Jori Lindell

Puhuvan avaimenperän ohjelmistokehitys

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

29.5.2013

Metropolia

Tekijä Otsikko Sivumäärä Aika	Jori Lindell Puhuvan avaimenperän ohjelmistokehitys 51 sivua + 1 liitettä 29.5.2013
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	sulautetut järjestelmät
Ohjaaja	yliopettaja Jarkko Vuori

Insinöörityön aiheena oli suunnitella ja toteuttaa vaikeasti toisistaan erotettavien esineiden tunnistamista helpottava apuväline heikkonäköisille. Tunnistettavat esineet merkitään RFID-tunnisteilla. Tunnisteen ensimmäisellä lukukerralla nauhoitetaan äänitiedosto, joka yhdistetään tunnisteeseen. Seuraavilla lukukerroilla laite toistaa tunnisteeseen yhdistetyn äänitiedoston.

Toteutettavan laitteen tärkeimmät vaatimukset olivat helppokäyttöisyys, pieni koko sekä tyylikäs ulkonäkö. Laitteen tuli olla riittävän pieni, jotta sitä olisi helppo kuljettaa mukana. Käyttöliittymän tuli olla riittävän yksinkertainen, jotta sen toiminnan omaksuisi jo ensimmäisellä käyttökerralla.

Työssä keskityttiin laitteen ohjelmiston kehitykseen ja toteutukseen. Myös laitteen elektroniikan komponenttien ohjelmoinnin kannalta tärkeimmät ominaisuudet on esitelty dokumentissa, kuten myös niiden hallintaa käytettävät kommunikointiväylät.

Kehitystyön tuloksena oli prototyyppi, joka toimii hyvänä lähtökohtana jatkokehitystä varten. Laitetta testattiin yhteistyössä Näkövammaisten keskusliiton kanssa. Näissä käyttäjätesteissä laite sai kiitosta muun muassa helppokäyttöisyydestä, muotoilusta ja RFID-tunnisteiden nopeasta lukemisesta.

Avainsanat	RFID, prototyyppi, ohjelmistokehitys, STM32, I2C, SPI, I2S, SDIO

Author Title Number of Pages Date	Jori Lindell Software for a talking keychain 51 pages + 1 appendix 29 May 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Embedded Systems
Instructor	Jarkko Vuori, Principal Lecturer

The purpose of the project described in this thesis was to develop and to implement a tool for visually impaired persons which could be used to identify objects that are difficult to distinguish from each other. In general, different kind of items can be marked with RFID tags. When the RFID tag is first detected, the user records a sound file, which is associated with the tag. When the tag is read again, the device plays the sound file.

The main requirements for the device were ease of use, small size and elegant appearance. The device should be small enough so it is easy to carry around. The user interface should be simple enough so that the user can learn to use it the very first time.

The focus of this thesis is on the software development and implementation of the device. Also, the key features of electronics components are presented in the document from the programming perspective. In addition, the communication interfaces which are used to control them are described.

As a result, a prototype was developed, which serves as a good starting point for further development. The device has been tested in cooperation with the Federation of the Visually Impaired. In these tests, the device has received good feedback for ease of use, design and fast detection of RFID tags.

Keywords	RFID, prototype, software development, STM32, I2C, SPI, I2S, SDIO

Sisällys

Lyhenteet

EEPROM *Electronically Erasable Programmable Read-Only Memory.* Haihtumaton puolijohdemuisti, jota voidaan kirjoittaa uudelleen noin 10000–100000 kertaa. Muistia ei tarvitse tyhjentää lohkoittain, vaan uuden tiedon voi päällekirjoittaa tavu kerrallaan.

EPC *Electronic Product Code.* RFID-tunnisteissa käytössä oleva 64- tai 96-bittinen tuotekoodi, joka sisältää tietoa muun muassa valmistajasta ja tuoteen tyypistä.

HF *High Frequency.* HF-taajuusalueella käytännön standarditaajuus on 13,56 MHz, joka on kansainvälisesti vapaa taajuus. HF-taajuudella toimivaa RFID-tekniikkaa käytetään yleensä lähietäisyydellä tunnistamisessa, kuten kulunvalvonnassa.

