Olli-Pekka Niinimäki

Itse virittävän kitaravirittimen prototyyppi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkötekniikan koulutusohjelma

Insinöörityö

29.9.2014

Metropolia

Tekijä Otsikko Sivumäärä Aika	Olli-Pekka Niinimäki Itse virittävän kitaravirittimen prototyyppi 32 sivua + 3 liitettä 8 Lokakuu 2014
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Sähkötekniikan koulutusohjelma
Suuntautumisvaihtoehto	Elektroniikka ja terveydenhuollontekniikka
Ohjaaja	Lehtori Kai Lindgren

Insinöörityön aiheena oli suunnitella ja toteuttaa itse virittävän kitaravirittimen prototyyppi. Laitteen tarkoituksena on havaita sisään tuleva signaali ja virittää servomoottorin avulla kitaran kieli haluttuun vireeseen.

Työ toteutettiin käyttäen Arduino UNO -alustaa, jolla sisään tuleva signaali tunnistetaan. Sen avulla ohjataan moottori pyörimään oikeaan suuntaan.

Kytkennän suunnittelun jälkeen syövytettiin piirilevy, joka sijoitettiin Arduino UNO -alustan päälle. Tämän jälkeen Arduino-ohjelmoitiin.

Lopputuloksena saatiin laite, joka täyttää asetetut tavoitteet. Laite tunnistaa sisään tulevan signaalin ja laskee siitä taajuuden. Tämän jälkeen se vertailee sisään tulevaa taajuutta haluttuun taajuuteen ja ohjaa moottoria pyörimään tämän mukaan.

Avainsanat	Arduino, kitaraviritin, ohjelmointi

Author Title Number of Pages Date	Olli-Pekka Niinimäki Self-Tuning Guitar Tuner Prototype 32 pages + 3 appendices 8 October 2014
Degree	Bachelor of Engineering
Degree Programme	Electrical Engineering
Specialisation option	Medical Engineering
Instructor	Kai Lindgren, Senior Lecturer

The goal of this thesis was to design and produce a self-tuning guitar tuner prototype. The purpose of the device was to detect the incoming signal and tune the string to the desired tune using a servo motor.

The work was conducted using Arduino Uno – programming platform to recognize the incoming signal and to control the motor to rotate to the right direction. After designing of the wiring, a circuit board was etched and placed over the Arduino Uno programming platform. This was followed by programming the Arduino.

The outcome is a product that fulfills the set objectives. The product detects the incoming signal and calculates its frequency. After this it compares it to the desired frequency and guides the motor to rotate to the needed direction.

Keywords	Arduino, guita tuner, programming

Sisällys

Lyhenteet

PWM	Pulse-Width Modulation. Pulssileveysmodulaatio, jossa jännitettä säädetään pulssisuhteen avulla.
USB	Universal Serial Bus. Sarjaväyläarkkitehtuuri, jonka kautta laitteet voivat kommunikoida keskenään.
EEPROM	Electronically Erasable Programmable Read-Only Memory. Uudelleen kirjoitettava muisti, jossa data säilyy myös ilman virtaa. .
SPRAM	Static Random Acces Memory. Muisti, jota on virkistettävä säännöllisesti, jotta muistiin kirjoitetut tiedot säilyisivät.

1 Johdanto

Tässä insinöörityössä tavoitteena on suunnitella, ohjelmoida ja toteuttaa prototyyppilaite, joka virittää kitaran yksi kieli kerrallaan. Tämän saavuttamiseksi pitää havaita kitaran taajuudet ja sen mukaan osoittaa virityssuunta led-valoilla. Tämän jälkeen aloittaa viritin korjauksen tekemisen servomoottorin avulla. Laitteen tarkoitus on helpottaa ja nopeuttaa kitaran virittämistä. Erillisellä laitteella kitaran omaa virityskoneistoa ei tarvitse muuttaa.

Viritysmittarit havaitsevat taajuudet yleensä äänisignaalin kautta. Tämä välitetään mittariin joko suoraan piuhalla tai sitten mikrofonin avulla. Jotkut mittarit myös havaitsevat soitetun kielen aiheuttaman värähtelyn. Näin ne osoittavat, mihin suuntaan kitaran kielen kireyttä pitää korjata.

Kitaran voi virittää korvakuulolta tai sitten erilaisilla viritysmittareilla. Viritysmittarit vaativat pääsääntöisesti ihmisen tekemään itse virityksen. Mittari vain osoittaa, mihin suuntaan korjaus on tehtävä. Kaupallisia itse virittäviä mittareita on vain pari mallia. Nämä kyseiset mittarit tulevat kitaraan virityskoneiston tilalle ja toimivat itse virityskoneistona.

2 Laitteen rakentamisessa käytetyt osat ja komponentit

2.1 Arduino

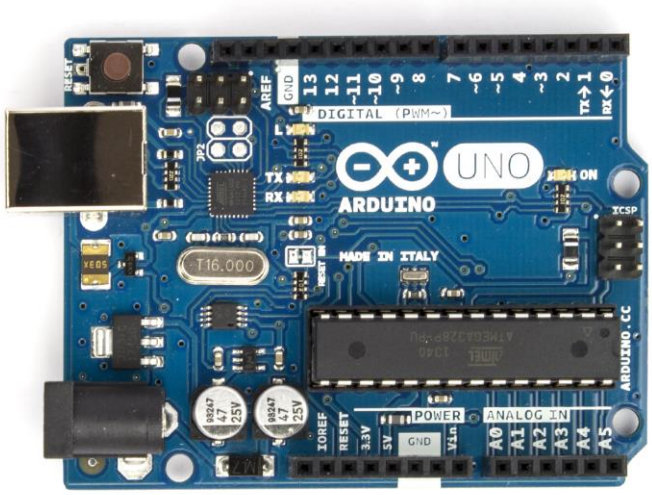
2.1.1 Yleistä Arduino-ohjelmointialustoista

Arduinot ovat ohjelmointialustoja, joiden avulla voidaan ohjata erilaista elektroniikkaa. Laitteen ohjelmointikieli perustuu C++ -ohjelmointikieleen, sen ohjelmointiin käytetään Arduinon omaa IDE-ohjelmaa. Ohjelma perustuu avoimeen lähdekoodiin. Arduinoon voidaan liittää lukuisia erilaisia kytkimiä, moottoreita ja muita fyysisiä laitteita, joiden avulla voidaan luoda erilaisia interaktiivisia laitteita. [1.]

Arduino on kehitetty Italiassa vuonna 2005. Kehitystiimiin kuuluivat Massimo Banzi, Hernando Barragan ja David Cuartiellies. Heidän tarkoituksenaan oli kehittää yksinkertainen ja edullinen ohjelmointialusta opiskelijoiden ja harrastelijoiden käyttöön. [2.]

2.1.2 Arduino UNO R3

Arduino Uno R3 (kuva1) perustuu ATmega328-mikrokontrolleriin ja on kolmas kehitysversio Arduino Uno -piirilevystä.



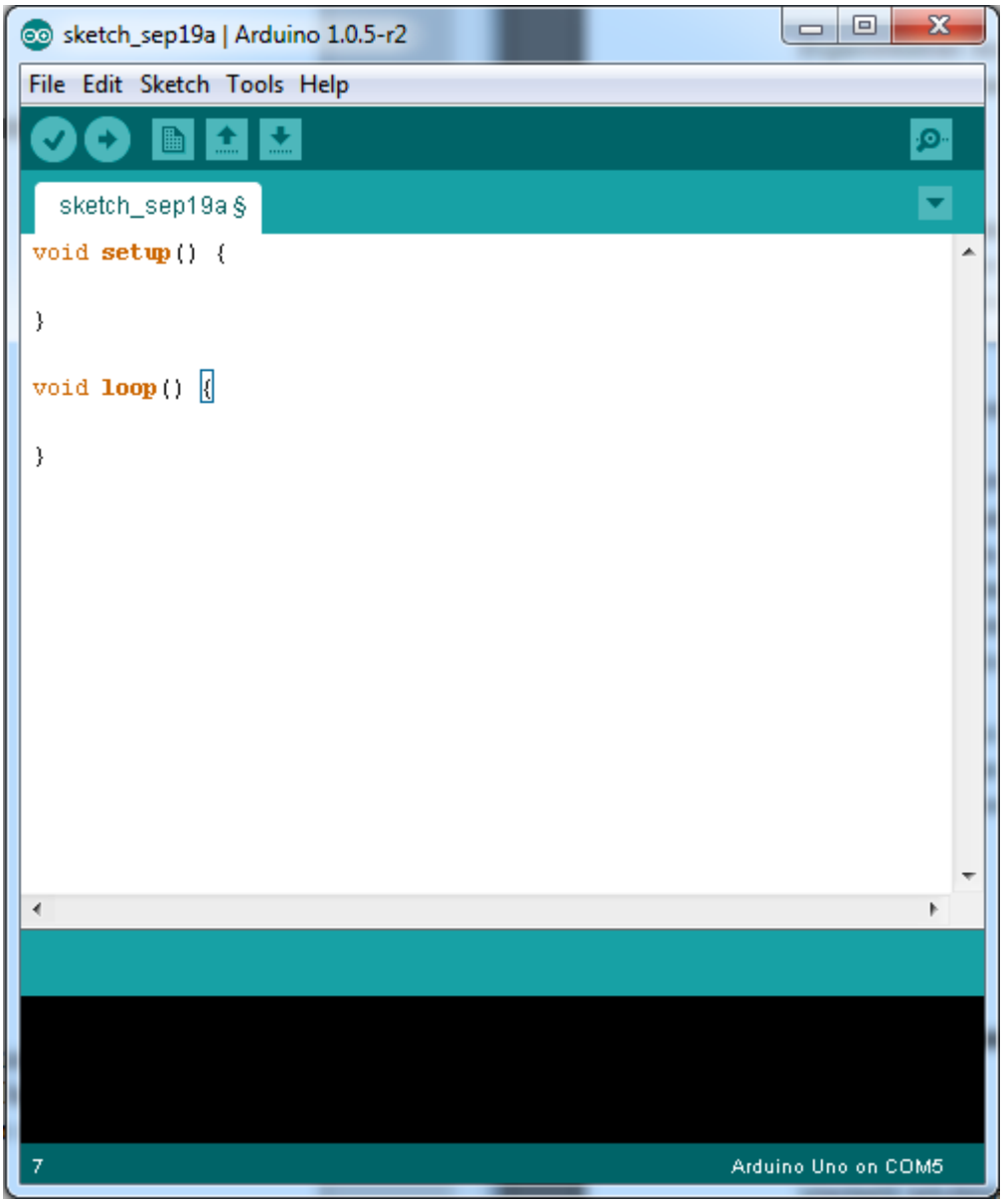
Kuva 1. Arduino Uno R3 edestä kuvattuna [3]

Arduino Uno R3 toimii 16 Mhz:n kellotaajuudella. Se sisältää 14 Digitaalista I/0-pinniä, joita voidaan käyttää sisääntuloina ja lähtöinä. 6 näistä nastoista toimii 8-bittisinä PWM-lähtöinä. Uno R3 sisältää myös 6 analogista sisääntuloa ja 5 V:n ja 3,3 V:n nastat. Se toimii joko USB-liittimen kautta tulevalla 5 V:n jännitteellä tai sitten DC-virtaliittimeen tulevalla jännitteellä, jolloin suositusrajat ovat 7-12 V. Alustassa on 34 kB Flash-muistia, 2 kB SPRAM-muistia ja 1 kB EEPROM-muistia. [3.]

