Antti Jumpponen

OHJELMARADION SPEKTRIANALYSAATTORI

Insinöörityö
Kajaanin ammattikorkeakoulu
Tekniikan ja liikenteen ala
Tietotekniikan koulutusohjelma
Kevät 2008

Kajaanin ammattikorkeakoulu

OPINNÄYTETYÖ
TIIVISTELMÄ

Koulutusala Tekniikan ja liikenteen ala	Koulutusohjelma Tietotekniikka koulutusohjelma
Tekijä(t) Antti Jumpponen	
Työn nimi Ohjelmaradion spektrianalysaattori	
Vaihtoehtoiset ammattiopinnot Konenäkö ja mittaustekniikka	Ohjaaja(t) Jukka Heino
	Toimeksiantaja Jukka Heino
Aika Kevät 2008	Sivumäärä ja liitteet 40 + 3

Tämän insinöörityön tarkoituksena oli ohjelmoida spektrianalysaattori ohjelmaradiolle. Insinöörityön tilaajana toimi Jukka Heino Kajaanin ammattikorkeakoulusta.

Ohjelmaradio on tapa kuunnella radiota käyttäen tietokonetta radion toiminnan määrittämiseen. Työn tavoitteena oli rakentaa ohjelma, joka vastaanottaa radion ääniaaltoa käyttäjän ohjelmassa määräämältä taajuudelta ja piirtää tietokoneen ruudulle ääniaaltosignaalin spektrin. Ohjelma on kirjoitettu C-kielellä käyttäen Windowsin ohjelmointikirjastoja ikkuna-funktioissa ja piirrossa.

Työn teoriaosuudessa käsitellään yleisesti spektrianalysaattoreita ja niihin liittyvää tekniikkaa. Ohjelmointiosuudessa tarkastellaan perustaajuuden asettamista, äänikortilta lukemista, nopean Fourier-muunnoksen suorittamista ja Windows-ikkunalle piirtämistä.

Toteutettu ohjelma täyttää työlle asetetut vaatimukset. Se toimii samalla mallina siitä, miten hoidetaan ikkunafunktiot ja piirto ilman, että käytetään korkeampia kieliä tai kirjastoja.

Kieli	Suomi
Asiasanat	FFT, DFT, SDR, kello-oskillaattori
Säilytyspaikka	☒ Kajaanin ammattikorkeakoulun Kaktus-tietokanta ☒ Kajaanin ammattikorkeakoulun kirjasto

Kajaanin ammattikorkeakoulu
University of Applied Sciences

THESIS
ABSTRACT

School School of Engineering	Degree Programme Information Technology
Author(s) Antti Jumpponen	
Title A Spectrum Analyzer for a Software Defined Radio	
Optional Professional Studies Machine Vision and Measurement Technology	Instructor(s) Jukka Heino
	Commissioned by Jukka Heino
Date Spring 2008	Total Number of Pages and Appendices 40 + 3

The purpose of this Bachelor's thesis was to program a spectrum analyzer for a software defined radio. This thesis was commissioned by Jukka Heino from the Kajaani University of Applied Sciences.

The Elektror magazine contained working instructions for a software defined radio circuit. The finished circuit needs a program, which enables the user to feed the counter values of the clock-oscillator and therefore get the required output frequency. The program also needs to utilize fast Fourier transform for an array of data which was read from the audio card of the computer and draw the signal spectrum using the transformed data.

The program was written using the C language with Windows application programming interface and graphics device interface providing window functions and drawing functions, respectively.

The thesis provides the reader common information and specifications in relation to the spectrum analyzers. The programming part shows the reader how to program clock oscillator, read audio card, utilize fast Fourier transform and draw on window.

Language of Thesis	Finnish
Keywords	FFT, DFT, SDR, Clock Oscillator
Deposited at	☒ Kaktus Database at Kajaani University of Applied Sciences ☒ Library of Kajaani University of Applied Sciences

LYHENTEET

ADC	Analog-Digital Converter
AM	Amplitude Modulation
DAC	Digital-Analog Converter
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
FM	Frequency Modulation
GPS	Global Positioning System
I²C	Inter-Integrated Circuit
IF	Intermediate Frequency
PCM	Pulse Code Modulation
PLL	Phase-locked Loop
RAM	Random Access Memory
RBW	Resolution Bandwidth
RF	Radio Frequency
SDR	Software Defined Radio
SPI	Serial Programming Interface
USB	Universal Serial Bus
VCO	Voltage Controlled Oscillator

SISÄLLYS

1 JOHDANTO

Tämän insinöörityön tarkoituksena oli ohjelmoida spektrianalysaattori ohjelmaradiolle. Insinöörityön tilaajana toimi Jukka Heino Kajaanin ammattikorkeakoulusta.

Ohjelmistopohjainen radio on tapa kuunnella radiotaajuuksia niin, että suurin osa radion ominaisuuksista on luotu ohjelmallisesti. Antennia ei voi nykyteknologialla kytkeä suoraan tietokoneen A/D-muuntimeen, vaan se vaatii lisäksi erillisen radiotaajuuden vastaanottokytkennän. Tässä työssä käytetään Elektror-lehden ohjelmaradiokytkentää, joka koostuu pääasiassa USB-sarjaliikennemuuntimesta, kello-oskillaattorista ja superheterodyne-vastaanottimesta.

Tämän työn tarkoituksena on rakentaa toimiva ohjelma, joka vastaanottaa radioääniaaltoa käyttäjän ohjelmassa määräämältä taajuudelta ja piirtää tietokoneen ruudulle ääniaaltosignaalin spektrin. Tähän päästäkseen ohjelman on kyettävä asettamaan kytkennän kello-oskillaattorin laskurinarvot oikeiksi halutun lähtötaajuuden aikaansaamiseksi, lukea äänikorttia, suorittaa taulukolliselle äänitietoa nopea Fourier-muunnos ja piirtää muunnoksen aikaan saamasta tiedosta spektri.

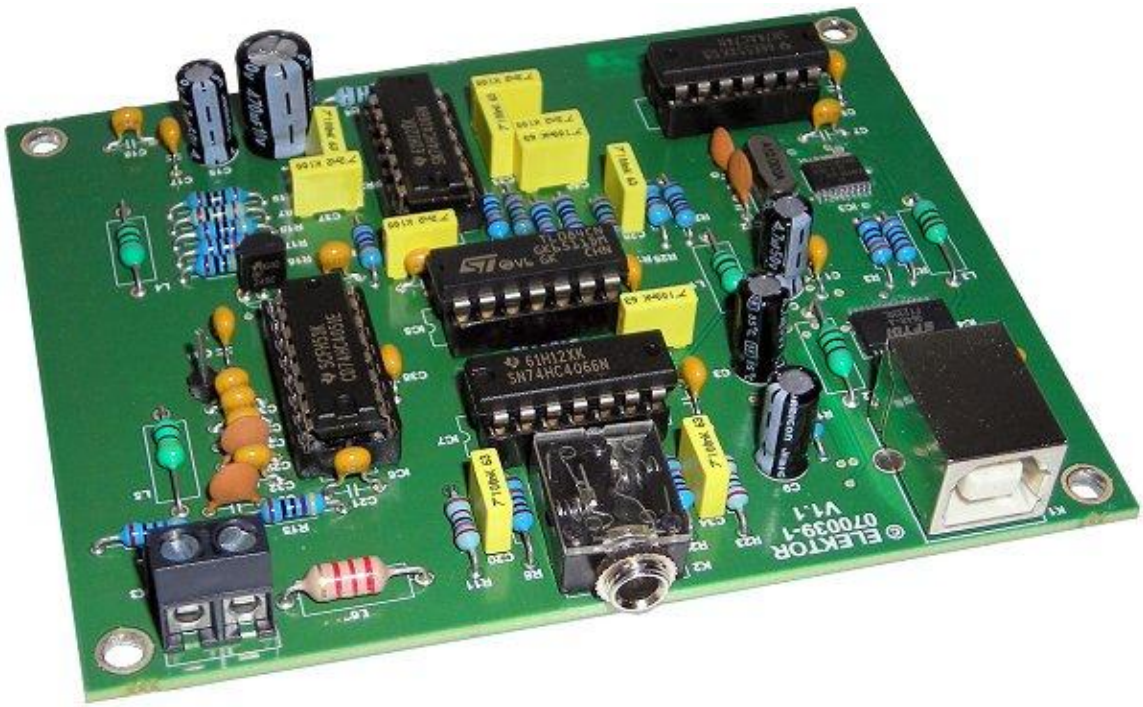
Ohjelmointi suoritetaan C-kielellä. Ikkunointi hoidetaan käyttämällä Windows API-kirjastoa ja piirto GDI-kirjastolla. Tämä on Assembly-ohjelmoinnin jälkeen alin tapa ohjelmoida Windows-käyttöjärjestelmää. Ohjelmointiympäristönä on Visual Studio 2005.

Työn teoriaosuudessa käsitellään yleisesti spektrianalysaattoreita ja niihin liittyvää tekniikkaa. Ohjelmointiosuudessa tarkastellaan perustaajuuden asettamista, äänikortin lukemista, nopean Fourier-muunnoksen suorittamista ja Windows-ikkunalle piirtämistä.

Lopputuloksena on ohjelma, joka täyttää vaaditut ehdot. Ohjelman koodi toimii myös mallinna siitä, miten hoidetaan ikkunafunktiot ja piirto ilman, että käytetään korkeampia kieliä (Visual Basic, Delphi) tai kirjastoja (MFC). Ohjelman spektrin piirtoa voidaan hyödyntää myös muulloin, kuin ohjelmaradion yhteydessä. Ohjelma vaatii ainoastaan taulukollisen näytteitä ja nämä näytteet voivat hyvin olla mistä tahansa peräisin.

Ohjelmoinnissa on hyödynnetty valmista koodia. Elektror-lehden mukana tuli Pascal-kielellä ohjelmoitu kello-oskillaattorin taajuuden syöttöohjelma. Tässä työssä hyödynnetään kyseisen ohjelman pohjakoodirakennetta. Nopeaan Fourier-muunnokseen sain Kajaanin ammattikorkeakoulun lehtori Jukka Heinolta Qbasic-kielisen metodin.

Työssä käytetään vastaanotinta, joka pystyy vastaanottamaan kaikkia kaistoja 150 kilohertsistä 30 megahertsiin. Vastaanotin on optimoitu vastaanottamaan DRM- ja AM-lähetyksiä, mutta sopii myös amatööriradion kuunteluun. Ohjelma vaatii Windows XP -käyttöjärjestelmän, ohjelmaradioon sopivan äänikortin ja USB-sarjaliikennemuuntimen ajurit. Kuvassa 1 on esitelty käytetty ohjelmaradiokytkentä.



Kuva 1. Elektor-lehden SDR-kytkentä [1.]

2 OHJELMAPOHJAINEN RADIO

Ohjelmapohjaisessa radiossa (SDR) ideana on käyttää pienintä mahdollista laitteistoa ja luoda suurin osa radion toiminnasta ohjelmallisesti. Laitteistoon kuuluu minimiosista rakennettu piiri ja kaksikanavaisella äänikortilla varustettu tietokone. [2.]

Radiolla tarkoitetaan laitetta, joka voi lähettää tai vastaanottaa elektromagneettispektrin radiotaajuudella. Näitä ovat AM- ja FM-radiot, TV, matkapuhelimet, johdottomat puhelimet, WiFI, GPS, jne. [3.]

Ohjelmaradiot käyttävät digitaalista signaalin käsittelyä (DSP) signaalien suodatukseen, modulaatioon ja demodulaatioon. DSP pystyy tekemään kaiken, minkä analoginen radio samalla mahdollistaen asioita, jotka olisivat vaikeita tai mahdottomia analogisen radion kanssa. Esimerkiksi sekä analogia että ohjelmistoradiot omaavat vastaanotinsuodattimia vaihtelevalle kaistanleveydelle. Laitteistoon pohjautuvissa radioissa tämä tehdään kidesuodattimella tai mekaanisella resonaattorilla jokaiselle kaistanleveydelle. Ohjelmaradiossa käytetään matemaattista algoritmia, jossa muuttujien arvoja vaihtamalla saadaan aikaiseksi haluttu kaistanleveys ja muoto. [4.]

Ohjelmistoradion etuja normaaliin radioon verrattuna ovat:

- Muuttamalla tai uusimalla ohjelmistoa voidaan muuttaa toiminnallisuutta. Tämän ansiosta on helppo lisätä uusia toimintamuotoja ja parantaa suorituskykyä ilman rautatason muutoksia. [2.]
- Ohjelmallinen päivitys on halvempaa kuin raudan johtuen siitä, että komponentteihin ei tarvitse sijoittaa rahaa.
- SDR tarvitsee laitteistokseen tietokoneen ja siihen äänikortin. Tämän lisäksi tarvitaan pieni laitteisto, jossa on vähimmillään kolme integroitua piiriä ja Tayloe-havaitsija. [2.]
- Kanavia, joihin puhutaan tai kuunnellaan, voi olla useampia. [3.]
- Mahdollisuus rakentaa uudenlainen radio. Esimerkiksi älykäs radio, joka osaa tutkia lähiseudun RF-spektriä ja ohjelmoida itsensä paremmaksi. [3.]

Ensimmäiseksi ohjelmistoradiota alkoi käyttää Yhdysvaltojen armeija SpeakEasy-projektissa vuonna 1992. Myös Suomen armeija on aloittanut ohjelmistoradion käytön. [5.]

Ideaalisessa tapauksessa antenni kytkettäisiin suoraan A/D-muuntimeen ja ohjelmallisesti muokattaisiin tietoa omaan tarpeeseen. Nykyisen teknologian puitteissa tämä ei ole mahdollista. Yli 40 MHz:n taajuuksille ongelmia tuottaa A/D-muuntimen hitaus. Antennin ja A/D-muuntimen välissä käytetään superheterodyne RF-etuosaa, joka koostuu taajuussekoittimesta ja referenssioskillaattorista. [6.]

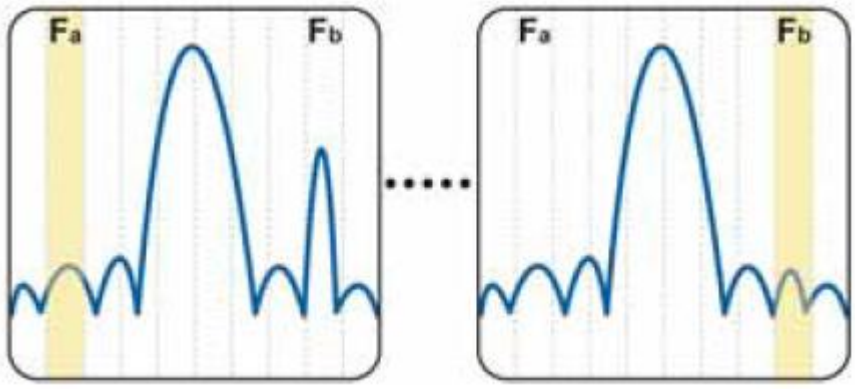
3 SPEKTRIANALYSAATTORITYYPIT

3.1 Pyyhkäisevä spektrianalysaattori

Pyyhkäisevä spektrianalysaattori (*Swept Spectrum Analyzer*) on superheterodyne-vastaanottimeen perustuva, ensimmäinen taajuustasomittauksia mahdollistanut analysaattori. Se on tarkoitettu mittaamaan tarkasti ohjattuja, staattisia signaaleita. Alun perin pyyhkäisevä spektrianalysaattori rakennettiin täysin analogisista osista mutta nykyään se sisältää digitaalisia komponentteja, kuten A/D-muuntimia ja mikroprosessoreita. [7.]