I2C *Inter-Integrated Circuit.* Yksinkertainen kaksisuuntainen ohjaus- ja tiedonsiirtoväylä, jota käytetään muun muassa kulutuselektroniikassa.

I2S *Inter-IC Sound.* Sarjaliikenneväylä, joka on suunniteltu digitaalisten audiolaitteiden yhdistämistä varten.

JTAG *Joint Test Action Group.* JTAG-porttia käytetään yleisesti mikropiirien testaus- ja kehitysapuvälineenä.

LF *Low Frequency*. LF-taajuusalueetta käyttävät RFID-järjestelmät toimivat yleensä 125 kHz:n taajuudella. Nykyisin käyttö rajoittuu lähinnä tiettyihin kulunvalvonnan ja eläintunnistuksen sovelluksiin.

RFID *Radio Frequency IDentification.* Radiotaajuinen etätunnistus on menetelmä tiedon etälukuun ja -tallentamiseen.

SDIO *Secure Digital Input Output.* Väylä muistikortin esimerkiksi matkapuhelimiin ja kameroihin liittämistä varten.

SPI *Serial Peripheral Interface.* Synkroninen sarjaliikenneväylä, jonka avulla on mahdollista lähettää sekä vastaanottaa tietoa samanaikaisesti.

SRAM *Static Random Access Memory.* Staattinen RAM-muisto on puolijohdetekniikalla toteutettu muistityyppi, jota ei tarvitse virkistää säännöllisesti.

UHF *Ultra High Frequency.* UHF-taajuusalueen RFID-tunnisteita käytetään yleisesti logistiikkasovelluksissa, kuten kuormalavojen ja konttien jäljityksessä. Suomessa käytetty taajuus on 868 MHz.

USB *Universal Serial Bus.* Sarjaväyläarkkitehtuuri oheislaitteiden liittämiseksi tietokoneeseen.

1 Johdanto

Yksi yleinen haaste kaikille sokeille tai heikkonäköisile henkilöille on sellaisten esineiden tunnistaminen, joita voi olla mahdoton erottaa toisistaan koskettamalla. Ovatko lääkepurkin pillerit lääkkeitä päänsärkyyn vai korkeaan verenpaineeseen? Ovatko jääkaapin sivuovessa olevat tölkit maitoa vai tuoremehua? Onko vaatekaapista valittu paita punainen vai sininen? Luettelo vaikeasti tunnistettavista esineistä on loputon.

Työn tarkoituksena on suunnitella ja toteuttaa prototyyppi laitteesta, jonka avulla heikkonäköinen henkilö voi tunnistaa näitä vaikeasti toisistaan erotettavia esineitä. Työ on toteutettu osana laajempaa HEA-hanketta (Hyvinvointia ja energiatehokkuutta asumiseen), jossa etsittiin innovatiivisia ratkaisuja ihmisten hyvinvointiin ja asumiseen kestävän kehityksen näkökulmasta.

Tehtäväni Puhuvan avaimenperä -projektissa oli laitteen ohjelmiston suunnittelu ja toteutus. Insinöörityössä kerrotaankin laitteen toteutusprosessista ohjelmoinnin näkökulmasta. Myös ohjelmointityön kannalta merkitykselliset elektroniikan komponentit esitellään työssä.

Minun lisäkseni projektissa työskenteli elektroniikkasuunnittelija ja kaksi teollista muotoilijaa. Tärkeä asia onnistuneen lopputuloksen kannalta oli se, että koko tiimillä oli mahdollisuus työskennellä samoissa tiloissa, jolloin kommunikointi oli vaivatonta. Työskentely tapahtui Metropolian Electria-yksikössä. Projektin edetessä kaikilla oli mahdollisuus kommentoida toistensa töitä, jolloin ei rajoituttu yhteen näkökulmaan. Lisäksi eri alojen ihmisten välinen kommunikointi opetti huomiomaan toisten henkilöiden työn asettamien vaatimusten ottamisen huomioon myös omassa työssä. Varsinkin minun ja elektroniikkasuunnittelijan välinen kommunikointi oli todella tarpeellista, sillä näin saimme yhdessä sovittua, miten eri komponentit liitetään mikrokontrolleriin kiinni.