Kaikki Arduino Uno R3-alustan nastat operoivat oletuksena 0-5 V jännitteellä. 5 V on samalla pinnien jännitteen yläraja, jota ei saa ylittää. Nastat pystyvät syöttämään ja kestämään noin 40 mA:n tasavirtaa. Analogiset nastat tarjoavat 10-bittisen resoluution eli palauttavat arvoja 0:n ja 1024:n väliltä. Uno R3 sisältää myös RESET-nastan, joka nimensä mukaisesti asettaa levyn aloitustilaan Tämä nasta mahdollistaa levyn alustamisen ilman että RESET-kytkintä painetaan. [3.]

2.1.3 Ohjelmointiympäristö

Arduinon omana ohjelmointiympäristönä toimii ARDUNO IDE (kuva 2). IDE.n tarkoituksena on yksinkertaistaa koodin kirjoittamista ja sen lataamista Arduino-laitteisiin. Se perustuu avoimen lähdekoodin ohjelmiin. Sitä voi myös laajentaa erilaisilla C++ -kirjastoilla. Koska Arduino-laitteet perustuvat Atmelin ATmega-mikrokontrollereihin, niin siitä syystä ohjelmointikielenä voi käyttää myös AVR C -kieltä. Ohjelmakoodia kutsutaan nimellä Sketch. [3,4.]



Kuva 2. IDE-ohjelmointiympäristö

Käytin Arduino Uno ohjelmointialustaa prototyypin suunnittelussa, koska se on edullinen ja sisältää tarvittavat ominaisuudet. Laitteen jatkokehittäminen suoraan käyttämällä Arduinon ATmega328 mikrokontrolleria onnistuu helposti.

2.2 Operaatiovahvistin

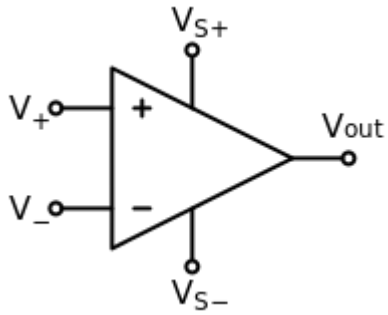
Operaatiovahvistin on analogiaelektroniikan peruskomponentti ja sitä kutsutaankin yleensä opariksi tai opampiksi. Operaatiovahvistin itsessään sisältää useita transistoreita ja fettejä sekä muita komponentteja, ja sitä voidaan silti käsitellä yhtenä elektronii-

kan komponenttina. Operaatiovahvistimen tarkoituksena on vahvistaa kahden sisäänTulon välistä jännitettä tietyllä vahvistinkertoimella A. Opari sisältää tavallisesti kaksi sisäänmenoa ja yhden ulostulon. Operaatiovahvistimessa käytetään tavallisesti kaksipuoleista jännitesyöttöä, jolloin myös operaatiovahvistimen ulostulojännite on sisään tulevasta signaalista riippuen joko positiivinen tai negatiivinen (kuva 3). [5.]

Operaatiovahvistin vertailee tuloliittimien välistä jännite-eroa. Kytkentätyyppeinä käytetään joko suljettua silmukkaa tai avointa silmukkaa. Avoin silmukka tarkoittaa, että kytkentä ei sisällä paluujohdinta ulostulosta. Avoimen signaalin lähtöjännite voidaan kirjoittaa muodossa. [4.]

$$V_{out} = A(V_+ - V_-) \quad (1)$$

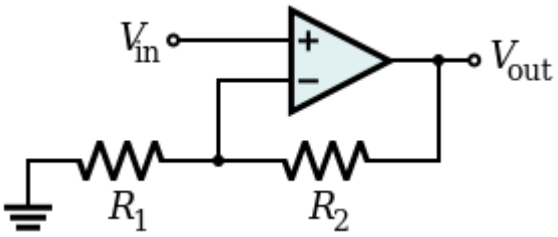
Kaavassa 1 A on avoimen silmukan vahvistus ja $V_+ ja V_-$ ovat tulojännitteet. V_{out} on lähtöjännite.



Kuva 3. Operaatiovahvistimen piirrossymboli [5].

Suljettua silmukkaa käytetään yleensä, jotta saadaan vahvistus halutunlaiseksi. Suljettu silmukka voi olla joko negatiivinen tai positiivinen takaisinkytkentä. Negatiivisen takaisinkytkentä parantaa vahvistimen ominaisuuksia kasvattamalla kaistanleveyttä, pienentämällä säröä ja vähentämällä häiriöitä. Tällä saadaan myös parannettua vahvistimen vakautta. Positiivista takaisinkytkentää käytetään yleensä komparaattoreissa [5].

Työssäni käytän ei-invertoivaa negatiivista takaisinkytkentää (kuva 4), jonka avulla lähtösignaali pysyy samassa vaiheessa tulosignaalin kanssa. Tässä kytkennässä operaatiovahvistimen jännitevahvistusta muutetaan vastussuhteen avulla R2/R1 [6].



Kuva 4. Ei-invertoiva vahvistinkytkentä [4].

Ei-invertoivassa vahvistinkytkennässä on tarkoituksena pitää vahvistus aina positiivisena. Samoin kyseinen kytkentä synnyttää stabiilin vahvistimen [6].

$$A = 1 + \frac{R_2}{R_1} \tag{2}$$

Kaavassa 2 A on vahvistuskerroin, R1 ja R2 ovat valitut vastukset. Tällä voimme laskea halutun jännitevahvistuksen[6].

$$V_{out} = (1 + \frac{R_2}{R_1})V_{in} \tag{3}$$

Kaavassa 3 V_{out} on ulostuleva jännite ja V_{in} on positiiviseen tuloon tuleva jännite, jota halutaan vahvistaa. R1 ja R2 muodostavat vastussuhteen, jolla saadaan määritettyä haluttu vahvistuskerroin [5].

Käytän laitteessani Texas Instrumentsin TL082:n kaksikanavaista operaatiovahvistinta, kotelotyyppinä käytän DIP8. TL082 omaa laajan käyttöjännitealueen, joka on +/- 7V – 36V. Opari on virran kulutukseltaan erittäin pieni, joten se sopii hyvin omalla akulla toimiviin kannettaviin laitteisiin. Valitsin tämän operaatiovahvistimen siksi, että se on itselleni entuudestaan tuttu ja niitä oli itselläni helposti saatavilla.

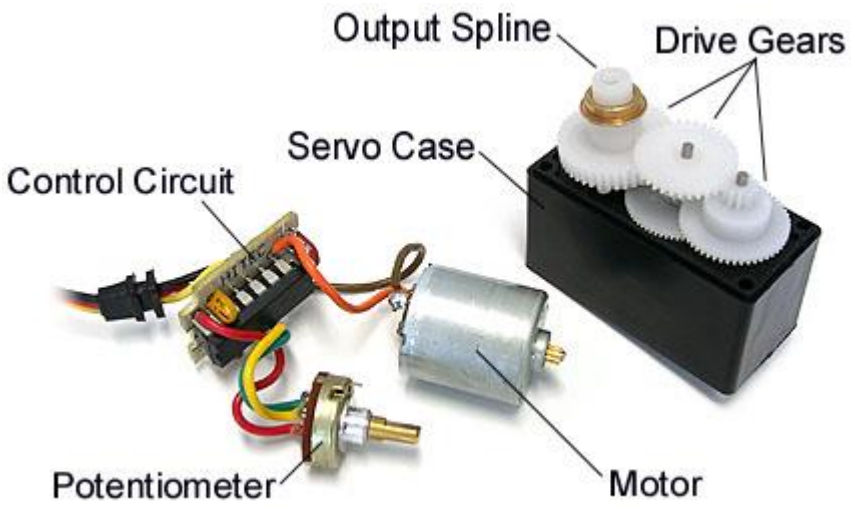
2.3 Servomoottori

Servomoottorilla tarkoitetaan takaisinkytkennällä varustettua moottorin ja moottorinohjauksen yhdistelmää. Moottorinohjauspiiriin on yleensä kytketty potentiometri, josta

saadaan tieto, missä asennossa servomoottori on. Yleisesti servomoottori ei siis käänny kuin tietyn astemäärän molempiin suuntiin. On kuitenkin olemassa myös servoja, joilla onnistuu täysi kääntyminen. Näiden tarkassa paikka ohjauksessa voidaan käyttää pulssilaskentaa.[8, 9.]

Servomoottoriin kuuluvat yleensä seuraavat pääosat (kuva 5):

- moottori
- vaihteisto
- moottorinohjaus
- potentiometri
- suojakuori.



Kuva 5. Servomoottorin sisältämät osat[8].

Servomoottoreita on mahdollista saada erilaisilla moottoreilla, mutta yleisesti on käytössä tavallinen DC-moottori. Moottoria ohjataan H-siltakytkennällä, jonka avulla voidaan moottorin suuntaa ja nopeutta muuttaa pelkästään ohjauspulssia säätämällä.

Tämä säätö tehdään mikrokontrollerin PWM-ohjauksella. Moottorin vääntöä muutetaan vaihteiston rattaiden avulla. Yleisesti servomoottoreista tehdään hitaita, mutta ne omaavat suuren vääntömomentin. [9.]

