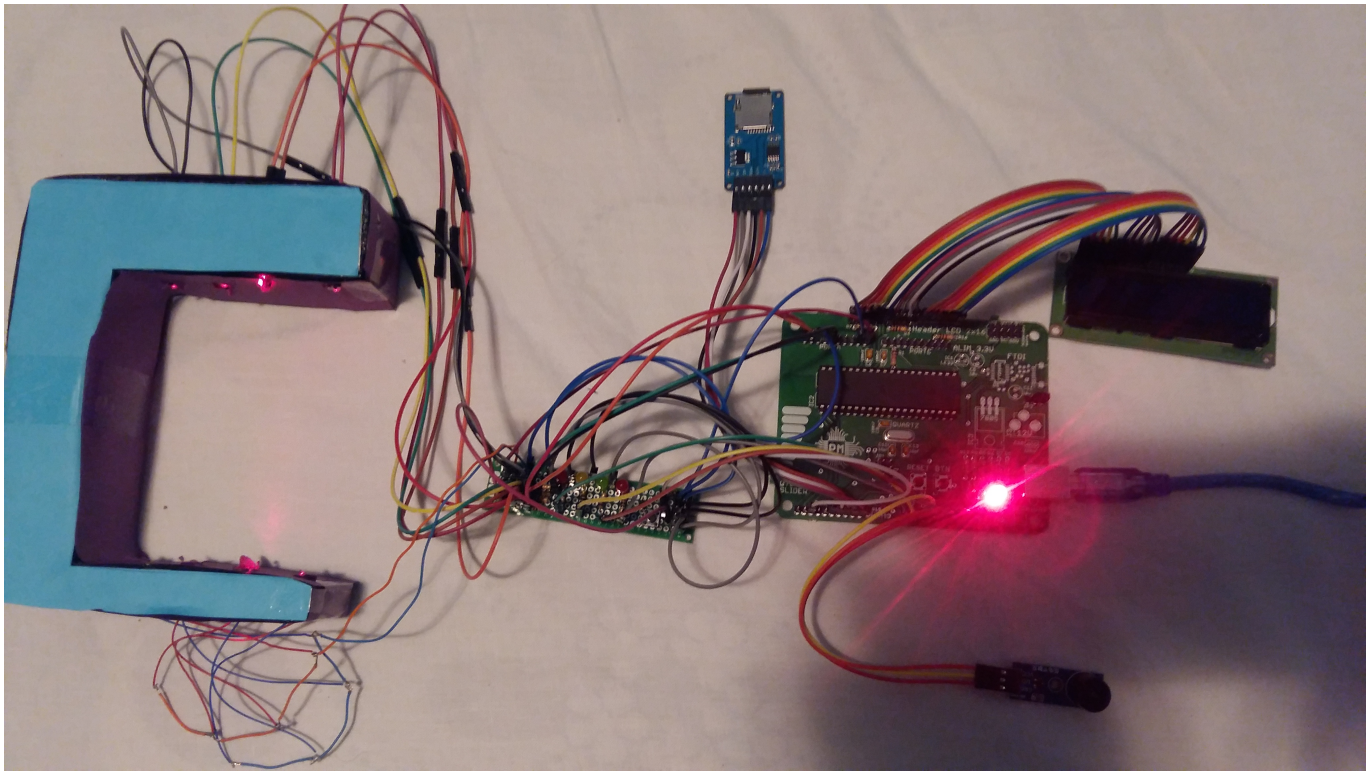
Observatii:

