TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Thomas Johann Seebecki elektroonikainstituut

IEE40LT

Ingel-Marie Hiiekivi 134341IALB

EKSPERIMENTAALNE ANDMESALVESTUSEGA NAHKHIIREDETEKTOR MIKROPROTSESSORI ATSAMD21G18 BAASIL

Bakalaureusetöö

Juhendaja: lektor Reeno Reeder, PhD

Kaasjuhendaja: teadur Veljo Sinivee, PhD

Tallinn 2016

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Ingel-Marie Hiiekivi

06.06.2016

Annotatsioon

Antud töö eesmärgiks on signaali salvestusvõimalusega nahkhiirte detektori skeemi väljatöötamine. Valmistatud seade baseerub mikroprotsessoril ATSAMD21G18 ja kasutab info salvestamiseks SD kaardi moodulit. Salvestatud andmeid saab kasutada hilisemaks signaalitöötluseks või analüüsiks. Töö lõpus on esitatud disainitud seadme töö tõhusust iseloomustavad katsetulemused.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 30 leheküljel, 9 peatükki, 16 joonist, 2 tabelit.

Abstract

Experimental Bat Detector with Data Logging Based on ATSAMD21G18 Microprocessor

Present thesis explores methods for making bat high-frequency echolocation signals audible to human researchers. Three widely used approaches (super heterodyne detection, frequency division and time expansion) are covered explaining their working principles on typical circuits. An enhanced experimental bat detector design, which is the main goal of present thesis, is then explained in more detail covering instruments analog-, digital and power supply sections. Signal processing (Analog to Digital conversion) is carried out in ATSAMD21G18 microprocessor on an "Arduino M0" controller board. Processed data is recorded on a separate SD-card shield enabling later analysis and identification of bat species.

The thesis is in Estonian and contains 30 pages of text, 9 chapters, 16 figures, 2 tables.

Lühendite ja mõistete sõnastik

ADC - *Analog to Digital Converter* – seade, mis muundab analoogsignaali ehk pideva signaali diskreetsesse vormi ehk digitaalsignaaliks;

Ajavenitus - *Time Expansion,* protsess, kus pikendatakse signaali kestvust;

Diskreetimissagedus - *Sampling Frequency*, suurus, mis määrab pideva signaali diskreetimisel võendite arvu sekundis;

Kajalokaator - bioloogiline sonar loomadel, mida kasutatakse ümbruskonnas orienteerumiseks ja omavaheliseks suhtluseks;

Kajalokatsioon - on mõnedel kõrgematel loomadel (vaalad, delfiinid, nahkhiired, mõned linnud jt) arenenud mingi kindla lainepikkusega heli tekitamine ja selle vastuvõtmine;

KMOP - Komplementaarne transistor, mis koosneb n-MOP ja p-MOP transistorist;

Konverteerimine - muundamine;

Mikroprotsessor - *Microprocessor*, mitmeotstarbeline programmeeritav seade, mis võtab sisendina vastu digitaalse info, töötleb seda vastavalt mälus salvestatud masinakäskudele;

Nahkhiire detektor - seade, mis on valmistatud nahkhiirte poolt tekitatud kõrgsagedusliku heli muundamiseks madalamasageduslikuks;

Ostsillaator - *Oscillator*, on elektrongeneraator, mis tekitab sumbumatuid harmoonilisi võnkumisi ehk siinusvõnkumisi;

PCB - *Printed Circuit Board*, trükkplaat;

Prescaler - taktijagur, mis on mõeldud väiksema taktikiiruse saavutamiseks;

Sagedusega jagamine - *Frequency Division*, sageduse jagamine mingi kindla arvuga madalama sageduse saamiseks;

Segusti - *Mixer*, on mittelineaarne ahel, kasutatakse sageduste nihutamiseks ühest sagedusalast teise, et lihtsustada signaalide ülekandmist või edasist signaalitöötlust;

SPI - Serial Peripheral Interface, on välisliidese ja välisseadmete vaheline andmevahetuse standard;

Superheterodüünvastuvõtja - *Superheterodyne receiver*, raadiovastuvõtja tüüp;

Taktsagedus - *clock rate*, protsessori kiirust iseloomustav tegur;

Tippsagedus - parima kuuldavuse sagedus;

TTL - transistor-transistor loogika;

Ultraheliandur - *Ultrasonic transducer*, ultraheli sagedusalas töötav andur;

Välkmälu - *Flash memory*, elektriliselt kustutatav ja programmeeritav mälu;

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tihtilugu arvatakse ekslikult, et nahkhiiri on ainult ühte liiki ning paljudel inimestel on nende suhtes negatiivne suhtumine. Nahkhiirte harjumuste kohta on vähe informatsiooni ning nende käitumist ei osata siiani täielikult põhjendada. Teada on, et nahkhiired, kes kuuluvad käsitiivaliste hulka on ainukesed tõeliselt lendavad imetajad. Eestis on kõik nahkhiired looduskaitse all. Toitumise järgi saab nahkhiiri jaotada kolmeks: herbivoorid ehk puuvilja- ja nektaritoidulised nahkhiired; karnivoorid, kes söövad putukaid ja teisi selgrootuid, ning parasiitse eluviisiga nahkhiired, kes toituvad teiste imetajate ja lindude verest. Kõik Eesti nahkhiired on karnivoorid, kes toituvad putukatest.[11]

Võib eeldada, et käsitiivalised kujunesid välja Eotseenis. Vanimad leiud pärinevad 52 miljoni aasta tagusest ajast. 1960 avastati Eotseenis elanud nahkhiire Icaronycteris jäänused.
Aastal 2008 leiti Lõuna-Dakotast käsitiivalise *Onychonycteris finneyi* hästi säilinud skelett. See nahkhiir suutis hästi lennata, kuid tema sisekõrva ehitus näitas, et ta polnud veel võimeline kajalokatsiooniks. See kinnitab hüpoteesi, et lennuvõime kujunes nahkhiirtel välja enne kajalokatsiooni. *Onychonycteris finneyi* nahkhiirel oli igal viiel varbal küünis, erinevalt tänapäeva nahkhiirtest, kel on kõige rohkem 2 küünist ühel jalal. Tal olid pikemad tagajalad ja lühemad käed ning sellega ta meenutas rohkem giboneid. Ta oli peopesasuurune ja tal olid märksa väiksemad tiivad kui tänapäevastel nahkhiirtel. Arvatavasti ei lennanud ta nii palju ja kiiresti kui tänapäevased nahkhiired, vaid üksnes aeg-ajalt puult puule. Enamjaolt ronis ta mööda puid ringi ja rippus puuokste küljes. Lendamise ajal ei vehkinud ta pidevalt tiibadega, vaid aeg-ajalt liugles õhus.[8]

Käsitiivaliste teine iseärasus on nende orienteerumine ruumis. Juba 1793 avastas itaalia uurija Lazzaro Spallanzani pärast mitmeid hoolikalt kontrollitud katseid, et väike-käsitiivalised suudavad vabalt lennata pimedas toas, kuid kakud on samades tingimustes täiesti abitud. Sealjuures lendasid suletud silmadega nahkhiired niisama hästi kui nägijad.[8]

1794 kinnitas šveitsi bioloog C. Jurine Spallanzani katseid ja avastas uue asjaolu: kui nahkhiire kõrvad olid vahaga kinni korgitud, siis muutus loom lennul abituks ja põrkus vastu takistusi. Jurine oletas, et nahkhiirtel on kuulmiselundid võtnud endale nägemise ülesanded. Samal aastal kordas neid katseid Spallanzani ja veendus, et need vastavad tõele, kuid teistele teadlastele tundusid need oletused absurdsetena: nad lükati tagasi, naerdi välja ja unustati peagi. Kuulmisteooria tagasilükkamist soodustas G. Cuvier' 1795 avaldatud taktiilsusteooria, mille kohaselt nahkhiired orienteeruvad pimedas kompimise kaudu, või, nagu hiljem täpsustati, kuuenda meele ehk kaugkompimise teel. Usk taktiilsusteooriasse vältas kogu maailmas üle 110 aasta.[8]

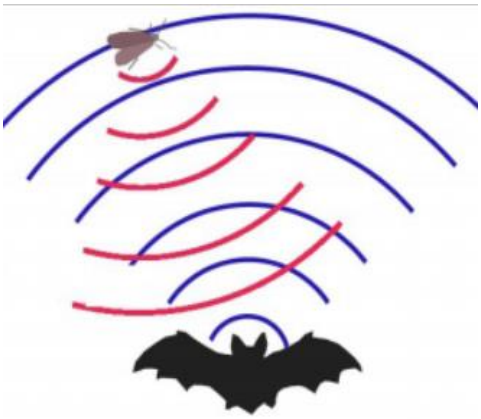
1912 tuli leiutaja Hiram Maxim ja 1920 inglise neurofüsioloog H. Hartridge mõttele, et "kõrvadega nägemist" saab seletada kajalokatsiooniga. Ka nende hüpoteesile ei pööratud tähelepanu.[8]

Alles 1938 tegi Harvardi ülikooli töötaja Griffin kindlaks, et leetlendlased ja hiid-nahkhiired tekitavad hulganisti ultrahelisid, mis on vahemikus 30–70 kHz. Niisuguseid helisid ei suuda inimene tajuda. Ühtlasi tuvastati, et nahkhiired tekitavad neid helisid katkendlike impulssidena, mis kestavad 10–20 ms ja mille sagedus vastavalt olukorrale muutub.[8]

Antud bakalaureuse töö eesmärk on analüüsida juba olemasolevaid lahendusi ning selle põhjal täiustatud lahenduse disainimine. Saadud lahendus peaks võimaldama sisendsignaali salvestamist reaalajas ning omama suuremat kasutusmugavust. Antud seade võib kasulikuks osutada määramaks kindlaks nahkhiirte eriliike hilisema signaalitöötluse abil.