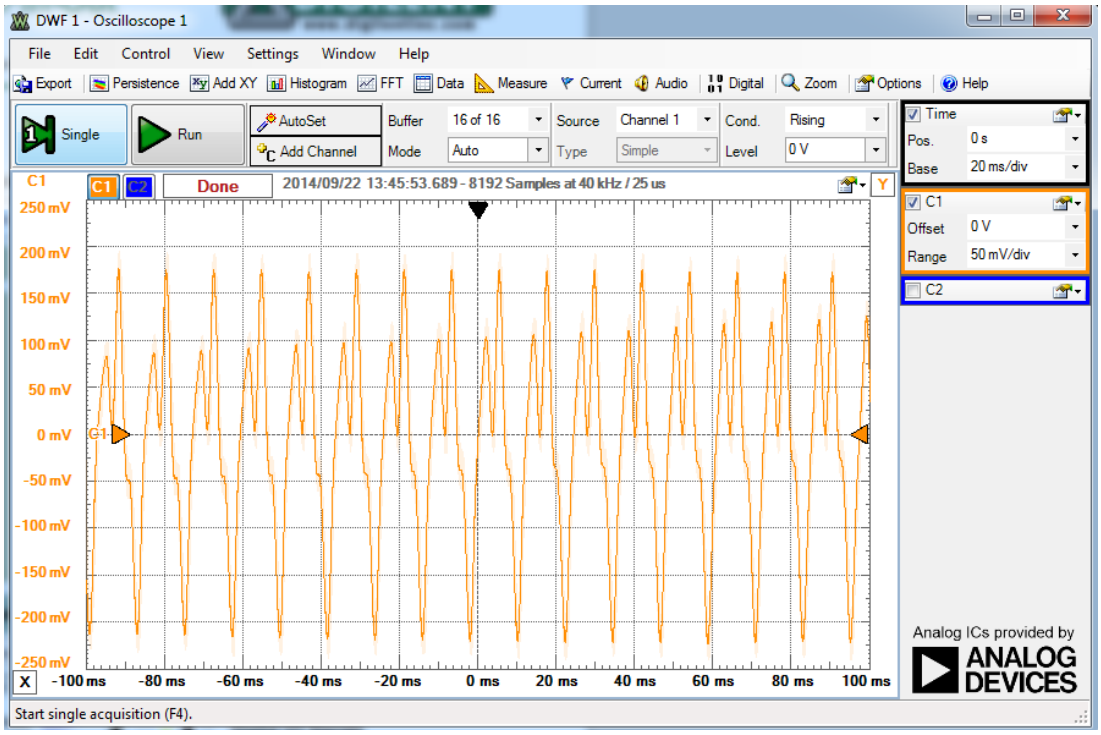
Prototyypissä käytän AAS-750MG-servomoottoria. Kyseinen servo sisältää metalliset vaihteiston rattaat, joilla saadaan vääntö 6 V:n jännitteellä 12,8 kgcm. Metalliset rattaat myös kestävät enemmän rasitusta. Alun perin moottori ei pyörinyt kuin 180°, joten jouduin poistamaan paikkatietoa antavan potentiometrin ja asentamaan tilalle kaksi samanarvoista vastusta, jotta ohjaus silti toimisi. Nyt voidaan asettaa servo menemään haluttuun kulmaan, mutta koska potentiometri ei anna tietoa siitä, milloin kulma on saavutettu, niin servomoottori jatkaa pyörimistään tähän suuntaan.

3 Laitteen rakentaminen

Tässä luvussa kerron laitteen suunnittelusta, rakentamisesta ja avaan käyttämääni koodia.

3.1 Signaalin sisääntulo ja operaatiovahvistinkytkentä

Mittasin ensimmäisenä kitarasta tulevan signaalin vahvuuden Digilentin Analog Discoveryn tietokoneeseen liitettävällä oskilloskoopilla. Ohjelmana toimii Waweform, joka on digilentin oma ohjelma. Mittauksista selvisi, että kitaran voimakkaimman signaalin amplitudin voimakkuus on 200 mV +/- 10 mV, joka saavutettiin E-kielellä (kuva 6). Signaali on siis kohtuullisen heikko, jotta sitä pystyttäisiin helposti käsittelemään.

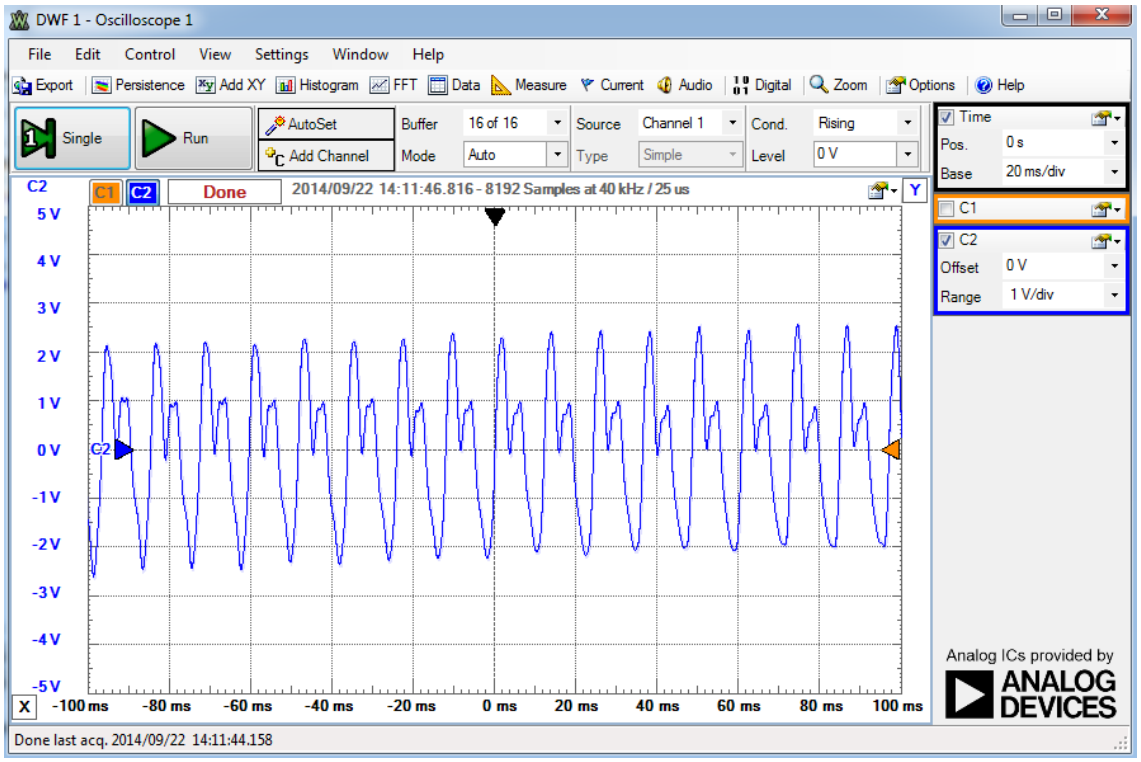


Kuva 6. Kitaran vahvistamaton signaali Waweform-ohjelmassa.

Signaalia vahvistetaan ei-invertoivalla opari-kytkennällä, jonka esittelin edellisessä luvussa. Kaavan 3 avulla lasketaan haluttu vahvistus seuraavasti:

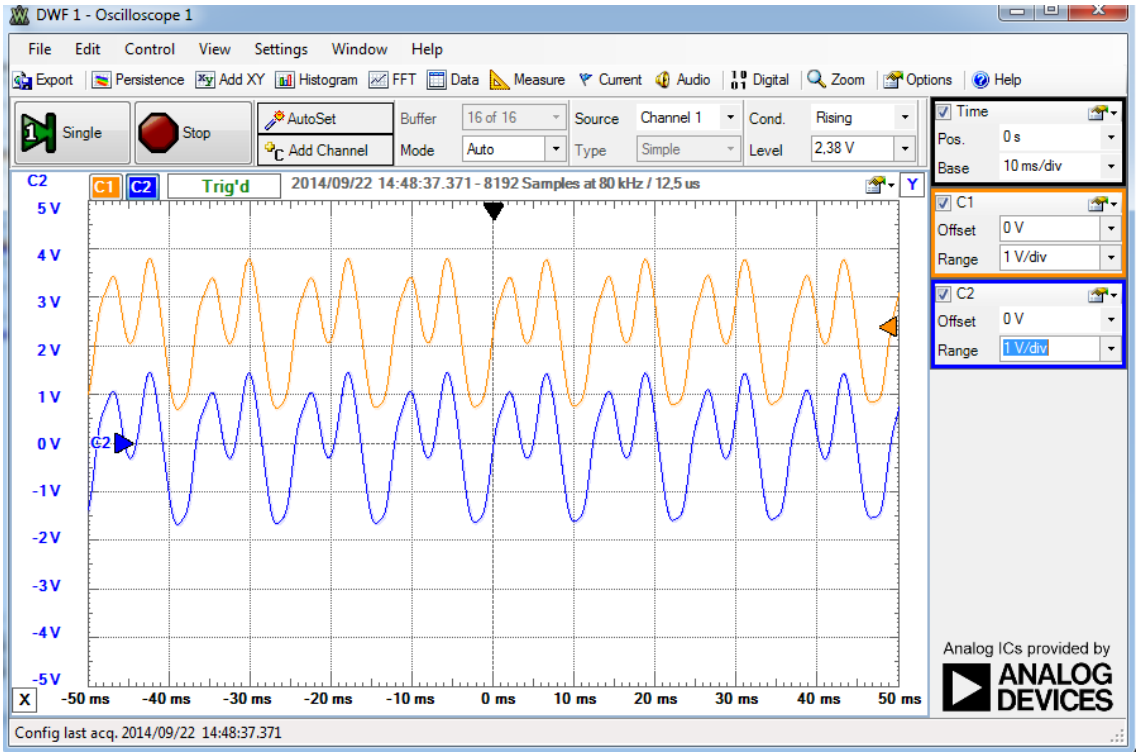
$$2{,}2V = \left(1 + \frac{100\ k\Omega}{10\ k\Omega}\right) 200mV$$

Operaatiovahvistimesta lähtevä korkein amplitudi on siis noin 2,2 V, koska heikoin sisäänmeno oli korkealla E-kielellä 100 mV +/- 20 mV (kuva 7) ja voimakkain matalalla E-kielellä, jonka amplitudi on noin 220 mV.



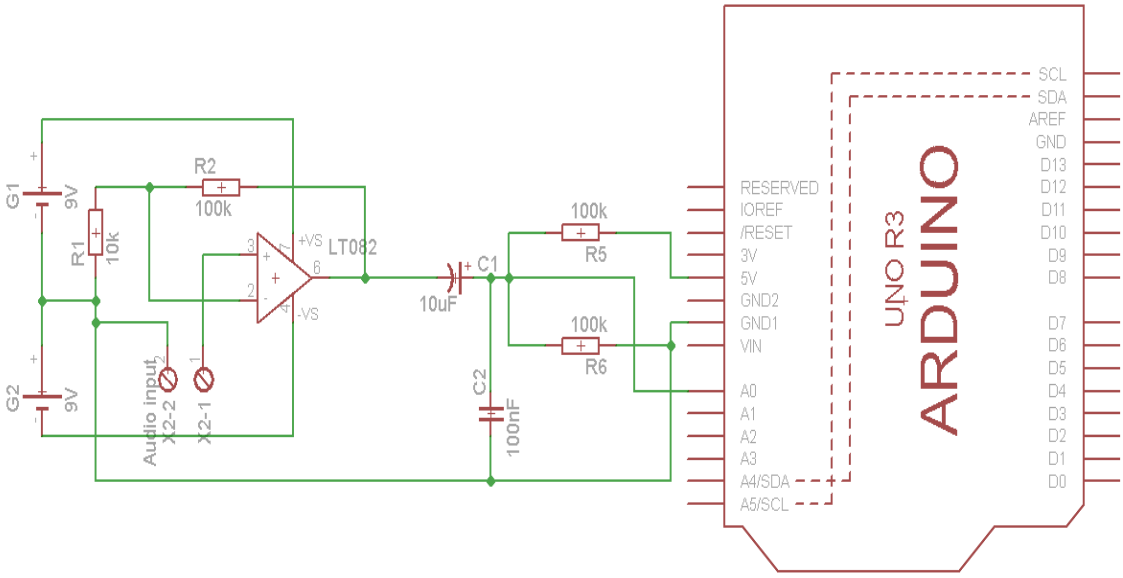
Kuva 7. Kitaran signaali operaatiovahvistimen jälkeen.

