Igor Oleynik

Bassokitaran virittimen toteutus mBed-mikrokontrollerin avulla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkötekniikan koulutusohjelma

Insinöörityö

1.9.2015

Metropolia

Tekijä Otsikko Sivumäärä Aika	Igor Oleynik Bassokitaran virittimen toteutus mBed-mikrokontrollerin avulla 19 sivua + 1 liite 1.9.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Sähkötekniikan koulutusohjelma
Suuntautumisvaihtoehto	Elektroniikka
Ohjaaja	Lehtori Kai Lindgren

Työn tavoitteena oli suunnitella ja ohjelmoida laite, joka auttaisi käyttäjää kielisoittimen, erityisesti bassokitaran, virittämisessä. Toissijaisena tavoitteena oli löytää FFT:lle vaihtoehtoinen menetelmä signaalin analysoimiseen.

Työssä käytettiin mBed LPC1768 -alustaa, jolla tapahtuu signaalin A/D-muunnos ja analyysi. Käyttäjän ohjaamiseen ja tulosten näyttämiseen käytettiin Adruino TFT LCD -näyttöä. Ohjelmointi tapahtui mBedin oman verkossa toimivan ohjelmointiympäristön avulla.

Vaihtoehtoiseksi menetelmäksi signaalin analyysille osoittautui Suomessa melkein tuntematon Goertzel-algoritmi. Menetelmä osoittautui toimivaksi ja työn ensimmäiseen tavoitteeseen täysin sopivaksi.

Lopputuloksena on laite, joka lukee sisään tulevan audiosignaalin, laskee signaalin taajuuden ja vertaa sitä haluttuun taajuuteen. Tämän jälkeen laite ohjaa käyttäjää kitaran virittämisessä. Valmista laitetta on helppo soveltaa mihin kielisoittimeen tahansa, pitää vain tietää soittimen kielten oikeat taajuudet.

Avainsanat	mBed, kitaraviritin, Goertzel-algoritmi

Author Title Number of Pages Date	Igor Oleynik Realization of Bass Guitar Tuner Using mBed Microcontroller 19 pages + 1 appendices 1 September 2015
Degree	Bachelor of Engineering
Degree Programme	Electrical Engineering
Specialisation option	Electronics
Instructor	Kai Lindgren, Senior Lecturer

Thesis includes realization of bass guitar tuner using mBed microcontroller. The goal of the study was to design and program a device that would help users to tune stringed instrument, especially bass guitar. The secondary objective was to find an optional method to analyze the signal beside a FFT.

The mBed LPC1768 microcontroller board was used in the work. AD conversion and signal analysis takes place with this board. For controlling the user and displaying results Arduino TFT LCD-monitor is used. Programming was performed in mBeds own, network based, integrated development environment.

An alternative method for signal analysis proved to be the Goertzel algorithm, which is almost unknown in Finland. The method proved to be effective and fully fitting to the first objective of the thesis.

The result is device that reads the input audio signal, calculates the frequency of the signal and compare it to desired frequency. After this the device guides the user in guitar tuning. The ready device is easy to apply to any string instrument; you only need to know the correct frequencies of the instrument strings.

Keywords	mBed, guitar tuner, Goertzel-algorithm

Sisällys

Lyhenteet

FFT	Fast Fourier Transform.
ARM	Advanced Risc Machines
USB	Universal Serial Bus
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
UART	Universal Asynchronous Receiver/Transmitter
CAN	Controller Area Network
PWM	Pulse Width Modulation
TFT LCD	Thin-Film-Transistor Liquid-Crystal Display
DFT	Discrete Furier Transform
DSP	Digital Signal Processing
IIR	Infinite Impulse Responce
ADC, A/D	Analog-Digital Converter
OFDM	Orthogonal Frequency-Division Multiplexing
DCT	Discrete Cosine Transform

1 Johdanto

Työn tavoitteena on suunnitella ja ohjelmoida laite, joka auttaisi bassokitaran virittämisessä. Laite havaitsisi soitetun kielen taajuuden ja sen mukaan ohjeistasi käyttäjää bassokitaran virittämisessä. Laitteen tarkoitus on mahdollistaa bassokitaran virittäminen niille, jotka eivät pysty virittämään sitä omatoimisesti. Vaikka yleensä äänen analysointiin käytetään Fast Fourier Transform (FFT) -algoritmeja, työn tavoitteena on myös löytää vaihtoehtoinen menetelmä. Tällaiseksi menetelmäksi ilmenee ns. Goertzel-algoritmi.

Laite on sopiva havaitsemaan äänisignaalin sekä piuhan että mikrofonin avulla. Laitteen ohjelmointia on myös helppo muuttaa muita kielisoittimia varten. Muutosta varten pitää vain tietää soittimien kielten taajuudet.

2 Laitteessa käytetyt osat ja komponentit

2.1 Kehitysalusta

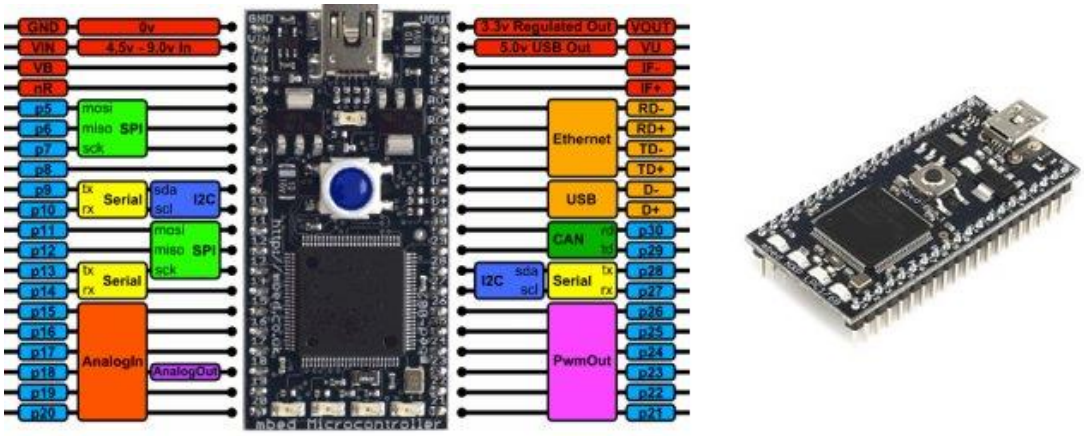
2.1.1 Yleistä mBed-systeemeistä

MBed on käyttöjärjestelmä ja alusta Internet-päätelaitteelle. Alusta perustuu 32-bittiseen ARM Cortex-M mikrokontrolleriin. Projektin ovat kehittäneet ARM ja sen yhteistyökumppanit. [1.]

Ohjelmat mBed-alustalle kirjoitetaan käyttäen ilmaista, verkossa (www.mbed.com) toimivaa koodieditoria ja ohjelmointikielen kääntäjää. Näitä voi käyttää tavallisen verkkoselaimen kautta ja käännös tapahtuu verkkopilvessä, käyttäen ARMCC C/C++ kääntäjää. Verkko-ohjelmointiympäristö mahdollistaa helpon koodien jakamisen. [1.]

On olemassa useita erilaisia vaihtoehtoja alustoille, mutta suosituin niistä on alkuperäinen "mBed Microcontroller board". "MBed Microcontroller Board" ("mbed NXP LPC17680") on NXP mikrokontrolleriin perustuva demo-alusta, jossa on ARM Cortex M3-ydin. [1.]

2.1.2 LPC1768-kehitysalusta



Kuva 1. LPC1768 ulkonäkö ja jalkojen kytkennät [2.]

Työssä käytetään LPC1768-mikrokontrolleria (kuva 1). Se on tehokas ja luotettava kehitysalusta. MBed LPC1768 sopii sekä kokeneelle, että vasta aloittaville käyttäjälle. Mikrokontrolleri on suunniteltu sovelluksiin jotka perustuvat ARM-mikroprosessoriarkkitehtuuriin.