2 Nahkhiired looduses

Nahkhiired kuuluvad keelikloomade hõimkonda, imetajate klassi ning käsitiivaliste seltsi. Väike-käsitiivalised kasutavad orienteerumiseks kajalokatsiooni, kuid suur-käsitiivaliste hulgas tarvitavad seda vaid mõned liigid. Kajalokatsiooni kasutavad nahkhiired on võimelised helide abil "nägema" nagu vaalad ja delfiinidki. Käsitiivalised tekitavad ning saadavad oma nina ja suu abil välja orienteerumiseks kasutatavat kõrge sagedusega ultraheli. Heli tagasipeegeldumise aeg annab teada eseme kauguse nahkhiirest. Ümbrusest tagasi peegeldunud heli tajuvad nahkhiired kõrvadega, selle tõttu ongi kajalokatsiooni kasutavatel väike-käsitiivalistel suured ning keeruka ehitusega kõrvad. Kajalokatsiooni nähtus esineb mõnedel kõrgemalt arenenud loomadel, ning seisneb mingi kindla lainepikkusega heli tekitamisel ja selle vastuvõtmisel. Kajalokatsiooni põhimõte seisneb emiteeritud heli tagasipeegeldumisel kajana ning peegeldatud heli vastuvõtmisel ja töötlemisel. [9]



Joonis 1. Nahkhiire kajalokatsiooni illustratsioon[1]

Antud joonisel on kujutatud nahkhiirt tegemas kajalokatsiooni abil kindlaks putuka suurust ja asukohta keskkonnas. Väiksemate putukate püüdmiseks kasutatakse kõrgemaid helisid ja suuremate püüdmiseks madalamaid helisid.[12]

Peale objektide kauguse kindlakstegemist kasutavad nahkhiired kajalokatsiooni ka suhtluseks liigikaaslastega.[11]

2.1 Eestis leiduvad nahkhiired ja nende kajalokaatorite sagedused

Eestis leiduvate nahkhiirte kajalokaatorite tippsagedused jäävad vahemikku 20-55kHz. Järgnevas tabelis toon välja Eestis talvituvate nahkhiirte kajalokaatorite sagedused ja tippsagedused. Tippsageduseks nimetatakse tugevaima helirõhuga piirkonda kajalokaator-impulsi sagedusvahemikus. See on oluline määramistunnus pikkade impulsside korral.

LIIK	Sagedusvahemik	Tippsagedus
Tiigilendlane	28-74kHz	36kHz
Veelendlane	32-85kHz	45-50kHz
Tõmmulendlane	32-103 kHz	52kHz
Habelendlane	34-102kHz	53kHz
Natterilendlane	23-115kHz	53kHz
Pruun-suurkõrv	27-56kHz	45kHz
Põhja-nahkhiir	30-40kHz	35kHz

Tabel 1. Eestis levivate nahkhiireliikide kajalokaatorite sagedused

3 Ultraheli muundamistehnikad nahkhiirte detektorites

Vaadeldes tabelit 1 on näha, et enamus nahkhiirte kajalokaatorite töösagedusi on inimesele kuuldamatus sagedusvahemikus. Inimene ei kuule 20kHz.st kõrgemaid helisagedusi. Nende kuuldavaks muutmiseks tuleb sagedus nihutada inimesele kuuldavasse sagedusvahemikku, milleks on 16Hz-20kHz. Selleks on ehitatud mitmeid erinevaid põhimõtteid kasutavaid nahkhiiredetektoreid. Kõige levinumad printsiibid on superheterodüünvastuvõtja, sageduse jagamine ja ajavenitus. Igal meetodil on oma eelised ja puudused, seega kasutavad mõned nahkhiiredetektorid samaaegselt kahte või isegi kolme üksteisest sõltumatut muundamistehnikat.

3.1 Superheterodüünvastuvõtja

Superheterodüünvastuvõtja kasutamine on kõige levinum ultraheli signaali konverteerimismeetod nahkhiiredetektorites. See on kitsasriba tehnika, mis tähendab et korraga on võimalik audiosignaaliks muundada vaid väga väike osa kogu sagedusvahemikust. Ribalaius mille vahemikus olevad impulsse konverteeritakse määratakse nahkhiiredetektoris oleva filtriga ja selle suurus jääb tavaliselt umbes 10kHz lähedale. Seadistades näiteks kesksageduseks 40kHz muudetakse see kuuldavaks sageduseks vahemikus 35-45kHz.[13], [14]

Superheterodüünvastuvõtjaga detektori elektriskeemi kuulub signaaligeneraator, mida nimetatakse sisemiseks ostsillaatoriks, mille sagedus on muudetav kasutaja poolt. Segustis korrutatakse generaatori sagedus *f0* ultraheli signaali sagedusega *fs*. Saadud tulemus sisaldab kahte sageduskomponenti. *fs-f0* ja *fs+f0*. Viimane komponent kõrvaldatakse suunates signaali läbi madalpääsfiltri. Juhul kui *fs* ja *f0* ei erine üksteisest liialt, siis tulemus *fs-f0* on kuuldav signaal. Näide: Kasutaja seadistab ostsillaatori 40kHz ja sisend signaal on 42kHz siis 42-40 = 2kHz ning see on kuuldav. [13], [14]

Sellised detektorid on võimelised avastama ka väga nõrku signaale. Samuti on müratase madal, sest kuuldavaks tehakse korraga väike osa ultraheli sagedusvahemikust ja seega satub ka väljundisse väiksem osa müra. [13], [14]

3.2 Sagedusega jagamise põhimõtet kasutav detektor

Võrreldes superheterodüünvastuvõtjaga, muudetakse sagedusega jagamisel korraga kogu siseneva ultrahelisignaali sagedusvahemiku. Algne signaal võimendatakse piisava tasemeni TTL või KMOP loogikalülituste rakendamiseks, kus toimub kogu vastu võetud signaali jagamine mingi kindla arvuga nt 10-ga. Seega näiteks 40kHz sagedusega nahkhiire signaal saavutab sageduse 4 kHz ehk on inimesele kuuldavas sagedusvahemikus ning samuti jagatakse 10-ga ka teiste nahkhiirte poolt tekitatud helid mõnel muul sagedusel. Seega saab kuulda kõikide ümbruskonnas olevate nahkhiirte poolt tekitatud heli korraga ühendades seadmega kõrvaklapi väljundi (kui nad pärast jagamist ikka kuuldevahemikku kuuluvad). Kuna tavaliselt on detektorseadmes kasutusel rohkem kui üks nendest põhimõtetest, siis lisaks sagedusega jagamisele võib kasutada lisaks ka näiteks eelnevalt mainitud heterodüünvastuvõtjat määramaks isendi lokaatori tegelikku töösagedust. Sel viisil suundub heterodüünvastuvõtja poolt muundatud heli teise kõrvaklapi väljundisse.[13], [14]

.