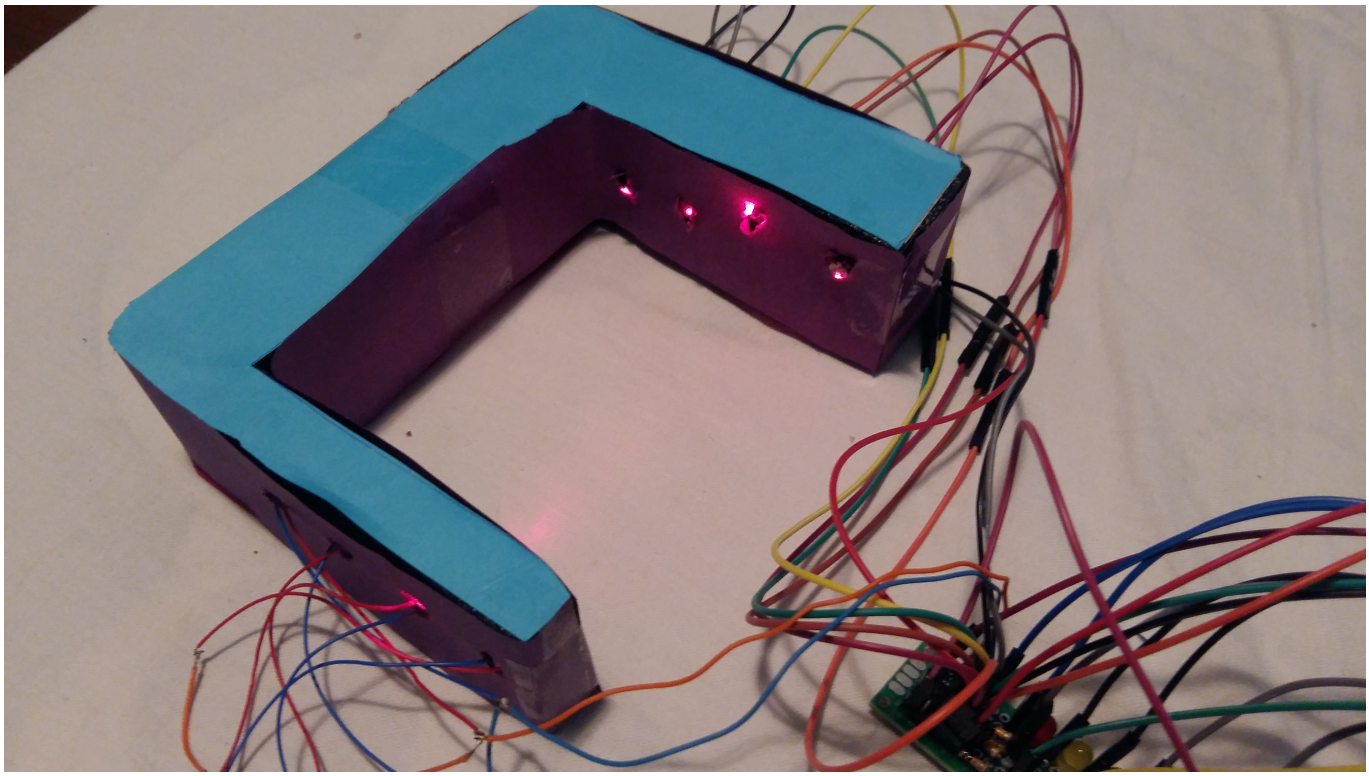
- Inca nu am reusit sa redau corect o melodie (e mai degraba huruit decat muzica)
- Exista sansa ca butonul de schimbat melodia sa nu selecteze fisierul dorit si sa strice programul :(