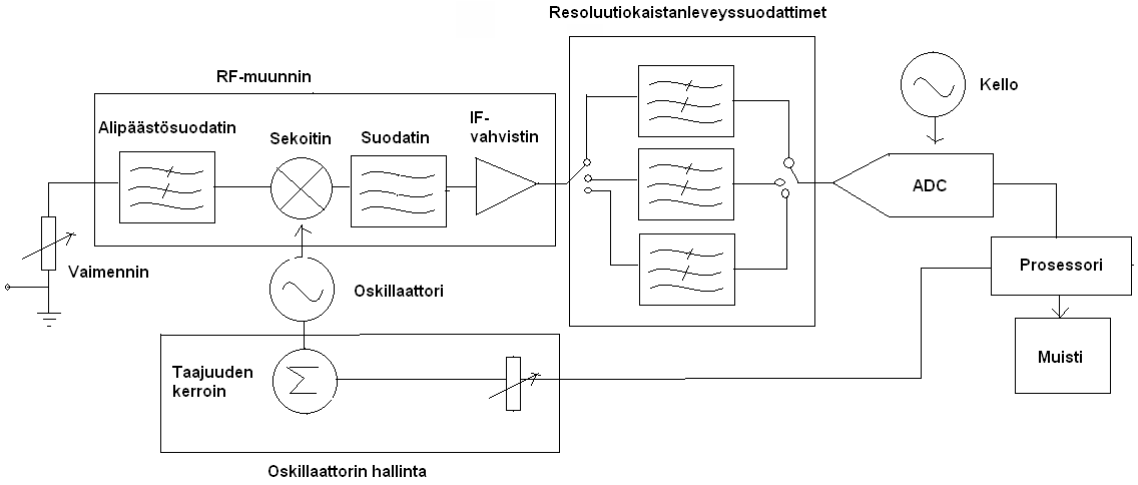
Analysaattori muuntaa signaalin keskitaajuudelle ja vie sen läpi resoluutiokaistanleveyssuodattimen (RBW). Suodatinta seuraa havaitsija, joka laskee amplitudin jokaiselle taajuuspisteelle valitulla pyyhkäisyalueella. Tällä saadaan aikaiseksi korkea dynaaminen alue, mutta haittana on vain yhden taajuuspisteen laskeminen kerralla. Pyyhkäisyalueen analyysi saattaa kestää pahimmillaan sekunteja. Signaali ei saa pyyhkäisyn aikana muuttua suuresti. Tilastollisesti ja käytännössä on osoitettu, että liian rankalla signaalinmuutoksella muutos saattaa jäädä huomioimatta. [7.]

Kuvassa 2 on esitetty tilanne, jossa pyyhkäisy etsii taajuutta segmentistä *Fa,* minkä aikana hetkellinen spektraalinen piikki tapahtuu *Fb*:lle. Kun pyyhkäisy saapuu *Fb*:lle, on tapahtuma jo ehtinyt kadota, eikä sitä huomata pyyhkäisyssä. [7.]



Kuva 2. Hetkellinen spektraalinen ilmiö jää havaitsematta pyyhkäisyssä [7.]

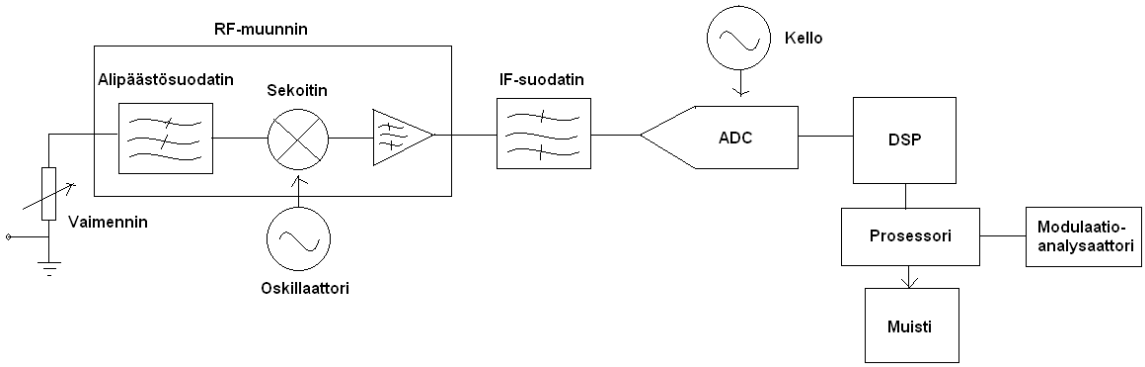
Kuvassa 3 on tyypillinen nykyaikainen pyyhkäisevän spektrianalysaattorin lohkokaavio. Ennen A/D-muunninta on suodatus, sekoitus ja vahvistus, jotka ovat analogisia prosesseja kaistanleveyksille BW1, BW2 ja BW3. Jos tarvitaan vieläkin kapeampia suodattimia, on turvauduttava digitaaliseen signaalinkäsittelyyn. [7.]



Kuva 3. Nykyaikainen pyyhkäisyyn pohjautuva spektrianalysaattori

3.2 Vektorisignaalianalysaattori

Perinteinen pyyhkäisyyn pohjautuva spektrianalyysi mahdollistaa skalaarimittaukset, jotka antavat tietoa vain tulosignaalin suuruudesta. Digitaalista modulaatiota sisältävien signaalien analysointi vaatii vektorimittauksia, jotka antavat sekä amplitudin että vaiheen. Tämä analysaattori on tarkoitettu nimenomaan digitaalisen modulaation analysoimiseen. Analysaattorin lohkokaavio on esitetty kuvassa 4. [7.]



Kuva 4. Tyypillisen vektorisignaalianalysaattorin lohkokaavio

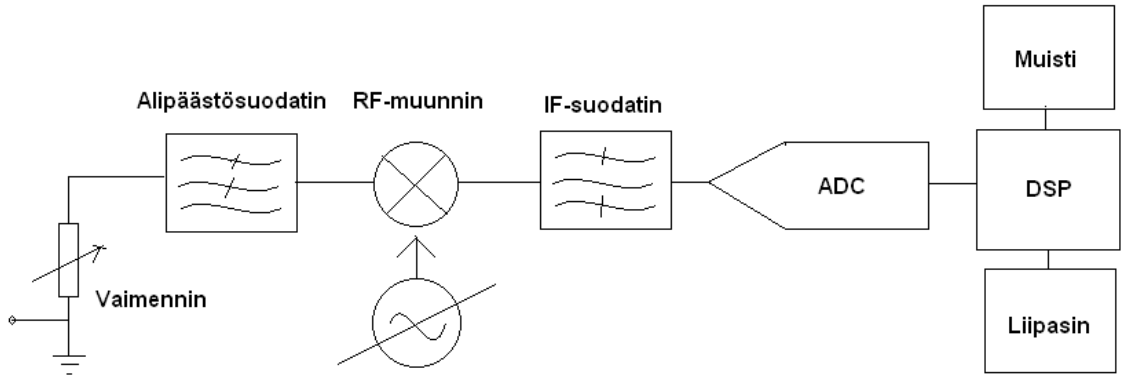
Analysaattorissa digitisoidaan kaistanpäästön sisällä oleva RF-taajuus, josta saadaan amplitudi- ja vaihetiedot digitaalisen modulaation mittaamiseen. Analysaattorin rakenne on suunniteltu ottamaan tulosignaalista tilannekuvia tietyn pistein välein, mistä seuraa vaikeus tai mahdottomuus tallentaa pitkiä tietoja vastaanotosta. Tämän takia tietoa signaalin käyttäytymisestä ajan kanssa ei juurikaan saada. Kuten pyyhkäisyanalysaattorin kanssa, liipaisumahdollisuudet on tyypillisesti rajattu IF-tason liipaisuun ja ulkoiseen liipaisuun. [7.]

Analysaattorin sisällä A/D-muunnin digitisoi IF-signaalin ja suorittaa taajuuden alennusmuunnoksen. Suodatus ja vastaanotto tapahtuvat numeerisesti. Nopeaa Fourier-muunnosta käytetään siirtymiseen aikatasosta taajuustasoon. Analysaattorin kannalta kriittisintä on A/D-muuntimen lineaarisuus ja dynaaminen alue. Lisäksi nopea mittaus vaatii riittävästi prosessorivoimaa digitaalisen signaalinkäsittelyn suorittamiselle. [7.]

Tämän analysaattorin modulaatiomittausparametrit, kuten virhevektorisuuruus (EVM, *error Vector Magnitude*), mahdollistavat muita esitysmuotoja esim. tähtikuviodiagrammin. Vektorisignaalianalysaattoria käytetään yleensä pyyhkäisysignaalianalysaattorin kanssa. [7.]

3.3 Reaaliaikaspektrianalysaattori

Reaaliaikaspektrianalysaattori (*Realtime Spectral Analyzer*) on yleinen ratkaisu hetkellisten ja dynaamisten RF-signaalien tuomaan ongelmaan. Perusajatus on mahdollisuus liipaista RF-signaalin kohdalla, tallentaa se muistiin ja analysoida sitä monessa tasossa. Tämä tekee mahdolliseksi luotettavasti havaita ja ilmentää RF-signaalit, jotka muuttuvat ajan kanssa. [7.]



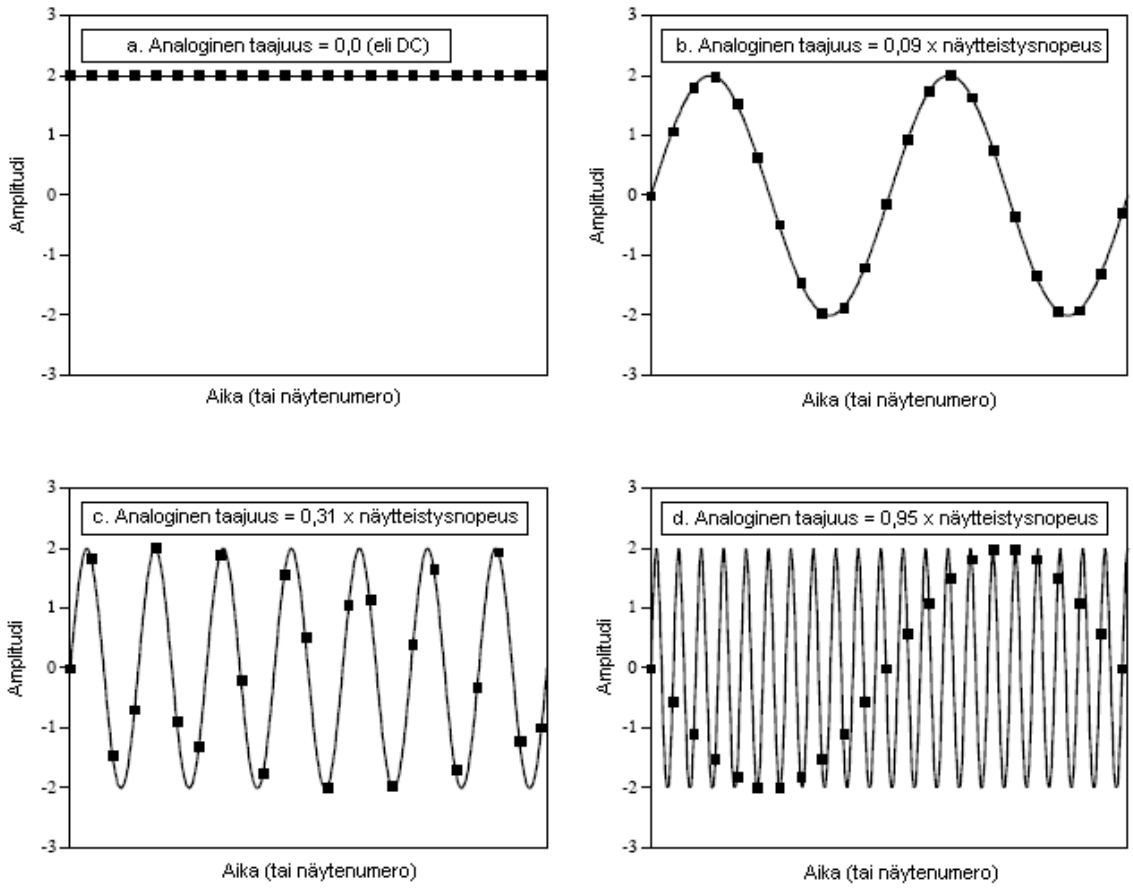
Kuva 5. Reaaliaikaspektrianalysaattorin arkkitehtuuri

Kuvassa 5 on yksinkertaistettu lohkokaavio analysaattorin toiminnasta. RF-etuosalla alennetaan tulosignaali IF-taajuudelle. Tämän jälkeen signaali suodatetaan, digitisoidaan A/D-muuntimella ja viedään lopuksi digitaaliseen signaalinkäsittelyyn, joka hoitaa analysaattorin liipaisun, muistin ja analyysifunktiot. Reaaliaikaspektrianalysaattori on tarkoitettu reaaliaikaiseen liipaisuun. Nykyään A/D-muuntimen kehitys on mahdollistanut muunnoksen laajalla dynaamisella alueella ja alhaisella kohinalla. [7.]

4 NÄYTTEISTYSTEORIA

Nyquistin näytteenottoteoreema sanoo, että kaistarajoitettu jatkuva-aikainen signaali voidaan näytteistää onnistuneesti, jos näytteistysnopeus on kaksi kertaa suurempi kuin suurin taajuus näytteistettävässä kaistarajoitetussa signaalissa. Esimerkiksi 1000 Hz:n taajuuden näytteistäminen vaatii 2000 Hz:n näytteistysnopeuden. [8.]

Näytteistys on onnistunut, jos näytteistä voidaan palauttaa takaisin vääristymätön signaali. Kuvassa 6 näkyy sinisignaaleja ennen ja jälkeen digitisoinnin. Jatkuva viiva merkitsee A/D-muuntimelle menevää analogiasignaalia ja neliöt muuntimelta ulostulevaa digitaalisignaalia. [8.]



Kuva 6. Näytteistysnopeus eri taajuuksilla [8.]

Kohdassa a analogiasignaali on pysyvä DC arvo, 0-taajuuden kosiniaalto. Koska analogiasignaali on sarja suoria viivoja näytteiden välissä, kaikki tieto pystytään rakentamaan uudelleen. [8.]

Kohdassa b olevan siniaallon taajuus on 0,09 näytteistysnopeudesta. Tämä voidaan esittää esimerkiksi siniaaltona, jossa on 90 jaksoa sekunnissa ja jota näytteistetään 100 näytettä sekunnissa. Tämä tarkoittaa 11 näytettä jokaisen jakson aikana eli riittävää näytteistystä. [8.]

Kohdassa c siniaallon suhde näytteistysnopeuteen on 0,31. Tästä seuraa 3,2 näytettä per jakso. Kuvassa näytteet eivät näytä seuraavan siniaaltoa johtuen niiden etäisyydestä toisiinsa. Tämäkin signaali on riittävästi näytteistetty. [8.]

Kohdassa d analogiataajuuden suhde näytteistykseen on 0,95 eli 1,05 näytettä per sinijakso. Näistä näytteistä muodostuu erilainen siniaalto, kuin syötetty analogiasignaali. Alkuperäinen 0,95-suhde näkyy 0,05-suhteena digitaalisessa signaalissa. Siniaallon taajuuden muutosta näytteistyksen aikana kutsutaan laskostumiseksi. Koska digitoitu tieto ei ole enää ainutlaatuinen suhteessa tiettyyn analogiasignaaliin, signaalin takaisin rakentaminen voi tapahtua monella eri tapaa. [8.]

5 FOURIER-MUUNNOS

Fourier-muunnoksen avulla mikä tahansa jatkuva-aikainen signaali voidaan esittää siniaaltojen summana. Fourier-analyysi on matemaattisten tekniikoiden joukko, jossa kaikki perustuu siniaallon paloittelemiseen osiksi. Siniaalloilla on hyödyllistä se, että sisään syötetty siniaalto takaa myös siniaallon lähtöön. Ainoastaan signaalin amplitudi tai vaihe voi muuttua. Taajuus ja muoto säilyvät samanlaisena. [8.]

Fourier-muunnos voidaan jakaa neljään kategoriaan riippuen signaalin tyypistä. Signaali voi olla jatkuva-aikainen tai diskreetti, ja se voi olla jaksollinen tai jaksoton. Taulukossa 1 on esitelty Fourier-analyysimuodot. [8.]

Taulukko 1. Fourier-analyysimuodon käytöt

Muunnos	**Esimerkkisignaali**
Fourier-muunnos Signaalit ovat jatkuvia ja jaksottomia.	
Fourier-sarja Signaalit ovat jatkuvia ja jaksollisia.	
Diskreettiaikainen Fourier-muunnos (DTFT) Signaalit ovat diskreettejä ja jaksottomia.	
Diskreetti Fourier-muunnos (DFT) Signaalit ovat diskreettejä ja jaksollisia.	