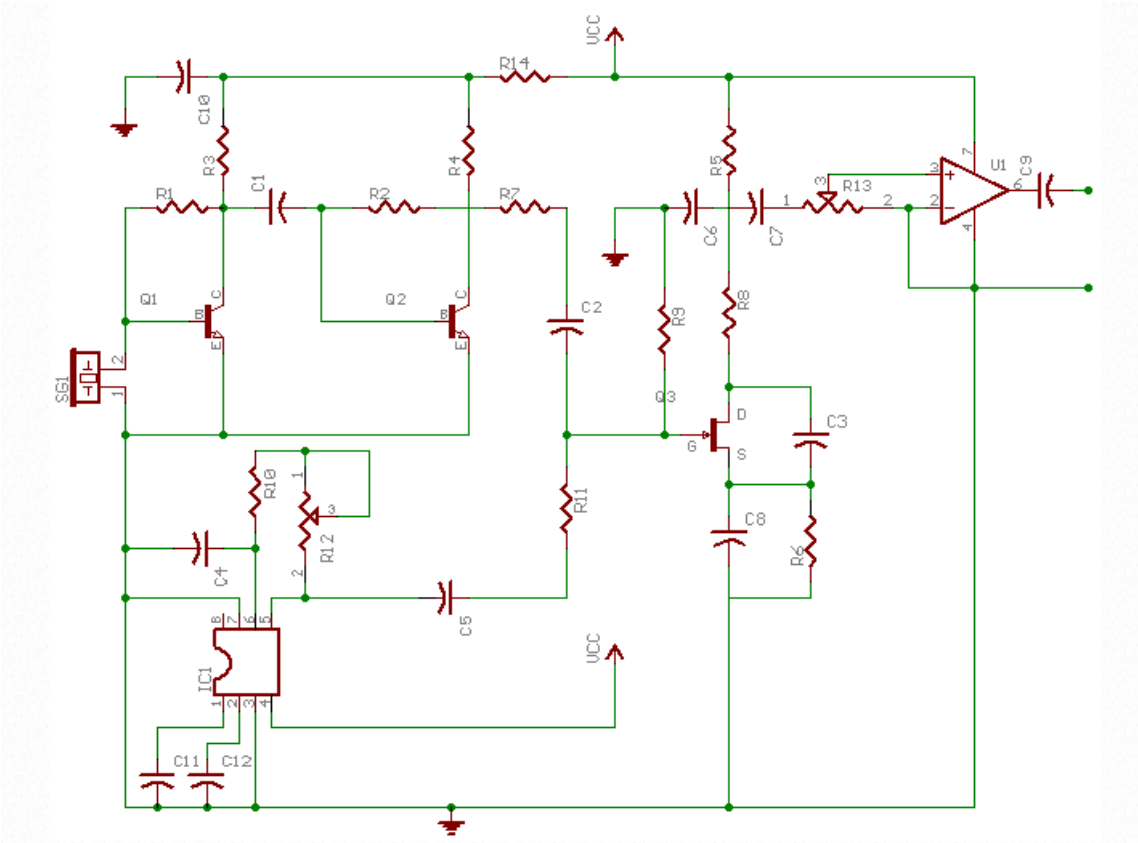
3.3 Ajavenitus

Ajavenitus meetod põhineb sellel, et ultraheli signaal salvestatakse lairiba salvestusseadmega ning taasesitatakse seejärel aeglasemal kiirusel. Saadav signaal on ajaliselt pikem ja madalama sagedusega kui algsignaal. Signaalitöötlus ei toimu reaalajas, vaid see digitaliseeritakse ja osa sellest salvestatakse vajadusel mällu. Selliselt saadud signaale on mugavam hiljem analüüsida. [13], [14]

Signaalide venitamine võimaldab kuulata rohkem ja detailsemalt kui teised meetodid. Et kasutada ajavenitust veelgi tõhusamalt võib seda kombineerida ka teiste eelnimetatud detektoritega. Sel juhul leitakse huvipakkuv isend, salvestatakse tema poolt tekitatud heli ning uuritakse hiljem ajavenitus meetodil. [13]

4 Tüüpilised skeemilahendused

Järgnevalt analüüsin tüüpiliselt kasutatavate eri liiki detektorite skeemilahendusi.



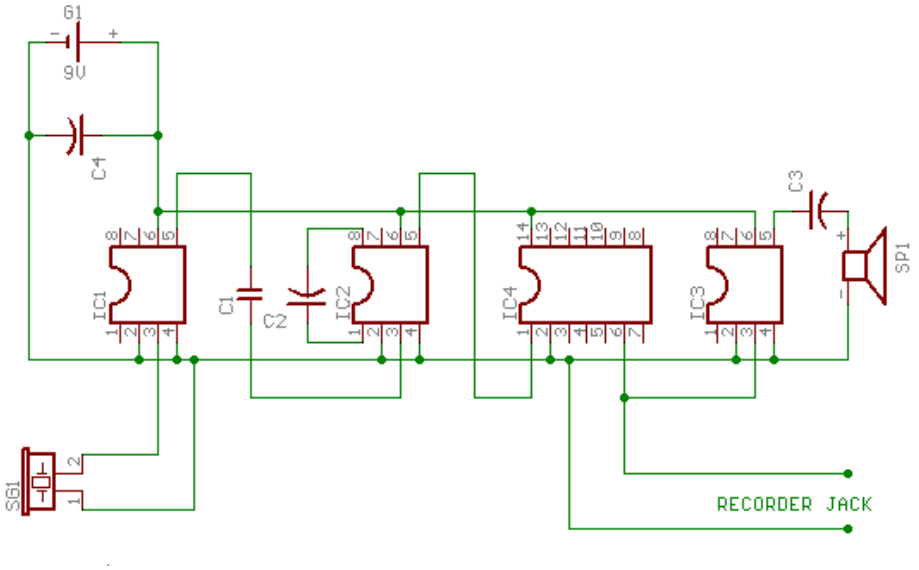
Joonis 2. Näide: Heterodüünvastuvõtja skeem [4]

Joonisel 2 on heterodüündetektor. Mikrofoni SG1 jõudvad signaalid võimendatakse kaheastmelise transistorvõimendiga (elemendid Q1 ja Q2). IC1 on ostsillaator, mille poolt genereeritud signaalisagedus on leitav valemiga

$$f = \frac{1}{1.1 \times C4 \times (R10 + R12)}.$$

Transistoriga Q3 segustatakse mikrofonist tulev võimendatud signaal ja ostsillaatori signaal. Seejärel signaal suundub läbi vahesagedusfiltri, millega lõigatakse ära summa

komponent ning sellele järgneb vahesageduse võimendamine operatsioonivõimendiga U1. Tema väljundis on kuuldav madalamasageduslik signaal. [4]



Joonis 3. Sagedusjagamisega printsiibiga nahkhiiredetektori skeem [7]

Skeem joonisel 3 kasutab sisendis nahkhiirte signaalide püüdmiseks mikrofoni asemel ultraheliandurit. Ultraheliandurit signaali võimendatakse kahes astmes võimenditega IC1 ja IC2. Võimendatud signaal suundub digitaalsesse sagedusjagurisse IC4, kus toimub sageduse jagamine 16.ga. Saadud madalama sagedusega heli on võimalik kuulata kõlarist. Detektori toiteks on 9V patarei. [7]

Antud skeemi positiivsed küljed:

1) skeem on lihtne;
2) vajaminevate komponentide arv on väike.

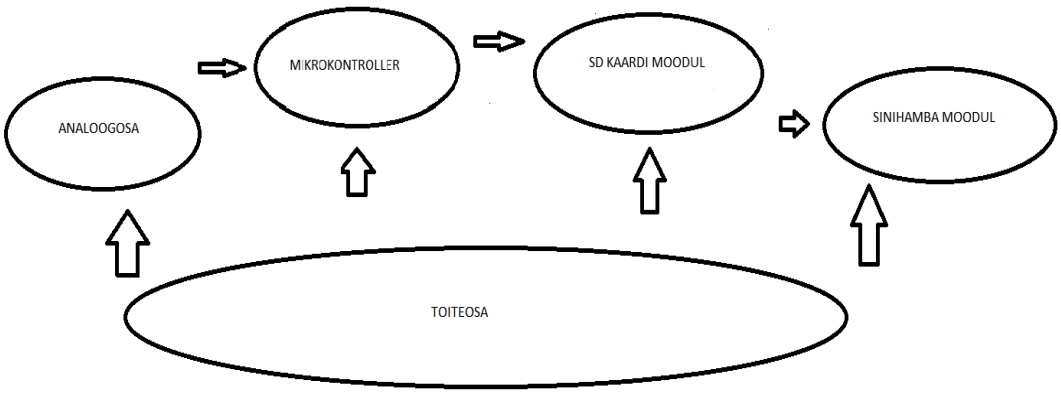
Antud skeemi negatiivsed küljed:

1) Väljundsignaali on võimalik kuulata ainult reaalajas, sest lisatud ei ole heli salvestuse võimalust ning hilisem signaalitöötlus on võimatu;
2) Heli võetakse vastu ainult anduri töösagedusele lähedasel sagedusel, puudub võimalus anduri hüveteguri vähendamiseks ning suurema ribalaiuse saavutamiseks.

5 Nahkhiirte detektori elektriskeem

5.1 Tööpõhimõte

Tööpõhimõtte väljatöötamisel on juhindutud sagedusega jagamise printsiipi kasutavast detektori skeemide tüüpidest. Eelnevalt mainitud lahendusel puudub salvestuse võimalus hilisemaks signaalitöötluseks ning kasutajamugavus tavakasutajale. Nende kõrvaldamiseks on vajalik välja töötada uudsem lahendus. Juhindudes tänapäeval enimkasutatavatele andmeside –ja andmesalvestus meetoditele on välja töötatud alljärgnev aseskeem.