Signaalin vahvistamisen jälkeen signaali vaihtelee -2,2 V:n ja +2,2 V:n välillä, joten Offset-taso täytyy nostaa 2,5 V, jotta vaihtelu olisi noin 0-5 V:n välillä. Tämä siitä syystä, että Arduinon sisäänmenot tunnistavat vain 0-5 V:n välillä olevaa signaalia. Näiden rajojen ulkopuolella olevat signaalit leikkaantuvat pois. Offsetin nosto tapahtuu jännitteen jaolla. Tämä suoritetaan kytkemällä 100 kohm vastus Arduinon +5 V nastaan ja 100 kohm vastus GND nastaan. Näiden vastusten väliin syntyy 2,5 V jännite. Signaali tuodaan näiden vastusten väliin 10 uF kondensaattorin kautta. Tämä kytkentä tasaa tulevan jännitteen. Vastusten välistä lähtee maahan myös 100 nF suodatuskondensaattori. Tarkoituksena on vähentää häiriöitä signaalissa. Tämän jälkeen jännite vaihtelee 0-5 V:n välillä. Kuvassa 8 sininen signaali on suoraan operaatiovahvistimelta tuleva, ja oranssi on Offset-noston kautta tuleva signaali.



Kuva 8. Signaali operaatiovahvistimelta ja Offset-noston jälkeen.

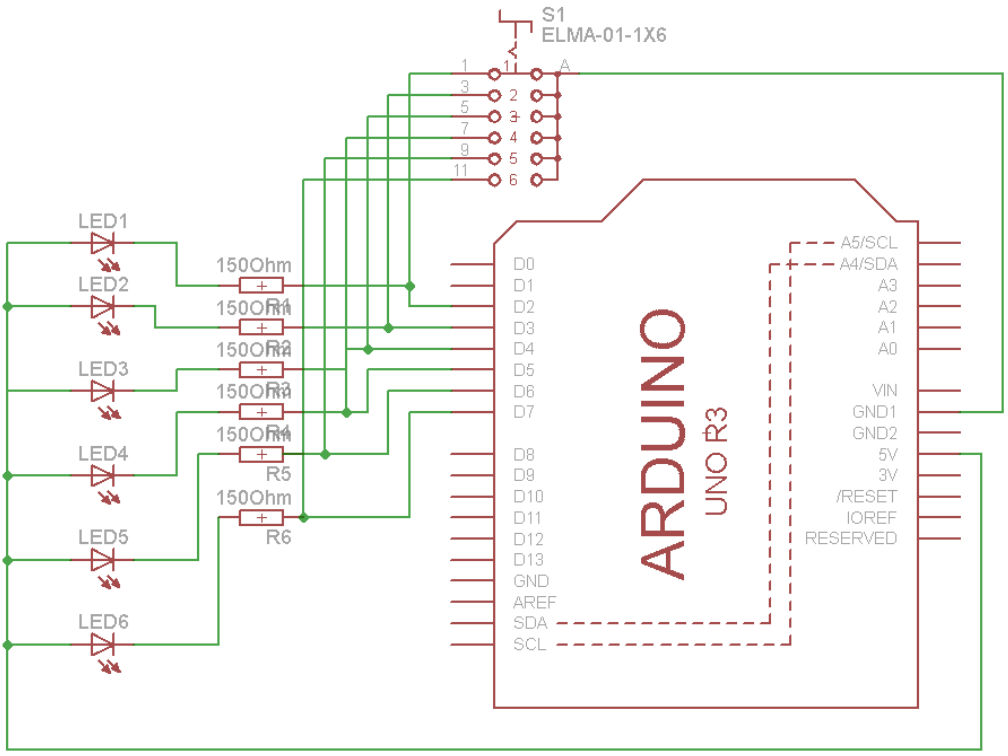
Operaatiovahvistin toimii +/- 7 – 36 V:n jännitealueella. Jännitelähteenä toimii kaksi 9 V:n paristoa, joista toinen antaa -9 V jännitteen operaatiovahvistimelle. Kuvassa 9 on-kytkentäkaavio sisäänmenosta Arduinolle asti. Kytkentäkaavio on piirretty Cad Soft Eagle PCB -ohjelmistolla. Ohjelmistolla on myös suunniteltu prototyyppi piirilevystä.



Kuva 9. Operaatiovahvistin ja Offset-kytkentä Arduinolle.

3.2 Kytkimen ja kytkimeen liitettyjen ledien kytkentä Arduinoon

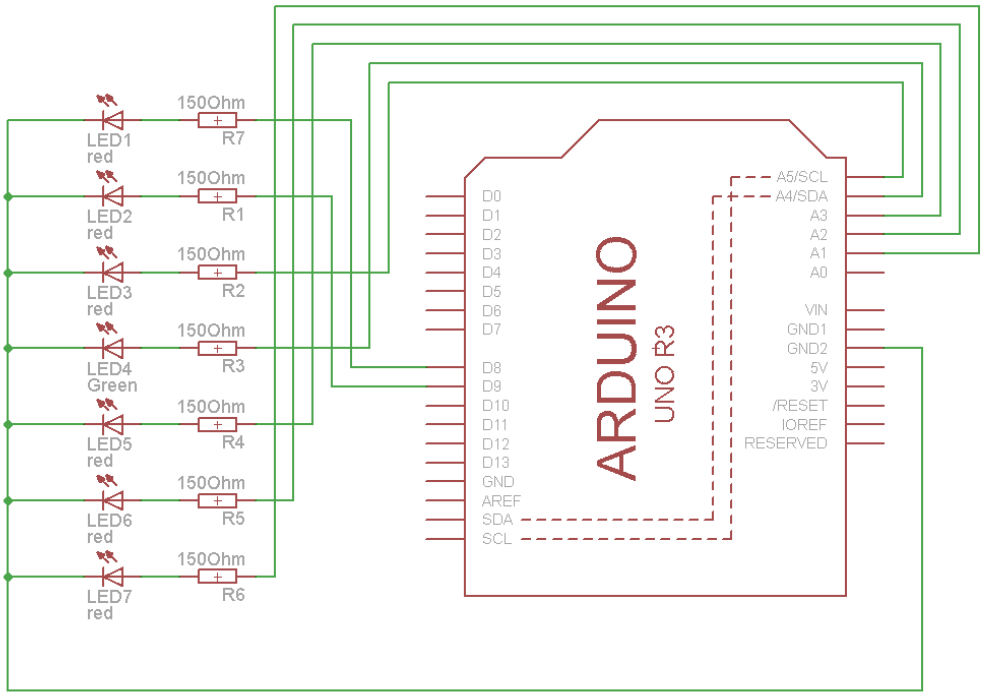
Kytkimenä toimii 6-asentoinen kiertokytkin. Tällä valitaan haluttu kieli viritettäväksi. Ledit syttyvät valitun kielen merkiksi. Kiertokytkimen keskinasta kytketään maahan ja valintanastat kytketään Arduinon digitaalisiin nastoihin numerot 2-7. Nastoihin tuodaan 5 V jännite, jotta Arduino lukee signaalin ylös. Ohjelmassa Arduinon nastojen numeroiden 2-7 jännite nostetaan sisäisesti arvoon 1, jotta kun kytkin valitsee nastan, se saa arvon 0. Ledit kytkeytyvät kytkimen kautta maahan, ja jokaiseen lediin on laitettu 150 Ω etuvastus (kuva 10).



Kuva 10. Kiertokytkimen ja ledien kytkentä Arduinoon.

3.3 Virityksen osoittavien ledien kytkentä

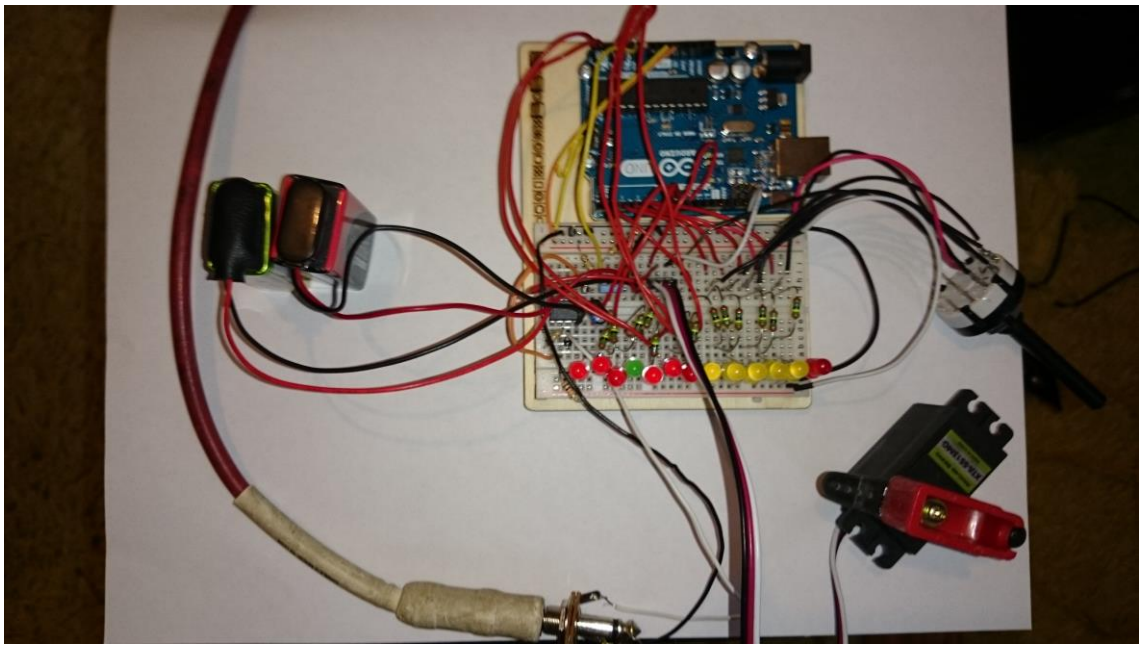
Virityksen osoittavia ledejä on 7 kappaletta. Yksi näistä on vihreä, ja tämä osoittaa, että kitara on vireessä. Molemmin puolin on kolme punaista lediä, jotka osoittavat, mihin suuntaan kitara on epävireessä ja kuinka paljon. Jokaiselle kitarankielelle on koodissa määrätty oikea taajuus. Sisään tulevaa taajuutta verrataan oikeaan taajuuteen. Ledit kytketään Arduinon kahteen digitaaliseen nastaan numerot 8 ja 9, sekä neljään analogiseen nastaan numerot A1-A5. Ledeille on asetettu 150 Ω:n etuvastukset (kuva 11).