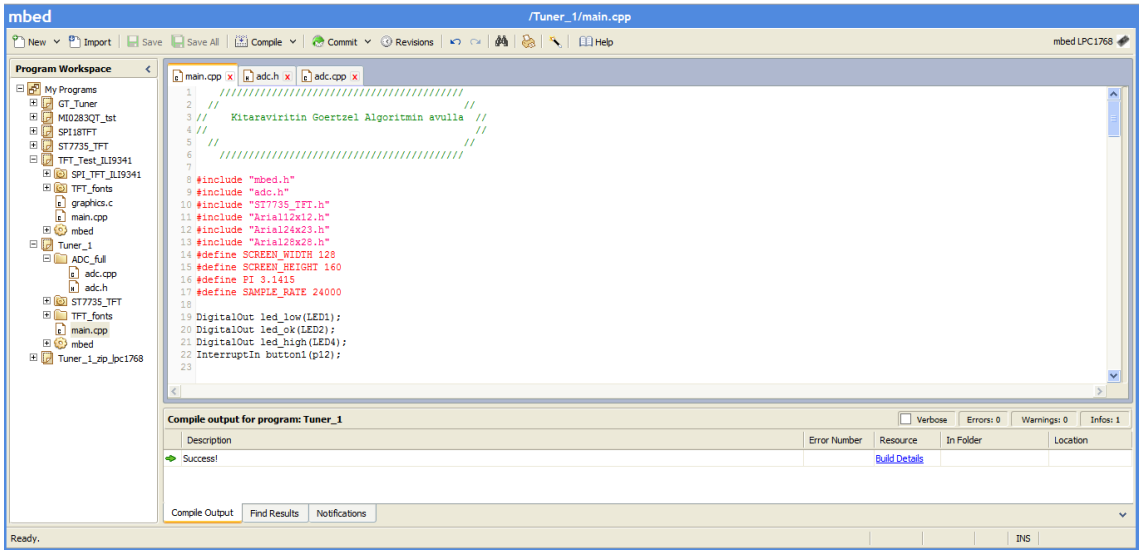
ARM (Advanced Risc Machines) on 32-bittinen mikroprosessoriarkkitehtuuri, joka on nykyään suosittu sulautettujen järjestelmien suorittimissa. MBed NXP LPC1768 -alusta käytetään erilaisissa automaattisissa ja teollisuuden ohjausjärjestelmissä, antureissa sekä mikrokontrollereissa. Se sopii sovelluksiin, jotka vaativat suurta suorituskykyä sekä reaaliaikaista toimintaa. Se sisältää 512 kilotavua flash- ja 32 kilotavua RAM-muistia. MBed NXP LPC1768 sisältää sisäänrakennetut Ethernet- ja USB-liitännät, portit SPI-, I2C-, UART- ja CAN- väyliä varten, kuusi jalkaa ADC-muunninta ja kuusi jalkaa PWM-ohjausta (pulssinleveysmodulaatio) varten (kuva 2). Mikrokontrollerin jalkoja 5–30 voidaan myös käyttää tavallisina digitaalisina tulo/lähtö portteina. [2.]

ARM Cortex-M3 core	• 100 MHz operation • Nested Vectored Interrupt Controller for fast deterministic interrupts • Wakeup Interrupt Controller allows automatic wake from any priority interrupt • Memory Protection Unit • Four reduced-power modes: sleep, deep sleep, power-down and deep power-down
Memories	• 512 KB of Flash memory • 64 KB of SRAM
Serial peripherals	• 10/100 Ethernet MAC • USB 2.0 full-speed device/Host/ OTG controller with on-chip PHY • Four UARTs with fractional baud rate generation, RS-48, modem control, and IrDA • Two CAN 2.0B controllers • Three SSP/SPI controllers • Three I²C-bus interfaces with one supporting Fast Mode Plus (1-Mbit/s data rates) • I²S interface for digital audio
Analog peripherals	• 12-bit ADC with eight channels • 10-bit DAC
Other peripherals	• Ultra-low-power (< 1 uA) RTC • General-purpose DMA controller with eight channels • Up to 70 GPIO • Motor control PWM and Quadrature Encoder Interface to support three-phase motors • Four 32-bit general-purpose timers/counters
Package	• 100-pin LQFP (14 x 14 x 1.4 mm)

Kuva 2. LPC1768 mikrokontrollerin tekniset ominaisuudet [2.]

2.1.3 MBed-ohjelmointiympäristö (mBed Compiler)

MBed Compiler tarjoaa kevyen, verkossa toimivan C / C++ -ohjelmointiympäristön, joka on valmiiksi konfiguroitu, jotta voi nopeasti kirjoittaa ohjelmia, kääntää niitä ja ladata ne mBed-mikrokontrolleriin (kuva 3). Itse asiassa, käyttäjän ei tarvitse asentaa tai määrittää mitään saadakseen ohjelman toimimaan mBedin kanssa. Koska se on web-sovellus, voi siihen kirjautua sisään mistä tahansa ja jatkaa siitä, mihin jäi. Ohjelmointiympäristö toimii kaikilla käyttöjärjestelmillä. [3.]



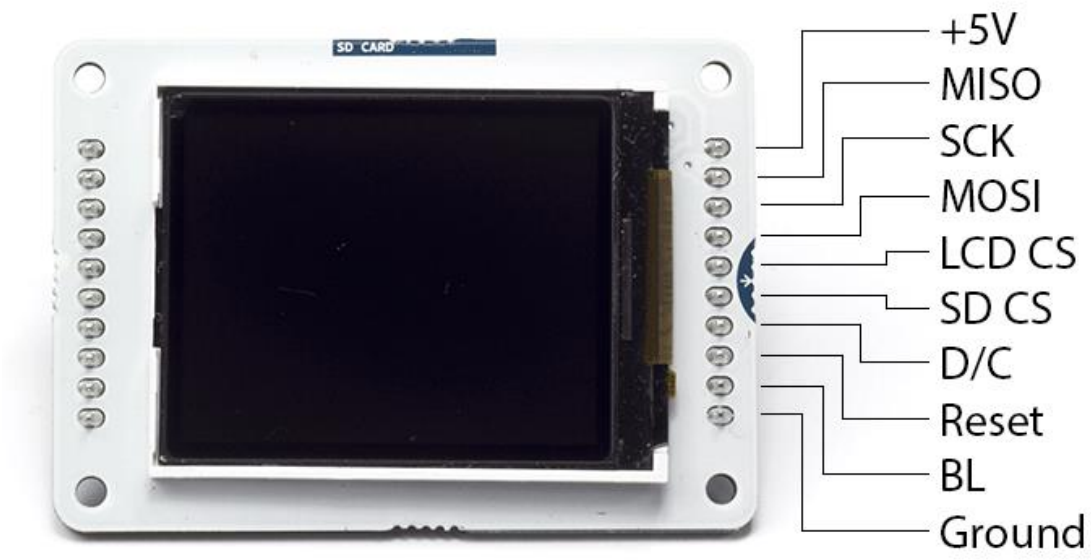
Kuva 3. MBed Compiler-ohjelmointiympäristö

Kääntäjä käyttää ammatillista ARMCC-moottoria, joten se tuottaa tehokkaita koodeja, joita voidaan käyttää ilman mitään maksuja myös kaupallisissa sovelluksissa. Ohjelmistoympäristö sisältää työtilan version ohjauksen, koodin muotoilun ja automaattisen dokumentoinnin julkaisevia kirjastoja varten. MBed-työkalut keskittyvät prototyyppeihin, ja ovat suunniteltu nopeaa kokeilua varten. Ne täydentävät muita ammattilaisia tuotantotason työkaluja. Käyttäjä voi julkaista projektia suoraan Compiler-työtilan mbed.org-verkkosivuilla jakaakseen koodia muiden kanssa ja viedä jo valmiita kirjastoja työtilaan saadakseen hyvän startin. [3.]

2.2 Näyttö

2.2.1 Arduino TFT LCD

Arduino TFT -näyttö on taustavalaistu LCD-näyttö (kuva 4). Näyttöön voi piirtää tekstiä, kuvia ja muotoja TFT-kirjaston avulla. Näytössä on sisäänrakennettu micro-SD-korttipaikka, johon voi esimerkiksi varastoida kuvia näyttämistä varten. Vaikka näyttö on suunniteltu Arduino Esploraa varten, sitä voi käyttää minkä tahansa AVR:n kanssa. Näyttö on 1.77-tuumainen ja näytön resoluutio on 160 x 128 pikseliä. Näyttö toimii +5 V:n DC:llä. [5.]



Kuva 4. Arduino LCD TFT [5.]

2.2.2 Näytön kirjasto

Toimiakseen mBed-mikrokontrollerin kanssa näyttö vaati erikoiskirjaston. Kirjasto on kokoelma koodeja, joiden tarkoitus on mahdollistaa erityisen osan toimintaa. Tärkein kirjaston tavoite on pakata jotain niin, ettei joka kerta tarvitse kirjoittaa uutta koodia, jotta saisi lisäosan tai funktion toimimaan. Kirjastoa voi pitää ohjelman rakennuspalikkana, mutta ei ohjelmana itsenään, sillä kirjastolla ei esimerkiksi ole main() funktiota. [5.]

Työssäni käytin Jonne Valolan tekemää kirjastoa Arduino TFT LCD -näytön ohjelmoimiseen [6.].

2.3 Muut komponentit

2.3.1 Äänen kaappaus

Äänisignaalin kaappausta varten työssäni käytetään kahta vaihtoehtoista laitetta: mikrofonia tai 6-millimetristä mono audio plugia. Mikrofonina on Adafruit MAX4466 -mikrofoni.