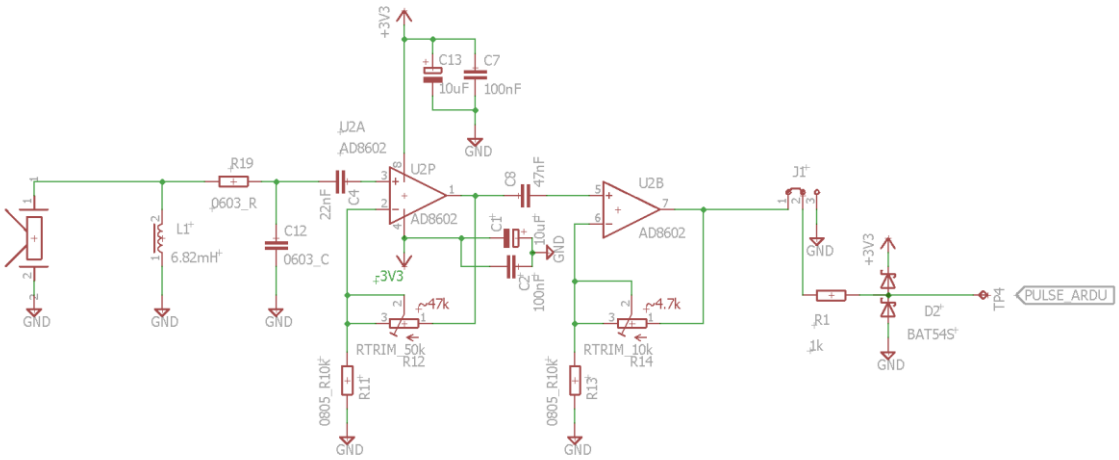
Joonis 4. Väljatöötatatud detektori blokkskeem

Käesoleva bakalaureuse töö käigus välja töötatud nahkhiirte detektor koosneb signaale vastu võtvast analoogosast [1] (vt. joonist 4), signaalitöötluse ja seadme üldise juhtimisega tegelevast mikrokontrollerist [2], SD-kaardi moodulist kuhu kasutaja soovi korral digitaliseeritud signaal salvestatakse [3] (lisana või alternatiivina saaks kasutada ka sinihamba moodulit andmete edastuseks [4]) ning aku laadimiseks ja toitepingete formeerimiseks mõeldud toiteosast [5].

Mikrokontrolleriks on valitud populaarse firma Atmel mikrokontroller makettplaadil “Arduino M0“ mis on ehituselt sarnane Zeroga ja M0 pro-ga. Arduino M0-i iseloomustab

ATSAMD21G18 mikroprotsessor taktsagedusega 48MHz, välkmälu 256KB, ja tööpinge 3,3V. Arduino Uno mikroprotsessoril ATmega328 on taktsagedus ainult 16MHz, välkmälu on 32KB ja tööpinge on 5V. Võttes arvesse protsessori suurema töökiiruse ja suurema välkmälu olemasolu, siis kasulikum on kasutada Arduino M0. [15], [16]

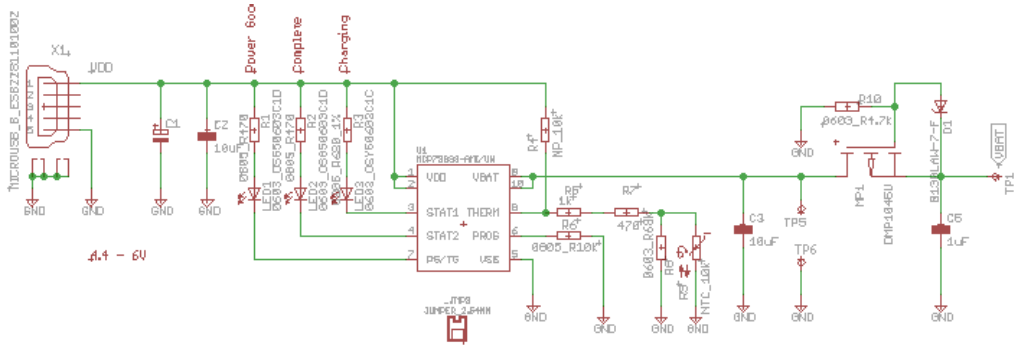
5.2 Seadme analoogosa elektriskeem



Joonis 5. Nahkhiirte detektori analoogosa elektriskeem.

Antud skeemis on kasutatud ultraheliandurit MA40A5R, mille nominaalsagedus on 40 kHz. Anduri eesmärk on vastu võtta ümbritsevast keskkonnast nahkhiirte poolt tekitatud häälitsusi. Anduriga on paralleelselt ühendatud induktiivpool induktiivsusega 6.82mH, mis häälestab anduri vastu võtma signaale laiemas sagedusvahemikus, vähendades anduri sageduskarakteristiku hüvetegurit. Vastuvõetud signaal juhitakse kahte järjestikusesse operatsioonivõimendisse-AD8602, mis on mitteinverteeriva võimendi režiimis. Mikrokontrolleri ADC-sse mineva signaali nivood piiratakse 2 kiiretoimelise Schottky dioodiga toitepinge ja üldmaa vahelisele nivoole et vältida ADC ülekoormamist või isegi läbi põlemist. Antud juhul piiravad dioodid protsessorisse mineva signaali amplituudi, mis ei saa minna kõrgemaks kui toitepinge ja ka alla 0V.

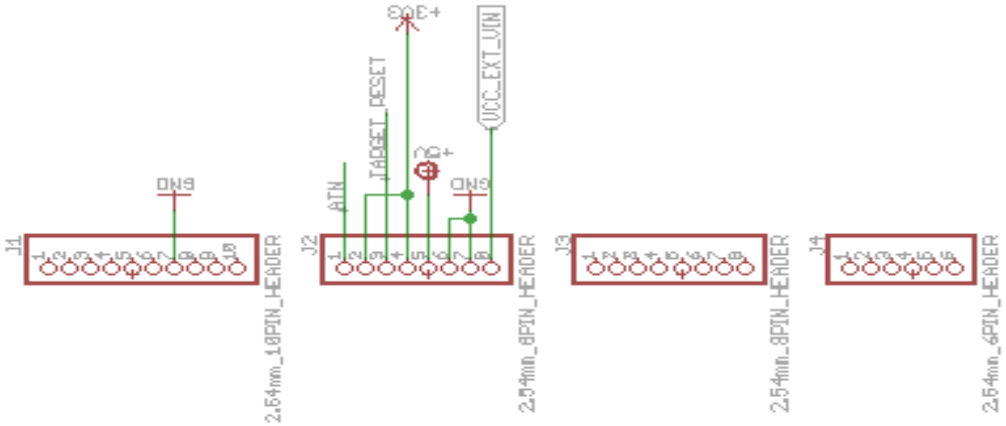
5.3 Toiteosa elektriskeem



Joonis 6. Nahkhiiredetektori toiteosa põhimõtteskeem

Seadmel on kaks teineteisest sõltumatut toiteallikat. Skeemi on võimalik pingestada arvutist standartse USB-liidese kaudu (pistik X1). Lisaks saab USB kaudu laadida ka akut. Aku laadimise tööd juhib spetsiaalne kontroller MCP73833, kontrolleri sisendpinge tuleb USB pistikust ning erinevad laadimisseisundid on visuaalselt kontrollitavad ühendades valgusdioodid LED1, LED2 ja LED3 vastavalt STAT1, STAT2 ja STAT3. Laadimisvoolu vähendatakse automaatselt, kui akuga soojuslikus kontaktis olev termotakisti annab kontrollerile märku ületemperatuurist.