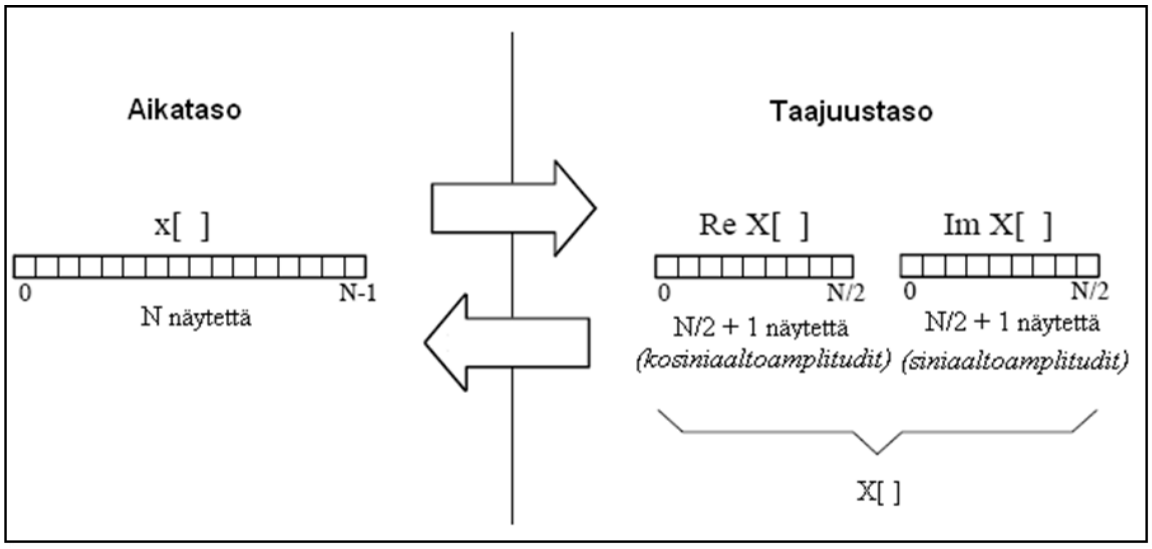
Jokainen Fourier-muunnos voidaan jakaa reaali- ja kompleksiseen-versioon. Reaali-versio on yksinkertaisin, käyttäen tavallisia numeroita ja algebraa syntesointiin ja osiin jakoon. Kompleksisessa versiossa käytetään kompleksinumeroita. [8.]

Diskreettiä Fourier-muunnosta käytetään digitoitujen signaalien kanssa.

5.1 Diskreetti Fourier-muunnos

Kuva 7 esittää, miten diskreetti Fourier-muunnos muuttaa N-pisteisen tulosignaalin kahteen N/2 + 1-pisteiseen lähtösignaaliin, jotka sisältävät sini- ja kosiniaallon amplitudiarvot. Tulosignaalin sanotaan olevan aikatasossa ja lähtösignaalien taajuustasossa. [8.]

Taajuustaso sisältää saman tiedon kuin aikataso mutta eri muodossa. Tietämällä jompikumpi taso voidaan laskea toinen. [8.]



Kuva 7. Aikatasosignaalin muutos taajuustasoon [8.]

Aikatasossa olevien näytteiden määrää esittää muuttuja N. N on yleensä kahden potenssi (esim. 128, 256, 512, 1025, jne). Tämä johtuu digitaalisen tiedon käsittelyn binäärisestä olomuodosta ja siitä, että FFT toimii yleensä niin, että N on kahden potenssi. N on yleensä 32:n ja 4096:n väliltä ja näytteet ovat väliltä 0–N-1. [8.]

Jos x[ ]-taulukossa on aikatasojen arvoja N kappaletta, taajuustasossa on kaksi taulukkoa, joiden koko on N/2 + 1. Nämä ovat reaaliosa eli kosiniaaltoamplitudit ReX[ ] ja imaginääriosa eli siniaaltoamplitudit ImX[ ]. [8.]

Vaikka taulukoita kutsutaan reaali- ja imaginääri-osiksi, niillä ei ole mitään tekemistä kompleksilukujen kanssa, kun käytetään reaaliluku DFT:tä. [8.]

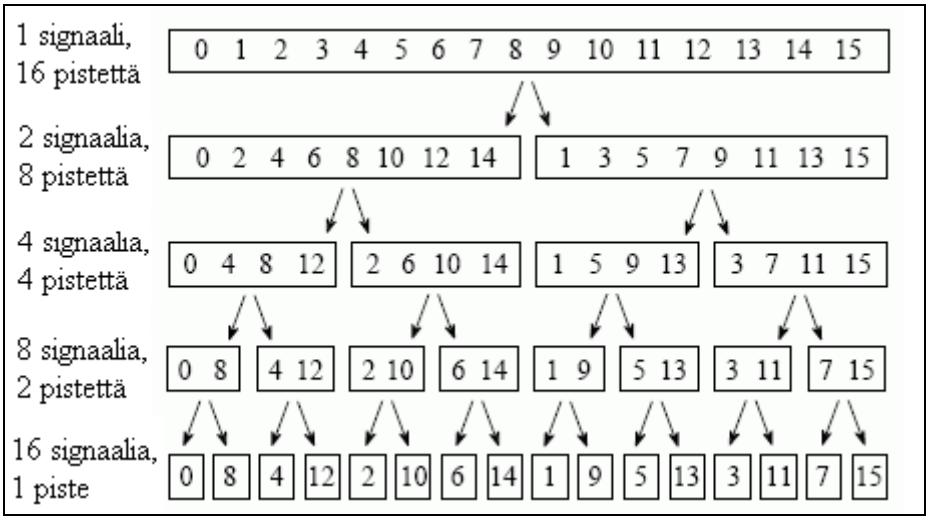
5.2 Nopea Fourier-muunnos

Nopea Fourier-muunnos on tapa laskea diskreetti Fourier-muunnos satoja kertoja nopeammin.

FFT:n laskemiseen tarvitaan kolme vaihetta:

1. Jaetaan N kpl pisteitä omaava aikatasosignaali N osaan, joissa jokaisessa osasignaalissa on yksi piste.
2. Lasketaan N kpl taajuusspektrejä, jotka vastaavat näitä aikatasosignaaleja.
3. N kpl spektrejä syntesoidaan yhdeksi taajuusspektriksi.

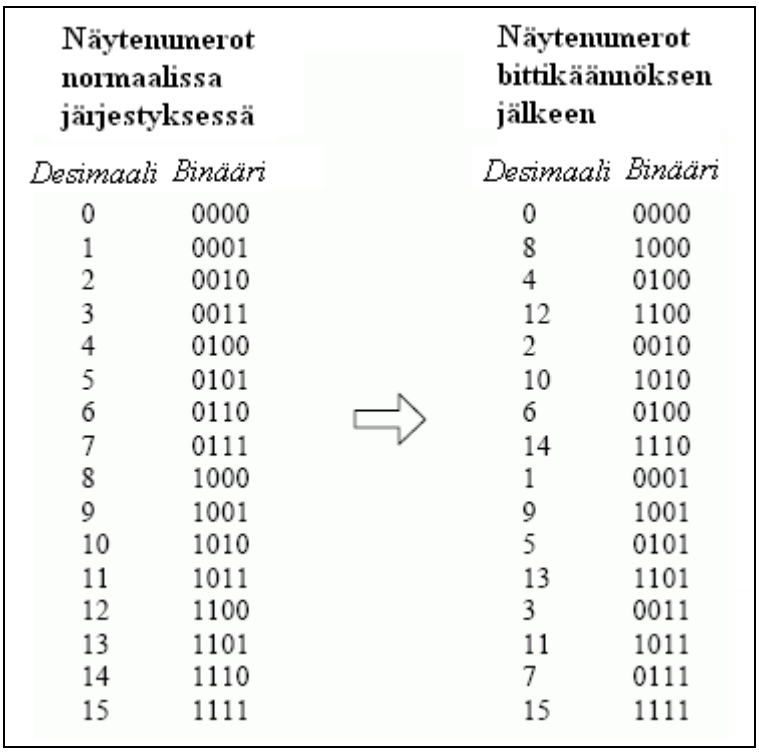
Kuvassa 8 on esitetty signaalin jakaminen N osaan. Siinä 16-pisteinen signaali muutetaan 16 signaaliin, joissa jokaisessa on yksi piste. [8.]



Kuva 8. N-pisteisen signaalin jakaminen osasignaaleihin [8.]

Osiin jakaminen toteutetaan lomittaen parilliset ja parittomat erilleen. Tasojen määrä saadaan $log_2 N$-säännöstä. 16-pisteinen signaali tarvitsee neljä tasoa, koska 2^4 on 16. 4096-pisteinen signaali tarvitsee 12 tasoa (2^{12}). Potenssiluku on tasojen määrä. [8.]

Osiin jakoa voidaan yksinkertaistaa käyttämällä binääristä päinvastaiseksi vaihtoa. Kuva 9 esittää tämän.



Näytenumerot normaalissa järjestyksessä		Näytenumerot bittikäännöksen jälkeen	
Desimaali	*Binääri*	*Desimaali*	*Binääri*
0	0000	0	0000
1	0001	8	1000
2	0010	4	0100
3	0011	12	1100
4	0100	2	0010
5	0101	10	1010
6	0110	6	0100
7	0111	14	1110
8	1000	1	0001
9	1001	9	1001
10	1010	5	0101
11	1011	13	1101
12	1100	3	0011
13	1101	11	1011
14	1110	7	0111
15	1111	15	1111

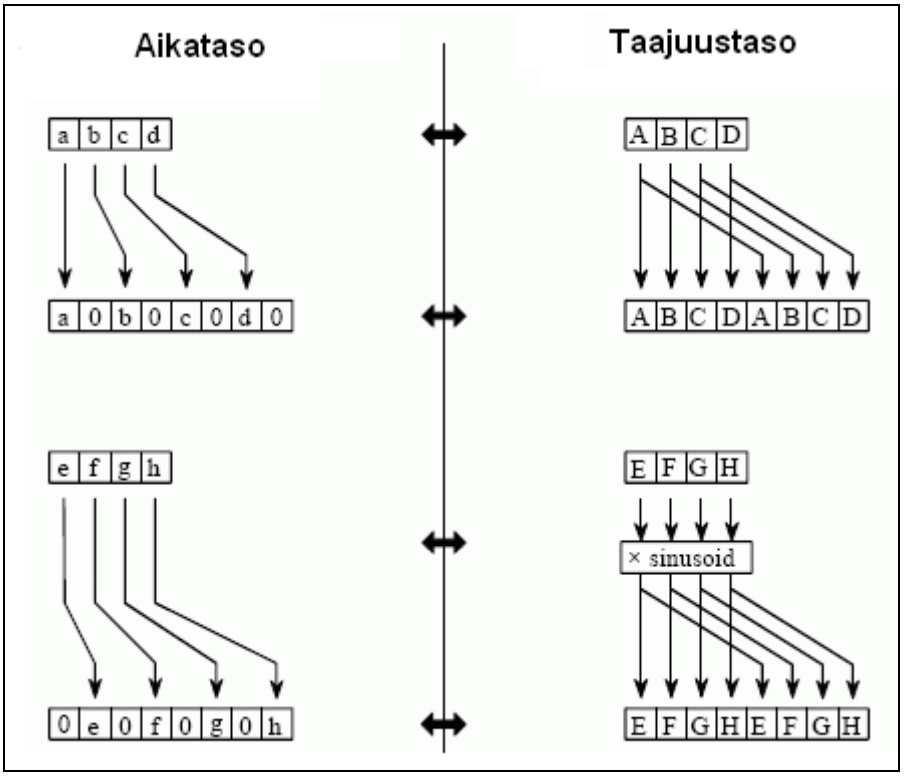
Kuva 9. Osiin jako tapahtuu kääntämällä binääriluku [8.]

Vertaamalla kuvia 8 ja 9 huomataan binäärisen päinvastaiseksi vaihdon tuovan suuren helpotuksen osiin jakoon.

Osiin jakamisen jälkeen muodostetaan jokaisesta pistesignaalista taajuusspektri. Tämä on sama kuin signaali itse, ja se ei vaadi mitään erillisiä toimia. Tämän jälkeen on ajateltava pistesignaalin olevan taajuusspektri eikä aikatason signaali. [8.]

Lopuksi yhdistetään N kpl taajuusspektrejä päinvastaisessa järjestyksessä kuin osiin jaossa. Aikaisempi binäärinen oikotie ei ole käytettävissä vaan jokainen taso täytyy käsitellä yksi kerrallaan. Asia ilmenee tarkastelemalla kuvaa 8 alhaalta ylöspäin. [8.]

Kuvassa 10 on esitetty miten kaksi taajuusspektriä, joista kumpikin sisältää 4 pistettä, yhdistetään yhdeksi taajuusspektriksi, joka koostuu 8 pisteestä. Synteesin täytyy kumota aikaisemmin tapahtunut palasiin jako yhdistämällä 4-pisteiset signaalit lomittamalla. [8.]

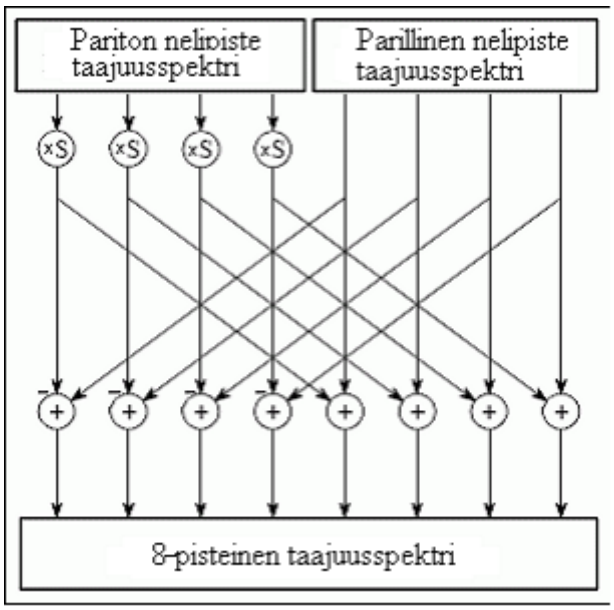


Kuva 10. Spektrien takaisin syntesointi [8.]

Kaksi aikatasossa olevaa signaalia abcd ja efgh voidaan yhdentää ohentamalla kumpaakin signaalia neljällä nollalla, jolloin kummastakin tulee kahdeksanpisteinen signaali. Laskemalla signaali a0b0c0d0 ja 0e0f0g0h yhteen saadaan abcdfgh. Kuva 10 esittää myös, miten taajuustasossa spektrin monistaminen vastaa ohentamista nollilla aikatasossa. Täten taajuusspektrit yhdistetään FFT:ssa monistamalla ja yhteenlaskemisella. [8.]

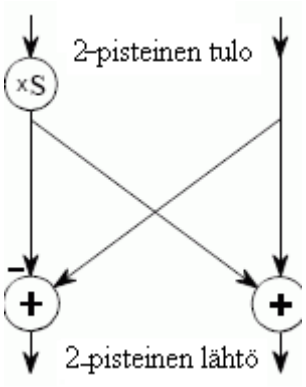
Jotta kaksi aikatasoista signaalia olisi yhdistymisessä sopivat, täytyy nollilla ohentaminen tehdä hieman erilailla. Ensimmäiselle signaalille parittomat pisteet ovat nollia, toiselle signaalille parilliset pisteet ovat nollia. Toisin sanoen toista signaalia siirretään yhdellä näytteellä oikealle. Taajuuspuolella tämä siirto vastaa spektrin kertomista siniaallolla. [8.]

Kuva 11 esittää synteesin kulkukaaviota, jossa yhdistetään kaksi 4-pisteistä spektriä yhdeksi 8-pisteiseksi spektriksi. [8.]



Kuva 11. Synteesin kulkukaavio [8.]

Kuva 11 on muodostettu kuvan 12 kuviosta, jota toistetaan koko ajan. Tätä kuviota kutsutaan sen muodon takia perhoseksi. Siinä otetaan kaksi kompleksista pistettä ja muutetaan ne kahdeksi muuksi kompleksiseksi pisteeksi. [8.]



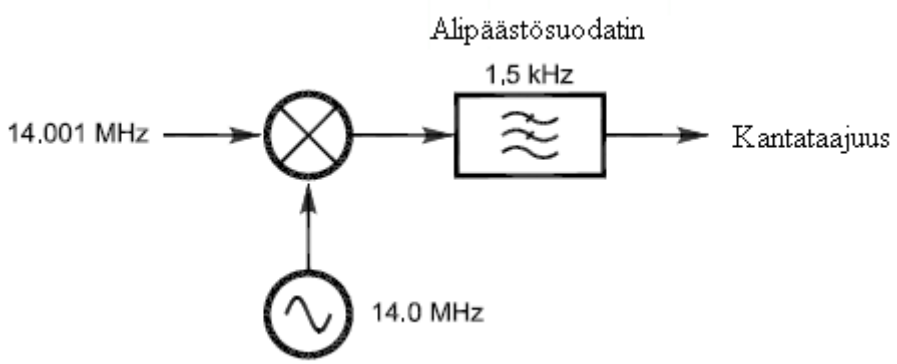
Kuva 12. FFT-perhonen [8.]

6 KVADRATUURI-SEKOITUS

Kvadratuuri-sekoitus on tapa muuntaa radiotaajuussignaali samalla poistaen peilitaajuudet. Peilitaajuuksista eroon pääseminen vaatii kahden 90°:n vaihesiirrossa olevien näytteiden välille tarkkaa balanssia amplitudin ja vaiheen osalta. [2.]