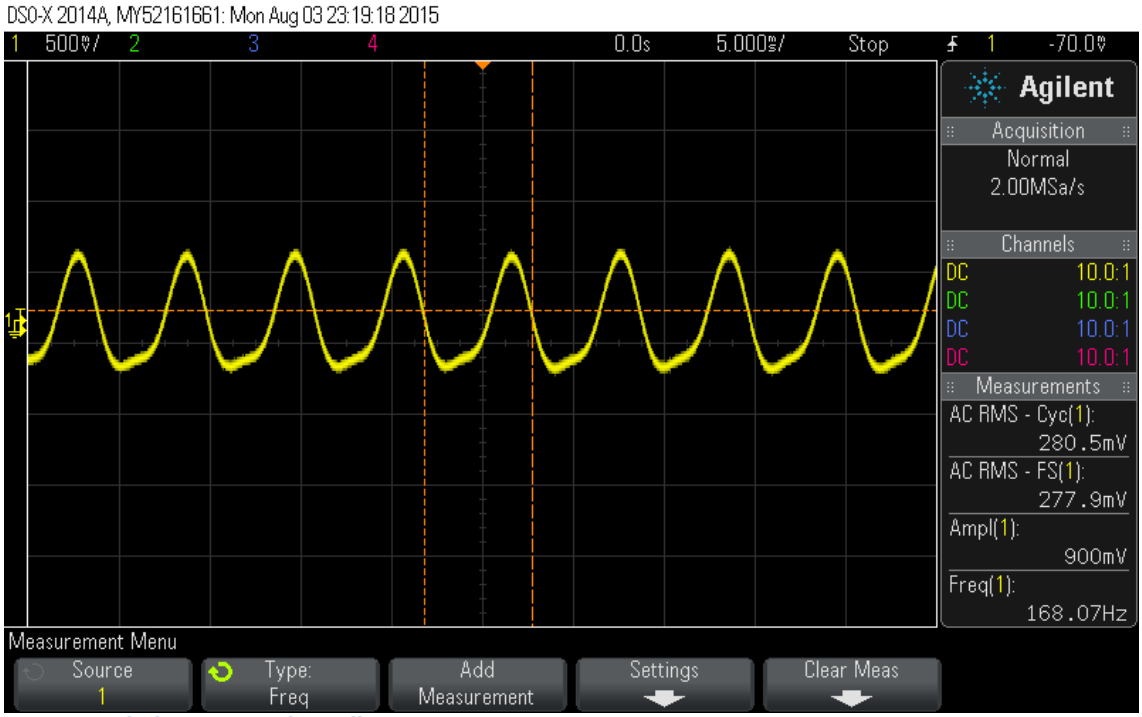
2.3.2 Signaalin vahvistaminen

Operaatiovahvistin on analogiaelektroniikan komponentti. Operaatiovahvistin sisältää useita transistoreita, vastuksia, kondensaattoreita, sekä muita komponentteja. Kuitenkin yleensä operaatiovahvistin myydään yhtenä komponenttina, vaikka esimerkiksi opetustarkoitukseen sitä voi rakentaa erillisistä komponenteista. Operaatiovahvistimen tarkoituksena on vahvistaa sisääntulon jännitettä tietyllä vahvistinkertoimella. Operaatiovahvistin sisältää tavallisesti kaksi sisäänmenoa ja yhden ulostulon. Tämän lisäksi on myös ns. ”käyttösähkön navat”. Operaatiovahvistimessa käytetään tavallisesti kaksipuoleista jännitesyöttöä, jolloin ulostulojännite on samassa vaiheessa kuin sisäänmenojännite. [11.]

Operaatiovahvistin vertailee tuloliittimien välistä jännite-eroa. Kytkentätyyppeinä käytetään joko suljettua silmukkaa tai avointa silmukkaa. Avoin silmukka tarkoittaa, että piirissä ei ole takaisinkytkentää ulostulosta sisääntuloon. [11.]

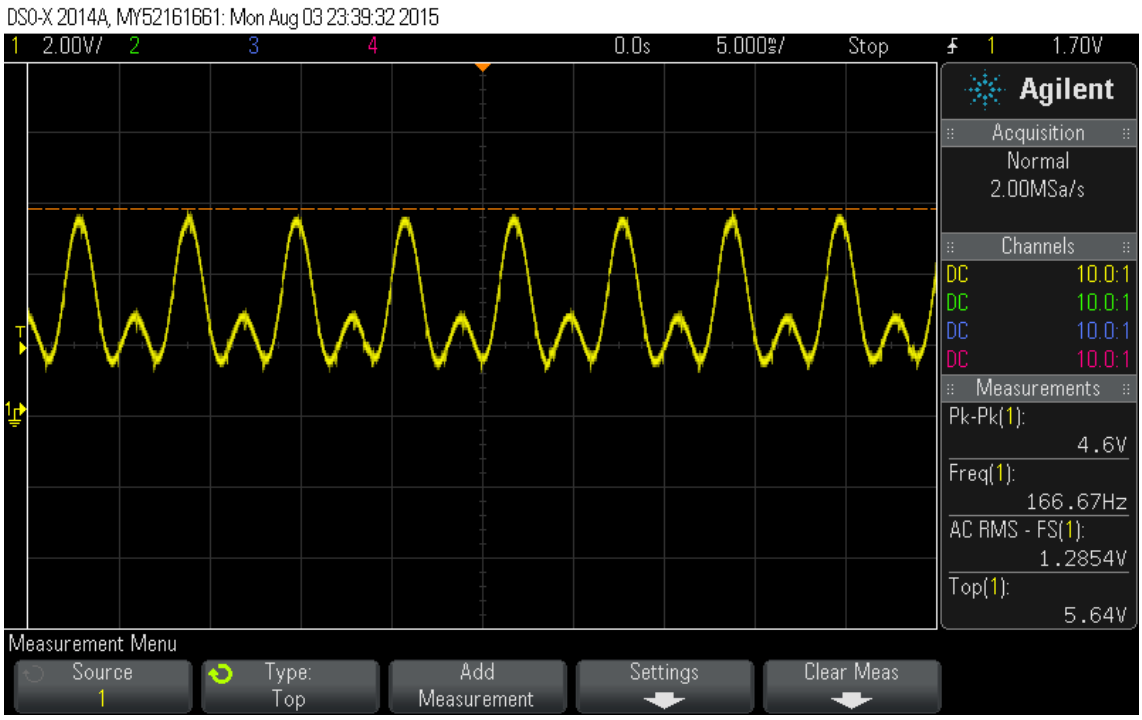
Operaatiovahvistimissa yleensä käytetään negatiivistä silmukkaa. Suljettu silmukka voi olla joko negatiivinen tai positiivinen takaisinkytkentä. Negatiivinen takaisinkytkentä parantaa vahvistinta kasvattamalla kaistanleveyttä ja vähentämällä häiriöitä. Negatiivisella takaisinkytkennällä myös saadaan suodatettua ei toivottuja signaalin komponentteja pois. Positiivista takaisinkytkennän tuloksena on hystereesipari. [11.]

Kitarasta tulevasta signaalin mittauksesta selvisi, että kitaran voimakkaimman signaalin amplitudin voimakkuus on 280mV +/- 10mV, joka saavutetaan 168 Hz:llä..



Kuva 5. Vahvistamaton signaali

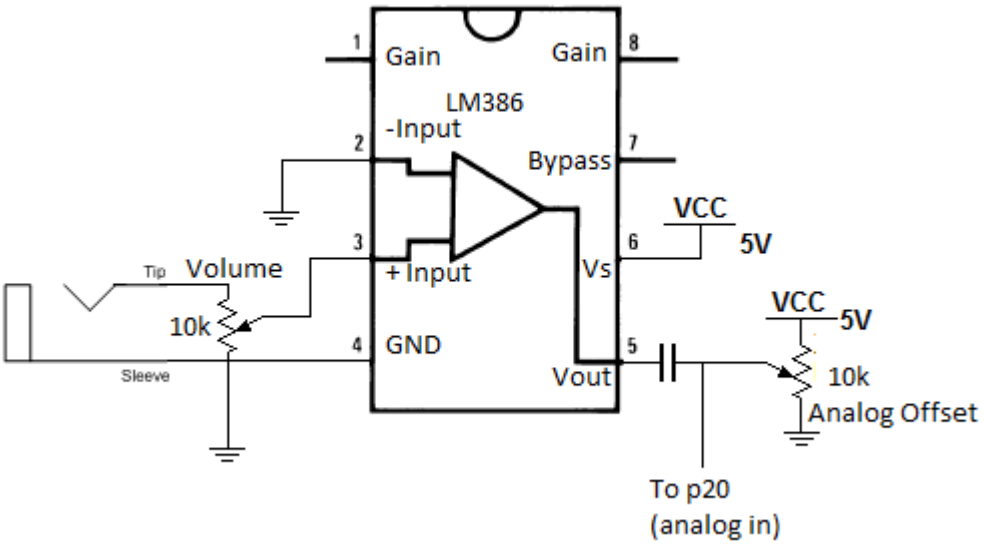
Koska kitarasta tuleva signaali on heikko, kytkentä tarvitsee erillisen vahvistimen. Signaalin vahvistamiseen käytettiin LM386-audiovahvistinta. Lisäksi käytettiin kahta 10K-potentiometriä ja yhtä 330nF-kondensaattoria. LM386-vahvistin on suunniteltu vahvistamaan 26 dB:n verran, eli vahvistuskerroin on 20. Vahvistuksen jälkeen kitarasta tuleva signaali on noin 1 V. Itse signaali on sinimuotoista ja vaihtelu on näin -1 V:n ja +1 V:n välillä. Koska LPC1768-mikrokontrollerin analoginen sisääntulo tunnistaa ainoastaan 0 V:n ja 5 V:n välillä olevat signaalit, on kitarasta tulevaan signaaliin kytkettävä erillinen jännite. Lopputuloksena on kuvan 6 mukainen signaali.