Kuva 11. Säätöledien kytkentä Arduinoon.

3.4 Kytkentä kokonaisuudessaan

Kytkentä toteutettiin testauksen takia ensin koekytkentälevylle (kuva 12). Servomoottorin ohjaus kytkettiin Arduinon digitaaliseen nastaan 10, jossa on PWM-ohjaus. Jännitteen servomoottori saa suoraan 9 V:n paristosta. Kytkentä todettiin toimivaksi. Tämän jälkeen suunniteltiin piirilevy laitteelle. Paristot, servomoottori, Arduino ja kääntökytkin liitetään piirilevyyn piikkirimaliittimien kautta.

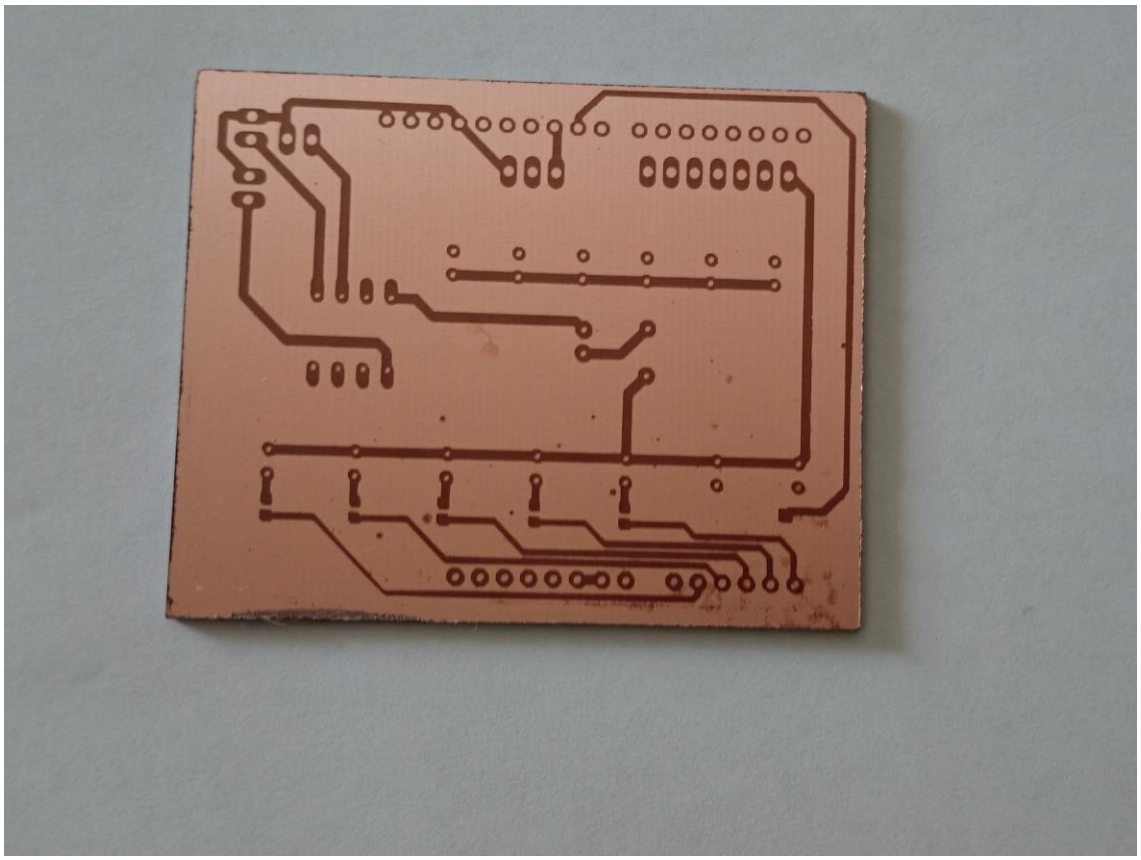


Kuva 12. Kitaraviritin koekytkentälevyllä.

3.5 Piirilevyn valmistaminen

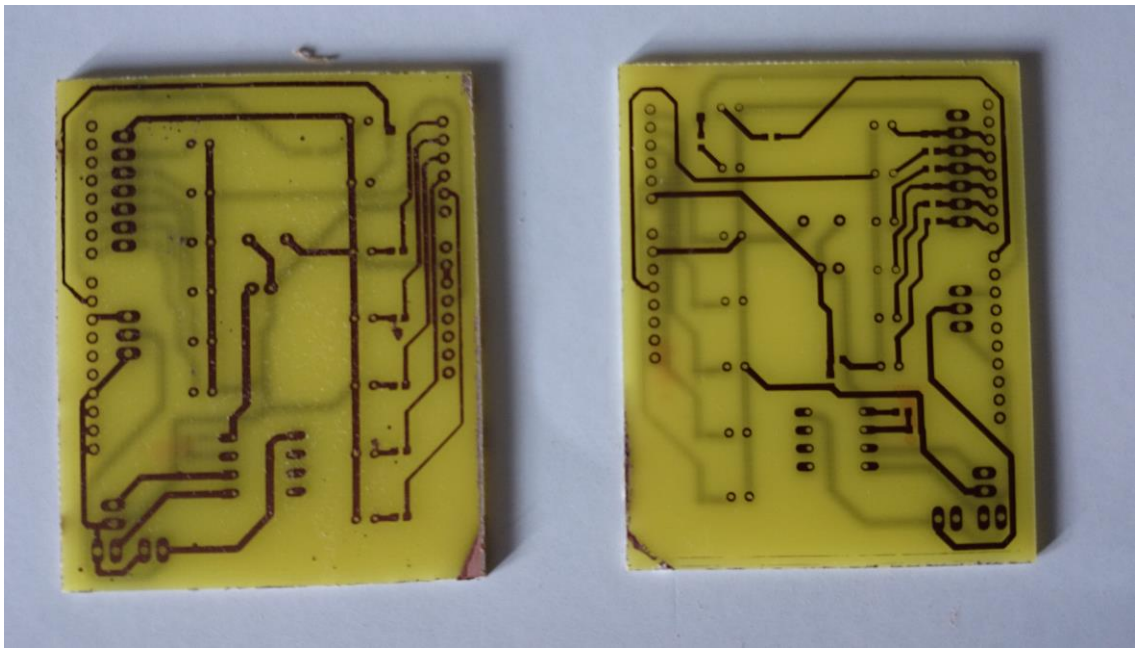
Prototyypin piirilevy valmistettiin syövyttämismenetelmällä. Aluksi levy suunniteltiin CadSoft Eagle PCB -ohjelmistolla (litteet 1 ja 2). Ensin tulostettiin piirilevyn Top- ja Bottom-kuvat piirtoheitinkalvolle, joka toimii valotusmaskina. Kalvo asetettiin valoherkän piirilevyn päälle ja tämän jälkeen altistettiin UV-valolle 5 minuutin ajan. Valotuksessa levyn pinnalla oleva lakka aktivoituu, ja ainoastaan ei-valottuneet osat tulevat kehityksessä esiin.

Valotuksen jälkeen piirilevy upotetaan kehitysliuokseen. Liuoksena toimii noin 1 %:n vahvuinen natriumhydroksidiliuos. Kun levyn pintaan on muodostunut näkyvä piirikuvio, huuhdellaan levy juoksevan veden alla (kuva 13).



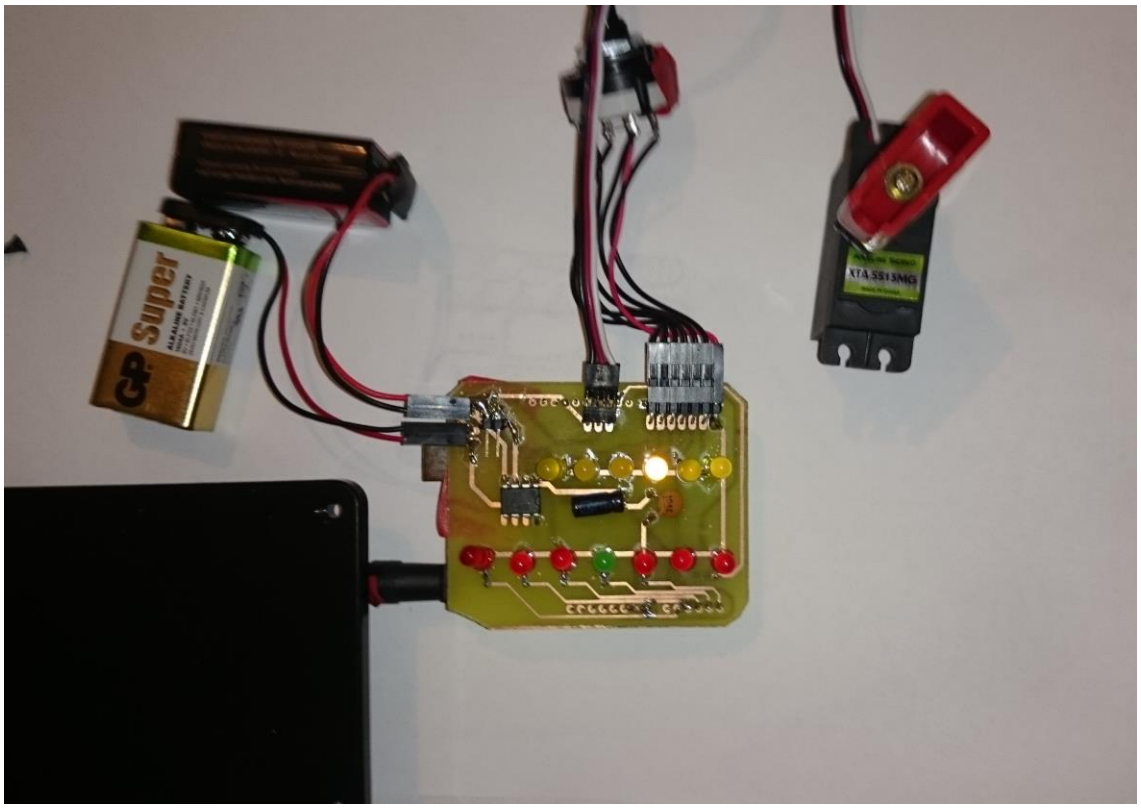
Kuva 13. Valotettu ja kehitetty piirilevy.

Kehityksen jälkeen aloitetaan syövyttäminen. Syövyttämisessä poistetaan kaikki ylimääräinen kupari levyltä. Vedot ja pädit jäävät syövytyksen jälkeen näkyviin (kuva 14).



Kuva 14. Piirilevyt syövytyksen jälkeen.

Syövyttämisen jälkeen pädeihin porataan reiät, jotta läpiladottavat komponentit saadaan paikalleen. Levy on valmis juotettavaksi.



Kuva 15. Valmis piirilevy.