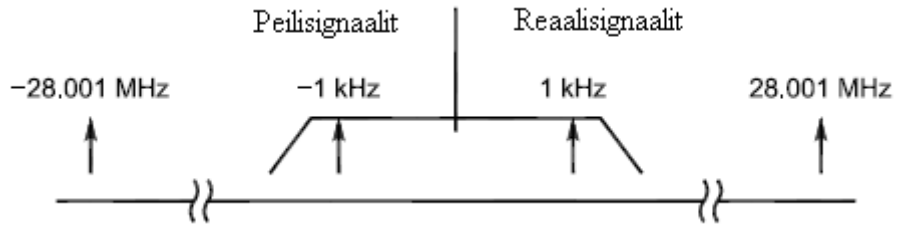
6.1 Peilitaajuus

Ohjelmaradion suunnittelussa pyritään maksimoimaan dynaaminen alue ja systeemin joustavuus. Näiden saamiseksi radiotaajuussignaali muunnetaan samalla poistaen siinä esiintyvät peilitaajuudet. Helpoiten tämä onnistuu käyttämällä DC-tekniikkaa, jossa moduloitu radiotaajuussignaali siirretään suoraan kantataajuudelle. Siirto tapahtuu sekoittamalla signaali radiotaajuuden kantoaaltotaajuudelle asetetun oskillaattorin kanssa, jolloin kaistarajoitettu signaali voidaan siirtää välitaajuudelle 0 Hz. Kuvassa 13 on esitetty signaalin siirto kantataajuudelle. [1.]



Kuva 13. Signaalin siirto kantataajuudelle [2.]

Kuvan 13 esimerkissä 14,001 MHz:n kantoaaltosignaali sekoitetaan 14,000 MHz:n oskillaattorisignaalin kanssa, jolloin kantataajuus siirtyy 1 kHz. Jos alipäästösuodattimen cutoff-taajuus on 1,5 kHz, vain signaalit väliltä 14,000–14,0015 MHz olisivat kaistanpäästön sisällä. Tämän menettelyn ongelma on se, että samalla pääsee läpi myös taajuudet välillä 13,9985–14,000 MHz peilitaajuutena kaistanpäästön sisällä. Kuva 14 esittää asian. [2.]



Kuva 14. Peilitaajuus [2.]

Kun kantotaajuus f_c sekoitetaan oskillaattorisignaalin f_{lo} kanssa, ne yhtyvät kaavan 1 mukaisesti:

$$f_c f_{lo} = \frac{1}{2}[(f_c + f_{lo}) + (f_c - f_{lo})] \quad (1)$$

Käyttämällä kuvan 13 mukaista sekoitinta vastaanotetaan lähtösignaalit:

$$f_c + f_{lo} = 14{,}001\ MHz + 14{,}000\ MHz = 28{,}001\ MHz$$

$$f_c - f_{lo} = 14{,}001\ MHz - 14{,}000\ MHz = 0{,}001\ MHz$$

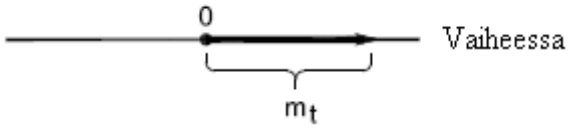
Näiden lisäksi vastaanotetaan myös peilitaajuus:

$$-f_c + f_{lo} = -14{,}001\ MHz + 14{,}000\ MHz = -0{,}001\ MHz$$

Alipäästösuodatin poistaa 28,001 MHz:n summataajuuden (*sum frequency*), mutta -0,001 MHz:n erotustaajuuspeili (*difference-frequency image*) pysyy lähdössä. Peili aiheuttaa myös kohinaa. [2.]

6.2 I- ja Q-signaalit

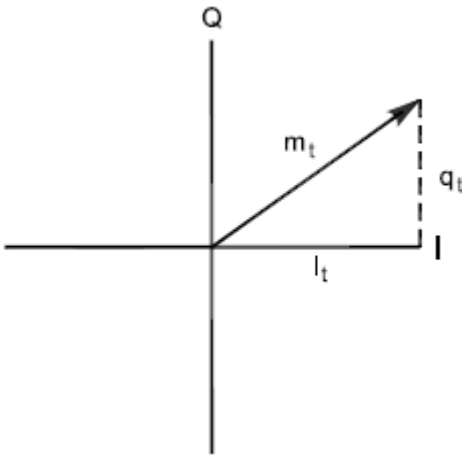
Kun radiotaajuussignaali muunnetaan kantataajuudelle käyttäen yhtä kanavaa, asia voidaan visualisoida muuttuvana amplitudina yhdellä akselilla, kuten kuva 15 esittää. Tätä kutsutaan vaiheessa (*inphase*) olevaksi signaaliksi eli I-signaaliksi. [2.]



Kuva 15. I-signaali [1.]

Jos radiotaajuuden kantosignaalia viivästytetään 90°, muodostuu kvadratuurisignaali eli Q-signaali. Käyttämällä I- ja Q-signaaleita saadaan selville alkuperäisen signaalin vaihe ja amplitudi. [2.]

Kuva 16 esittää radiotaajuuden kantosignaalia, jossa I-signaali on x-akselilla ja Q-signaali on y-akselilla. Tätä kutsutaan usein kompleksitasoksi (*complex plane*). Näiden kahden signaalin avulla voidaan piirtää osoitinviiva, joka esittää alkuperäisen signaalin hetkellistä suuruutta ja vaihetta. [2.]



Kuva 16. Kompleksitaso [2.]

m_t-vektorin laskemiseksi käytetään Pythagoraan lausetta:

$$m_t = \sqrt{I_t^2 + Q_t^2} \tag{2}$$

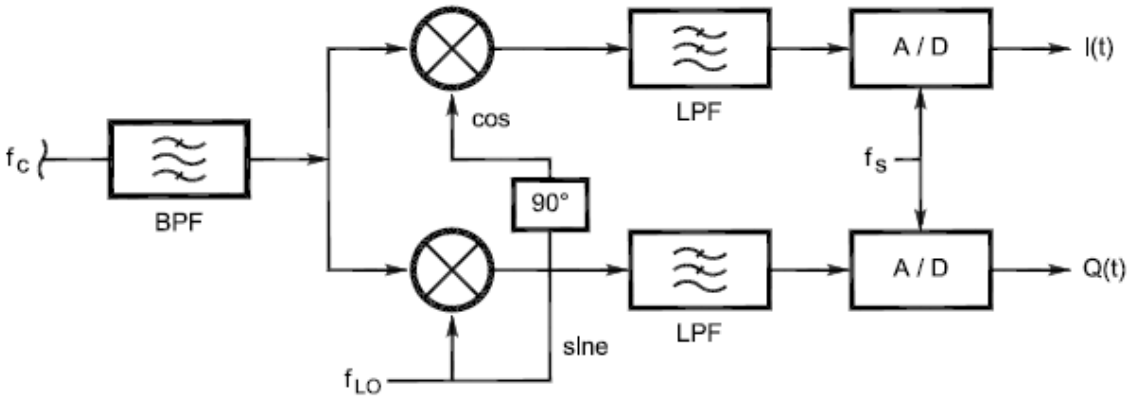
Vaihe lasketaan siirtämällä osoitinta kelloa päinvastaiseen suuntaan positiivisesta x-akselista.

$$\theta_t = tan^{-1}\left(\frac{Q_t}{I_t}\right) \quad (3)$$

Näin saadaan kaikki tarvittava tieto signaalista tietyssä kohtaa aikaa, oli kyseessä sitten jatkuva-aikainen signaali tai diskreetti signaali.

Kaavaa 2 käytetään AM-signaalien demoduloimiseen ja kaavaa 3 FM-signaalien demuloimiseen.

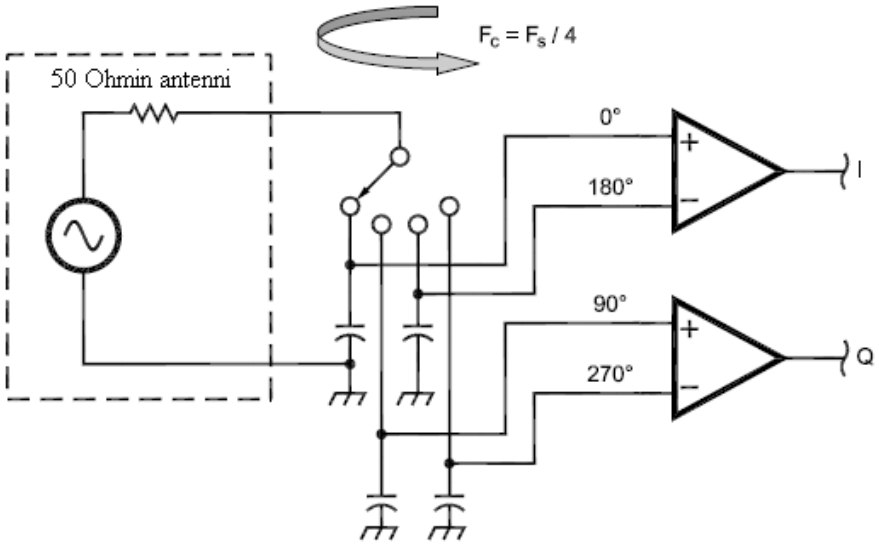
Kuva 17 esittää kvadratuuri I/Q-sekoitinta, jossa viedään radiotaajuussignaali kaistanpäästösuodattimelle ja suodattimen ulostulosta viedään signaali kahdelle rinnakkaiselle sekoitinkanavalle. Oskillaattorin taajuus viedään toiselle sekoittimelle 90° viivästyneenä, jolloin muodostuu kosiniaalto. Kantoaaltotaajuus $f_c(t)$ sekoitetaan sekä siniaalto että kosiniaalto oskillaattorin taajuuden kanssa. Molemmat signaalit kuljetetaan alipäästösuodattimen kautta, joka poistaa yli Nyqvistin taajuuden menevät taajuudet laskostumisen välttämiseksi. Suodattimilta ulostulosta saadaan I(t)- ja Q(t)-signaalit, ja ne ovat keskenään 90°:n vaihesiirrossa. [1.]



Kuva 17. Kvadratuuri I/Q-sekoitin [2.]

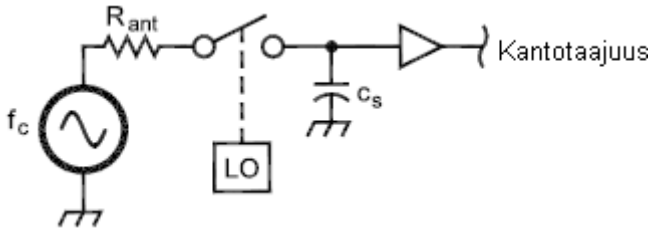
6.3 Tayloe-ilmaisin

Kuvassa 18 on esitetty Tayloe-ilmaisin. Ilmaisimen voidaan kuvitella olevan 4-sijainen pyörivä katkaisija, joka pyörii kantotaajuutta vastaavalla nopeudella. 50 Ω:n antenni-impedanssi on kytketty pyörijään, ja jokainen neljästä kytkimestä on yhdistetty näytteistyskondensaattoriin. Pyörivä kytkin näytteistää signaalia vaiheissa 0°, 90°, 180° ja 270°. Koska kytkinpyörijä pyörii radiotaajuuden kantoaaltotaajuudella, jokainen kondensaattori sisältää kantoaaltotaajuuden amplitudi-arvon, joka vastaa neljäsosa jaksoa. [2.]



Kuva 18. Tayloe-ilmaisin [1.]

50 Ω antenni-impedanssi ja näytteistyskondensaattori muodostavat R/C -alipäästösuodattimet sillä hetkellä jaksoa, kun kytkin on kiinni. Täten jokainen näyte on keskiarvojännite neljäsosa jakson aikana. Tämä on esitetty kuvassa 19. [2.]



Kuva. 19. Antenni-impedanssi ja näytteistyskondensaattori muodostavat RC-alipäästösuodattimen [2.]

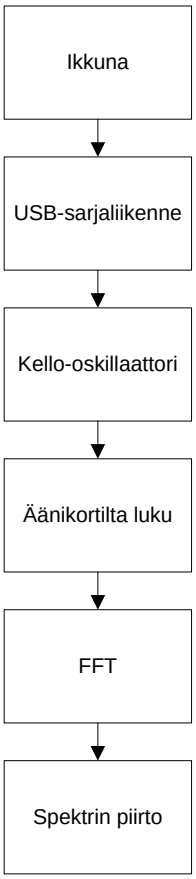
Jos radiotaajuuden kantotaajuudella ja pyörimistaajuudella on täsmälleen sama vaihe, jokaisen kondensaattorin lähtö on DC-taso, joka vastaa keskimääräistä näytteen arvoa.

Jos 0°:n ja 180°:n näytteistyskondensaattorien lähdöt summataan yhteen differentiaalisesti käyttäen operaatiovahvistinta, lähtö olisi DC-jännite, joka vastaa kahta itsenäisesti näytteistettyä arvoa. Tästä seuraa 6 dB:ä kohinasta vapaata vahvistusta. Sama pätee myös 90°:n ja 270°:n näytteistyskondensaattoreille. 0° / 180° summaus muodostaa I-kanavan ja 90° / 270° muodostaa Q-kanavan. [2.]

Kun kantotaajuutta siirretään pois näytteistystaajuudesta, invertoivan vaiheen arvot eivät ole enää DC-arvoja. Ulostulotaajuus vaihtelee eroavaisuustaajuuden mukaan kantoaaltotaajuuden ja kytkintaajuuden välissä antaen tarkan kantataajuudelle muutetun signaalin. [2.]

7 OHJELMOINNIN SUORITUS

Kuvassa 20 on esitelty ohjelmoimisen kulkukaavio.

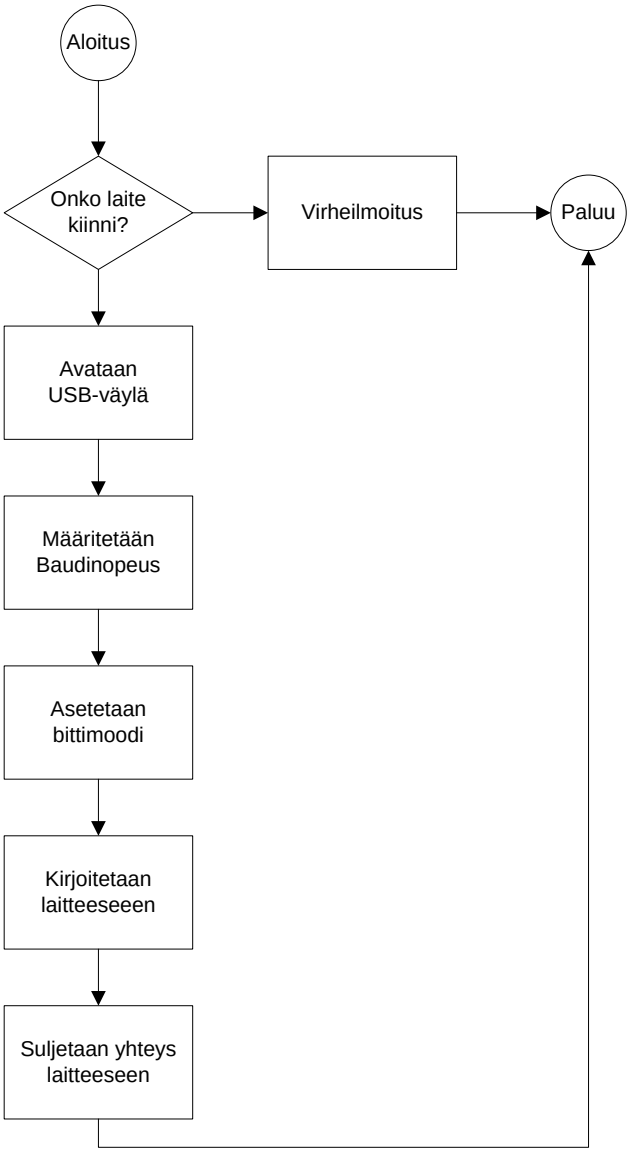


Kuva 20. Ohjelmoinnin kulkukaavio

Käyttäjälle ohjelman ytimenä on ikkuna, josta hän voi asettaa perustaajuuden haluamakseen. Tämä asetettu arvo kulkeutuu kellopiirille USB-liittimen kautta, jossa arvon avulla pyöritetään PLL:n sisäiset laskurit sopiviksi. Lähtötaajuus ajautuu sekoittimelle, jonka avulla saadaan I- ja Q-signaalit äänikortille. Ikkunaohjelmassa luetaan äänikortilta kummatkin mainitut signaalit ja suoritetaan niille nopea Fourier-muunnos. Lopuksi taajuustasossa olevan tiedon avulla piirretään signaalin spektri.