Kuva 6. Vahvistettu signaali

Koska laite on suunniteltu käytettäväksi erilaisten soittimien kanssa, tavallisten vastusten sijaan käytin 10k:n potentiometreja. Näin vahvistusta voi tarvittaessa pienentää.

Vahvistimen lopullinen kytkentä on kuvassa 7.



Kuva 7. Signaalivahvistuksen kytkentäkaavio

Mikrofonin tapauksessa kolmas jalka kytketään +5 V:n jännitteeseen.

2.3.3 Fourier-muunnos

”Fourier-muunnos on matematiikassa käytetty jatkuva integraalimuunnos. Muunnosta käytetään erityisesti analyysissä, differentiaaliyhtälöiden ratkaisemisessa ja signaalinkäsittelyssä erilaisiin taajuusanalyysiä vaativiin sovelluksiin. Muunnos perustuu teoreemaan, jonka mukaan mikä tahansa jatkuva riittävän säännöllinen funktio voidaan esittää sinimuotoisten funktioiden integraalina. Fourierin muunnoksella voidaan laskea näiden sinimuotoisten komponenttien amplitudi ja vaihe.” [7.]

Fourier-muunnos sopii parhaiten jatkuva-amplitudisten ja jatkuva-aikaisten signaalien taajuusanalyysiin. Käytännössä signaali tunnetaan usein vain mitattujen näytteiden muodossa, eikä signaalilauseketta ole käytettävissä. On kaksi lähestymistapaa taajuusanalyysin suorittamiseksi. Ensimmäinen tapa on approksimoida signaalia ja suorittaa approksimaatiolle analyyttisesti Fourier-muunnos. Toinen tapa on käyttää numeerista algoritmia, jolloin signaalispektrikin tunnetaan vain numeerisessa muodossa. Diskreetti Fourier muunnos on esimerkki tällaisesta algoritmista. [7.]

Diskreetti Fourier-muunnos, eli **DFT,** on Fourier-muutoksen diskreettiaikainen yleistys. Siinä signaali esitettään äärellisenä Fourier sarjana ja integraali korvautuu summalausekkeella.

$$F_n = \sum_{k=0}^{N-1} f_k e^{-2\pi i n k/N} \quad , n = 0, ..., N-1$$

Jossa $f_k$on N:n pituinen reaali- tai kompleksiarvoinen sarja.

Diskreetin käänteismuunnos on

$$f_k = \frac{1}{N} \sum_{n=0}^{N-1} F_n e^{2\pi i n k/N}$$

. [7.]

2.3.4 Goertzel-algoritmi

Goertzel-algoritmi on digitaalinen signaalikäsittelytekniikka (DSP), joka tarjoaa tehokkaan keinon yksittäisen taajuuskomponentin arviointiin diskreetti Fourier-muunnoksen puitteissa. Tämä tekee algoritmistä erityisen sopivan tiettyihin sovelluksiin, kuten puhe-

linlaitteissa käytettyyn numeroiden äänitaajuusvalintaan (DFT). Algoritmin on ensimmäisen kerran kuvannut Gerald Goertzel, vuonna 1958. [9.]

Kuten DFT, Goertzel-algoritmi analysoi yhden valittavissa olevan taajuuden komponentin diskreettisignaalista. Toisin kuin suorat DFT-laskelmat, Goertzel-algoritmi operoi jokaisella iteraatiolla yhdellä reaaliarvoisella kertoimella, käyttäen reaaliarvoista aritmeettista syötesekvenssiä. Koko spektrin kääntämiseksi Goertzel algoritmi on monimutkaisempi kuin Fast Fourier algoritmi, mutta muutaman valikoidun taajuuskomponentin laskemiseen, se on numeerisesti tehokas. Goertzel-algoritmin yksinkertainen rakenne tekee sen hyvin sopivaksi pienille prosessiohjelmille ja sulautettuihin sovelluksiin, eikä se rajoitu näihin. [9.]

Goertzel-algoritmia voidaan myös käyttää "käänteisesti" sinikäyrän muodostamista varten. Sinikäyrä tällöin vaatii vain yhden kerto- ja yhden vähennyslaskun yhtä generoitua näytettä kohden. [9.]

Peruslaskutoimitukset Goertzel-algoritmissä ovat digitaalisia suodattimia, ja tästä syystä algoritmia kutsutaan usein Goertzel-suodattimeksi. Suodatin toimii tulosekvenssissä $x(n)$, kahden tason kaskadissa. Parametri f antaa analysoitavan taajuuden, joka on taas normalisoitu näytesykliin.

Ensimmäinen vaihe laske välisekvenssin, $s(n)$:

$$s(n) = x(n) + 2\cos(2\pi f)s(n-1) - s(n-2)$$

Toisessa vaiheessa sovelletaan filtteri $s(n)$, jolloin tuloksena saadaan sekvenssiyhtälö $y(n)$.

$$y(n) = s(n) - e^{-2\pi i f}s(n-1)$$

Ensimmäistä vaihetta voidaan pitää toissijaisena lineaarisena IIR- (Infinite impulse response) suodattimena. Tälle rakenteelle on ominaista, että sen sisäiset tilamuuttujat ovat yhtä suuria kuin edellisen laskelman tulokset. Sisääntulot ehdolle $x(n)$, $n < 0$

ovat kaikki yhtä suuret kuin 0. Aloitusfiltterin alustamista varten yhtälöä voi aloittaa näytteellä $x(0)$. Suodattimen tilat ovat esiasennettuja arvoille

$$s(-2) = s(-1) = 0.$$

Toista vaihetta voi pitää FIR -filtterinä, koska sen laskutoimituksissa ei käytetä yhtään vanhempaa lähtöä.

Z-muunnoksen metodia voi soveltaa filterin tutkimiseen. Z-muunnos ensimmäisestä vaiheesta on

$$\frac{S(z)}{X(z)} = \frac{1}{1 - 2\cos(2\pi f)z^{-1} + z^{-2}} = \frac{1}{(1 - e^{+2\pi i f} z^{-1})(1 - e^{-2\pi i f} z^{-1})}.$$

Toisen vaiheen Z-muunnos on

$$\frac{Y(z)}{S(z)} = 1 - e^{-2\pi i f} z^{-1}$$

Yhdistetty muunnosfunktio kahdelle vaiheelle on näin ollen

$$\frac{S(z)}{X(z)}\frac{Y(z)}{S(z)} = \frac{Y(z)}{X(z)} = \frac{(1 - e^{-2\pi i f} z^{-1})}{(1 - e^{+2\pi i f} z^{-1})(1 - e^{-2\pi i f} z^{-1})} = \frac{1}{1 - e^{+2\pi i f} z^{-1}}.$$

Tämä voidaan muuntaa takaisin vastaavaan ajan funktioon:

$$\begin{aligned} y(n) &= x(n) + e^{+2\pi i f} y(n-1) \\ &= \sum_{k=-\infty}^{n} x(k) e^{+2\pi i f (n-k)} \\ &= e^{+2\pi i f n} \sum_{k=0}^{n} x(k) e^{-2\pi i f k} \end{aligned}$$ [9.]

2.3.5 Kitaran virittäminen

Viritetyssä neljäkielisessä bassokitarassa kielten taajuudet ovat seuraavat:

Kieli:	Nuotit:	Taajuus:
4	G2	97.999 Hz
3	D2	73.416 Hz
2	A	55 Hz
1	E	41.204 Hz

Taajuudet ovat perustaajuuksia, eikä yläsävelsarjoja oteta huomioon.

2.3.6 ADC

”A/D-muunnin eli analogia–digitaali–muunnin on elektroniikassa laite, joka muuntaa jatkuvan analogisen signaalin – esimerkiksi jännitteen – arvoja digitaalisiksi lukuarvoiksi. (D/A-muunnin eli digitaali-analogiamuunnin suorittaa päinvastaisen operaation.)” [10.]