Valmis piirilevy sijoitettiin Arduinon päälle, ja se kytkeytyy Arduinon nastoihin piikkirima-liitimillä (kuva 15). Levyyn saatiin kytkettyä kaikki tarvittava piikkirima- ja harwin-liittimillä.

4 Laitteen ohjelmointi

Laitteen ohjelmoinnissa käytetään Arduinon IDE-ohjelmointiympäristöä. Koodin kirjoittamisessa on käytetty apuna netistä löytyviä esimerkkikoodeja. Käytän koodissa myös netistä löytämääni frequency detection–koodia. [10.]

Koodissa otetaan aluksi käyttöön halutut kirjastot, tässä tapauksessa servomoottorin ohjauskirjasto.

```
#include <Servo.h> // lisää servo moottorin ohjaukses-
sa käytettävän kirjaston
```

Tämän jälkeen koodissa määritellään muuttujat:

```
//Datan tallennuksen muuttujat

byte newData = 0;

byte prevData = 0;

unsigned int time = 0;//keeps time and sends vales to
store in timer[] occasionally

int timer[10];//storage for timing of events

int slope[10];//storage for slope of events

unsigned int totalTimer;//used to calculate period

unsigned int period;//storage for period of wave

byte index = 0;//current storage index

float frequency;//storage for frequency calculations

int maxSlope = 0;//used to calculate max slope as
trigger point

int newSlope;//storage for incoming slope data



//muuttujat vastaavuuksien hakuun

byte noMatch = 0;//counts how many non-matches you've
received to reset variables if it's been too long

byte slopeTol = 3;

int timerTol = 10;



//muuttujat amplitudimuutoksille

unsigned int ampTimer = 0;
```

```
byte maxAmp = 0;

byte checkMaxAmp;

byte ampThreshold = 20;



//muuttuja oikealle taajuudelle

int correctFrequency;//Soitetun kielen haluttu taajuus



// muuttujat kääntälytkimelle

int firstRotaryPin = 2;

int lastRotaryPin = 7;



Servo myservo;// määritellään ohjattavan servomoottorin nimi
```

Koodiesimerkki 1. Muuttujien määrittely.

Tämän jälkeen määritellään, kuinka luetaan A0-nastaan tulevaa taajuutta ja hyväksytään keskeytysten toiminta. Seuraavaksi määritellään myös kytkimen ohjelma, ja kerrotaan Arduinolle nastojen käyttö, eli ovatko lukevia vai kirjoittavia.

```
void setup(){

for( int i=firstRotaryPin; i<= lastRotaryPin; i++) {

 pinMode( i, INPUT);

 digitalWrite( i, HIGH);

}

pinMode(A3,OUTPUT);
```

```
pinMode(A4,OUTPUT);

pinMode(A5,OUTPUT);

pinMode(A1,OUTPUT);

pinMode(A2,OUTPUT);

pinMode(8,OUTPUT);

pinMode(9,OUTPUT);
```

Koodiesimerkki 2. Nastojen käyttö.

Määritellään kytkimen nastat lukeviksi ja nostetaan ne suoraan arvoon 1. Ledien nastat määritellään ulostuloiksi.

```
 ADCSRA = 0;

 ADCSRB = 0;

 ADMUX |= (1 << REFS0);

 ADMUX |= (1 << ADLAR);

ADCSRA |= (1 << ADPS2) | (1 << ADPS0); //set ADC clock
with 32 prescaler- 16mHz/32=500kHz

ADCSRA |= (1 << ADATE);

ADCSRA |= (1 << ADIE);

ADCSRA |= (1 << ADEN);

ADCSRA |= (1 << ADSC);



sei();
```

Koodiesimerkki 3. A/D-muuntimen koodi.

Arduino lukee sisään tulevan signaalin jännitteen A0-nastan kautta. Jännitteen luku tapahtuu 0 V ja 5 V välillä. Arduino kuitenkin lukee nämä lukuarvoina 0:n ja 255:n välillä. Koodissa skaalataan sisäisen AD-muuntimen nopeudeksi 500 kHz. Koska A/D-muunnin saa uuden arvon 13 syklin välein, saadaan näytteistys taajuus n. 38,5 kHz. Arduino siis saa uuden arvon 26 mikrosekuntin välein.

```
ISR(ADC_vect)

  prevData = newData;//store previous value

  newData = ADCH;//get value from A0

  if (prevData < 127 && newData >=127){//if increasing and
crossing midpoint

    newSlope = newData - prevData;

    if (abs(newSlope-maxSlope)<slopeTol){//if slopes are ==

      //record new data and reset time

      slope[index] = newSlope;

      timer[index] = time;

      time = 0;

      if (index == 0){//new max slope just reset

         noMatch = 0;

        index++;//increment index

      }

           else if (abs(timer[0]-timer[index])<timerTol &&
          abs(slope[0]-newSlope)<slopeTol){

        totalTimer = 0;
```

```
      for (byte i=0;i<index;i++){
        totalTimer+=timer[i];
      }
      period = totalTimer;
      //reset new zero index values to compare with
      timer[0] = timer[index];
      slope[0] = slope[index];
      index = 1;//set index to 1
      noMatch = 0;
    }
    else{
      index++;//increment index
      if (index > 9){
        reset();
      }
    }
  }
  else if (newSlope>maxSlope){
    maxSlope = newSlope;
    time = 0;//reset clock
    noMatch = 0;
    index = 0;//reset index
```

```
    }
    else{
      noMatch++;//increment no match counter
      if (noMatch>9){
        reset();
      }
    }
  }
  time++;//increment timer at rate of 38.5kHz

  ampTimer++;//increment amplitude timer
  if (abs(127-ADCH)>maxAmp){
    maxAmp = abs(127-ADCH);
  }
  if (ampTimer==1000){
    ampTimer = 0;
    checkMaxAmp = maxAmp;
    maxAmp = 0;
  }

}
```

```
        void reset(){//clean out some variables

        index = 0;//reset index

        noMatch = 0;//reset match counter

        maxSlope = 0;//reset slope

}
```

Koodiesimerkki 4. Keskeytys.

Edellä olevassa koodissa määritellään keskeytykset. Arduino käy normaalisti läpi `setup()`-funktion ja aloittaa loop() -funktion suorittamisen, joka on ikuinen silmukka. Mutta joka 26 mikrosekunti, kun uusi arvo on valmiina, Arduino lopettaa `loop ()` -funktion suorittamisen ja siirtyy suorittamaan määriteltyä koodia keskeytysosiossa. Tässä kyseisessä koodissa Arduino vertaa signaalin aallon nousuja ja laskuja edelliseen signaaliin. Tämän jälkeen Arduino palaa 26us:ksi takaisin `loop()` -funktioon. Tämä kierto pysyy niin kauan, kun laite on päällä.[10]

Määritellään seuraavaksi kiertokytkimen asennoille kitarankielten oikeat taajuudet. Taajuudet menevät seuraavasti:

- E – 82.4Hz
- A – 110Hz
- D – 146.8Hz
- G – 196Hz
- B – 246.9Hz
- E – 329.6Hz.

Kun kiertokytkimestä on valittu oikea taajuus, niin tämän jälkeen valittua taajuutta verrataan Arduinolle sisään tulevaan taajuuteen. Ledit on ohjelmoitu tunnistamaan taajuutta ja osoittamaan, mihin suuntaan kitaraa pitää virittää. Koodiin on myös määrätty, että

kun taajuus on pienempää tai yhtä suurta kuin 0Hz, niin kaikki ledit ovat sammuksissa. Kun oikea taajuus saavutetaan, syttyy vihreä ledi tämän merkiksi.

```
//määrittää kaantokytkimella oikean kielen mita soite-
taan ja maarittaa oikean taajuuden

void stringCheck(){


int rotaryPos = getRotaryValue();


 if(rotaryPos == 1){

   correctFrequency = 82.4;

 }

  else if( rotaryPos == 2 ){

  correctFrequency = 110;

 }

  else if(rotaryPos == 3){

  correctFrequency = 146.8;

  }

  else if( rotaryPos == 4){

    correctFrequency = 196;

  }

  else if( rotaryPos ==5 ){

    correctFrequency = 246.9;

  }

  else if(rotaryPos == 6){

    correctFrequency = 329.6;

  }

}
```

```
//vertaa taajuuttaa oikeaan taajuuteen ja sytyttaa
nain halutun ledin

void frequencyCheck(){

  if(frequency>correctFrequency+1){

    analogWrite(A3,255);

  }

  if(frequency>correctFrequency+4){

    analogWrite(A2,255);

  }

  if(frequency>correctFrequency+6){

    analogWrite(A1,255);

  }

  if(frequency<correctFrequency-1){

    analogWrite(A5,255);

  }

  if(frequency<correctFrequency-4){

    digitalWrite(9,1);

  }

  if(frequency<correctFrequency-6){

    digitalWrite(8,1);

  }


  if(frequency>correctFrequency-
1&frequency<correctFrequency+1){

    analogWrite(A4,255);

}

   else if (frequency<=0){

digitalWrite(8, LOW);

  digitalWrite(9, LOW);
```

```
            analogWrite(A1,0);
            analogWrite(A2,0);
            analogWrite(A3,0);
           analogWrite(A4,0);
           analogWrite(A5,0);
           delay(100);
          }

}
```

Koodiesimerkki 5. Kytkin ja ledit.

Koodissa määritellään myös moottorin ohjaus. Aluksi Arduinolle kerrotaan, mihin nastaan moottori on kytketty. Myös moottori määritellään pysähtyneeksi, jos taajuus on 0 Hz tai sen alle. Moottori on ohjelmoitu tekemään korjauksia riippuen siitä, mihin suuntaan kitarankieli on epävireessä. Moottori lähtee kiristämään tai löysentämään kitarankieltä sen mukaan, onko taajuus liian korkea tai matala. Moottori pysähtyy, kun haluttu taajuus saavutetaan.

```
void loop(){
  allLEDsOff();
if (checkMaxAmp>ampThreshold){
frequency = 38462/float(period);//calculate frequency
timer rate/period
}
stringCheck();
  frequencyCheck();
 tuning();
```

```
            delay(100);

    }
```

Koodiesimerkki 6. Loop-funktio.

Seuraavaksi kirjoitettiin Arduinolle ikuinen `loop()` -funktio. Arduino tavallisesti pyörittää `loop()` -funktiota loputtomasti, mutta aikaisemmin koodissa oli toteutettu keskeytys, joka katkaisee tämän pyörittämisen. Kaikki määritellyt ohjelmat kutsutaan loop () -funktiossa. Taajuuden laskukaava asetetaan myös funktion sisälle. Koodi on kokonaisuudessaan liitteenä (liite 3).

5 Loppusanat

5.1 Prototyyppi

Tavoitteena oli suunnitella laite, joka tunnistaa kitarasta tulevan signaalin ja tämän mukaan ohjaa moottoria pyörimään oikeaan suuntaan kitaran virittämiseksi.