7.1 USB-sarjaliikenne

USB-sarjaliikenteen ohjelmoimiseen käytetään Future Technology Devices International Ltd:n luomia aliohjelmia, joiden käyttäminen vaatii FTD2XX-kirjaston. Kuvassa 21 on esitelty USB-sarjaliikenteen vuokaavio.



Kuva 21. USB-sarjaliikenteen vuokaavio

7.2 Kello-oskillaattorin perustaajuuden laskenta

Kello-oskillaattorin täytyy käydä neljä kertaa vastaanotettua taajuutta korkeammalla, jotta tarvittava vaihesuodatus voidaan jakaa neljällä. Kytkentä on suunniteltu vastaanottamaan taajuuksia aina 30 megahertsiin asti, mikä tarkoittaa maksimissaan 120 MHz:n lähtötaajuutta oskillaattorilta. Ohjelmallisesti tämä tarkoittaa sitä, että käyttäjän haluama perustaajuus täytyy olla kello-oskillaattorin lähdössä nelinkertainen.

Perustaajuuden laskemiseksi on tiedettävä referenssikiteen taajuus, P- ja Q-laskurien arvot ja jälkijakaja. Referenssitaajuus on käytetyllä piirillä 10 MHz. Q-laskuria käytetään referenssitaajuuden jakamiseksi ja P-laskuri on tulolaskuri, jolla kerrotaan (REF / Qtot) jänniteohjatun oskillaattorin taajuuden saamiseksi. Jälkijakaja on nimensä mukaisesti viimeinen taajuuteen vaikuttava rekisteri, joka jakaa lähtötaajuuden tietyllä arvolla. Lopullinen taajuus saadaan kaavalla 4.

$$CLK = \frac{\left(\frac{REF \cdot P}{Q}\right)}{Post\ Divider} \tag{4}$$

P-laskuri on tulolaskuri, jolla kerrotaan (REF/Qtot) VCO-taajuuden saamiseksi. P-laskuri saadaan kahdella sisäisellä muuttujalla PB ja PO. Kokonaisarvon laskemisessa käytetään kaavaa 5.

$$P_{yht} = (2(PB + 4) + P0) \tag{5}$$

PB on 10-bittinen muuttuja, joka on määritelty rekistereissä 40H ja 41H. Rekisterin 40H kaksi LSB-bittiä ovat muuttujan PB kaksi MSB-bittiä. P0 on yksi bittinen muuttuja rekisterissä 42H, jolla saadaan *Pyht* parittomaksi tarpeen vaatiessa. *Pyht* saa olla välillä 8–2055. Miniarvo saadaan jos PB ja P0 on 0 ja maksimiarvo, jos PB on 1023 ja P0 on 1. P-laskuri käyttää rekistereitä 40H–42H, kuten taulukko 2 esittää.

Taulukko 2. Pumpun, P-laskurin, Q-laskurin ja jälkijakajien rekisterit

Osoite	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
40H	1	1	0	Pumppu(2)	Pumppu(1)	Pumppu(0)	PB(9)	PB(8)
41H	PB(7)	PB(6)	PB(5)	PB(4)	PB(3)	PB(2)	PB(1)	PB(0)
42H	P0	Q(6)	Q(5)	Q(4)	Q(3)	Q(2)	Q(1)	Q(0)
OCH	DIV1SRC	DIV1N(6)	DIV1N(5)	DIV1N(4)	DIV1N(3)	DIV1N(2)	DIV1N(1)	DIV1N(0)
47H	DIV2SRC	DIV2N(6)	DIV2N(5)	DIV2N(4)	DIV2N(3)	DIV2N(2)	DIV2N(1)	DIV2N(0)

Q-laskurilla jaetaan referenssitaajuus. Q määritellään 7 bitillä rekisterissä 42H. Lopulliseen arvoon lisätään 2. Q:n arvo tulee olla välillä 2–129. Jos REF / Q_{yht} putoaa alle 250 kHz, piiri ei pakosti toimi kunnolla. Q:n arvo tallennetaan rekisteriin 42H.

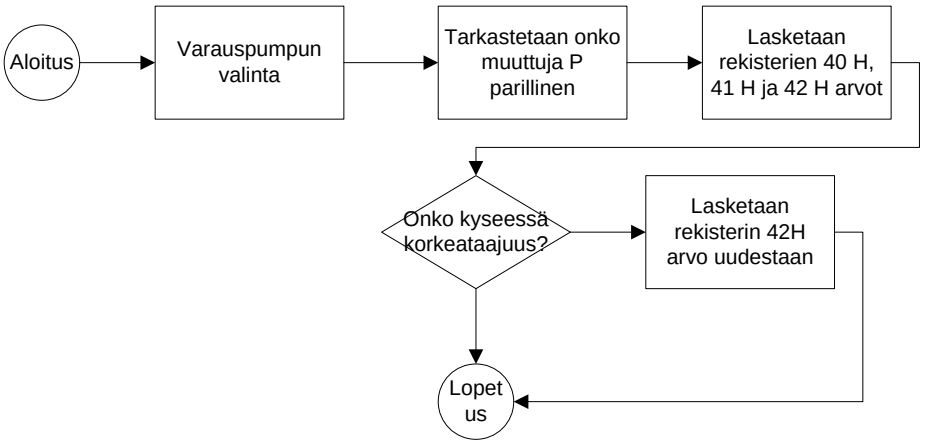
Jakajapankkeja on kaksi. Jakajapankeille on neljä erilaista jälkijakajaa: /2, /3, /4 ja DIVXN. Näistä jälkimmäinen on suuruudeltaan ohjelmoijan itsensä määrittelemä, jonka arvo on väliltä 4–127. Jälkijakajat määritellään rekistereihin OCH ja 47H.

Varauspumpun arvo riippuu lasketusta PB-arvosta. Pumpun arvot määritellään rekisteriin 40H. Taulukko 3 esittää pumpun arvot eri P arvoilla.

Taulukko 3. Varauspumpun arvo eri P:n arvoilla

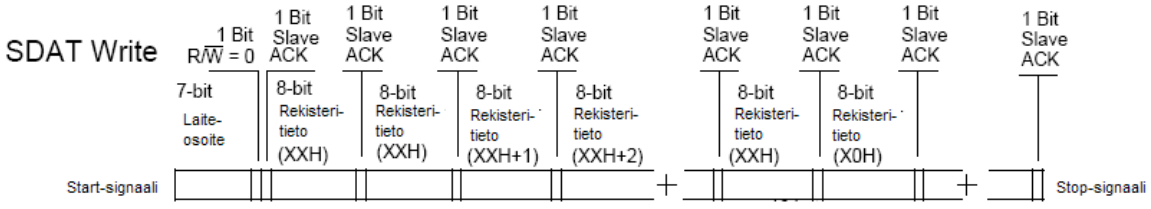
Varauspumppu	**Pyht**
000	16 - 44
001	45 - 479
010	480 - 639
011	640 - 799
100	800 - 1023
101, 110, 111	Ei saa käyttää

Ohjelmassa on globaalimuuttuja P, johon ikkunan taajuusvierityspalkki ja taajuusnapit vaikuttavat. Tämän muuttujan perusteella lasketaan varauspumpun arvo, P- ja Q-kertoimien arvot. Jälkijakajan arvo riippuu senhetkisestä taajuusalueesta. Kuvassa 22 on esitetty prosessin vuokaavio.



Kuva 22. Kello-oskillaattorin taajuuden laskenta

Kello-oskillaattori käyttää sarjaliikenteeseen kahta linjaa (SDAT ja SCLK), jotka toimivat 400 kbit/s nopeudella sekä kirjoitus että lukutilassa. Datan tietokehys on kuvan 23 mukainen.

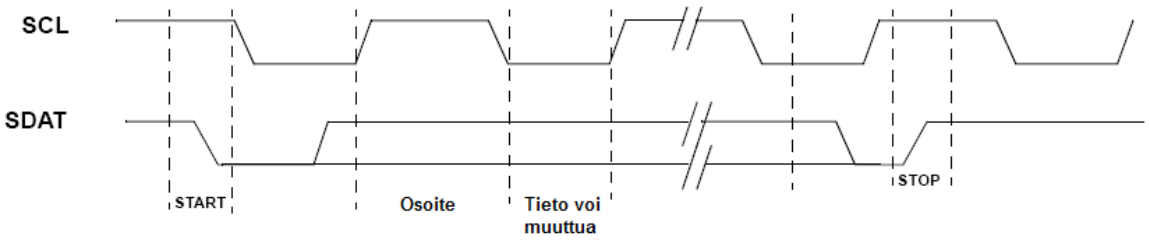


Kuva 23. Datan tietokehys [8.]

Jokainen tietokehys merkitään START- ja STOP-sekvensseillä. Aloitussekvenssi saadaan viemällä SDAT alas silloin kuin SCLK on ylhäällä. Lopetussekvenssi toteutetaan viemällä SDAT ylös, kun SCLK on ylhäällä.

Tietokehys alkaa START-sekvenssillä, joka on yhden bitin kokoinen. Tämän jälkeen tulee 7-bittinen laiteosoite, jolla kerrotaan mihinpäin piiriä tieto kirjoitettaan (RAM, EEPROM). Perään kerrotaan onko kyseessä luku- vai kirjoitusoperaatio yhdellä bitillä. Seuraavaksi kirjoitetaan rekisterin osoite (8 bittiä) ja tähän rekisteriin laitettava data (8 bittiä). Jos laiteosoitteeseen kirjoitetaan moneen eri rekisteriin tietoa, voidaan tässä kohtaa jatkaa yksinkertaisesti toistamalla rekisterin osoite- ja data-sekvenssejä. Lopuksi tietokehys päätetään STOP-sekvenssillä.

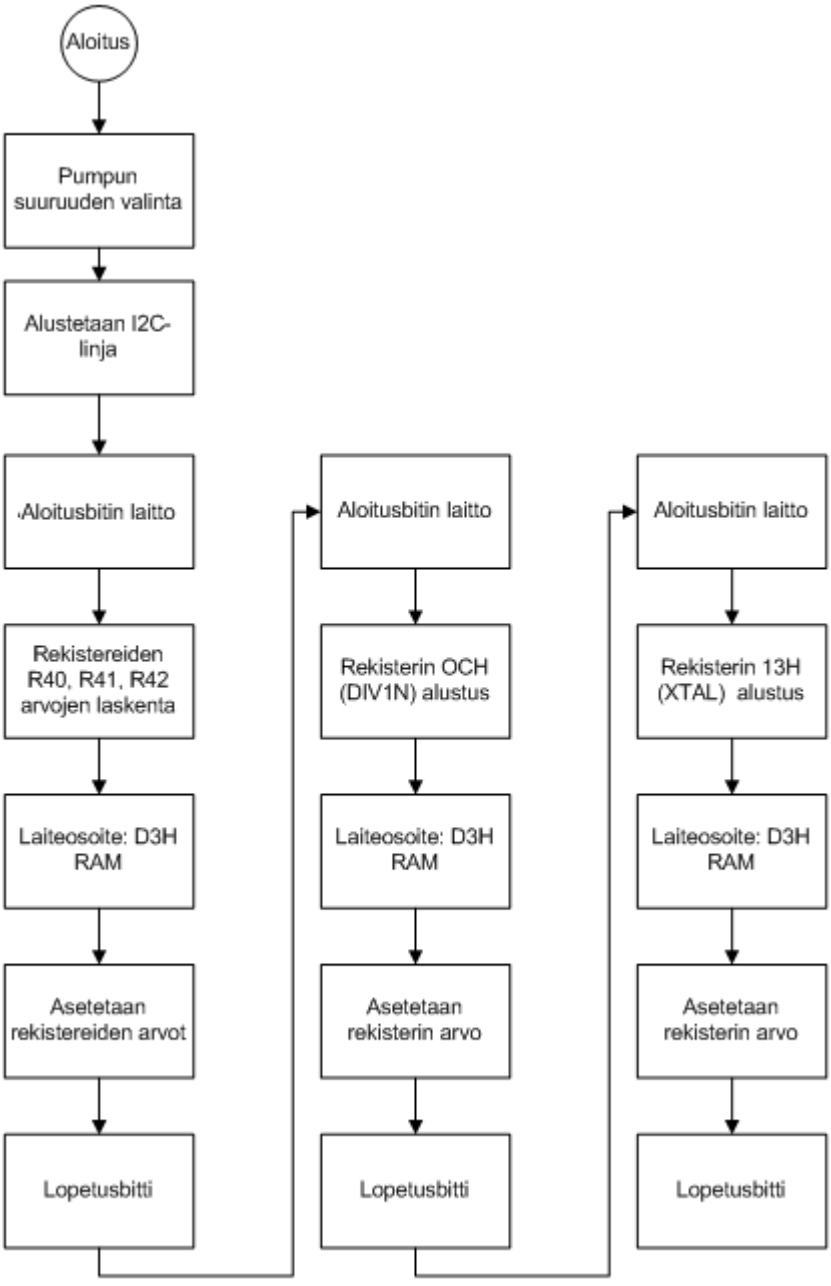
Kuvassa 24 on esitelty SPI-ajastus.



Kuva 24. SPI:n ajastus [8.]

Ohjelmassa on iso puskuri, johon tallennetaan vuoronperään sarjaliikenteen kello- ja data-arvot eli onko kyseinen linja ylhäällä vai alhaalla. Arvot seuraavat kuvan 22 datakehystietoa ja kellotus kuvan 23 ajastuskuvaa. Sarjaliikenteen arvoja ohjaa funktio, jolle voidaan syöttää niin rekisteriosoitteet kuin rekisteriin laitettavat arvot. Täytetty puskuri lähetetään lopuksi laitteelle.

Kuvassa 25 on esitelty kokonaisuudessaan kello-oskillaattorin kantataajuuden syöttöfunktion vuokaavio. Liitteestä 1 löytyy funktioiden koodi.



Kuva 25. Perustaajuuden syöttö laitteelle

7.3 Ikkunan ohjelmointi

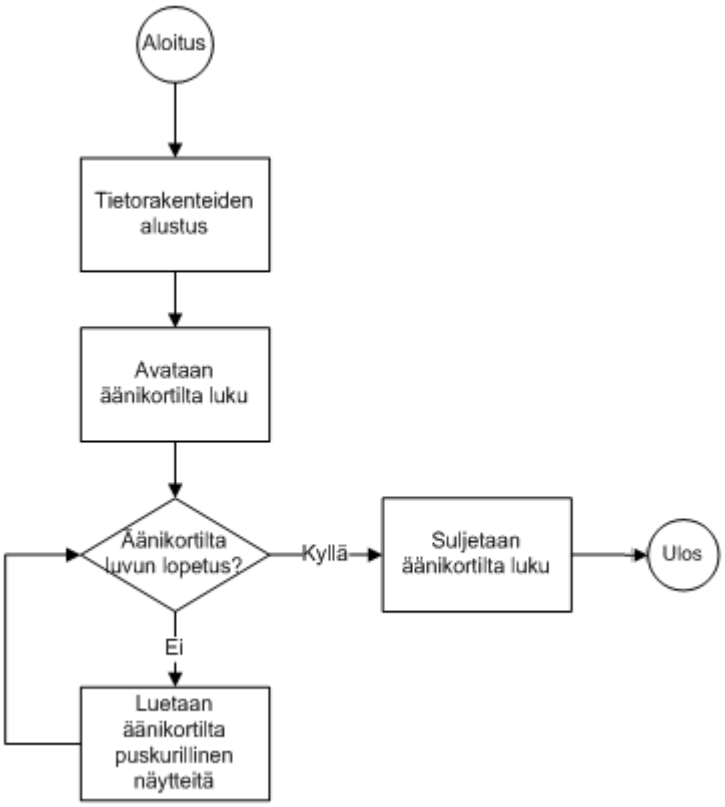
Ohjelmointi toteutetaan käyttämällä C-kieltä ja ohjelmoinnissa käytetään Windows API-kirjastoa ikkunan luomiseen ja GDI-kirjastoa piirtämiseen. Window API määrittelee ikkunafunktiot, rakenteet ja tietotyypit, joita tarvitaan käyttöjärjestelmän kanssakäymiseen. Windows-ohjelmointi keskittyy pääasiassa viestien käsittelyyn, jotka lähetetään käyttäjän tekemisen pohjalta. Jos käyttäjä esimerkiksi painaa nappia, ohjelmoijan on käsiteltävä tätä vastaava viesti halutun lopputuloksen aikaansaamiseksi. Eräät viestit käyttöjärjestelmä lähettää itse. Tällaiset viestit saapuvat esimerkiksi ohjelman luonnin yhteydessä tai ikkunan koon muuttuessa. Tässä ohjelmassa ikkuna on tehty vakiokokoiseksi, eikä sitä voi suurentaa kokoruudulle. Taulukossa 4 on esitelty perusviestit, jotka ohjelma käsittelee.