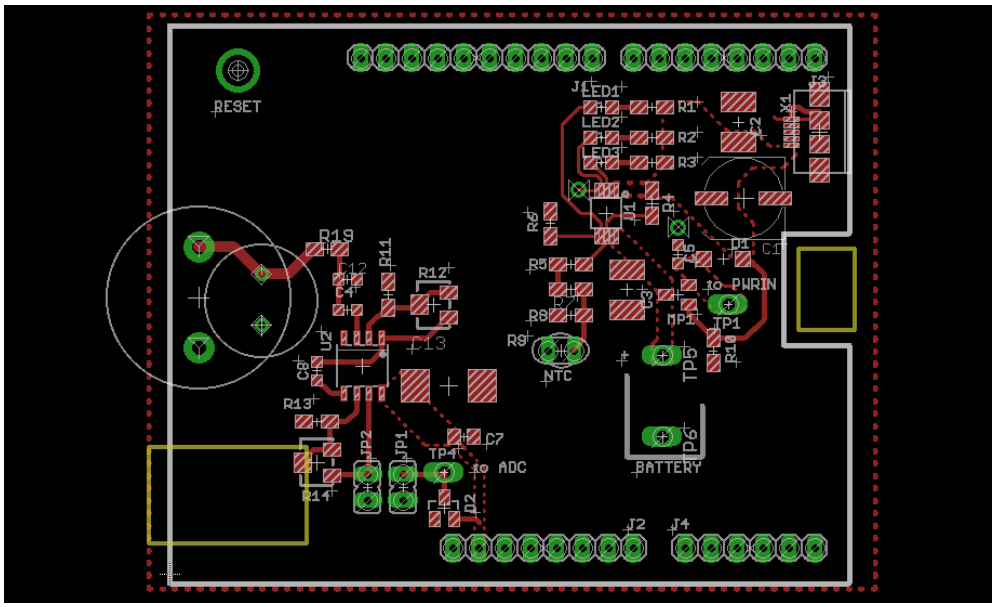
5.4 Ühendus „Arduino“ protsessorplaadiga



Joonis 7. Analoogosa ühendus ATSAMD21G18 mikroprotsessorit sisaldava plaadiga

Mikroprotsessorit toidetakse 5V-ga. Skeemi jaoks kasutatakse 3.3V-st väljundtoidet. Analoogsignaal juhitakse sisendosast mikroprotsessori analoogsidendisse A1. Peale digitaliseerimist mikroprotsessori analoog-digitaal muunduriga ja mõningast töötlust juhitakse signaal digitaalväljundisse D4, kust saab seda vajadusel edastada SD kaardile.

5.5 Trükkplaadi disain



Joonis 8. PCB

Trükkplaat on valmistatud suuremas osas SMD komponentidest(takistid, kondensaatorid, USB-pistik, ledid) ja mõnedest aukmontaaži komponentidest (ultraheliandur, jumperid, induktiivne element). Mikroprotsessoriga ühendamiseks on pistikud J1-J4. Analoogosa elektriskeem paikneb põhiliselt vasakul ning toiteosa komponendid paremal.

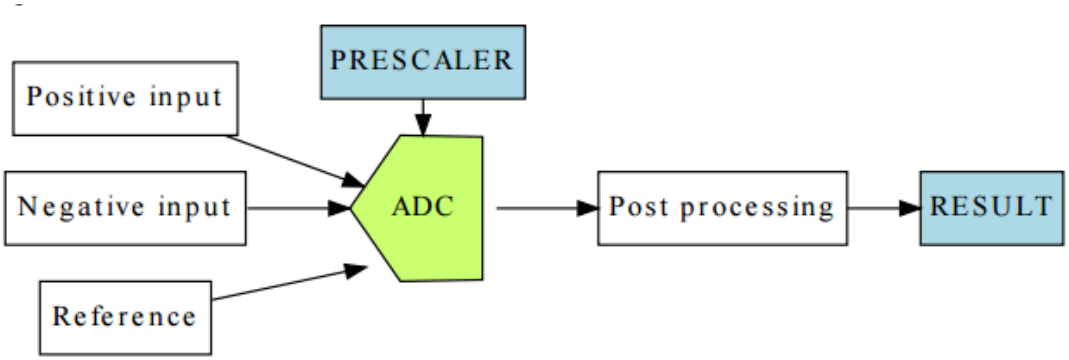
Kasutatud komponendid:

Komponent	Kogus
Ultraheliandur, MA40A5R	1
MCP73833-AMI/UN (4.2V)	1
Dual precicion op/amp AD8602DRZ	1
P-MOSFET DMP1045U	2
Schottky diood B130LAW-7-F	2
LED SMD kollane 20mA 0603	1
LED SMD roheline 20mA 0603	2
Micro USB B konnektor	1
NTC_10k	1
BAT54S	1
2.54mm male header long strips	1
Induktor 6.82mH	1
Takisti 0805_R470	2
Takisti 0805_R680	1
Takisti 0805_R10k	5
Takisti 0805_R5k	1
Takisti 0805_R1.54k	1
Takisti 0805_R69.8k	1
Trimmer potentsiomeeter 5k	1
Trimmer potentsiomeeter. 100k	1
Kondensaator 10uF 0805 SMD	2
Kondensaator 47nF SMD	1
Kondensaator 22nF SMD	1
Kondensaator 100uF SMD	1
Kondensaator 1uF SMD	1
Jumper 2pin header	4

Tabel 2. Kasutatud komponentide nimikiri

6 Analoogsignaali digitaliseerimine

Ultraheli sensori poolt vastu võetud analoogsignaali digitaliseerimiseks kasutatakse ATSAMD21G18 mikroprotsessoris olevat analoog-digitaal muundurit. Enne muundamist ADC-ga läbib signaal sagedusjaguri, sest muundur ei suuda piisavalt kõrgel sagedusel rahuldavalt toimida. [5]



Joonis 9. Analoog-digitaal muunduri skeem[5]

Signaali diskreetimise tingimus on, et diskreetimissagedus peab olema vähemalt 2 korda suurem signaali enda sagedusest. Võttes arvesse tabelis 1 toodud Natterilendlase maksimaalse kajalokaatori sageduse, mis on 115kHz võib leida, et minimaalne diskreetimissagedus on sellest kaks korda suurem seega peab ADC olema diskreetimissagedus 230kHz. Sellist diskreetimissagedust ei saa reaaliseerida, sest alates 100kHz muutub ADC ebastabiilseks. Stabiilsemaks saab olukorda muuta kasutades diskreedi pikkust 0-63 bitti. [6]

$$Diskreetimise\ aeg = \frac{ADCCLK\ 2}{(diskreedi\ pikkus + 1)}$$ [5]

7 Ühendus SD kaardi mooduliga

SD kaardi mooduli ühendamiseks Arduino mikrokontrolleriga on kasutatud järgmist pistikut. Kasutusel on toitepinge 5V. Signaal SDCS on ühendatud mikrokontrolleri digitaalväljundisse D4.

Joonis 10. SD kaardi mooduli ühendamise juhis [17], [18]

Andmete salvestuseks on programmikood kirjutatud „Arduino M0“ protsessorplaadile. Koodi eesmärgiks on ADC.st tulevate andmete salvestamine SD kaardile. Kaasatud koodid on ära toodud lisades (LISA 1, LISA 2 ja LISA 3). SD kaardile salvestatakse andmed teksti faili kujul.

```
const char *name = "F"; uint32_t filenum = 0;
const char *format = ".txt";
void SDPrepare(File &file, uint8_t chipSelect){
     bool Ready; uint8_t i;
  SerialUSB.println("12MHz SPI Ready");
  SerialUSB.print("Initializing SD card...");
for(i =0; (i<NUMBER_OF_TRIES)&&(!SD.begin(chipSelect));i++)
    {
    SerialUSB.println("Card failed, or not present");
     delay(100);
  }
```

Joonis 11. Programmikoodi lõik SD kaardiga ühendamiseks[19]

Analoogsignaal suundub ADC-sse protsessorplaadi analoogsisendi A1 kaudu (`#define ADC_PIN A1`). Töödeldud andmed salvestatakse ajutiselt ühte deklareeritud puhvritest. Puhvrite täitumist kontrollitakse pidevalt, kui puhver on saanud täis ja andmeid rohkem ei mahu hakatakse andmeid saatma SD kaardile ning samal ajal hakatakse andmeid salvestama teise tühja puhvrisse.