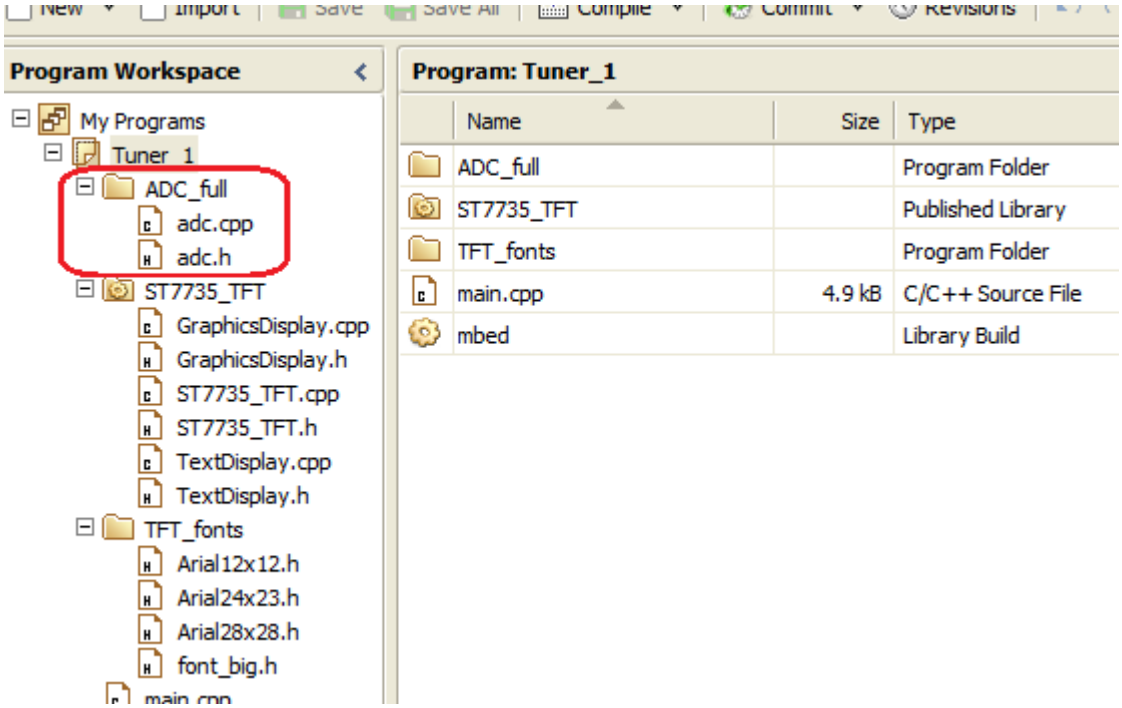
Toisin kuin analogista signaalia, digitaalista signaalia voi käsitellä esimerkiksi tietokoneella tai jollain muulla erityisellä digitaalisella signaaliprosessorilla. A/D-muuntimia käytetään hyvin erilaisissa sovelluksissa. Tämänlaisia ovat esimerkiksi matkapuhelimet, mittalaitteet, äänentalletuslaitteet ja radiomodeemit. [10.]

Muuntimien ominaisuuksia ovat resoluutio ja näytteentaajuudesta riippuva muunnosaika.

Muuntimen resoluutio, eli erottelukyky näyttää kuinka monella bitillä muunnin muuntaa signaalia digitaaliseksi arvoksi.

Näytteenottotaajuus kertoo, kuinka usein muunnos tehdään. A/D muuntimen suurin näytteenottotaajuus riippuu siitä mikä on yhteen muunnokseen tarvittava aika. Nyquistin teoreeman mukaan digitoitavasta signaalista tulee ottaa näytteitä taajuudella, joka on suurempi kuin kaksinkertainen alkuperäinen signaali [10.]

Koska ohjelmassa käytettävät taajuudet ovat erittäin matalia (< 1 kHz), muunnosajasta ei tarvitse välittää. Ohjelmassa A/D muunnosta vastaa ADC_full-ohjain (kuva 8).



Kuva 8. ADC_full-ohjain

3 Laitteen ohjelmointi

3.1 Algoritmin toteutus C-kielellä

Goertzel-algoritmi käyttää alla esitettyä koodia. Koko koodi ilman kommentteja löytyy liitteestä yksi.

```
int string_select = 0;                                    //Valitun kielen nollaus
float high, high1, in_tune, in_tune1, in_tune2, in_tune3,
low, low1, note, low_mod, high_mod;
char* key;
int Counter = 0;
int Buffer[6000];

float goertzelFilter(int samples[], float frequency, int N) {  //frequency =
kielen taajuus
            float s_preview = 0.0;
            float s_preview2 = 0.0;                          //Väliaikainen muisti
            float coefficent, normalizedfrequency, power, s;//Lisää muuttujaa
            int i;
            normalizedfrequency = frequency / SAMPLE_RATE;
            coefficent = 2 * cos(2 * PI*normalizedfreqency);//Itse laskutoimitus
            for (i = 0; i<N; i++) {
```

```
                        s = samples[i] + coefficent * s_preview - s_prevpriew2;
                        s_preview2 = s_preview;
                        s_preview = s;
                }
                power = s_preview2*s_preview2 + s_preview*s_preview - coeffi-
                cent*s_preview*s_preview2;
                return power;
}

ADC adc(SAMPLE_RATE, 1);        //Määritetään puskuri

void sample_audio(int chan, uint32_t value) {
                Buffer[Counter] = adc.read(p20);
                Counter += 1;
}
while (1) {         // Määritetään analysoivat kielet.

                switch (string_select) {
                case 0:
                        note = 70;
                        key = "E";
                        break;
                case 1:
                        note = 93;
                        key = "B";
                        break;
                case 2:
                        note = 123;
                        key = "F#";
                        break;
                case 3:
                        note = 166;
                        key = "C#";
                        break;
                }

//Valmistellaan pinnit A/D muunnosta varten ja asetetaan keskeytyksen ohjaus
(käyttäen Simon Blandfordin ADC draiveria)
adc.append(sample_audio);
adc.startmode(0, 0);
adc.burst(1);
adc.setup(p20, 1);

//Aloitetaan keskeytys ja odotetaan noin 4096 näytettä
adc.interrupt_state(p20, 1);
wait(.2);

//A/D muunoksen lopetus, 20 pinnin alustus
adc.interrupt_state(p20, 0);
adc.setup(p20, 0);
int actual_rate = adc.actual_sample_rate();

high = 0;
low = 0;
//Määritetään liian korkean ja matalan äänen rajat
for (int i = 2; i<46; i += 2) {
                high1 = goertzelFilter(Buffer, (note + i), Counter);
                if (high1 > high) high = high1;
}
//Määritetään liian korkean ja matalan äänen rajat
```

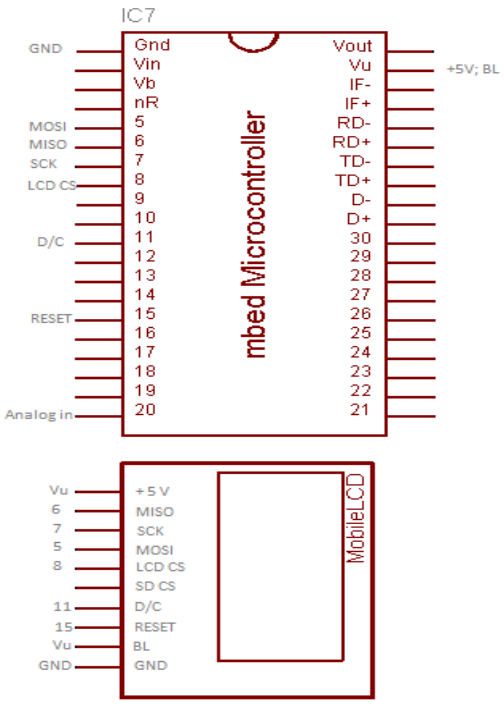
```
for (int i = 2; i<46; i += 2) {
            low1 = goertzelFilter(Buffer, (note - i), Counter);
            if (low1 > low) low = low1;
}
//Määritetään nuottiin osunutta ääntä
in_tune1 = goertzelFilter(Buffer, (note + 1), Counter);
in_tune2 = goertzelFilter(Buffer, note, Counter);
in_tune3 = goertzelFilter(Buffer, (note - 1), Counter);
if ((in_tune1 > in_tune2) && (in_tune1 > in_tune3)) in_tune = in_tune1;
else if ((in_tune2 > in_tune1) && (in_tune2 > in_tune3)) in_tune = in_tune2;
```

3.2 Näytettävä teksti

Näyttöön ilmestyvästä tekstistä vastaa seuraava koodi:

```
// Näytön teksti
TFT.locate(0, 0);
TFT.printf("::Kitaraviritin::");
TFT.locate(0, 15);
TFT.printf("Virittävä kieli: %i", (4 - string_select));
TFT.locate(0, 30);
TFT.printf("%s kielen taajuus: %i Hz", key, (int)note);
TFT.locate(0, 45);
if (led_ok) TFT.printf("Vireessä!");
else if (led_low) TFT.printf("Liian matala!");
else if (led_high) TFT.printf("Liian korkea!");
else TFT.printf("~~~~~~~~");
```

3.3 Kytkentäkaavio



Kuva 9. Näytön ja mikroprosessorin kytkentä

Kytkennässä (kuva 9) audiojakin voi vaihtaa mikrofoniksi. Koodissa ei tarvitse vaihtaa mitään.

Laite soveltuu myös muiden kielisoittimen virittämiseen, pitää vain tietää soittimen kielten taajuudet. Ohjelmassa pitää tällöin muuttaa kohtaa, jossa määritetään analysoivat kielet.