Taulukko 4. Viestien merkitykset

Viesti	**Merkitys**
WM_CREATE	Tapahtuu ikkunan luonnin yhteydessä. Tämän viestin aikana luodaan lapsi-ikkunat, alustetaan muuttujat ja varataan dynaamisesti muistia äänenkaappaukseen.
WM_TIMER	Viesti saapuu ajastimen alustuksen mukaisin ajanhetkin. Tämän viestin saapuessa annetaan spektrinpäivityslupa.
WM_SIZE	Tapahtuu ikkunan luonnin yhteydessä. Viestin aikana otetaan ylös ruudun koko ja sijoitetaan lapsi-ikkunat paikalleen ikkunaan.
WM_HSCROLL	Tapahtuu vierityspalkkia painettaessa. Viestin aikana tarkastetaan mikä viidestä vierityspalkista oli muutoksen kohteena, ja tehdään tarvittava muutos. Samalla tarkastetaan ettei maksimiarvoa ylitetä ja minimiarvoa aliteta.
WM_COMMAND	Tapahtuu nappia painettaessa. Tarkastetaan mikä nappi oli painalluksen kohteena ja suoritetaan tämän perusteella nappia vastaava muutos.
WM_PAINT	Tämän viestin aikana suoritetaan kaikki piirtäminen ruudulle. Piirrettävänä on spektrialue, tekstit, akselit ja spektri.
WM_DESTROY	Tapahtuu ohjelmaa sulkiessa. Viestin aikana tuhotaan laskuri.

7.4 Äänikortilta luku

Kuvassa 26 on esitetty äänikortilta luvun vuokaavio.



Kuva 26. Äänikortilta luvun vuokaavio

Ääniaallon kaappaamiseksi täytetään kaksi rakennetta (WAVEFORM ja WAVEHEADER), joilla kerrotaan Windowsille miten kaappaus tulisi tapahtua. Ohjelmassa äänimuodon käsittely on PCM, äänikanavia on 2, näytteistystaajuus on 44100 Hz ja näytteen bittikoko on 16. Näiden valintojen pohjalta tarvitaan 16384 tavun kokoinen puskuri. Äänen kaappauksessa käytettään tuplapuskurointia, jossa käytetään kahta puskuria niin, että kaappauksessa ei esiinny taukoa. Koska API käyttää viestijärjestelmää, pitää ääniaallon kaappaukseen käsitellä kolme viestiä: MM_WIM_OPEN, MM_WIM_CLOSE ja MM_WIM_DATA. Taulukossa 5 on esitelty viestien merkitykset. Liitteestä 2 löytyy kaappaukseen käytettävien funktioiden koodit.

Taulukko 5. Äänen kaappaukseen liittyvät viestit

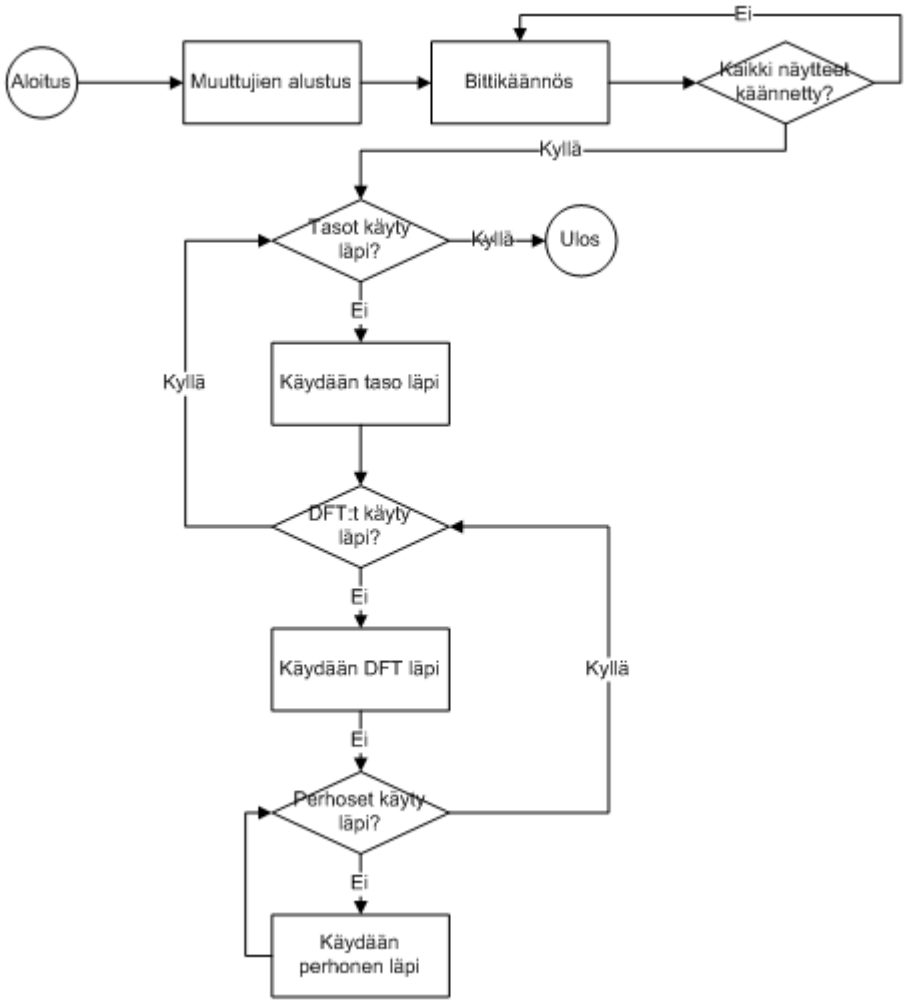
Viesti	**Merkitys**
MM_WIM_OPEN	Tämä viesti tapahtuu ON-nappia painettaessa ja se ilmoittaa äänikortin lukuprosessin alusta. Tämän viestin aikana alustetaan puskureille uudestaan tila, alustetaan tuplapuskurointi ja käynnistetään näytteistys.
MM_WIM_DATA	Tapahtuu, kun äänenkaappauspuskuri on tullut täyteen. Tämän viestin aikana jaetaan puskurin tieto kahteen taulukkoon, joista toisessa on reaaliosa ja toisessa imaginääriosa. Jakamisen jälkeen lisätään uusi puskuri täytettäväksi.
MM_WIM_CLOSE	Tapahtuu painettaessa OFF-näppäintä tai jos ohjelma suljetaan. Viestin aikana vapautetaan puskurien viemä tilanvaraus ja lopetetaan laskuri.

Äänikortilta saatavat I- ja Q-signaalit tulevat yhdessä vektorissa. Tämä vektori täytyy jakaa vektoreiksi I[ ] ja Q[ ], joille suoritetaan nopea Fourier-muunnos.

7.5 Nopea Fourier-muunnos

Nopeassa Fourier-muunnoksessa muutetaan reaali- ja imaginääri-taulukoiden arvot aikatasosta taajuustasoon. Aliohjelmalle annetaan sisään reaali- ja imaginääri-taulukot ja FFT:n koko. Aliohjelma tekee aluksi bittikäännöksen taulukoiden sisällölle. Tämän jälkeen taulukoiden tiedot syntesoidaan, johon tarvitaan kolme for-silmukkaa. Ensimmäisellä käydään läpi jokainen taso, toisella jokainen DFT ja kolmannella jokainen perhonen. Koska taulukot käsitellään automaattisesti osoittimilla, taulukon tiedot ovat päivittyneet aliohjelmasta poistuessa.

Kuvassa 27 on esitetty FFT-aliohjelman vuokaavio. Liitteestä 2 löytyy funktion koodi.

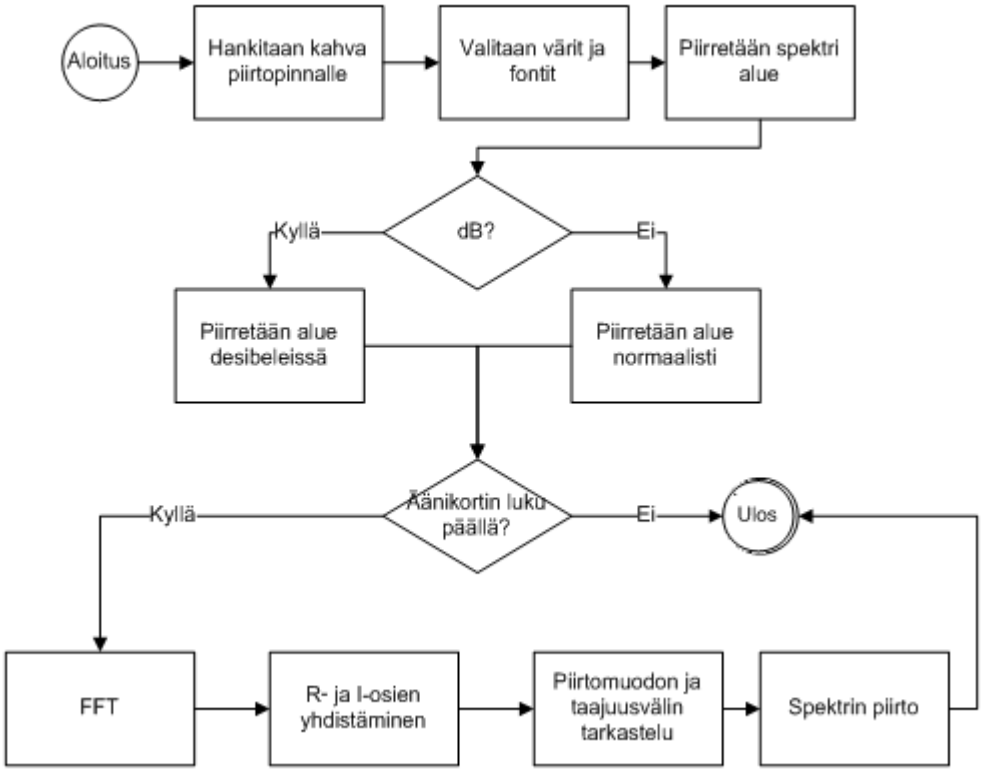


Kuva 27. FFT-aliohjelman vuokaavio

7.6 Spektrin piirto

Spektrin piirto tapahtuu WM_PAINT-viestin käsittelyn aikana. Viestin käsittelyssä valitaan piirtoon käytettävät värit ja fontit, piirretään y- ja x-akselit ja spektri. GDI-kirjasto tarjoaa suorat funktiot piirtämiseen, joita käyttämällä piirto on helppo suorittaa. Reaali- ja imaginääri-taulukoille tehdään nopea Fourier-muunnos ja tämän jälkeen taulukon arvot yhdistetään käyttämällä kaavaa 2. Yhdistämisen jälkeen voidaan piirtää spektri.

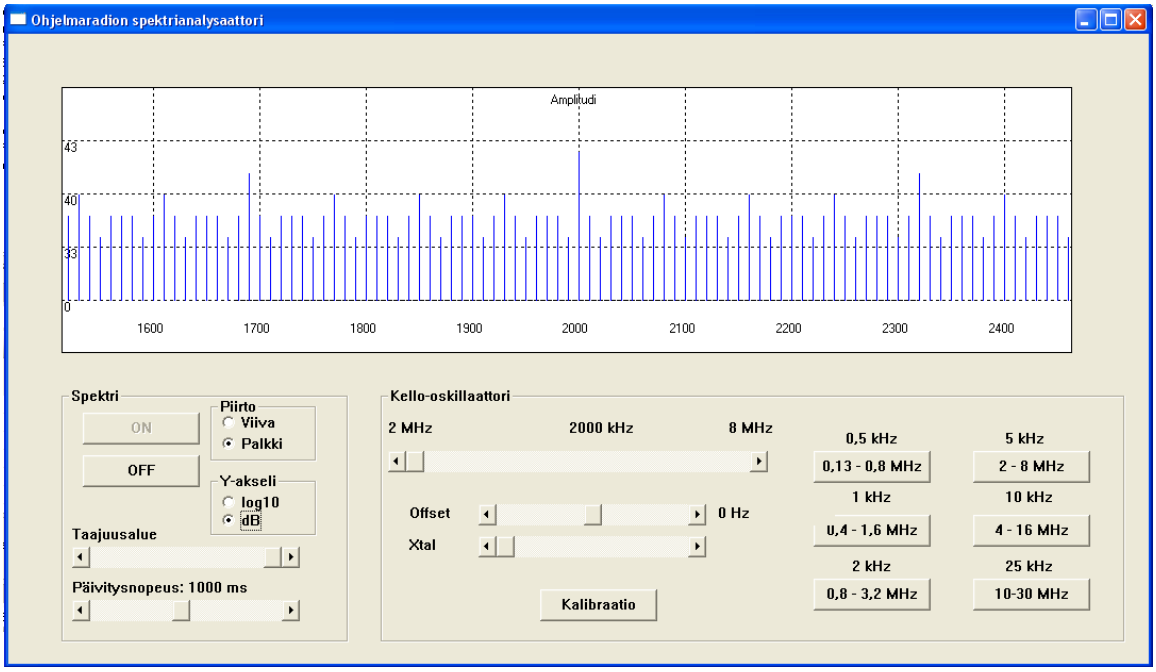
Kuvassa 28 on esitelty piirtämisen vuokaavio. Liitteestä 3 löytyy piirtofunktioiden koodi.



Kuva 28. Spektrin piirron vuokaavio

8 OHJELMAN TOIMINTA

Kuvassa 29 on esitelty kirjoitetun ohjelman ikkuna. Ikkuna on jaettu kolmeen osaan: ensimmäinen osa on spektrin piirtoruutu, toinen osa on spektrin säätönapit ja kolmas osa on kello-oskillaattorin taajuuden muutossäätimet.



Kuva 29. Kirjoitetun ohjelman ikkuna

Spektrin säätönapeilla voidaan muuntaa spektrin piirron esitystapaa. ON- ja OFF-napeilla kytketään spektrin piirto päälle ja pois päältä. Piirto-kohdan alta löytyy kaksi mahdollista spektrin piirron esitysmuotoa: palkkimainen tai viivamainen. Y-akseli-napeilla voidaan muuttaa y-akselin esitystapaa joko normaaliksi 10-kantaiseksi logaritmiksi tai desibelityyliseksi. Taajuusalue-vierityspalkilla muutetaan piirrettävää taajuusaluetta. Kuvassa 29 alue on 12300–13800 Hz. Päivitysnopeus-vierityspalkilla säädetään spektrin piirron nopeutta. Se voi olla suurimmillaan 10 kertaa sekunnissa ja pienemmillään kerran 2 sekunnissa.

Kello-oskillaattorin säätimillä muunnetaan oskillaattorin taajuutta. Taajuus voi olla pienimmillään 130 Hz ja suurimmillaan 30 MHz. Suurta vierityspalkkia siirtämällä taajuus muuttuu askelittain. Palkin vasemmalla ja oikealla yläpuolella on ilmoitettu taajuuden siirtorajat. Palkin oikealla puolella olevilla kuudella näppäimellä vaikutetaan vierityspalkin taajuusrajoihin. Offset-vierityspalkilla voidaan hienosäätää taajuutta. Xtal-säätimellä muutetaan oskillaattorin kondensaattoreiden arvoa. Kalibraatio näppäin tekee 5 MHz:n kalibraatiosignaalin.

9 OHJELMAN TESTAUS

Ohjelmaa testattiin käyttäen Visual Studion debuggeria. Debuggeri on kätevä tapa tarkastella ohjelman toimintaa rivi kerralla. Testauksessa tarkasteltiin erityisesti perustaajuuden syöttöä kello-oskillaattorille ja äänikortilta lukua, jotka ovat ohjelman toimimisen kannalta kriittisimmät kohdat.

Perustaajuuden syöttöä testattiin antamalla ohjelmassa eri taajuuksia kello-oskillaattorille. Taulukossa 6 on esitetty syötetyt taajuudet ja ohjelman käyttämät kello-oskillaattorin kertoimet. Testauksessa saatiin lähtötaajuudeksi jokaisella annetulla taajuudella nelinkertainen lähtötaajuus, kuten pitääkin. Ohjelma käyttää laskemisessa kaavoja 4 ja 5.