Laitteen rakentaminen sujui hyvin helpon kytkennän avulla. Kun laitteen toimivuus oli todettu oikeaksi koekytkentäalustalla, niin sen pohjalta toteutettiin piirilevy. Signaalin vahvistus sujui ongelmitta. Tätä helpottivat TL082-operaativahvistimen hyvä datalehti ja jännitealueen laajuus.

Servomoottori olisi ollut parempi, jos olisi löytynyt suoraan edullinen täyskääntyvä moottori. Kuitenkin laitteeseen muokattu servomoottori toimii hyvin, vaikkakin on havaittavissa pientä nykivää liikettä, vaikka moottorin pitäisi olla pysähtyneenä. Tähän vaikuttavat sisälle vaihdetut vastukset.

Ohjelmointi oli haastavin osuus, koska oma ohjelmoinnin tasoni ei ole kovin hyvä ja käyttämäni Arduinon ohjelmointialusta oli itselleni täysin uusi. Myös käytetty ohjelmointiohjelmisto oli uusi. Työ alkoikin sillä, että Arduinolla harjoiteltiin koodin perusteita, joiden avulla kasvatettiin koodikielen ymmärrystä.

Ensimmäinen ohjelmisto ei toiminutkaan täysin luotettavasti, vaan moottori pyöri jatkuvasti myös silloin, kun sisään ei tullut signaalia. Tämä korjattiin lisäämällä koodiin osuus, joka estää moottorin pyörinnän, kun sisääntuleva taajuus on pienempää tai yhtä suurta kuin nolla. Ohjelmistoon myös lisättiin osuus, joka resetoi ohjelman, kun oikea taajuus saavutetaan ja pidetään yhden sekunnin ajan.

Työn lopputuloksena saatiin toteutettua suunniteltu prototyyppilaite, joka toimii hyvin kitaran virittämisessä perusvireeseen. Työn aikana opittiin paljon myös Arduinon käyttömahdollisuuksista.

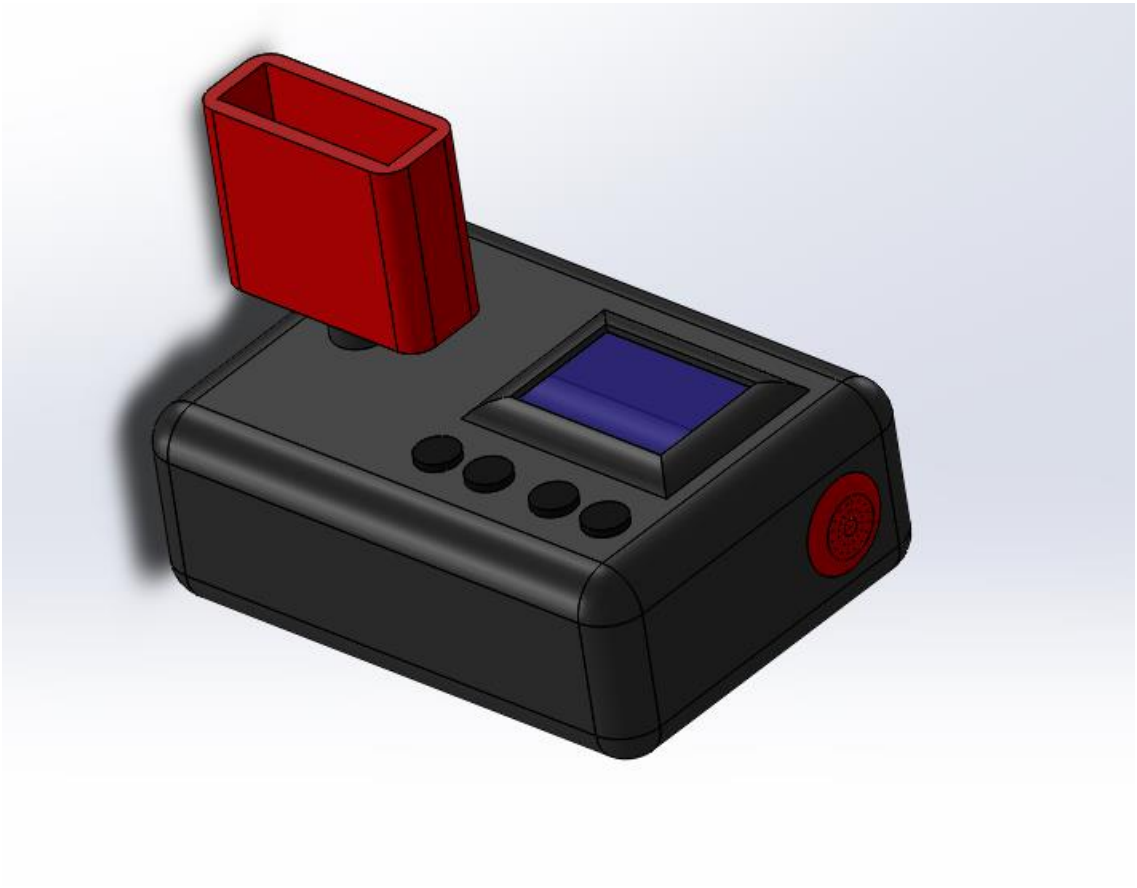
5.2 Jatkokehitys

Laitteessa on paljon potentiaalia jatkokehitystä varten. Ohjelmisto on toimiva ja myös itse piirilevy on toimiva. Piirilevyä voisi kuitenkin pienentää käyttämällä pelkkää ATmega328-mikrokontrolleripiiriä, jolloin Arduinon alusta poistuisi kokonaan. Levyn voisi näin pienentää jopa 3 cm x 4 cm kokoon. Näin pystyttäisiin tekemään helposti mukana kulkeva virityslaite (kuva 16). Kuva ehdotelmasta on suunniteltu Solid Works -mallinnusohjelmalla.

Laitteeseen voitaisiin myös suunnitella erillinen A/D-muunnin, jolloin luettaisiin pelkkää kantti-aaltoa. Tämä suodattaisi signaalista pois kaiken ylimääräisen. Tällöin laitteen tarkkuutta voitaisiin parantaa.

Laitteeseen voitaisiin myös ohjelmoida suoraan halutun kielen tunnistus valintakytkimen sijaan. Signaali voitaisiin myös tuoda laitteeseen mikrofonin kautta, jolloin laitetta ei tarvitse kytkeä kitaraan johdolla.

Laitteeseen voitaisiin myös ledien sijasta sijoittaa Lcd-näyttö, josta nähtäisiin, mitä kieltä ollaan virittämässä ja mihin suuntaan kieltä pitää virittää. Tässäkin moottori virittäisi kielen, mutta antaisi myös mahdollisuuden käsin virittämiseen.



Kuva 16. Ehdotelma miltä laite voisi näyttää.

Lähteet

1 Arduino. FAQ. Verkkodokumentti <http://arduino.cc/en/Main/FAQ>. Luettu 19.9.2014.

2 Wikipedia. Arduino. Verkkodokumentti < http://en.wikipedia.org/wiki/Arduino>. Luettu 19.9.2014.

3 Arduino. Arduino UNO REV 3. Verkkodokumentti < http://arduino.cc/en/Main/arduinoBoardUno>. Luettu 19.9.2014.

4 Koivisto,Jami. 2014. Opinnäytetyö. <https://www.theseus.fi/bitstream/handle/10024/79144/Koivisto_Jami.pdf?sequence=1>. Luettu 18.9.2014.

5 Wikipedia. Operational amplifier. Verkkodokumentti <http://en.wikipedia.org/wiki/Operational_amplifier>. Luettu 19.9.2014.

6 Silvonen, Kimmo, Tiilikainen, Matti & Helenius, Kari. 2002. Analogiaelektroniikka. Helsinki: Edita Prima Oy, s. 188 -195.

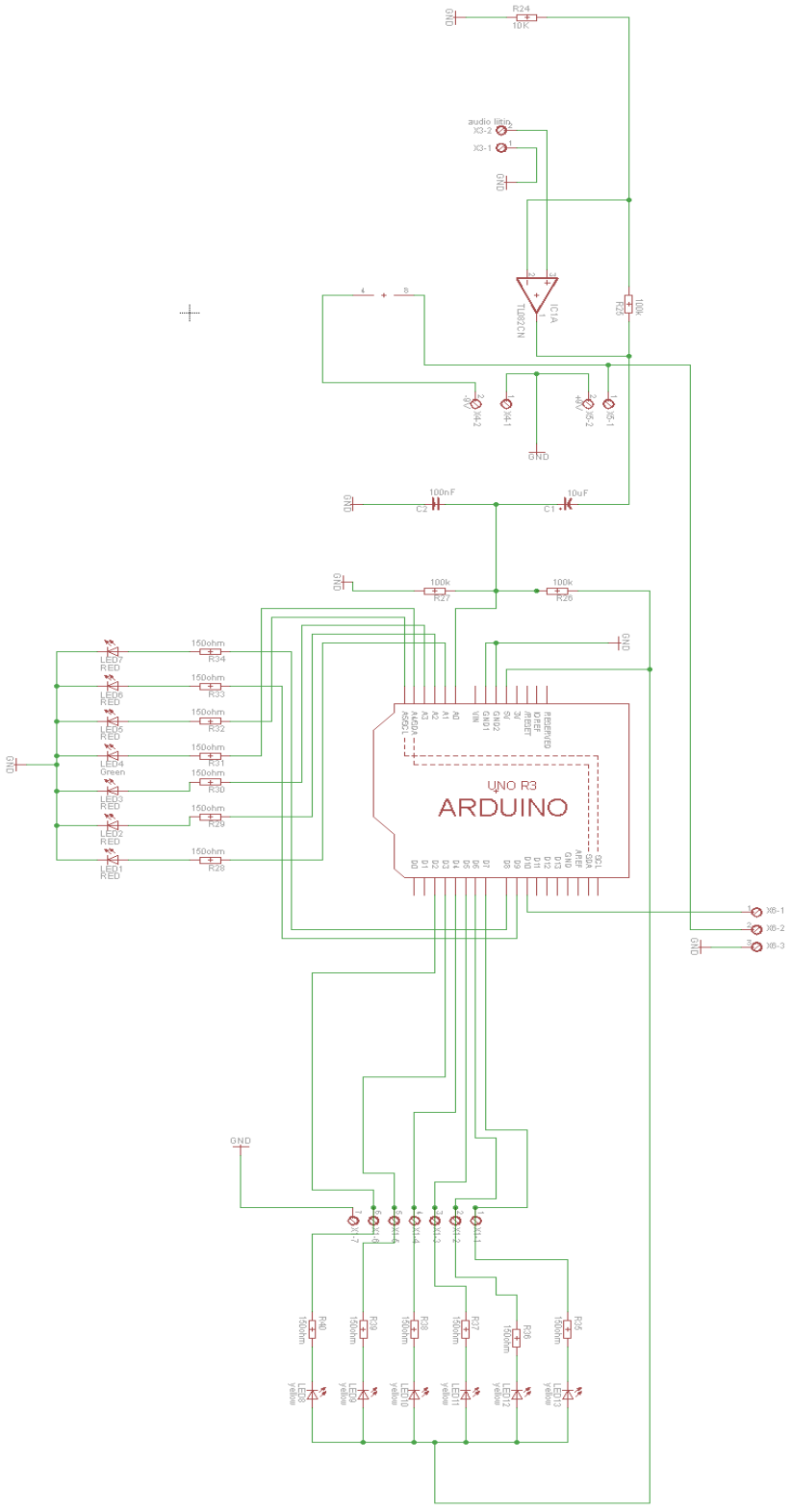
7 TL08xx JFET-Input Operational Amplifiers. 2014. Datalehti. <https://www.theseus.fi/bitstream/handle/10024/61775/Hifi-tas.pdf?sequence=1>. Luettu 22.9.2014.