Uudessa sovelluksessa voi olla enemmän kuin neljä kieltä. Tällöin lisätään vain puuttuvat case kohdat ja muokataan kohtaa:

```
TFT.printf("Tuning String: %i", (4-string_select));
```

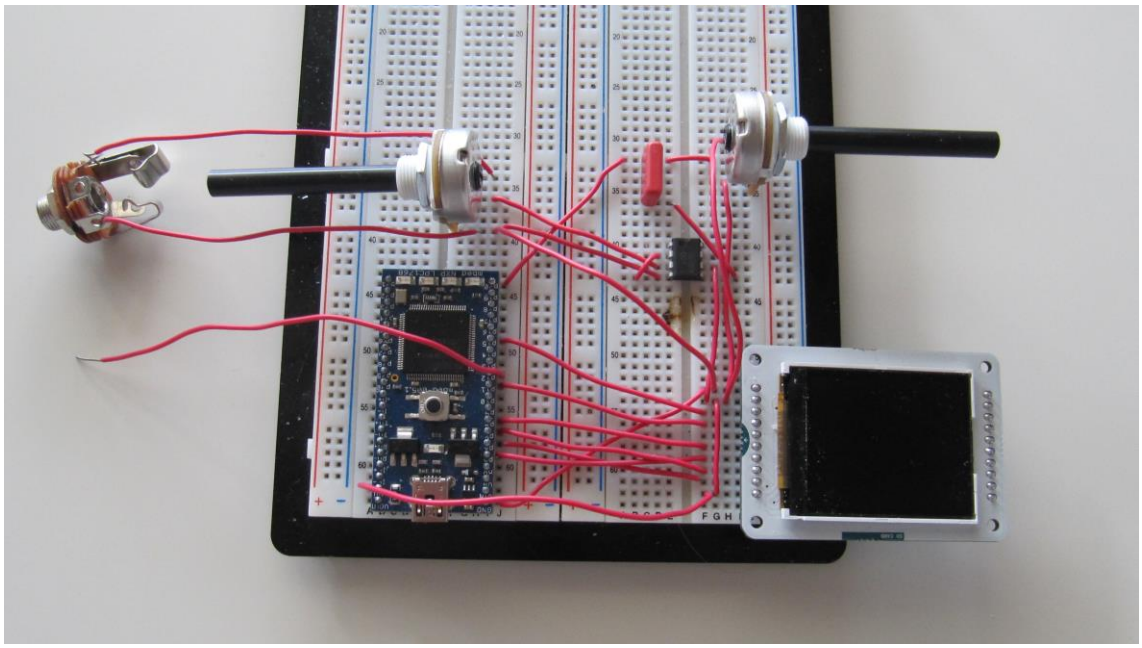
Esimerkiksi tavallisen kuusikielisen kitaran koodi näyttäisi tältä:

```
while (1) {

            switch (string_select) {
            case 0:
                        note = 82;
                        key = "E";
                        break;
            case 1:
```

```
                note = 110;
                key = "A";
                break;
        case 2:
                note = 146;
                key = "D";
                break;
        case 3:
                note = 196;
                key = "G";
                break;
        case 4:
                note = 247;
                key = "B";
                break;
        case 5:
                note = 330;
                key = "E";
                break;

        }
```

3.4 Laitteen kokoonpano

Laite on koottu tavalliselle PCB-alustalle (kuva 10). Pinnien välisiä yhteyksiä varten on käytetty tavallista 2 mm:n alumiinikaapelia. Virtaa laite saa USB-kaapelin välityksellä. Laite myös voi toimia 5 V:n paristojen tai akun kautta.



Kuva 10. Valmis kytkentä

4 Loppusanat

Työn tavoitteena oli suunnitella laite, joka tunnistaa kitarasta tulevan signaalin ja tämän mukaan neuvoo käyttäjää pyörittämään virityskoneiston nuppeja oikeaan suuntaan. Laitteen rakentaminen ei tuottanut suuria haasteita. Haasteellisinta oli ohjelmointi. Vaikeuksia tuotti näytön sovittaminen mBed-mikrokontrollerin kanssa.

Lopputuloksena on kuitenkin täysin toimiva laite, jonka parhaimpana ominaisuutena pidän hyviä muokkaamismahdollisuuksia. Laitetta on helppo soveltaa erilaisten soittimien kanssa. Tämä vaatii ainoastaan pienen muutokseen koodissa, eikä itse laitteen osiin tarvitse koskea.

Erityisen kiinnostavaa työstä teki se, että pääsi tutustumaan erilaisiin analysointimenetelmiin. Työssä käytetty Goertzel-algoritmi on Suomessa melkein tuntematon, vaikka tietyissä tapauksissa, kuten tässä työssä, se on paljon kätevämpi ja yksinkertaisempi kuin esimerkiksi FFT.

Lähteet

1 Wikipedia. MBed. 2015. Verkkodokumentti <https://en.wikipedia.org/wiki/Mbed>. Luettu 3.7.2015.

2 MBed. LPC1768 alusta. 2015. Verkkodokumentti. <https://developer.mbed.org/platforms/mbed-LPC1768/>. Luettu 3.7.2015.

3 MBed. Compiler. 2015. Verkkodokumentti. <https://developer.mbed.org/handbook/mbed-Compiler>. Luettu 19.6.2015.

4 Arduino. TFT LCD Näyttö. 2015. Verkkodokumentti. <https://www.arduino.cc/en/Main/GTFT>. Luettu 3.7.2015.

5 MBed. Kirjastot. 2014. Verkkodokumentti. <https://developer.mbed.org/cookbook/Writing-a-Library>. Luettu 3.7.2015.

6 MBed. Näytön ohjauskirjasto. 2015. Verkkodokumentti. <https://developer.mbed.org/users/smultron1977/notebook/spi-18-tft--mbed-work-in-progress/>. Luettu 3.7.2015.

7 Fourier’n muunnos. 2015. Verkkodokumentti. <https://fi.wikipedia.org/wiki/Fourier%E2%80%99n_muunnos>. Luettu 3.6.2015.

8 Nopea Fourier’n muunnos. 2015. Verkkodokumentti. <https://ru.wikipedia.org/wiki/%D0%91%D1%8B%D1%81%D1%82%D1%80%D0%BE%D0%B5_%D0%BF%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%A4%D1%83%D1%80%D1%8C%D0%B5>. Luettu 19.6.2015.

9 Aalto. Diskreetti Fourier’n muunnos. 2013. Verkkodokumentti <https://noppa.aalto.fi/noppa/kurssi/s-72.1110/harjoitustyot/S-72_1110_dft__osa_1.pdf>. Luettu 19.6.2015.

10 A/D muunnin. 2015. Verkkodokumentti. <https://fi.wikipedia.org/wiki/A/D-muunnin>. Luettu 3.7.2015.

11 Operaatiovahvistin. 2015. Verkkodokumentti. <https://fi.wikipedia.org/wiki/Operaatiovahvistin>. Luettu 17.7.2015

1 Main.cpp

```
#include "mbed.h"
#include "adc.h"
#include "ST7735_TFT.h"
#include "Arial12x12.h"
#include "Arial24x23.h"
#include "Arial28x28.h"
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 160
#define PI 3.1415926
#define SAMPLE_RATE 24000

DigitalOut led_low(LED1);
DigitalOut led_ok(LED2);
DigitalOut led_high(LED4);
InterruptIn button1(p12);

 ST7735_TFT TFT(p5, p6, p7, p8, p11, p15,"TFT");
int string_select = 0;
float high, high1, in_tune, in_tune1, in_tune2, in_tune3,
low, low1, note, low_mod, high_mod;
char* key;
int Counter = 0;
int Buffer[6000];

float goertzelFilter(int samples[], float frequency, int N) {
            float s_preview = 0.0;
            float s_preview2 = 0.0;
            float coefficent, normalizedfrequency, power, s;
            int i;
            normalizedfrequency = frequency / SAMPLE_RATE;
            coefficent = 2 * cos(2 * PI*normalizedfrequency);
            for (i = 0; i<N; i++) {
                        s = samples[i] + coefficent * s_preview - s_prevpriew2;
                        s_preview2 = s_preview;
                        s_preview = s;
            }
            power = s_preview2*s_preview2 + s_preview*s_preview – coeffi-
            cient*s_preview*s_preview2;
            return power;
}

ADC adc(SAMPLE_RATE, 1);

void sample_audio(int chan, uint32_t value) {
    Buffer[Counter] = adc.read(p20);
    Counter += 1;
}

void button1_pressed() {
    string_select++;
    if (string_select > 3) string_select = 0;
}

int main() {
    button1.mode(PullDown);
```

```
    button1.rise(&button1_pressed);

    TFT.cls();
    TFT.set_font((unsigned char*) Arial12x12);

    while (1) {

        switch (string_select) {
            case 0:
                note = 70;
                key= "E1";
                break;
            case 1:
                note = 93;
                key= "A1";
                break;
            case 2:
                note = 123;
                key= "D2";
                break;
            case 3:
                note = 166;
                key= "G2";
                break;
        }

        adc.append(sample_audio);
        adc.startmode(0,0);
        adc.burst(1);
        adc.setup(p20,1);

        adc.interrupt_state(p20,1);
        wait(.2);

        adc.interrupt_state(p20,0);
        adc.setup(p20,0);
        int actual_rate = adc.actual_sample_rate();

        printf("Requested max sample rate is %u, actual max sample rate is
%u.\n",
               SAMPLE_RATE, actual_rate);
        printf("We did %i samples\n",Counter);

high = 0;
low = 0;
for (int i=2; i<46; i+=2) {
    high1 = goertzelFilter(Buffer, (note + i ), Counter);
    if (high1 > high) high=high1;
}
for (int i=2; i<46; i+=2) {
    low1 = goertzelFilter(Buffer, (note - i ), Counter);
    if (low1 > low) low=low1;
}
        in_tune1 =  goertzelFilter(Buffer, (note+1), Counter);
        in_tune2 =  goertzelFilter(Buffer, note, Counter);
        in_tune3 =  goertzelFilter(Buffer, (note-1), Counter);

        if ((in_tune1 > in_tune2) && (in_tune1 > in_tune3)) in_tune = in_tune1;
        else if ((in_tune2 > in_tune1) && (in_tune2 > in_tune3)) in_tune =
in_tune2;
```

```
        else in_tune = in_tune3;
        if ((in_tune >= high) && (in_tune >= low)) {
            led_high = 0;
            led_ok = 1;
            led_low = 0;
        } else if (high >= in_tune) {
            led_high = 1;
            led_ok = 0;
            led_low = 0;
        } else if (low >= in_tune) {
            led_high = 0;
            led_ok = 0;
            led_low = 1;

        TFT.locate(0,0);
        TFT.printf("::Guitar Tuner::");
        TFT.locate(0,15);
        TFT.printf("Tuning String: %i", (4-string_select));
        TFT.locate(0,30);
        TFT.printf("%s at %i Hz",key, (int) note);
        TFT.locate(0,45);
        if (led_ok) TFT.printf("In Tune!");
        else if (led_low) TFT.printf("Too Low ");
        else if (led_high) TFT.printf("Too High");
        else TFT.printf("~~~~~~~~");
TFT.locate(0,60);
        TFT.printf("%.0f high %.0f low %.0f tune", high, low, in_tune);
        Counter = 0;

    }

}
```