Taulukko 6. Perustaajuuden syöttötesti

Annettu taajuus	**REF**	**P**	**Q**	**DIV1N**	**Lähtötaajuus**
2 MHz	10 MHz	400	50	10	8 MHz
10 MHz	10 MHz	500	50	5	40 MHz
15 MHz	10 MHz	1500	50	5	60 MHz
25 MHz	10 MHz	1000	25	4	100 MHz

Ohjelma osaa laskea perustaajuuden oikein. Seuraavaksi tarkastellaan saadaanko se oikein puskuriin eli niin, että sarjaliikenne sujuu. Tämä tapahtuu tarkastelemalla debuggerilla puskurin sisältöä siinä vaiheessa, kun se ollaan lähettämässä laitteelle. Tarkastelussa havaittiin puskurin sisältävän sarjaliikenteen mukaisesti tietoa. Samalla tarkasteltiin puskurin syöttämistä laitteelle. Puskurissa oli 284 tavua tietoa ja laitteen kirjoitusfunktio ilmoitti kirjoitetuksi määräksi 284 tavua eli laite vastaanotti kaiken tiedon.

Debuggerilla tarkasteltiin myös äänikortin lukua äänitietopuskurin tullessa täyteen. Ensinnäkin havaittiin puskurin jakamisen reaali- ja imaginääri-osiin tapahtuvan onnistuneesti. Sen sijaan FFT-aliohjelmasta löydettiin paha bugi, joka vääristi lopputuloksen. Ongelma liittyi for-silmukan parametriin, jonka ansiosta silmukoiden kertamäärä oli väärä. Ongelma korjattiin ja ohjelma toimi tämän jälkeen oikein.

10 TYÖN ANALYSOINTI

Valmis ohjelma täytti sille asetetut toiminnalliset vaatimukset ja se toimi kuten piti. Ohjelman avulla käyttäjä pystyy syöttämään vastaanotinkytkennälle kantataajuuden ja ohjelma tulostaa taajuutta vastaavaan ääniaallon spektrin. Ongelmana ohjelmassa saattaa olla se, että se ei ohjeista käyttäjää mitenkään. Käyttäjän on tiedettävä mitä tekee ennen ohjelman käyttöä.

Työn valmistusprosessi oli melko ongelmaton ja ohjelman toimintaan ei tullut mitään muutoksia alkuvaatimuksista. Suurimmat ongelmat esiintyivät kello-oskillaattorin taajuuden laskemisessa, sillä oskillaattorin rekisterien hallinta ei ollut selkein mahdollinen. Haastetta tuli myös englanninkielisestä termistöstä, joka ei kaikilta osin ollut täysin tuttua. Ongelmia aiheutti myös valmiit Pascal- ja Qbasic-koodit, jotka käänsin C-kielelle. Varsinkin jälkimmäinen kieli oli jotain todella kamalaa johtuen hyppylauseista, eikä Pascal C-läheisyydestä huolimatta ollut kovinkaan kiinnostavaa.

Työssä mukavinta oli se, ettei tarvinnut koskea rautaan muuten kuin asettaessa piuhat kiinni kytkentään. Kytkennän toiminnan ymmärtäminen oli toki välttämätöntä.

Tätä työtä tehdessä opin melkoisesti erityisesti kello-oskillaattorista, jonka kanssa tappelin kuukauden verran. Ohjelmaa tehdessä tuli kerrattua Windows-ohjelmoinnin perusteet C-kielellä. Nykyään Windows API-ohjelmoinnin on syrjäyttänyt melkolailla .NET-kehitysympäristö. API-ohjelmointia näkee nykyään enimmäkseen ajuritasolla.

Sovellus saatiin valmiiksi aikataulussa eli kolmessa kuukaudessa. Ohjelmaan voitaisiin tulevaisuudessa päivittää mahdollisuus kuunnella radiota sekä rakentaa käyttäjän opastustoiminnot. Molempien uudistuksien rakentaminen pitäisi olla melko helppoa.

11 YHTEENVETO

Työn tarkoituksena oli ohjelmoida spektrianalysaattori ohjelmaradiolle. Työn tavoitteena oli rakentaa toimiva ohjelma, joka vastaanottaa radioääniaaltoa käyttäjän ohjelmassa määräämältä taajuudelta ja piirtää tietokoneen ruudulle ääniaaltosignaalin spektrin. Ohjelmalle asetetut tavoitteet saavutettiin.

Tässä työssä käsiteltiin melkoisesti spektrianalysaattoreihin liittyvää tekniikkaa ja toimintatapoja ohjelman toiminnan kannalta. Ensiksi tarkasteltiin analogisen radion eroja ohjelmaradioon verrattuna ja tämän jälkeen hahmoteltiin spektrianalysaattorit kolmeen kategoriaan. Tarkastelussa oli myös ohjelmaradion vaatima vastaanottokytkentä, jossa katsottiin erityisesti I- ja Q-signaalien luomista. Lisäksi tarkasteltiin signaalin muuttamista aikatasosta taajuustasoon.

Ohjelmointiosuudessa tarkasteltiin ohjelman kriittisimpien kohtien ratkaisuja. Tärkeimmiksi nähtiin kello-oskillaattorin perustaajuuden syöttö, äänikortilta äänisignaalin lukeminen ja spektrin piirtoon kuuluva tapahtumasarja. Ohjelmakoodiin listaamisen sijaan käytettiin lohkokaavioita, joiden avulla on helppo havainnoida kunkin ohjelmakohdan toiminta. Liitteenä ovat ohjelman kolmen tärkeimmän osuuden koodit.

Vaikka ohjelma saatiin toimimaan vaatimusten mukaisesti, sitä voisi edelleen kehittää kattamaan esimerkiksi äänentoisto.

LÄHTEET

1. Burkhard Kainka, Software Defined Radio mit USB-Interface, luettu 9.4.2008, [www-dokumentti] http://www.b-kainka.de/sdrusb.html

2. Youngblood, Gerald, A Software-Defined Radio for the Masses, Part 1, luettu 5.3.2007, [pdf-dokumentti] http://www.arrl.org/tis/info/pdf/020708qex013.pdf

3. Blossom, Eric, Software Radio, luettu 7.3.2008, [www-dokumentti] http://comsec.com/software-radio.html

4. Marcus, Mike, Linux, Software Radio and the Radio Amateur, luettu 7.3.200 8, [pdf-dokumentti] http://www.arrl.org/tis/info/pdf/0210033.pdf

5. Finish Software Radio Programme, luettu 7.3.2007, [pdf-dokumentti] http://www.mil.fi/laitokset/pvtt/fsrpbook.pdf

6. Reed, Jeffrey H, Software Radio: A Modern Approach to Radio Engineering, luettu 7.3.2008, päivitetty 15.5.2002, [pdf-dokumentti] http://vig.pearsoned.com/samplechapter/0130811580.pdf

7. Fundamentals of the Real-time Spectrum Analysis, luettu 8.3.2008, [pdf-dokumentti] http://www.tek.com/Measurement/App_Notes/37_17249/eng/37W_17249_1.pdf

8. Smith, Steven W, The Scientist and Engineer's Guide to Digital Signal Processing, luettu 5.3.2008, [www-dokumentti] http://www.dspguide.com/ch12/2.htm

9. 1 PLL In-System Programmable Clock Generator with Individual 16K EEPROM, Document #: 38-07440 Rev. *B, luettu 8.3.2008, päivitetty 30.6.2003, [pdf-dokumentti]