8 Servocity. 2014. Verkkodoku-mentti.<http://www.servocity.com/html/how_do_servos_work_.html#.VB_zX_l_voc>. Luettu 22.9.2014.

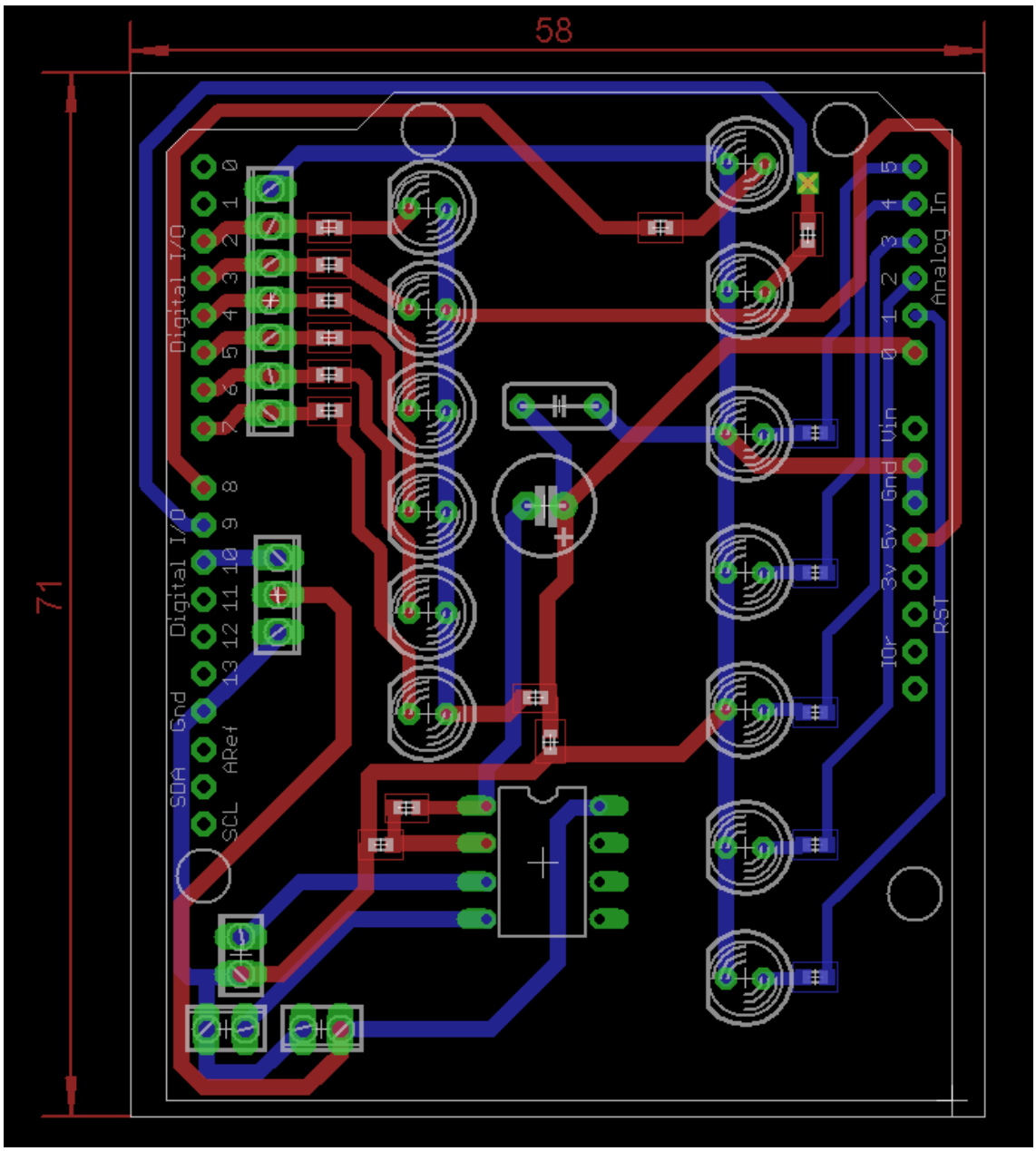
9 PCBheaven. 2014. Verkkodoku-mentti<http://pcbheaven.com/wikipages/How_RC_Servos_Works/>. Luettu 22.9.2014.

10 Amandaghassaei. 2012. Verkkodokumentti. < http://www.instructables.com/id/Arduino-Frequency-Detection/>. Luettu 9.9.2014.

Kitaran virittimen kytkentäkaavio



Kitaranvirittimen piirilevykuva



Arduinon ohjelmakoodi

```
////////////////////////////////////////////////////////////

// Itse virittävä kitaraviritin

// Olli-Pekka Niinimäki

// 2014

////////////////////////////////////////////////////////////

#include <Servo.h>

byte newData = 0;

byte prevData = 0;

unsigned int time = 0;

int timer[10];

int slope[10];

unsigned int totalTimer;

unsigned int period;

byte index = 0;

float frequency;

int maxSlope = 0;

int newSlope;

byte noMatch = 0;

byte slopeTol = 3;
```

```
int timerTol = 10;

unsigned int ampTimer = 0;

byte maxAmp = 0;

byte checkMaxAmp;

byte ampThreshold = 20;

int correctFrequency;

int firstRotaryPin = 2;

int lastRotaryPin = 7;

Servo myservo;

int getRotaryValue() {

for( int i=firstRotaryPin; i<= lastRotaryPin; i++) {

int val = digitalRead( i );

if( val == LOW ) {

return (i - firstRotaryPin + 1);

}

}

}

void setup(){

  for( int i=firstRotaryPin; i<= lastRotaryPin; i++) {

  pinMode( i, INPUT);

  digitalWrite( i, HIGH);}
```

```
  pinMode(A3,OUTPUT);
  pinMode(A4,OUTPUT);
  pinMode(A5,OUTPUT);
  pinMode(A1,OUTPUT);
  pinMode(A2,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  cli();
  ADCSRA = 0;
  ADCSRB = 0;
  ADMUX |= (1 << REFS0);
  ADMUX |= (1 << ADLAR);
  ADCSRA |= (1 << ADPS2) | (1 << ADPS0);
  ADCSRA |= (1 << ADATE);
  ADCSRA |= (1 << ADIE);
  ADCSRA |= (1 << ADEN);
  ADCSRA |= (1 << ADSC);
  sei();
}
ISR(ADC_vect) {
```

```
  PORTB &= B11101111;
  prevData = newData;
  newData = ADCH;
  if (prevData < 127 && newData >=127){
    newSlope = newData - prevData;
    if (abs(newSlope-maxSlope)<slopeTol){
      slope[index] = newSlope;
      timer[index] = time;
      time = 0;
      if (index == 0){
        PORTB |= B00010000;
        noMatch = 0;
        index++;
      }
      else if (abs(timer[0]-timer[index])<timerTol &&
abs(slope[0]-newSlope)<slopeTol){
        totalTimer = 0;
        for (byte i=0;i<index;i++){
          totalTimer+=timer[i];
        }
        period = totalTimer;
```

```
      timer[0] = timer[index];
      slope[0] = slope[index];
      index = 1;
      PORTB |= B00010000;
      noMatch = 0;
    }
    else{
      index++;
      if (index > 9){
        reset();
      }
    }
  }
  else if (newSlope>maxSlope){
    maxSlope = newSlope;
    time = 0;
    noMatch = 0;
    index = 0;
  }
  else{
```

```
      noMatch++;
      if (noMatch>9){
        reset();
      }
    }
  }

  time++;

  ampTimer++;
  if (abs(127-ADCH)>maxAmp){
    maxAmp = abs(127-ADCH);
  }
  if (ampTimer==1000){
    ampTimer = 0;
    checkMaxAmp = maxAmp;
    maxAmp = 0;
  }

}
```

```
void reset(){
  index = 0;
  noMatch = 0;
  maxSlope = 0;
}
void stringCheck(){
int rotaryPos = getRotaryValue();
  if(rotaryPos == 1){
    correctFrequency = 82.4;
  }
  else if( rotaryPos == 2 ){
    correctFrequency = 110;
  }
  else if(rotaryPos == 3){
    correctFrequency = 146.8;
  }
  else if( rotaryPos == 4){
    correctFrequency = 196;
  }
  else if( rotaryPos ==5 ){
    correctFrequency = 246.9;
```

```
  }
  else if(rotaryPos == 6){
    correctFrequency = 329.6;
  }
}
void frequencyCheck(){
  if(frequency>correctFrequency+1){
    analogWrite(A3,255);
  }
  if(frequency>correctFrequency+4){
    analogWrite(A2,255);
  }
  if(frequency>correctFrequency+6){
    analogWrite(A1,255);
  }
  if(frequency<correctFrequency-1){
    analogWrite(A5,255);
  }
  if(frequency<correctFrequency-4){
    digitalWrite(9,1);
  }
```

```
  if(frequency<correctFrequency-6){

    digitalWrite(8,1);

  }

  if(frequency>correctFrequency-
1&frequency<correctFrequency+1){

   analogWrite(A4,255);

  }

   else if (frequency<=0){

   digitalWrite(8, LOW);

   digitalWrite(9, LOW);

   analogWrite(A1,0);

   analogWrite(A2,0);

   analogWrite(A3,0);

   analogWrite(A4,0);

   analogWrite(A5,0);

   delay(100);

  }

}

void allLEDsOff(){

  digitalWrite(8, LOW);

  digitalWrite(9, LOW);
```

```
  analogWrite(A1,0);
  analogWrite(A2,0);
  analogWrite(A3,0);
  analogWrite(A4,0);
  analogWrite(A5,0);
}
void tuning(){
  myservo.attach(10);
  if(frequency>correctFrequency+1){
    myservo.write(95);
 }
  if(frequency<correctFrequency-1){
    myservo.write(90);
  }
  if(frequency>correctFrequency-
1&frequency<correctFrequency+1){
    myservo.write(93);
  }
  else if(frequency<=10){
   myservo.write(93);
  }
```

```
}


void loop(){
  allLEDsOff();
  if (checkMaxAmp>ampThreshold){
    frequency = 38462/float(period);
  }
  stringCheck();
  frequencyCheck();
  tuning();
  delay(100);

}
```