1.1 adc.h

```
#ifndef MBED_ADC_H
#define MBED_ADC_H

#include "mbed.h"
#define XTAL_FREQ       12000000
#define MAX_ADC_CLOCK   13000000
#define CLKS_PER_SAMPLE 64

class ADC {
public:

            ADC(int sample_rate, int cclk_div);

            void setup(PinName pin, int state);

            int setup(PinName pin);

            void burst(int state);

            void select(PinName pin);
```

```
        int burst(void);

        void startmode(int mode, int edge);

        int startmode(int mode_edge);

        void start(void);

        void interrupt_state(PinName pin, int state);

        int interrupt_state(PinName pin);

        void attach(void(*fptr)(void));

        void detach(void);

        void append(PinName pin, void(*fptr)(uint32_t value));

        void unappend(PinName pin);

        void append(void(*fptr)(int chan, uint32_t value));

        void unappend(void);

        void offset(int offset);

        int offset(void);

        int read(PinName pin);

        int done(PinName pin);

        int overrun(PinName pin);

        int actual_adc_clock(void);

        int actual_sample_rate(void);

        PinName channel_to_pin(int chan);

        int channel_to_pin_number(int chan);

private:
        int _pin_to_channel(PinName pin);
        uint32_t _data_of_pin(PinName pin);

        int _adc_clk_freq;
        void adcisr(void);
        static void _adcisr(void);
        static ADC *instance;

        uint32_t _adc_data[8];
        void(*_adc_isr[8])(uint32_t value);
        void(*_adc_g_isr)(int chan, uint32_t value);
        void(*_adc_m_isr)(void);
};
```

```
#endif
```