10. D2XX Programmer's Guide, luettu 8.3.2008, päivitetty 2006, [pdf-dokumentti]

LIITTEIDEN LUETTELO

```
/* Laitteeseen (SDR-kytkentä) päästään käsiksi käyttämällä seuraavia
funktioita. Näillä hoidetaan keskusteku USB-sarjaporttimuuntimen kanssa
ja tallennetaan kello-oskillaattorin sisälle haluttu kantataajuus. */

void RAM()// Asetetaan koneen RAM-muistiin haluttu tieto
{
      int Pump = 1;

      Buf_adr = 0;      // aloitetaan taulukon ensimmäisestä alkiosta
      I2Cadr = 210;     // laitteen osoite on D2H

      if (P > 479) Pump = 2;
      if (P > 639) Pump = 3;
      if (P > 799) Pump = 4;

      P0 = P % 2; // P0 on joko 0 tai 1 riippuen jakojäännöksestä

      /* Ptotal = (2(PB + 4) + PO) */
      /* Rekisterin 40 kolmen MSB-bitin arvo on 192 */

      R40 = ((P / 2 - 4) / 256) + 4 * Pump + 192;
      R41 = (P / 2 - 4) & 255;
      R42 = 48 + (128 * P0);      //q+2=50, 10 MHz / 50 = 200 kHz

      if (VFOhigh)
            R42 = 23 + 128 * P0;      //q+2=25, 10 MHz / 25 = 400 kHz

      I2C_Init();       // Alustetaan I2C-linjat
      Start();          // Aloitus bitti

      Use(I2Cadr);      // Laiteosoite eli 210 = D2H RAM:ssa
      Use(64); // Rekisteriosoite on 40H eli Charge Pump and PB counter
      Use(R40);
      Use(R41);
      Use(R42);
      Stop();           // pysäytysbitti

      Start();          // Aloitus bitti
      Use(I2Cadr);      // Laiteosoite eli 210 = D2H RAM:ssa
      // Rekisteriosoite on 0CH eli DIV1SRC mux and DIV1N divider
      Use(12);
      Use(Div1N);       // DIV1N=50
      Stop();                 // pysäytysbitti

      Start();          // Aloitus bitti
      Use(I2Cadr);      // Laiteosoite eli 210 = D2H RAM:ssa
      Use(19);          // Rekisteriosoite on 13H, Input load capacitor
      Use(xtal);        // Säätökondensaattoreiden arvo
      Stop();           // pysäytysbitti

      Open_USB_Device();
      Set_USB_Device_BitMode(255, 1);
      Set_USB_Device_BaudRate(38400);
      Write_USB_Device_Buffer(Buf_adr);
      Close_USB_Device();
}
```

```
// suljetaan laite
void Close_USB_Device()
{
	ftResult = FT_Close(ftHandle);

	if(ftResult != FT_OK)
		MessageBox(NULL, L"Laitteen sulkeminen epäonnistui",
		L"VIRHE", MB_OK);
}

// asetetaan laitteen baudinopeus
void Set_USB_Device_BaudRate(int baudrate)
{
	ftResult = FT_SetBaudRate(ftHandle, baudrate);

	if(ftResult != FT_OK)
		MessageBox(NULL, L"Baudinopeuden asettaminen epäonnistui",
		L"VIRHE", MB_OK);
}



// asetetaan laitteen bittimoodi
void Set_USB_Device_BitMode(unsigned char Mask, unsigned char Enable)
{
	ftResult = FT_SetBitMode(ftHandle, Mask, Enable);

	if(ftResult != FT_OK)
		MessageBox(NULL, L"Bittimoodin asettaminen epäonnistui",
		L"VIRHE", MB_OK);
}

// avataan laite
void Open_USB_Device()
{
	DWORD DevIndex = 0;
	ftResult = FT_Open(DevIndex, &ftHandle);

	if(ftResult == FT_DEVICE_NOT_FOUND)
	{
		MessageBox(NULL, L"Laitetta ei ole kytketty!", L"VIRHE",
		MB_OK);
	}
}

// kirjoitetaan laitteeseen
void Write_USB_Device_Buffer(int Write_Count)
{
	DWORD Write_Result;
	ftResult = FT_Write(ftHandle, &FT_Out_Buffer, Write_Count,
	&Write_Result);

	if(ftResult != FT_OK)
		MessageBox(NULL, L"Kirjoittaminen epäonnistui!", L"VIRHE",
		MB_OK);
}
```

```
/* SCL ja SDA ovat CY27EE16-piirin tulot johon FT232R-piirin TXT- ja
RTX-ulostulot menevät. Näillä funktioilla pyöritetään sarjaliikennettä.
*/

void SCL(int d)   // Onko kellolinja 0 vai 1
{
      if(d == 0)
            FT_port = FT_port & (255 - 2); // toinen bitti on aina 0
      else
            FT_port = FT_port | 2;         // toinen bitti on aina 1

      FT_Out_Buffer[Buf_adr] = FT_port;
      Buf_adr = Buf_adr + 1;
}

void SDA(int d)   // Onko datalinja 0 vai 1
{
      if(d == 0)
            FT_port = FT_port & (255 - 1); // eka bitti on aina 0
      else
            FT_port = FT_port | 1;         // eka bitti on aina 1

      FT_Out_Buffer[Buf_adr] = FT_port;
      Buf_adr = Buf_adr + 1;
}

// Alustetaan I2C-linjat eli kummatkin nostetaan 1:ksi
void I2C_Init()
{
      SCL(1);
      SDA(1);
}

// Aloitusbitti: kello lasketaan nollaan kun datalinja on 0:ssa
void Start()
{
      SDA(0);
      SCL(0);
}

/* Pysähdys-bitti: molempien linjojen ollessa alhaalla ensin nousee
kello ja tämän jälkeen data. */

void Stop()
{
      SCL(0); SDA(0);
      SCL(1);
      SDA(1);
}
```

```
/* Muuttaa annetun luvun SDA / SCL-muotoon:

Luvun 210 binäärimuoto on 11010010. Tälle tehdään and-bittioperaatio
jota silmukan joka kohdassa siirretään vertailulukui bittimuodossa
yhden verran oikeilla kunnes koko luku on saatu SDA / SCL-muotoon.

11010010                11010010
10000000                00100000
10000000                00000000
=                               =
sama eli SDA(1)         eri eli SDA(0) */

bool Use(unsigned char convnum)
{
      int bitdiv = 128;

      for(int n = 1 ; n<8 ; n++)
      {
            if((convnum & bitdiv) == bitdiv)
                  SDA(1);
            else
                  SDA(0);

            SCL(1);
            SCL(0);
            bitdiv = bitdiv / 2;
      }

      SDA(1);
      SCL(1);
      SCL(0);

      return true;
}
```

```
/* Äänenkaappausfunktiot, joilla ohjataan äänenkaappauksen alustusta,
käynnistystä, puskurin täyttymistä ja lopetusta. */

/* Äänenkaappauksen aloitusfunktio, jota kutsutaan OK-nappia
painettaessa */

void InitAudioCapture()
{

      // aloitetaan nauhoitus ja varataan puskureille tilaa

      pBuffer1 = (PBYTE)malloc (INP_BUFFER_SIZE) ;
      pBuffer2 = (PBYTE)malloc (INP_BUFFER_SIZE) ;

      /* jos tilanvaraus epäonnistui, tyhjennetään puskurit ja annetaan
      varoitus */

      if (!pBuffer1 || !pBuffer2)
      {
            if (pBuffer1) free (pBuffer1) ;
            if (pBuffer2) free (pBuffer2) ;

            MessageBeep (MB_ICONEXCLAMATION);
            MessageBox (hwnd, szMemError, L"VIRHE", MB_ICONEXCLAMATION
            | MB_OK) ;
            return TRUE ;
      }

      // täytetään äänikortin kaappaukseen käytettävä äänimuotorakenne
      waveform.wFormatTag = WAVE_FORMAT_PCM ;    // äänimuoto on PCM
      waveform.nChannels  = 2;                   // stereo
      waveform.nSamplesPerSec = 44100;           // näytteistysnopeus
      waveform.nAvgBytesPerSec = 44100 * 4;      // tavumäärä sekunnissa
      waveform.nBlockAlign     = 4;
      waveform.wBitsPerSample  = 16 ;            // näytteen bittikoko
      waveform.cbSize          = 0 ;

      // avataan äänikortti, jos ei onnistu annetaan ilmoitus

      if (waveInOpen (&hWaveIn, WAVE_MAPPER, &waveform,
      (DWORD_PTR)hwnd, 0,  CALLBACK_WINDOW))
      {
            free (pBuffer1) ;
            free (pBuffer2) ;
            MessageBeep (MB_ICONEXCLAMATION) ;
            MessageBox (hwnd, szOpenError, L"VIRHE", MB_ICONEXCLAMATION
            | MB_OK) ;
      }

      /* täytetään puskurien otsikkorakenteet ja annetaan ne
      valmistelufunktiolle */

      pWaveHdr1->lpData          = (LPSTR)pBuffer1 ;
      pWaveHdr1->dwBufferLength  = INP_BUFFER_SIZE ;
      pWaveHdr1->dwBytesRecorded = 0 ;
      pWaveHdr1->dwUser          = 0 ;
      pWaveHdr1->dwFlags         = 0 ;
      pWaveHdr1->dwLoops         = 1 ;
      pWaveHdr1->lpNext          = NULL ;
```

```
      pWaveHdr1->reserved        = 0 ;

      waveInPrepareHeader (hWaveIn, pWaveHdr1, sizeof (WAVEHDR)) ;

      pWaveHdr2->lpData          = (LPSTR)pBuffer2 ;
      pWaveHdr2->dwBufferLength  = INP_BUFFER_SIZE ;
      pWaveHdr2->dwBytesRecorded = 0 ;
      pWaveHdr2->dwUser          = 0 ;
      pWaveHdr2->dwFlags         = 0 ;
      pWaveHdr2->dwLoops         = 1 ;
      pWaveHdr2->lpNext          = NULL ;
      pWaveHdr2->reserved        = 0 ;

      waveInPrepareHeader (hWaveIn, pWaveHdr2, sizeof (WAVEHDR)) ;

}

/* Äänenkaappauksen käsittelyssä tarvitsee hoitaa kolmen viestin
sisältö: aloistus, puskurin täyttyminen ja lopetus. */

case MM_WIM_OPEN: // äänikortin luku alkaa

      // varataan tallennuspuskurille uudestaan tilaa
      pSaveBuffer = (PBYTE)realloc (pSaveBuffer, 1) ;

      // lisätään puskurit käsittelyyn
      waveInAddBuffer (hWaveIn, pWaveHdr1, sizeof (WAVEHDR)) ;
      waveInAddBuffer (hWaveIn, pWaveHdr2, sizeof (WAVEHDR)) ;

      // aloitetaan näytteistys
      bRecording = TRUE ;
      bEnding = FALSE ;
      dwDataLength = 0 ;
      waveInStart (hWaveIn) ;

      return TRUE ;

case MM_WIM_DATA: // äänikortilta lukupuskuri on täynnä

/* jaetaan puskuri kahteen taulukkoon, joista toinen sisältää
reaaliosan ja toinen imaginaariosan */

      for(int i = 0 ; i < 4095 ; i += 2)
      {
            Real[i/2] = ((PWAVEHDR)lParam)->lpData[i];
            Imag[i/2] = ((PWAVEHDR)lParam)->lpData[i + 1];
      }

      // tarkastetaan onko äänikortilta luku päättynyt varmuudeksi
      if (bEnding)
      {
            waveInClose (hWaveIn) ;
            return TRUE ;
      }

      // lähetetään uusi puskuri käsiteltäväksi
      waveInAddBuffer (hWaveIn, (PWAVEHDR) lParam, sizeof (WAVEHDR)) ;

      return TRUE ;
```

```
case MM_WIM_CLOSE:      // äänikortilta luku loppuu

/* vapautetaan äänikortilta lukuun käytettyjen otsikoiden ja puskurien
tilanvaraus */

      waveInUnprepareHeader (hWaveIn, pWaveHdr1, sizeof (WAVEHDR)) ;
      waveInUnprepareHeader (hWaveIn, pWaveHdr2, sizeof (WAVEHDR)) ;

      free (pBuffer1) ;
      free (pBuffer2) ;

      // ei sallita äänikortilta lukua ja pysäytetään ajastin

      bRecording = FALSE ;

      if (bTerminating)
            SendMessage (hwnd, WM_SYSCOMMAND, SC_CLOSE, 0L) ;

      KillTimer(hwnd, 1);

return TRUE;


/* Nopea Fourier-muunnos aliohjelma. I -ja Q-signaaleiden taulukot
   annetaan sisään ja ohjelma muuttaa taulukon sisällön. */

void FFT(int FFTsize, int real[], int imaginary[])
{
      int i, j, l, k;
      int Nminus1, Ndiv2;     // N - 1 ja N / 2 -muuttujat
      float stageCount;       // vaihemäärä
      int stagePointCount;    // vaiheen pistemäärä;
      int SPCdiv2;            // vaihepistemäärä jaettuna 2
      int ur, ui;
      int ip;
      // i + SPCdiv2
      int sr, si;             // sini -ja kosini-arvot
      int jminus1;            // j - 1
      // säiliöt tilapäisille reaali -ja imaginääri-arvoille
      int tr, ti;
      Nminus1 = FFTsize - 1;  // FFT-kokoa pienennetään yhdellä
      Ndiv2 = FFTsize / 2;    // FFT-koko jaetaan kahdella

      // monta vaihetta käydään läpi, ennenkuin N signaalia koostuu
      // yhdestä pisteestä
      stageCount = log((float)FFTsize) / log((float)2);
      j = Ndiv2;

      for(i=1 ; i<FFTsize-2 ; i++)  // bittien kääntäminen
      {
            // jos iteraattori on pienempi kuin puolet FFT:n koosta
            if(i < j)
            {
                  // otetaan talteen reaali -ja imaginääriarvot
                  puolesta välistä
                  tr = real[j];
                  ti = imaginary[j];
```

```
                    // sijoitetaan puoleenväliin aikaisemmat arvot
                    real[j] = real[i];
                    imaginary[j] = imaginary[i];
                    real[i] = tr; // sijoitetaan alkuun puolenvälin arvot
                    imaginary[i] = ti;
              }

              k = Ndiv2;

              while (k <= j)
              {
                          j = j - k;
                          k = k / 2;
              }

              j = j + k;
        }

        // käydään kaikki vaiheet lävitse
        for( l = 1 ; l <= stageCount ; l++)
        {
              // montako pistettä on tässä vaiheessa
              stagePointCount = (int)pow((float)2, (float)l);

              SPCdiv2 = stagePointCount / 2; // pisteet jaetaan kahtia
              ur = 1;
              ui = 0;
              // sini-kertoimen reaaliosa (SIN)
              sr =  (int)cos(PI / SPCdiv2);
              // sini-kertoinen imaginääriosa (COS)
              si = (int)-sin(PI / SPCdiv2);

              for(j = 1 ; j <= SPCdiv2 ; j++)
              {
                    jminus1 = j - 1;

                    for(i = jminus1 ; i <= Nminus1 ; i +=
                    stagePointCount)  // BUTTERFLY
                    {
                          ip = i + SPCdiv2;
                          tr = real[ip] * ur - imaginary[ip] * ui;
                          ti = real[ip] * ui + imaginary[ip] * ur;

                          real[ip]          = real[i]         - tr;
                          imaginary[ip]     = imaginary[i]    - ti;
                          real[i]           = real[i]         + tr;
                          imaginary[i]      = imaginary[i]    + ti;
                    }

                    tr = ur;
                    ur = tr * sr - ui * si;
                    ui = tr * si + ui * sr;
              }
        }
}
```

```
/* Piirtofunktiot, joilla hoidetaan spektrin piirtämine ruudulle.
   WM_PAINT-viesti lähetetään aina kun on tarpeen päivittää ohjelman
   ikkuna. */

 case WM_PAINT:

      // hankitaan kahva piirtopintaan
      hdc = BeginPaint (hwnd, &ps);

      OnPaint(hdc);

      // vapautetaan piirtopinta ja tuhotaan kynä ja fontti
      EndPaint (hwnd, &ps);
      DeleteObject(hPen);
      DeleteObject(hFont);

      return 0;


void OnPaint(HDC hdc)   // Piirto-aliohjelma
{
      /* piirretään spektrialue: täytetään se valkoiselle ja
         kehystetään mustalla reunuksella */

      FillRect(hdc, &spectrumArea, WHITE_BRUSH);
      FrameRect(hdc, &spectrumArea,
      (HBRUSH)GetStockObject(BLACK_BRUSH));

      // valitaan tekstiksi windowsin oletuksena käyttämä GUI-fontti
      hFont = (HFONT)GetStockObject(DEFAULT_GUI_FONT);
      SelectObject(hdc, hFont);

      // spektrialueen tausta on läpinäkyvä ja taustaväri on valkoinen
      SetBkMode(hdc, TRANSPARENT);
      SetBkColor(hdc, RGB(0, 0, 0));

      // valitaan viivaksi valkoinen piste, jolla piirretään akselit
      hPen = CreatePen(PS_DOT, 1, RGB(0, 0, 0));
      SelectObject(hdc, hPen);

      // piirtää asteikko
      DrawScale();

      // valitaan viivaksi musta kiinteä
      hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
      SelectObject(hdc, hPen);

      // jos äänikortilta luku on sallittu, piirretään spektri
      if(bMeasure)
            DrawSpectrum();

}
```

```
void DrawScale(bool bDecibel)
{
	if(bDecibel)
	{
		TextOut(hdc, 52, 100, szBuffer, wsprintf(szBuffer, L"43"));
		TextOut(hdc, 52, 150, szBuffer, wsprintf(szBuffer, L"40"));
		TextOut(hdc, 52, 200, szBuffer, wsprintf(szBuffer, L"33"));
		TextOut(hdc, 52, 250, szBuffer, wsprintf(szBuffer, L"0"));

		MoveToEx(hdc, cxClient / 2, 50, NULL);
		LineTo(hdc, cxClient / 2, 250);
		MoveToEx(hdc, 50, 250, NULL);
		LineTo(hdc, 1000, 250);
		MoveToEx(hdc, 50, 150, NULL);
		LineTo(hdc, 1000, 150);
		MoveToEx(hdc, 50, 200, NULL);
		LineTo(hdc, 1000, 200);
		MoveToEx(hdc, 50, 100, NULL);
		LineTo(hdc, 1000, 100);
		MoveToEx(hdc, cxClient / 5, 250, NULL);
		LineTo(hdc, cxClient / 5 * 4, 250);

		MoveToEx(hdc, cxClient / 5, 250, NULL);
		LineTo(hdc, cxClient / 5 * 4, 250);
	}
	else  // piirretään log10-asteikko
	{
		TextOut(hdc, 52, 100, szBuffer, wsprintf(szBuffer,
		L"150"));
		TextOut(hdc, 52, 150, szBuffer, wsprintf(szBuffer,
		L"100"));
		TextOut(hdc, 52, 200, szBuffer, wsprintf(szBuffer, L"50"));
		TextOut(hdc, 52, 250, szBuffer, wsprintf(szBuffer, L"0"));

		MoveToEx(hdc, cxClient / 2, 50, NULL);
		LineTo(hdc, cxClient / 2, 250);
		MoveToEx(hdc, 50, 250, NULL);
		LineTo(hdc, 1000, 250);
		MoveToEx(hdc, 50, 150, NULL);
		LineTo(hdc, 1000, 150);
		MoveToEx(hdc, 50, 200, NULL);
		LineTo(hdc, 1000, 200);
		MoveToEx(hdc, 50, 100, NULL);
		LineTo(hdc, 1000, 100);
		MoveToEx(hdc, cxClient / 5, 250, NULL);
		LineTo(hdc, cxClient / 5 * 4, 250);
		MoveToEx(hdc, cxClient / 5, 250, NULL);
		LineTo(hdc, cxClient / 5 * 4, 250);
	}

	// piirretään 10 Hz välein amplitudiasteikko
	for(int i = 0 ; i < 460 ; i+=10)
	{
		MoveToEx(hdc, (cxClient / 2) + i*spectrumScale, 50, NULL);
		LineTo(hdc, (cxClient / 2) + i*spectrumScale, 250);

		MoveToEx(hdc, (cxClient / 2) + -i*spectrumScale, 50, NULL);
		LineTo(hdc, (cxClient / 2) + -i*spectrumScale, 250);
	}
}
```

```
void DrawSpectrum()
{
	// muutetaan aikatasossa olevan arvot taajuustasoon

	FFT(2048, Real, Imag);

	// piirto alkaa keskeltä

	MoveToEx(hdc, (cxClient / 2), 150, NULL);

	// piirretään 700 taulukon "positiivista" arvoa

	for(int i = 0 ; i < 700 ; i ++)
	{
		int temp;

		// muutetaan reaali -ja imaginaariarvot amplitudi arvoksi

		temp = (int)sqrt(pow((double)Real[i], 2) +
		pow((double)Imag[i], 2));

		// muutetaan desibeliksi jos tarpeen

		if(bDecibel)
			temp = (int)20 * (int)log((double)temp);

		/* jos piirtomuotona on palkki, siirretään se x-akselin
		oikeaan kohtaan */

		if(bPalkki)
			MoveToEx(hdc, (cxClient / 2)+(i*spectrumScale), 250,
			NULL);

		// piirretään spektrialueen "positiivinen" puoli

		LineTo(hdc, (cxClient / 2)+(i*spectrumScale), -temp + 250);

		/* spektrin piirto riippuu sen skaalasta - tässä voitaisiin
		käyttää PolyLineTo-funktiota nopeutukseen */

		if(spectrumScale > 4)
		{
			TextOut(hdc, (cxClient / 2) + 10*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 100));
			TextOut(hdc, (cxClient / 2) + 20*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 200));
			TextOut(hdc, (cxClient / 2) + 30*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 300));
			TextOut(hdc, (cxClient / 2) + 40*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 400));
			TextOut(hdc, (cxClient / 2) + 50*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 500));
			TextOut(hdc, (cxClient / 2) + 60*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 600));
			TextOut(hdc, (cxClient / 2) + 70*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 700));
			TextOut(hdc, (cxClient / 2) + 80*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 800));
			TextOut(hdc, (cxClient / 2) + 90*spectrumScale - 15,
```

```
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 900));
                    TextOut(hdc, (cxClient / 2) + -10*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset - 100));
                    TextOut(hdc, (cxClient / 2) + -20*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset - 200));
                    TextOut(hdc, (cxClient / 2) + -30*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset - 300));
                    TextOut(hdc, (cxClient / 2) + -40*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset - 400));
                    TextOut(hdc, (cxClient / 2) + -50*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset - 500));
                    TextOut(hdc, (cxClient / 2) + -60*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset - 600));
                    TextOut(hdc, (cxClient / 2) + -70*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset - 700));
                    TextOut(hdc, (cxClient / 2) + -80*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset - 800));
                    TextOut(hdc, (cxClient / 2) + -90*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset - 900));
            }
            else if(spectrumScale > 1)
            {
                    TextOut(hdc, (cxClient / 2) + 20*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 200));
                    TextOut(hdc, (cxClient / 2) + 40*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 400));
                    TextOut(hdc, (cxClient / 2) + 60*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 600));
                    TextOut(hdc, (cxClient / 2) + 80*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 800));
                    TextOut(hdc, (cxClient / 2) + 100*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 1000));
                    TextOut(hdc, (cxClient / 2) + 120*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 1200));
                    TextOut(hdc, (cxClient / 2) + 140*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 1400));
                    TextOut(hdc, (cxClient / 2) + 160*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 1600));
                    TextOut(hdc, (cxClient / 2) + 180*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 1800));
                    TextOut(hdc, (cxClient / 2) + -20*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset - 200));
                    TextOut(hdc, (cxClient / 2) + -40*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset - 400));
                    TextOut(hdc, (cxClient / 2) + -60*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset - 600));
                    TextOut(hdc, (cxClient / 2) + -80*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset - 800));
                    TextOut(hdc, (cxClient / 2) + -100*spectrumScale -
15, 270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset -
1000));
                    TextOut(hdc, (cxClient / 2) + -120*spectrumScale -
15, 270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset -
1200));
                    TextOut(hdc, (cxClient / 2) + -140*spectrumScale -
15, 270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset -
1400));
                    TextOut(hdc, (cxClient / 2) + -160*spectrumScale -
15, 270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset -
1600));
```

```
                  TextOut(hdc, (cxClient / 2) + -180*spectrumScale -
15, 270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset -
1800));
                  TextOut(hdc, (cxClient / 2) + -200*spectrumScale -
15, 270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset -
2000));
                  TextOut(hdc, (cxClient / 2) + -220*spectrumScale -
15, 270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset -
2200));
            }
            else if(spectrumScale == 1)
            {
                  TextOut(hdc, (cxClient / 2) + 50*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 500));
                  TextOut(hdc, (cxClient / 2) + 100*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 1000));
                  TextOut(hdc, (cxClient / 2) + 150*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 1500));
                  TextOut(hdc, (cxClient / 2) + 200*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset + 2000));
                  TextOut(hdc, (cxClient / 2) + -50*spectrumScale - 15,
270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset - 500));
                  TextOut(hdc, (cxClient / 2) + -100*spectrumScale -
15, 270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset -
1000));
                  TextOut(hdc, (cxClient / 2) + -150*spectrumScale -
15, 270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset -
1500));
                  TextOut(hdc, (cxClient / 2) + -200*spectrumScale -
15, 270, szBuffer, wsprintf(szBuffer, L"%i", P*(Step/4) + offset -
2000));
            }
      }
      MoveToEx(hdc, cxClient / 2, 150, NULL);

      /* piirretään taulukon 700 "negatiivista" arvoa (keskitaajuuden
      vasenpuoli) */
      for(int i = 2047 ; i > 1747 ; i--)
      {
            int temp;

            temp = (int)sqrt(pow((double)Real[i], 2) +
            pow((double)Imag[i], 2));

            if(bDecibel)
                  temp = 20 * (int)log((double)temp);

            if(bPalkki)
                  MoveToEx(hdc, cxClient / 2 + ((i-2047)*
                  spectrumScale), 250, NULL);

            LineTo(hdc, (cxClient / 2) + ((i-2047)* spectrumScale), -
            temp + 250);

      }
      // mittaus suoritettu, WM_TIMER antaa luvan uudestaan
      bMeasure = false;
}
```