2 Työn tavoite sekä laitteen vaatimusten määrittely

Insinöörityön tarkoituksena oli kehittää heikkonäköisille ja sokeille henkilöille apuväline, jonka avulla he voivat tunnistaa veikeasti toisistaan erotettavia esineitä. Laitteen käyttämisen tulisi olla helppoa ja nopeaa, ja sen käyttämisen omaksuminen tulisi olla mut-

katonta. Lisäksi laitteen tulisi olla pienikokoinen, jotta sitä olisi helppo kuljettaa mukana. Laitteen nimi, Puhuva avaimenperä, tulikin juuri siitä, että laitteen tulisi olla niin pieni, että sitä voi kuljettaa taskussa, esimerkiksi avainnippuun kiinnitettynä.

2.1 Markkinoilla olevat vastaavat laitteet

Markkinoilla on tällä hetkellä kaksi laitetta, jotka ovat näkövammaisilla laajemmassa käytössä esineiden merkitsemistä ja tunnistamista varten: Touch memo ja Penfriend. Molempien laitteiden tunnistamistekniikka perustuu optiseen lukemiseen. Tunnistettavaan esineeseen kiinnitetään tarra, jolla on yksilöllisen tunnistekuvio. Tarran kuvio luetaan laitteen toisessa päässä sijaitsevalla lukijapiirillä. Tunnisteita tulee osoittaa lähes kosketusetäisyydeltä, joten käyttäjän tulee tietää tarkalleen, missä tarra sijaitsee. Laitteet ovat suurin piirtein samankokoisia. Molemmat ovat noin 15 cm:n pituisin ja 3 cm:n paksuisia. Kummassakin laitteessa on liitännät ulkoiselle mikrofonille ja korvakuulokkeille. Laitteiden akun lataus sekä varmuuskopiointi tapahtuu USB-liittimen avulla. Laitteet muistuttavat paljon toisiaan, ja erot löytyvät lähinnä käyttöliittymästä sekä painikkeiden lukumäärästä ja sijainnista. Seuraavaksi esitellään kummankin laitteen toiminta.

Touch memo

Touch memon kyljessä on liukukytkin, josta laitteeseen kytketään virta. Kytkimellä laite saadaan myös nollattua mahdollisissa ongelmatilanteissa. Virran tulee olla kytkettynä laitetta ladattaessa. Jotta laitetta voidaan käyttää, tulee se vielä erikseen käynnistää toisesta painikkeesta pitämällä sitä pohjassa sekunnin ajan. Samasta painikkeesta laite voidaan sammuttaa pitämällä painiketta pohjassa sekunnin ajan. Laite sammuu itsestään 2–3 minuutin kuluttua, jollei käyttäjä ole sammuttanut sitä manuaalisesti. [1.]

Käynnistymisen jälkeen laitteella voidaan lukea tunnisteita osoittamalla tunnistetarraa laitteen päässä olevalla lukijapiirillä. Laite toistaa nauhoituksen, jos tunnisteeseen on jo aiemmin äänitetty. Äänitteen toistamisen voi keskeyttää painamalla tallennuspainiketta. Tunnisteen ollessa tyhjä laite soittaa merkkiäänen. [1.]

Laitteeseen voidaan tallentaa uusi äänitunniste painamalla tallennuspainike pohjaan ja pitämällä se pohjassa. Tämän jälkeen haluttu tunniste luetaan edelleen pitämällä paini-

ke pohjassa. Äänitys alkaa laiteen antaman merkkiäänen jälkeen. Äänitys päättyy vapauttamalla tallennuspainike. [1.]