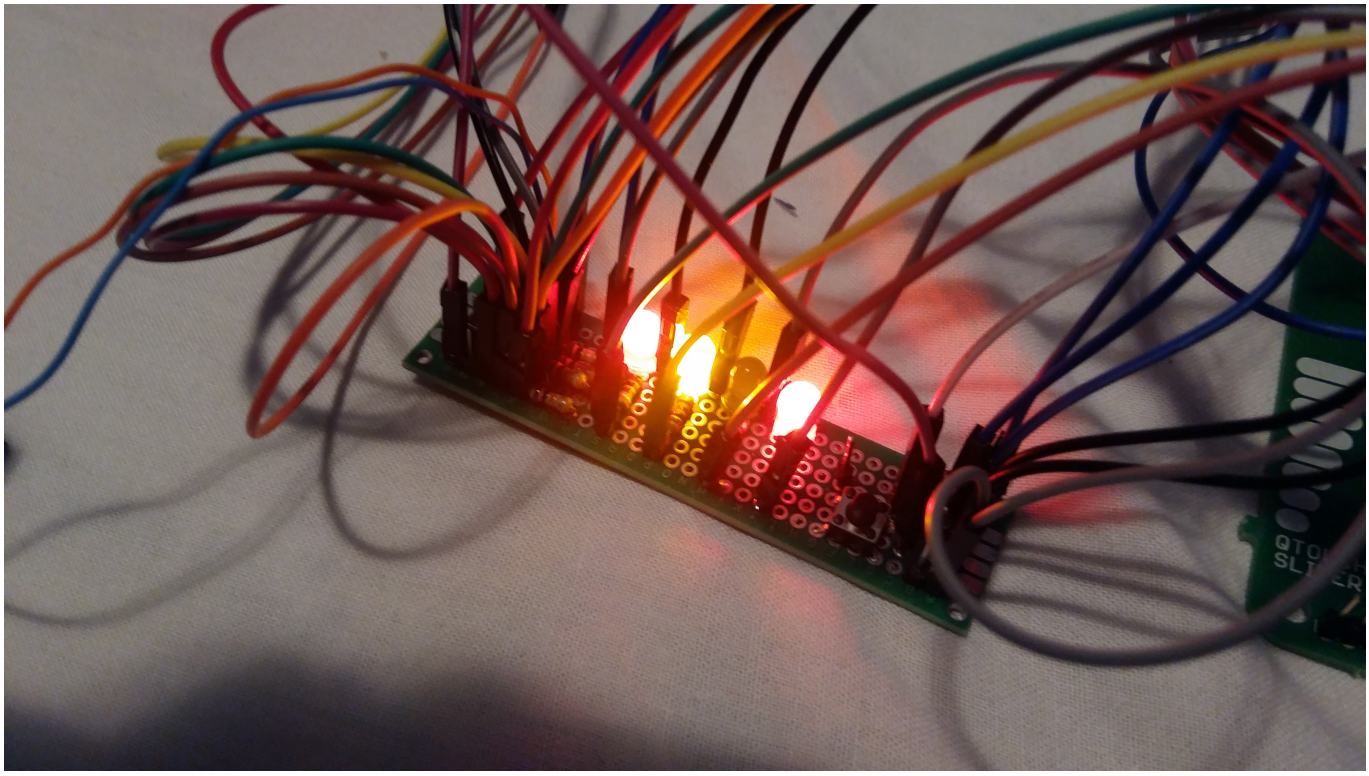
Rezultate Obţinute

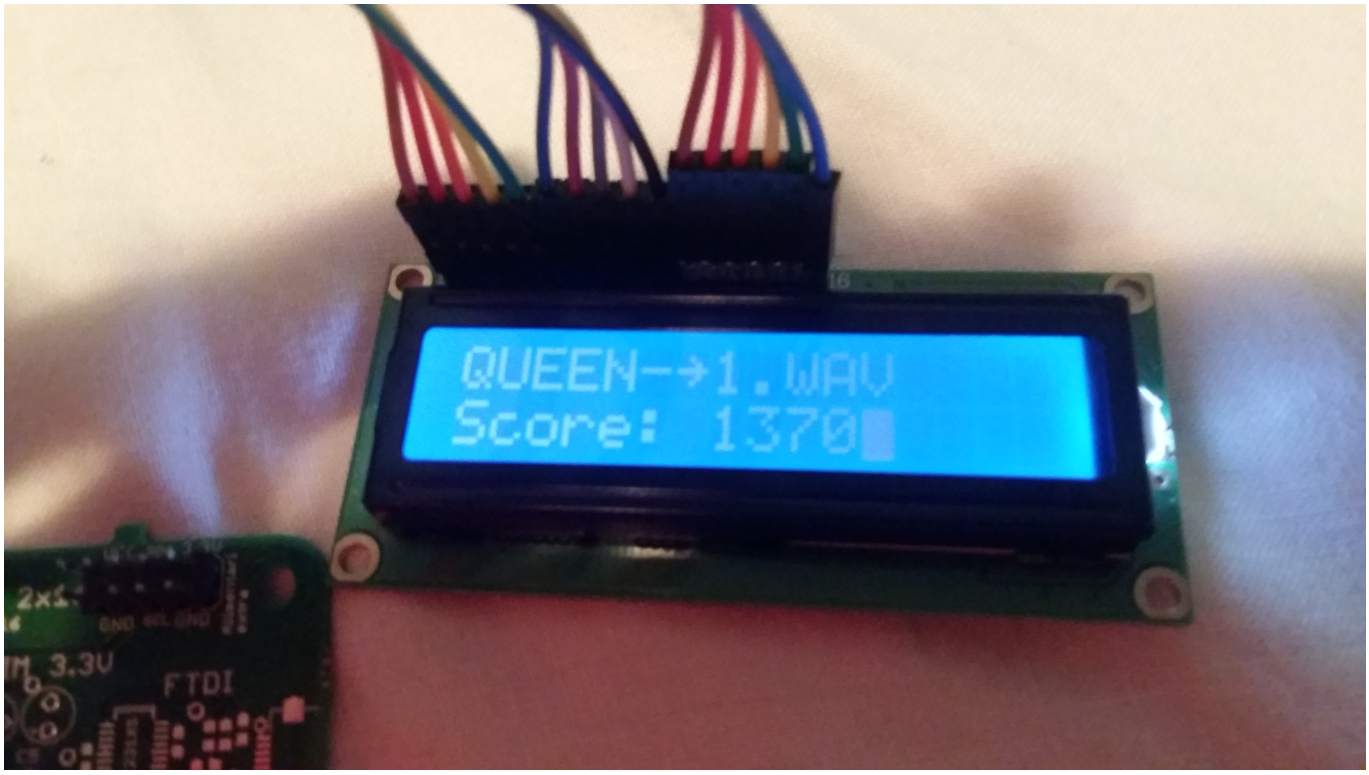
Voi pune cateva poze si un video pentru a arata starea proiectului momentan (cum am spus, muzica nu este redata bine):

Poze:









Video:

https://www.youtube.com/watch?v=Y4rqXtTIABA&feature=youtu.be

Concluzii

As fi vrut sa am mai mult cable management, sa imi fi luat o placuta de test mai mare si sa fi lipit conectorii la placuta mai bine. Nu faceti ca mine, o sa regretati!

Download

popa_cristian_331cb_proiect.zip

Bibliografie/Resurse

Listă cu documente, datasheet-uri, resurse Internet folosite, eventual grupate pe **Resurse Software** şi **Resurse Hardware**.

- Documentația în format PDF