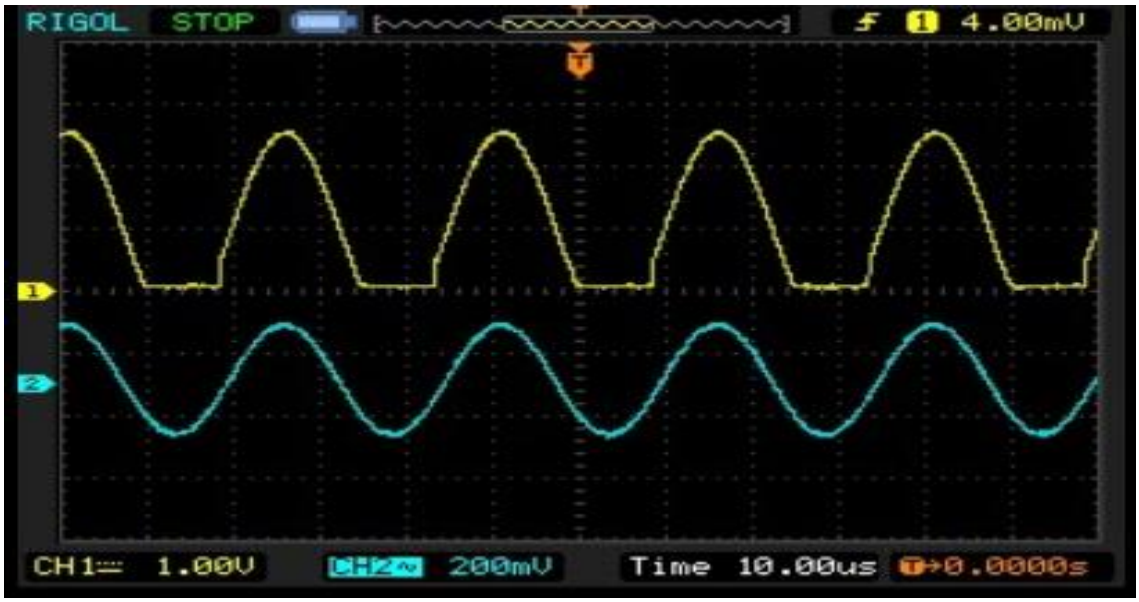
```
  if ((*otherBufferStatus) == BUFFER_FULL)
 {
    (*bufferStatus) = BUFFER_ADC_RUNNING;
    adc_dma(buffer, BUFFER_SIZE);
    writeBufferToSD(otherBuffer, BUFFER_SIZE);
  }
}
void writeBufferToSD(uint8_t* buffer, size_t size) {
  uint32_t t2 = micros();
  uint32_t t1 = micros();
  uint16_t bytesWritten = sdFile.write(buffer, size);
  t1 = micros() - t1;
  SDFlush(sdFile, t1);
  t2 = micros() - t2;
status
  handleBufferWrittenToSDCard(buffer);
```

Joonis 12. Programmikoodi lõik, kui puhver on saanud täis ja sealsed andmed kirjutatakse SD kaardile[19]

8 Katsetulemused

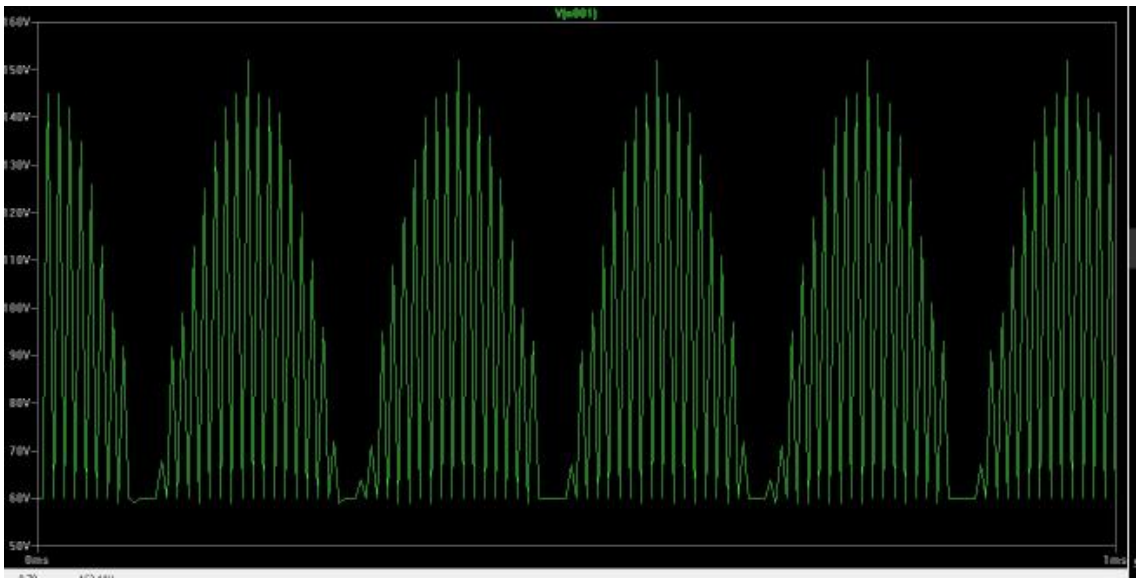
Katsed viidi läbi laboritingimustes toites mikroprotsessorit USB-liidese kaudu ja kasutades funktsiooni generaatorit nahkhiire heli imiteerimiseks.

1. Katseks on genereeritud funktsiooni generaatoriga signaal sagedusega 40kHz, mis on joonisel kujutatud sinise joonega. Kollase joonega on esitatud mikrokontrolleri analoogsisendisse A1 suunduv signaal. Võttes arvesse, et ADC töötleb ainult positiivset osa signaalist, siis signaalide negatiivne osa joonistel puudub.



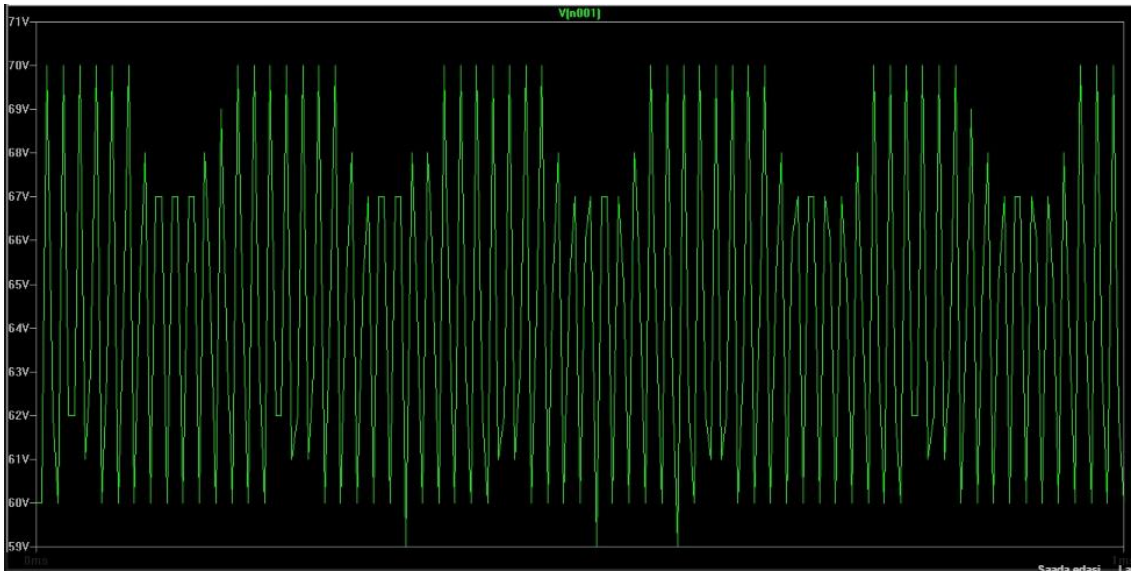
Joonis 13. Signaal ostsilloskoobiga mõõdetuna –kollane on mikrokontrollerisse suunduv analoogsignaal - sinine on funktsiooni generaatoriga genereeritud signaal

Järgmistel joonistel on SD kaardile salvestatud andmetest arvuti poolt genereeritud joonised, mis kujutavad diskreetseid signaale. Joonisel 12 on signaal mille puhul ADC töötab stabiilsena. Selleks on seatud diskreedi pikkuseks 47 bitti. Joonisel 13 olev signaal märgib ADC ebastabiilset käitumist, kus diskreedi pikkus on 63 bitti. Joonisel 14 on kujutatud olukord, mis võib esineda kui diskreedi pikkus on liiga väikseks seadistatud. Sellisel juhul võivad tekkida Nyquisti kriteeriumi kohaselt aliassignaalid, mis tähendab seda, et tekivad ülekattuvused.



Joonis 14. Digitaliseeritud signaali stabiilne kuju võetuna mikrokontrolleri digitaalväljundist

Sellise kuju saavutab väljund, kui diskreetimissagedus on väiksem või võrdne 100kHz piiriga.



Joonis 15. Digitaliseeritud signaali ebastabiilne kuju võetuna mikrokontrolleri digitaalväljundist

Sellise kuju saavutab väljund, kui diskreetimissagedus on suurem kui 100kHz.

Joonis 16. Signaali moonutatud kuju väiksema aja lõikes

Joonisel 15 kujutab digitaliseeritud signaali väiksema aja lõikes, millest järeldub, et liiga suure diskreetimissageduse korral tekivad ülekattuvused.

9 Kokkuvõte

Käesolevas töös võrreldi eri liiki nahkhiirte kajalokatsiooni sagedusi, mille põhjal järeldati, et nende helisagedus on inimesele mittekuuldavas sagedusvahemikus ehk siis kõrgema helisagedusega. Seejärel esitati erinevaid võimalusi helisageduse madaldamiseks võttes vaatluse alla superheterodüünvastuvõtja, sagedusega jagamine ja ajavenituse tehnikad. Esimese kahe kohta esitati skeemid ning seletati nende üldist tööpõhimõtet ning skeemide eeliseid ja puudusi.