Tunnisteessa oleva äänitunniste voidaan päällekirjoittaa samalla tavalla kuin tallentaminen tapahtui. Tällä kertaa laite antaa pidemmän, noin kahden sekunnin mittaisen äänimerkin, jonka jälkeen vanha äänitunniste poistetaan ja uuden äänittäminen aloitetaan. Äänitunnisteen poistaminen voidaan perua vapauttamalla tallennuspainike äänimerkin soimisen aikana. Äänite voidaan poistaa myös lukemalla tunniste, ja toistamisen loputtua painamalla poistopainike pohjaan kahden sekunnin ajaksi. Laite antaa äänimerkin poistamisen onnistuessa. Kolmas tapa äänitteen poistamiseksi on painaa lyhyesti poistopainiketta, jolloin laite alkaa toistaa äänitteitä järjestyksessä vanhimmasta alkaen. Tallennuspainikkeella voidaan keskeyttää äänitteen toistaminen. Poistopainikkeella siirrytään seuraavaa ääniteeseen. Oikean äänityksen löydyttyä pidetään poistopainiketta pohjassa kahden sekunnin ajan, jolloin laite soittaa äänimerkin onnistuneen poistamisen merkiksi. [1.]

Touch memon äänenvoimakkuutta voidaan säätä laitteen kyljessä olevalla rullalla. Äänenvoimakkuutta suositellaan säädettävän jonkin työkalun, esimerkiksi kynän avulla. Kuvassa 1 on Touch memo -laite.

Kuva 1. Touch memo -laite [2].

Penfriend

Penfriend käynnistyy pitämällä käynnistyspainiketta pohjassa kolmen sekunnin ajan. Laite sammutetaan pitämällä käynnistyspainiketta pohjassa kolme sekunttia. Laite sammuttaa itsensä automaattisesti kolmen minuutin päästä edellisestä toiminnasta.

Äänitteen toistaminen tapahtuu samalla tavalla kuin Touch memolla. Tunnistetarraa osoitetaan laitteen päässä sijaitsevalla lukijapiirillä, jonka jälkeen laite toistaa tallennetun äänitteen. Tunnisteen ollessa tyhjä laite soittaa äänimerkin. [3.]

Myös äänittäminen tapahtuu samalla tavalla kuin Touch memossa. Äänityspainike painetaan pohjaan, jonka jälkeen osoitetaan haluttua tunnistetarraa. Äänitys alkaa äänimerkin jälkeen. Äänitys jatkuu kunnes käyttäjä vapauttaa äänityspainikkeen. Tunnisteen päällekirjoitettus tapahtuu samalla tavalla kuin äänityskin. Tyhjään tunnisteeseen äänittämisen ja päällekirjoituksen äänimerkit eivät poikkea mitenkään toisistaan, joten käyttäjän tulee olla varma, että tunnistetarra on oikea. [3.]