1.2 adc.cpp

```
#include "mbed.h"
#include "adc.h"


ADC *ADC::instance;

ADC::ADC(int sample_rate, int cclk_div)
{
            int i, adc_clk_freq, pclk, clock_div, max_div = 1;

            adc_clk_freq = CLKS_PER_SAMPLE*sample_rate;
            int m = (LPC_SC->PLL0CFG & 0xFFFF) + 1;
            int n = (LPC_SC->PLL0CFG >> 16) + 1;
            int cclkdiv = LPC_SC->CCLKCFG + 1;
            int Fcco = (2 * m * XTAL_FREQ) / n;
            int cclk = Fcco / cclkdiv;

            LPC_SC->PCONP |= (1 << 12);
            LPC_SC->PCLKSEL0 &= ~(0x3 << 24);
            switch (cclk_div) {
            case 1:
                        LPC_SC->PCLKSEL0 |= 0x1 << 24;
                        break;
            case 2:
                        LPC_SC->PCLKSEL0 |= 0x2 << 24;
                        break;
            case 4:
                        LPC_SC->PCLKSEL0 |= 0x0 << 24;
                        break;
            case 8:
                        LPC_SC->PCLKSEL0 |= 0x3 << 24;
                        break;
            default:
                        fprintf(stderr, "Warning: ADC CCLK clock divider must be
1, 2, 4 or 8. %u supplied.\n",
                                    cclk_div);
                        fprintf(stderr, "Defaulting to 1.\n");
                        LPC_SC->PCLKSEL0 |= 0x1 << 24;
                        break;
            }
            pclk = cclk / cclk_div;
            clock_div = pclk / adc_clk_freq;

            if (clock_div > 0xFF) {
                        fprintf(stderr, "Warning: Clock division is %u which is
above 255 limit. Re-Setting at limit.\n",
                                    clock_div);
                        clock_div = 0xFF;
            }
            if (clock_div == 0) {
```

```
                        fprintf(stderr, "Warning: Clock division is 0. Re-
Setting to 1.\n");
                        clock_div = 1;
                }

                _adc_clk_freq = pclk / clock_div;
                if (_adc_clk_freq > MAX_ADC_CLOCK) {
                        fprintf(stderr, "Warning: Actual ADC sample rate of %u
which is above %u limit\n",
                                        _adc_clk_freq / CLKS_PER_SAMPLE,
MAX_ADC_CLOCK / CLKS_PER_SAMPLE);
                        while ((pclk / max_div) > MAX_ADC_CLOCK) max_div++;
                        fprintf(stderr, "Maximum recommended sample rate is
%u\n", (pclk / max_div) / CLKS_PER_SAMPLE);
                }

                LPC_ADC->ADCR =
                        ((clock_div - 1) << 8) |
                        (1 << 21);

                LPC_ADC->ADCR &= ~0xFF;
                _adc_g_isr = NULL;
                for (i = 7; i >= 0; i--) {
                        _adc_data[i] = 0;
                        _adc_isr[i] = NULL;
                }

                instance = this;
                NVIC_SetVector(ADC_IRQn, (uint32_t)&_adcisr);

                LPC_ADC->ADINTEN &= ~0x100;

};

void ADC::_adcisr(void)
{
                instance->adcisr();
}


void ADC::adcisr(void)
{
                uint32_t stat;
                int chan;

                stat = LPC_ADC->ADSTAT;
                if (stat & 0x0101) _adc_data[0] = LPC_ADC->ADDR0;
                if (stat & 0x0202) _adc_data[1] = LPC_ADC->ADDR1;
                if (stat & 0x0404) _adc_data[2] = LPC_ADC->ADDR2;
                if (stat & 0x0808) _adc_data[3] = LPC_ADC->ADDR3;
                if (stat & 0x1010) _adc_data[4] = LPC_ADC->ADDR4;
                if (stat & 0x2020) _adc_data[5] = LPC_ADC->ADDR5;
                if (stat & 0x4040) _adc_data[6] = LPC_ADC->ADDR6;
                if (stat & 0x8080) _adc_data[7] = LPC_ADC->ADDR7;

                chan = (LPC_ADC->ADGDR >> 24) & 0x07;
                if (_adc_isr[chan] != NULL)
                        _adc_isr[chan](_adc_data[chan]);
                if (_adc_g_isr != NULL)
                        _adc_g_isr(chan, _adc_data[chan]);
```

```
            return;
}

int ADC::_pin_to_channel(PinName pin) {
            int chan;
            switch (pin) {
            case p15:
            default:
                        chan = 0;
                        break;
            case p16:
                        chan = 1;
                        break;
            case p17:
                        chan = 2;
                        break;
            case p18:
                        chan = 3;
                        break;
            case p19:
                        chan = 4;
                        break;
            case p20:
                        chan = 5;
                        break;
            }
            return(chan);
}

PinName ADC::channel_to_pin(int chan) {
            const PinName pin[8] = { p15, p16, p17, p18, p19, p20, p15, p15 };

            if ((chan < 0) || (chan > 5))
                        fprintf(stderr, "ADC channel %u is outside range availa-
ble to MBED pins.\n", chan);
            return(pin[chan & 0x07]);
}


int ADC::channel_to_pin_number(int chan) {
            const int pin[8] = { 15, 16, 17, 18, 19, 20, 0, 0 };

            if ((chan < 0) || (chan > 5))
                        fprintf(stderr, "ADC channel %u is outside range availa-
ble to MBED pins.\n", chan);
            return(pin[chan & 0x07]);
}


uint32_t ADC::_data_of_pin(PinName pin) {
            if (burst() && (LPC_ADC->ADINTEN & 0x3F)) {
                        return(_adc_data[_pin_to_channel(pin)]);
            }
            else {
                        switch (pin) {
                        case p15:
                        default:
                                    return(LPC_ADC->ADINTEN & 0x01 ?
_adc_data[0] : LPC_ADC->ADDR0);
                        case p16:
```

```
                                        return(LPC_ADC->ADINTEN & 0x02 ?
_adc_data[1] : LPC_ADC->ADDR1);
                        case p17:
                                        return(LPC_ADC->ADINTEN & 0x04 ?
_adc_data[2] : LPC_ADC->ADDR2);
                        case p18:
                                        return(LPC_ADC->ADINTEN & 0x08 ?
_adc_data[3] : LPC_ADC->ADDR3);
                        case p19:
                                        return(LPC_ADC->ADINTEN & 0x10 ?
_adc_data[4] : LPC_ADC->ADDR4);
                        case p20://
                                        return(LPC_ADC->ADINTEN & 0x20 ?
_adc_data[5] : LPC_ADC->ADDR5);
                        }
            }
}

void ADC::setup(PinName pin, int state) {
            int chan;
            chan = _pin_to_channel(pin);
            if ((state & 1) == 1) {
            switch (pin) {
            case p15:
            default:
                        LPC_PINCON->PINSEL1 &= ~((unsigned int)0x3 << 14);
                        LPC_PINCON->PINSEL1 |= (unsigned int)0x1 << 14;
                        LPC_PINCON->PINMODE1 &= ~((unsigned int)0x3 << 14);
                        LPC_PINCON->PINMODE1 |= (unsigned int)0x2 << 14;
                        break;
            case p16:
                        LPC_PINCON->PINSEL1 &= ~((unsigned int)0x3 << 16);
                        LPC_PINCON->PINSEL1 |= (unsigned int)0x1 << 16;
                        LPC_PINCON->PINMODE1 &= ~((unsigned int)0x3 << 16);
                        LPC_PINCON->PINMODE1 |= (unsigned int)0x2 << 16;
                        break;
            case p17:
                        LPC_PINCON->PINSEL1 &= ~((unsigned int)0x3 << 18);
                        LPC_PINCON->PINSEL1 |= (unsigned int)0x1 << 18;
                        LPC_PINCON->PINMODE1 &= ~((unsigned int)0x3 << 18);
                        LPC_PINCON->PINMODE1 |= (unsigned int)0x2 << 18;
                        break;
            case p18:
                        LPC_PINCON->PINSEL1 &= ~((unsigned int)0x3 << 20);
                        LPC_PINCON->PINSEL1 |= (unsigned int)0x1 << 20;
                        LPC_PINCON->PINMODE1 &= ~((unsigned int)0x3 << 20);
                        LPC_PINCON->PINMODE1 |= (unsigned int)0x2 << 20;
                        break;
            case p19:
                        LPC_PINCON->PINSEL3 &= ~((unsigned int)0x3 << 28);
                        LPC_PINCON->PINSEL3 |= (unsigned int)0x3 << 28;
                        LPC_PINCON->PINMODE3 &= ~((unsigned int)0x3 << 28);
                        LPC_PINCON->PINMODE3 |= (unsigned int)0x2 << 28;
                        break;
            case p20:
                        LPC_PINCON->PINSEL3 &= ~((unsigned int)0x3 << 30);
                        LPC_PINCON->PINSEL3 |= (unsigned int)0x3 << 30;
                        LPC_PINCON->PINMODE3 &= ~((unsigned int)0x3 << 30);
                        LPC_PINCON->PINMODE3 |= (unsigned int)0x2 << 30;
                        break;
```

```
                    }
            if (!burst()) LPC_ADC->ADCR &= ~0xFF;
            LPC_ADC->ADCR |= (1 << chan);
            }
else {
            switch (pin) {
            case p15:
            default:
                        LPC_PINCON->PINSEL1 &= ~((unsigned int)0x3 << 14);
                        LPC_PINCON->PINMODE1 &= ~((unsigned int)0x3 << 14);
                        break;
            case p16:
                        LPC_PINCON->PINSEL1 &= ~((unsigned int)0x3 << 16);
                        LPC_PINCON->PINMODE1 &= ~((unsigned int)0x3 << 16);
                        break;
            case p17:
                        LPC_PINCON->PINSEL1 &= ~((unsigned int)0x3 << 18);
                        LPC_PINCON->PINMODE1 &= ~((unsigned int)0x3 << 18);
                        break;
            case p18:
                        LPC_PINCON->PINSEL1 &= ~((unsigned int)0x3 << 20);
                        LPC_PINCON->PINMODE1 &= ~((unsigned int)0x3 << 20);
                        break;
            case p19:
                        LPC_PINCON->PINSEL3 &= ~((unsigned int)0x3 << 28);
                        LPC_PINCON->PINMODE3 &= ~((unsigned int)0x3 << 28);
                        break;
            case p20:
                        LPC_PINCON->PINSEL3 &= ~((unsigned int)0x3 << 30);
                        LPC_PINCON->PINMODE3 &= ~((unsigned int)0x3 << 30);
                        break;
                        }
                        LPC_ADC->ADCR &= ~(1 << chan);
            }
}

int ADC::setup(PinName pin) {
            int chan;

            chan = _pin_to_channel(pin);
            return((LPC_ADC->ADCR & (1 << chan)) >> chan);
}

void ADC::select(PinName pin) {
            int chan;

            if (!burst()) LPC_ADC->ADCR &= ~0xFF;
            chan = _pin_to_channel(pin);
            LPC_ADC->ADCR |= (1 << chan);
}

void ADC::burst(int state) {
            if ((state & 1) == 1) {
                        if (startmode(0) != 0)
                                    fprintf(stderr, "Warning. startmode is %u.
Must be 0 for burst mode.\n", startmode(0));
                        LPC_ADC->ADCR |= (1 << 16);
            }
            else
                        LPC_ADC->ADCR &= ~(1 << 16);
```

```
}

int  ADC::burst(void) {
            return((LPC_ADC->ADCR & (1 << 16)) >> 16);
}

void ADC::startmode(int mode, int edge) {
            int lpc_adc_temp;

            lpc_adc_temp = LPC_ADC->ADCR & ~(0x0F << 24);
            lpc_adc_temp |= ((mode & 7) << 24) | ((edge & 1) << 27);
            LPC_ADC->ADCR = lpc_adc_temp;
}

int ADC::startmode(int mode_edge){
            switch (mode_edge) {
            case 0:
            default:
                        return((LPC_ADC->ADCR >> 24) & 0x07);
            case 1:
                        return((LPC_ADC->ADCR >> 27) & 0x01);
            }
}

void ADC::start(void) {
            startmode(1, 0);
}

void ADC::interrupt_state(PinName pin, int state) {
            int chan;

            chan = _pin_to_channel(pin);
            if (state == 1) {
                        LPC_ADC->ADINTEN &= ~0x100;
                        LPC_ADC->ADINTEN |= 1 << chan;
                        NVIC_EnableIRQ(ADC_IRQn);
            }
            else {
                        LPC_ADC->ADINTEN &= ~(1 << chan);
                        if ((LPC_ADC->ADINTEN & 0xFF) == 0)
                                    NVIC_DisableIRQ(ADC_IRQn);
            }
}

int ADC::interrupt_state(PinName pin) {
            int chan;

            chan = _pin_to_channel(pin);
            return((LPC_ADC->ADINTEN >> chan) & 0x01);
}

void ADC::attach(void(*fptr)(void)) {
            NVIC_SetVector(ADC_IRQn, (uint32_t)fptr);
}

void ADC::detach(void) {
            instance = this;
            NVIC_SetVector(ADC_IRQn, (uint32_t)&_adcisr);
}
```

```
void ADC::append(PinName pin, void(*fptr)(uint32_t value)) {
            int chan;

            chan = _pin_to_channel(pin);
            _adc_isr[chan] = fptr;
}

void ADC::unappend(PinName pin) {
            int chan;

            chan = _pin_to_channel(pin);
            _adc_isr[chan] = NULL;
}

void ADC::append(void(*fptr)(int chan, uint32_t value)) {
            _adc_g_isr = fptr;
}

void ADC::unappend() {
            _adc_g_isr = NULL;
}

void offset(int offset) {
            LPC_ADC->ADTRM &= ~(0x07 << 4);
            LPC_ADC->ADTRM |= (offset & 0x07) << 4;
}

int offset(void) {
            return((LPC_ADC->ADTRM >> 4) & 0x07);
}

int ADC::read(PinName pin) {
            _adc_data[_pin_to_channel(pin)] &= ~(((uint32_t)0x01 << 31) |
((uint32_t)0x01 << 30));
            return((_data_of_pin(pin) >> 4) & 0xFFF);
}

int ADC::done(PinName pin) {
            return((_data_of_pin(pin) >> 31) & 0x01);
}

int ADC::overrun(PinName pin) {
            return((_data_of_pin(pin) >> 30) & 0x01);
}

int ADC::actual_adc_clock(void) {
            return(_adc_clk_freq);
}

int ADC::actual_sample_rate(void) {
            return(_adc_clk_freq / CLKS_PER_SAMPLE);


}
```