Ühe suurema puuduse kõrvaldamiseks, milleks oli andmete salvestamise võimalus, töötati välja eraldi skeem, mis kasutab Arduino M0 peal olevat mikroprotsessorit, kuhu juhitakse analoogsignaal. Kasutades analoog-digitaal muundurit signaal digitaliseeritakse ning edastatakse salvestusseadmesse, milleks kasutatakse SD kaardi moodulit.

Katsetulemuste saamiseks genereeriti funktsiooni generaatoriga 40kHz signaal, mis on skeemis kasutatava ultraheliandurit iseloomustav töösagedus. Katsetulemusest järeldub, et antud skeem töötab ootuspäraselt. Anduriga vastuvõetud signaal jõuab mikroprotsessori analoogsisendisse ning liigub edasi analoog-digitaal muundurisse, kus ta digitaliseeritakse. Digitaliseerimisel esinesid teatud probleemid piisavalt kõrge diskreetimissageduse saavutamisel, sest ADC töö muutus kõrgel sagedusel ebastabiilseks. Seda sai muuta stabiilsemaks vähendades diskreetimisaega, kuid liiga väikseks muutmisel tekkisid juhud, kus osa signaalist läks kaduma. Digitaliseeritud signaal edastatakse SD kaardi moodulisse. SD kaardi peal olevat infot saab kasutada hilisemaks signaalide võrdlemiseks.

Seadme veelgi kasutajamugavamaks muutmiseks võib kasutada Bluetoothi ühendust andmete ülekandeks, näiteks mõnda spetsiaalsesse äppi nutitelefonis, mis kasutab „Androidi“ operatsioonisüsteemi või teha server kuhu detektor mingi kindla aja tagant kogutud andmeid edastab.

Kasutatud kirjandus

[1] Eesti nahkhiired. [WWW] http://www.keskkonnaamet.ee/public/Keskkonnaamet_PP_tume.pdf (30.05.16)

[2] MCP73833 andmeleht. [WWW] http://www.tme.eu/en/Document/00199be6c30361447b781e2dcbb77bbb/mcp73833_4.pdf (30.05.16)

[3] Build a Simple Bat Detector. [WWW] http://home.netcom.com/~t-rex/BatDetector.html (30.05.16)

[4] Ultrasound Detector with 567PLL and LM386 ICs. [WWW] http://www.circuitdiagramworld.com/sensor_circuit_diagram/Ultrasound_Detector_with_567PLL_and_LM386_ICs_21965.html (30.05.16)

[5] AT07627: ASF Manual (SAM D21). [WWW] http://www.atmel.com/Images/Atmel-42258-ASF-Manual-SAM-D21_AP-Note_AT07627.pdf (30.05.16)

[6] Signaalitöötlus II, (Loengumaterjal 1). [WWW] http://lr.ttu.ee/signaalitootlus2/2015sygis/SIGNAALID_2_1.pdf (30.05.16)

[7] Belfry Frequency Division Detector. [WWW] http://njsas.org/projects/bat_detector/01/belfry_sch.html (30.05.16)

[8] Käsitiivalised. [WWW] https://et.wikipedia.org/wiki/K%C3%A4sitiivalised (30.05.16)

[9] Kuu loom – nahkhiir. [WWW] http://www.loodusajakiri.ee/eesti_loodus/EL/vanaweb/0010/nahkhiir.html (30.05.16)

[10] Eesti nahkhiired. [WWW] http://xazuhav.havike.eenet.ee/wordpress/eesti-nahkhiired/ (30.05.16)

[11] Nahkhiired on Eestis kaitse all. [WWW] http://www.loodusajakiri.ee/eesti_loodus/artikkel4960_4941.html (30.05.16)

[12] Echolocation by Insect-Eating Bats. [WWW]

http://bioscience.oxfordjournals.org/content/51/7/557.full (30.05.16)

[13] R. Mark Brigham, Elisabeth K. V. Kalko, Gareth Jones, Stuart Parsons, Herman J. G. A. Limpens, „Bat Echolocation Research“, Austin, Texas 2002.

[14] Bat detector. [WWW] https://en.wikipedia.org/wiki/Bat_detector (30.05.16)

[15] Arduino M0. [WWW] http://www.oomipood.ee/product/a000103/arduino-m0&s=Arduino%20M0 (30.05.16)

[16] Arduino Board Uno Overview. [WWW] https://www.arduino.cc/en/Main/ArduinoBoardUno (30.05.16)

[17] SD Card Reader/Writer Module. [WWW] http://artofcircuits.com/product/sd-card-readerwriter-module (30.05.16)

[18] Introducing the Adafruit Bluefruit LE UART Friend. [WWW] https://learn.adafruit.com/introducing-the-adafruit-bluefruit-le-uart-friend/wiring (30.05.16)

[19] Programmikoodid. [WWW] https://github.com/raunc/bat-detector (30.05.16)

Lisa 1 –Mikrokontrolleri kood ADC ja SD koodi koos kasutamiseks [19]

```
#include <SD.h>
#include <SPI.h>

#include "adc-dma.h"
#include "sd.h"


#define ADC_PIN A1
#define CHIP_SELECT_SD 4
#define BUFFER_SIZE 4096

// Buffer Statuses
#define BUFFER_EMPTY 0
#define BUFFER_FULL 1
#define BUFFER_ADC_RUNNING 2

/* GLOBAL VARIABLE DEFINITIONS */
// ADC buffers
uint8_t adcBuffer0[BUFFER_SIZE];
uint8_t adcBuffer1[BUFFER_SIZE];

uint8_t adcBuffer0Status = BUFFER_EMPTY;
uint8_t adcBuffer1Status = BUFFER_EMPTY;

// File on SD card
File sdFile;

/* Function declarations */
void handleBufferFull(void);
void handleBufferWrittenToSDCard(uint8_t* buffer);
```

```
void setup() {

  SerialUSB.begin(115200);
  while (!SerialUSB);

  // Initializing ADC
  adc_init(ADC_PIN, 63);
  dma_init();
  setADCDoneHandler(&handleBufferFull);

  // Initializing SD card
  pinMode(13, OUTPUT);// For using LED to indicate SD card
errors
  SDPrepare(sdFile, CHIP_SELECT_SD);
 // Fill one ADC buffer
  adcBuffer0Status = BUFFER_ADC_RUNNING;
  adc_dma(adcBuffer0, BUFFER_SIZE);
}
void loop() {
}
void handleBufferFull(void) {
//  SerialUSB.println("handleBufferFull");
  uint8_t* buffer;
  uint8_t* bufferStatus;
  uint8_t* otherBuffer;
  uint8_t* otherBufferStatus;

  if (adcBuffer0Status == BUFFER_ADC_RUNNING) {
    // Buffer 0 was running
    buffer = adcBuffer0;
    bufferStatus = &adcBuffer0Status;
    otherBuffer = adcBuffer1;
    otherBufferStatus = &adcBuffer1Status;
  } else if (adcBuffer1Status == BUFFER_ADC_RUNNING) {
```

```
    // Buffer 1 was running
    buffer = adcBuffer1;
    bufferStatus = &adcBuffer1Status;
    otherBuffer = adcBuffer0;
    otherBufferStatus = &adcBuffer0Status;
  } else {
    SerialUSB.println("Something went wrong with adc buffer
statuses");
    return;
  }
  (*bufferStatus) = BUFFER_FULL;
  if ((*otherBufferStatus) == BUFFER_EMPTY) {
    // Start filling other buffer and write this buffer to
SD card
    (*otherBufferStatus) = BUFFER_ADC_RUNNING;
    adc_dma(otherBuffer, BUFFER_SIZE);

    writeBufferToSD(buffer, BUFFER_SIZE);
  }
}
void handleBufferWrittenToSDCard(uint8_t* buffer) {
//  SerialUSB.println("handleBufferWrittenToSDCard");
  uint8_t* otherBuffer;
  uint8_t* otherBufferStatus;
  uint8_t* bufferStatus;

  if (buffer == adcBuffer0) {
    bufferStatus = &adcBuffer0Status;
    otherBuffer = adcBuffer1;
    otherBufferStatus = &adcBuffer1Status;
  } else {// buffer == &adcBuffer1
    bufferStatus = &adcBuffer1Status;
    otherBuffer = adcBuffer0;
    otherBufferStatus = &adcBuffer0Status;
  }
```

```
(*bufferStatus) = BUFFER_EMPTY;

  if ((*otherBufferStatus) == BUFFER_FULL)
 {
    (*bufferStatus) = BUFFER_ADC_RUNNING;
    adc_dma(buffer, BUFFER_SIZE);
    writeBufferToSD(otherBuffer, BUFFER_SIZE);
  }
}
void writeBufferToSD(uint8_t* buffer, size_t size) {
  uint32_t t2 = micros();
  uint32_t t1 = micros();
  uint16_t bytesWritten = sdFile.write(buffer, size);
  t1 = micros() - t1;
  SDFlush(sdFile, t1);
  t2 = micros() - t2;
status
  handleBufferWrittenToSDCard(buffer);
```