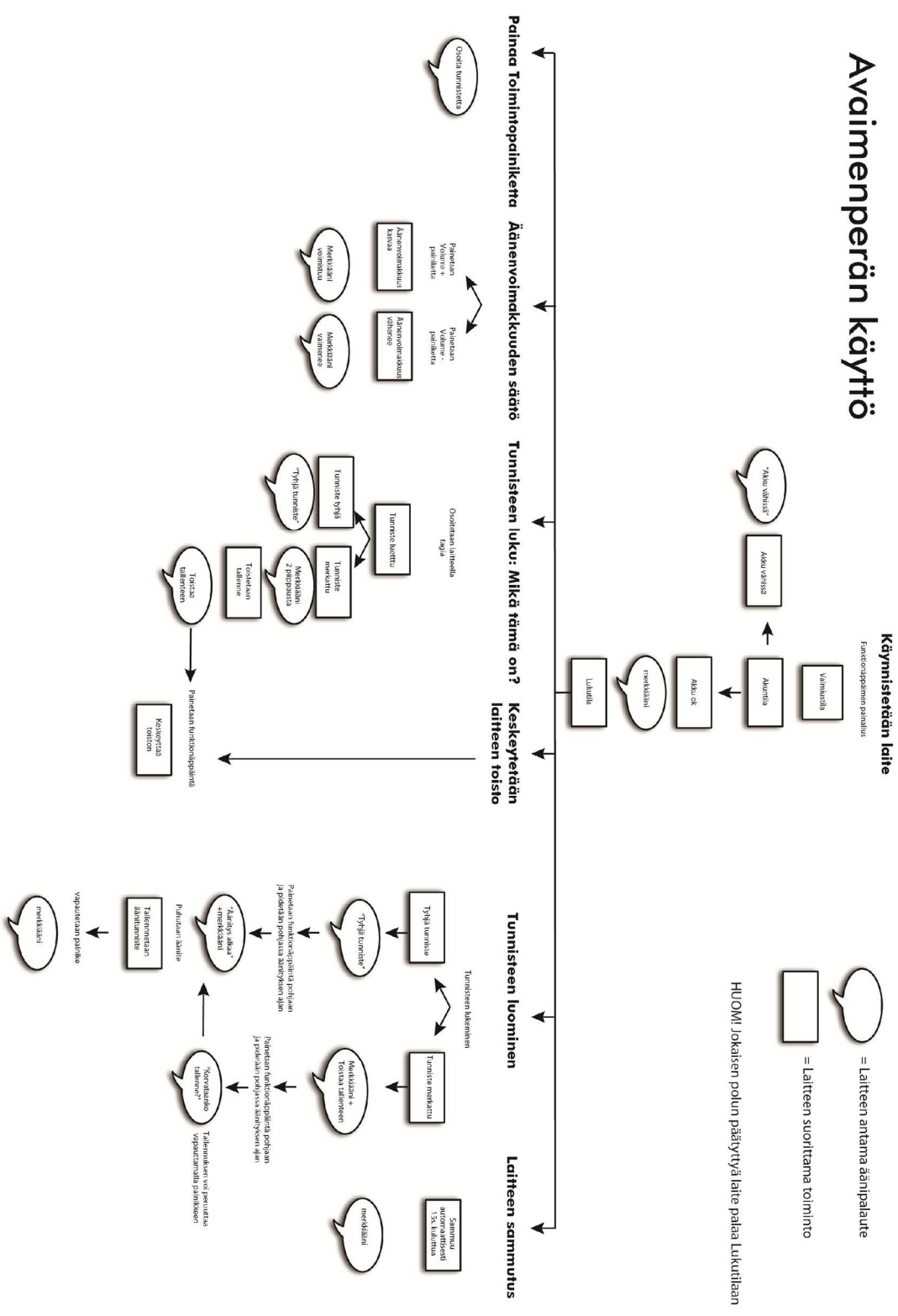
Penfriendissä on tilapainike, jolla voidaan muuttaa laitteen käyttötilaa. Aiemmin kerrotut toiminnot ovat laitteen normaalissa käyttötilassa. Pitämällä tilapainiketta pohjassa kahden sekunnin ajan voidaan vaihtaa seuraavan käyttötilaan. Laitteen käyttötila ilmaistaan piippausten lukumäärällä. Normaalin käyttötilan jälkeen tulee käyttötila, jossa voidaan kuunnella ennaltaäänitettyjä ohjeita. Käyttöohjeen mukaan tätä ei ole laitteeseen toistaiseksi toteutettu. Kolmas käyttötila on MP3-tiedostojen kuuntelua varten. USB-liittimen avulla laitteeseen saa tallennettua MP3-tiedostoja, joita voi kuunnella lailleella. Käyttöohje ei kerro, voiko kappaleiden järjestykseen vaikuttaa. [3.]

Äänenvoimakkuuden säätäminen tapahtuu kahden erillisen painikkeen avulla. Painikkeita painetaan toistuvasti, kunnes haluttu äänenvoimakkuus on saavutettu. Kuvassa 2 on Penfriend-laite.

Kuva 2. Penfriend-laite [4].

2.2 Laitteen vaatimusten määrittely

Vaikka markkinoilla oli jo valmiiksi esineiden tunnistamiseen suunniteltuja laitteita, eivät ne vastanneet täysin käyttäjien tarpeita. Sekä Touch memo että Penfriend koettiin liian monimutkaisiksi käyttää. Tärkeänä vaatimuksena Puhuvalle avaimenperälle olikin sen helppokäyttöisyys. Laitteen käytön pitäisi pystyä oppimaan ilman käyttöohjetta. Tähän tavoitteeseen pyrittiin rajoittamalla laitteen näppäimien ja toimintojen lukumäärää. Vaikka kilpailevien laitteiden käyttöä monipuolistavat ominaisuudet, kuten MP3-tiedostojen soitto sekä vanhojen tallenteiden selaaminen ja poistaminen kuullostavat hienoilta, ovat ne useiden käyttäjien mielestä turhia ja laitteen käyttöä monimutkaistavia. Niinpä Puhuvan avaimenperän toiminnallisuus haluttiin rajata vain välttämättömiin perustoimintoihin. Laitteella tulisi pystyä lukemaan tunnisteita, äänittämään uusia puhetunnisteita sekä tarvittaessa korvaamaan äänitteet uusilla. Puhuvan avaimenperän toiminnasta laadittiin toimintakaavio, jota muokattiin käyttäjiltä saadun palautteen perusteella kehitystyön aikana. Laitteen viimeisin toimintakaavio on esitetty kuvassa 3. Kaaviosta selviää lopullisen prototyypin toiminnallisuus.



Kuva 3. Kaavio Puhuvan avaimenperän toiminnasta.

Alkuperäinen suunnitelma oli käyttää laitteen toiminnan ohjaamiseen vain yhtä painiketta. Näkövammaisten joukossa on kuitenkin henkilöitä, jotka kärsivät heikentyneestä

kuulosta sekä henkilöitä, jotka kuulevat pienimmätkin äänet. Niinpä laitteessa tulisi olla mahdollisuus äänenvoimakkuuden säätämiseen. Tähän tarkoitukseen pohdittiin kahta eri vaihtoehtoa: Touch memon kaltaista rullalla toimivaa tai Penfriendin kaltaista painikkeilla toimivaa äänenvoimakkuuden säätöä. Näistä vaihtoehdoista päädyttiin painikkeiden käyttämiseen, sillä sen avulla on helpompi ilmaista uuden äänenvoimakkuuden taso. Jokaisen painalluksen jälkeen laite soittaa äänimerkin uudella äänenvoimakkuudella, minkä uskottiin olevan käyttäjän kannalta intuitiivista. Näin ollen päädyttiin kolmen painikkeen käyttämiseen: kaksi painiketta äänenvoimakkuuden säätämistä varten sekä toimintapainike.

Toimintapainiketta oli alun perin tarkoitus käyttää pelkästään äänitunnisteiden tallentamista varten. Tällöin laitteen olisi pitänyt olla jatkuvasti päällä tunnistetarrojen lukemista varten. Se ei olisi ollut mahdollista johtuen korkeasta virrankulutuksesta RFID-tunnisteita luettaessa. Laitetta olisi joutunut lataamaan jatkuvasti. Näin ollen päädyttiin ratkaisuus, jossa toimintapainiketta käytetään myös laitteen käynnistämiseen. Laite sammuttaa itsensä tietyn ajan kuluttua automaattisesti.

Laitteella äänittämisen logiikkaa haluttiin muuttaa kilpaileviin laitteisiin verrattuna. Puhuvassa avaimenperässä päädyttiin ratkaisuun, jossa tunnistetarra luetaan ensin ja vasta tämän jälkeen äänitys aloitetaan painamalla toimintapainike pohjaan. Äänitys jatkuu, kunnes äänityspainike vapautetaan. Tällöin käyttäjän ei tarvitse pitää painiketta pohjassa etsiessään oikeaa tunnistetta. Käyttäjä voi olla myös varma, että kyseessä on oikea tunnistetarra, sillä laite toistaa luetun tunnisteen sisällön ennen äänityksen aloittamista.