Lisa 2 – SD kaardile kirjutamise kood [19]

```
#ifndef SD_H_
#define SD_H_
#include <SD.h>

void SDPrepare(File&, uint8_t chipSelect);
bool SDFlush(File &, uint32_t);
bool SDNewOpen(File&);
void SDReCheck(File&, uint8_t chipSelect, uint32_t,
uint32_t);

#endif


#include <SD.h>
#include <SPI.h>
#include "sd.h"
#define NUMBER_OF_TRIES 3

const char *name = "F";
uint32_t filenum = 0;
const char *format = ".txt";
void SDPrepare(File &file, uint8_t chipSelect){
     bool Ready;
     uint8_t i;
  SerialUSB.println("12MHz SPI Ready");
  SerialUSB.print("Initializing SD card...");
  for(i =0;
(i<NUMBER_OF_TRIES)&&(!SD.begin(chipSelect));i++)
    {
    SerialUSB.println("Card failed, or not present");
     delay(100);
  }
  if (i == NUMBER_OF_TRIES) {
```

```
        while (1) {
             digitalWrite(13, HIGH);
             delay(500);
             digitalWrite(13, LOW);
             delay(500);
        }
  }
  SerialUSB.println("Card initialized.");
  SerialUSB.println("Opening file.");
   if (!SDNewOpen(file)) {
        SerialUSB.println("Error opening new file");
        return;
  }
  SerialUSB.println("File opened.");
}
bool SDFlush(File &myFile, uint32_t t)
{
     if (t > 150000) {
          SerialUSB.println("SDFlush");
          myFile.flush();
               }
}
bool SDNewOpen(File &myFile) {
     SerialUSB.println("SDNewOpen");
     char buffer[30];
     filenum++;
     if (filenum != 1) {
          myFile.close();
     }
     sprintf(buffer, "%s%d%s", name, filenum, format);
     SerialUSB.println(buffer);
     while (SD.exists(buffer)) {
          filenum++;
          sprintf(buffer, "%s%d%s", name, filenum, format);
          SerialUSB.println(buffer);
```

```
      }
      if ((myFile = SD.open(buffer, FILE_WRITE)) == 0) {
                return 0;
      }
      return 1;
}
void SDReCheck(File &file, uint8_t chipSelect, uint32_t t,
uint32_t t1) {
           if ((t < 1000) || (t1 < 1000)) {
           SerialUSB.println("SDReCheck: prepare");
           SDPrepare(file, chipSelect);
      }
}
```

Lisa 3 –ADC mikroprotsessori konfigureerimise kood [19]

```
#ifndef ADC_DMA_H_
#define ADC_DMA_H_

// Arduino header for board (includes also ASF headers)
#include <Arduino.h>

extern volatile uint32_t adc_done;
typedef void (*fnPtr)(void);

void setADCDoneHandler(fnPtr);
void adc_init(const uint8_t ADC_PIN, const uint8_t
samplingSpeed);
void dma_init();
void adc_dma(void *rxdata, size_t hwords);

#endif
#include <Arduino.h>
#include "adc-dma.h"

typedef struct {
    uint16_t btctrl;
    uint16_t btcnt;
    uint32_t srcaddr;
    uint32_t dstaddr;
    uint32_t descaddr;
} dmacdescriptor;

volatile dmacdescriptor wrb[12] __attribute__ ((aligned
(16)));
dmacdescriptor descriptor_section[12] __attribute__
((aligned (16)));
dmacdescriptor descriptor __attribute__ ((aligned (16)));
```

```
static uint32_t chnl = 0;  // DMA channel
volatile uint32_t adc_done = 0;
fnPtr doneHandler = 0;
void setADCDoneHandler(fnPtr handler) {
  doneHandler = handler;
}
void DMAC_Handler() {
    // interrupts DMAC_CHINTENCLR_TERR
DMAC_CHINTENCLR_TCMPL DMAC_CHINTENCLR_SUSP
    uint8_t active_channel;
    __disable_irq();
    active_channel = DMAC->INTPEND.reg &
DMAC_INTPEND_ID_Msk; // get channel number
    DMAC->CHID.reg = DMAC_CHID_ID(active_channel);
    adc_done = DMAC->CHINTFLAG.reg;
    DMAC->CHINTFLAG.reg = DMAC_CHINTENCLR_TCMPL; // clear
    DMAC->CHINTFLAG.reg = DMAC_CHINTENCLR_TERR;
    DMAC->CHINTFLAG.reg = DMAC_CHINTENCLR_SUSP;
    __enable_irq();

    if (doneHandler) {
      (*doneHandler)();
    }
}


void dma_init() {
    // probably on by default
    PM->AHBMASK.reg |= PM_AHBMASK_DMAC ;
    PM->APBBMASK.reg |= PM_APBBMASK_DMAC ;
    NVIC_EnableIRQ( DMAC_IRQn ) ;

    DMAC->BASEADDR.reg = (uint32_t)descriptor_section;
    DMAC->WRBADDR.reg = (uint32_t)wrb;
```

```
    DMAC->CTRL.reg = DMAC_CTRL_DMAENABLE |
DMAC_CTRL_LVLEN(0xf);
}

void adc_dma(void *rxdata, size_t dataLength) {

    DMAC->CHID.reg = DMAC_CHID_ID(chnl);
    DMAC->CHCTRLA.reg &= ~DMAC_CHCTRLA_ENABLE;
    DMAC->CHCTRLA.reg = DMAC_CHCTRLA_SWRST;
    DMAC->SWTRIGCTRL.reg &= (uint32_t)(~(1 << chnl));
    DMAC->CHCTRLB.reg = DMAC_CHCTRLB_LVL(0) |
DMAC_CHCTRLB_TRIGSRC(ADC_DMAC_ID_RESRDY) |
DMAC_CHCTRLB_TRIGACT_BEAT;
    DMAC->CHINTENSET.reg = DMAC_CHINTENSET_MASK ; // enable
all 3 interrupts

    adc_done = 0;
    descriptor.descaddr = 0;
    descriptor.srcaddr = (uint32_t) &ADC->RESULT.reg;
    descriptor.btcnt =  dataLength;
    descriptor.dstaddr = (uint32_t)rxdata + dataLength;
// end address // Why end address?
    descriptor.btctrl =  DMAC_BTCTRL_BEATSIZE_BYTE |
DMAC_BTCTRL_DSTINC | DMAC_BTCTRL_VALID;
    memcpy(&descriptor_section[chnl],&descriptor,
sizeof(dmacdescriptor));

    // start channel
    DMAC->CHID.reg = DMAC_CHID_ID(chnl);
    DMAC->CHCTRLA.reg |= DMAC_CHCTRLA_ENABLE;
}

static __inline__ void ADCsync()
__attribute__((always_inline, unused));
static void   ADCsync() {
```

```
  while (ADC->STATUS.bit.SYNCBUSY == 1); //Just wait till
the ADC is free
}
void adc_init(const uint8_t ADC_PIN, const uint8_t
samplingSpeed) {
  analogRead(ADC_PIN);  // do some pin init
pinPeripheral()
  ADC->CTRLA.bit.ENABLE = 0x00;             // Disable ADC
  ADCsync();
  //ADC->REFCTRL.bit.REFSEL =
ADC_REFCTRL_REFSEL_INTVCC0_Val; //  2.2297 V Supply VDDANA
  //ADC->INPUTCTRL.bit.GAIN = ADC_INPUTCTRL_GAIN_1X_Val;
// Gain select as 1X
  ADC->INPUTCTRL.bit.GAIN = ADC_INPUTCTRL_GAIN_DIV2_Val;
// default
  ADC->REFCTRL.bit.REFSEL = ADC_REFCTRL_REFSEL_INTVCC1_Val;
  ADCsync();    //  ref 31.6.16
  ADC->INPUTCTRL.bit.MUXPOS =
g_APinDescription[ADC_PIN].ulADCChannelNumber;
  ADC->INPUTCTRL.bit.INPUTSCAN = 0;
  ADC->INPUTCTRL.bit.INPUTOFFSET = 0;
  ADCsync();
  ADC->AVGCTRL.reg = 0x00;       //no averaging
  ADC->SAMPCTRL.reg = samplingSpeed; //sample length in 1/2
CLK_ADC cycles // Setting to zero does not work for 10bit
resolution
  ADCsync();
  ADC->CTRLB.reg = ADC_CTRLB_PRESCALER_DIV16 |
ADC_CTRLB_FREERUN | ADC_CTRLB_RESSEL_8BIT;

  ADCsync();
  ADC->CTRLA.bit.ENABLE = 0x01;
  ADCsync();
}
```