Yksi olennainen käyttömahdollisuuksien rajoite Touch memossa ja Penfriendissä on niiden suuri koko. Molemmat ovat noin 15 cm pitkiä ja 3 cm paksuja. Puhuvasta avaimeperän tulisi tehdä mahdollisimman pieni, suunnilleen tulitikkuaskin kokoinen. Tällöin sitä olisi helppo kuljettaa mukana. Tämä lisäisi sen käyttömahdollisuuksia huomattavasti. Laitteen tulisi myös olla tyylikkään näköinen, sillä vaikkei käyttäjä itse laitetta näkisikään, on muiden ihmisten mielipiteellä hänelle merkitystä. Laitteen muodon tulisi myös kertoa käyttäjälle, miten laite mahdollisesti toimii. Tällöin laitteen käytön oppiminen olisi mahdollisimman vaivatonta.

Olennainen vaatimus Puhuvalle avaimenperälle oli tunnisteiden nopea ja helppo lukeminen. Touch memossa ja Penfriendissä käytetty optinen lukeminen koettiin hankalaksi

käytön kannalta. Tunnistetarroja tulee näillä laitteilla osoittaa suoraan laitteen päässä olevalla lukijapiirillä. Lisäksi tunnistetarraa pitää melkeinpä koskettaa laitteella. Optiset tunnistetarrat tai laitteen lukijapiiri saattavat likaantua, jolloin tunnisteiden lukeminen vaikeutuu. Puhuvassa avaimenperässä haluttiin käyttää RFID-tekniikkaa esineiden merkitsemiseen. RFID-tunnisteiden lukeminen ei vaadi suoraa näköyhteyttä lukijan ja tunnistetarran välillä, ja lukuetäisyys on optista lukemista parempi. Tunnistetarroja ei tarvitse osoittaa yhtä tarkasti optisiin lukijoihin verrattuna. Lukuetäisyys tulee kuitenkin rajoittaa muutamaan senttimetriin, jotta voidaan olla varmoja, että luettu tunniste on haluttu.

Äänitiedostojen tallentamiseen haluttiin käyttää SD-muistikorttia. Ne ovat usein huomattavasti edullisempia kuin piirilevylle kiinteästi asennettavat muistipiirit. Kehitystyön aikana sen avulla on helppo siirtää äänitiedostoja laitteen ja tietokoneen avulla. Tästä olisi apua äänityksen suodatusta testattaessa. Äänitiedostoa voidaan tutkia esimerkiksi Audacity-nimisellä ohjelmalla. Ohjelman avulla voidaan selvittää äänitiedoston tai sen osan spektri. Tällöin eri suodatinvaihtoehtojen vaikutus äänityksessä poimittuihin taajuuksiin nähdään suoraan kuvaajasta. Lisäksi äänitiedostoja on hyvä kuunnella useammista eri kaiuttimista, jolloin voidaan paikantaa, johtuvatko äänenlaadun ongelmat äänityksestä vai toistosta. [31.]

Virransyöttöä varten Puhuvaan avaimenperään valittiin ladattava akku. Kertakäyttöisiä paristoja joutuisi vaihtamaan liian usein, johtuen RFID-tunnisteiden lukemisen melko korkeasta virrankulutuksesta. Laturi liitetään laitteeseen microUSB-liittimellä. Samaa liitintä olisi mahdollista käyttää myöhemmin esimerkiksi tiedostojen varmuuskopiointiin. Prototyyppiin tätä ominaisuutta ei haluttu vielä lisätä. Akun lataamista varten laiteessa tulee olla microUSB-liittimen lisäksi latauspiiri, jonka tehtävänä on kontrolloida akulle syötettävää sähkövirtaa.

3 RFID-tekniikan perusteet

RFID (Radio Frequency Identification) on yleisnimitys radiotaajuuksilla toimiville tekniikoille, joita käytetään esineiden merkitsemiseen ja havainnointiin. Tekniikka perustuu tiedon tallentamiseen RFID-tunnisteeseen ja tiedon lukemiseen RFID-lukijalla radioaaltojen avulla. Tärkeimmät osat RFID-järjestelmässä ovat tunniste, joka on kiinnitetty seurattavaksi haluttuun kohteeseen, ja lukija, joka havaitsee tunnisteen läsnäolon ja

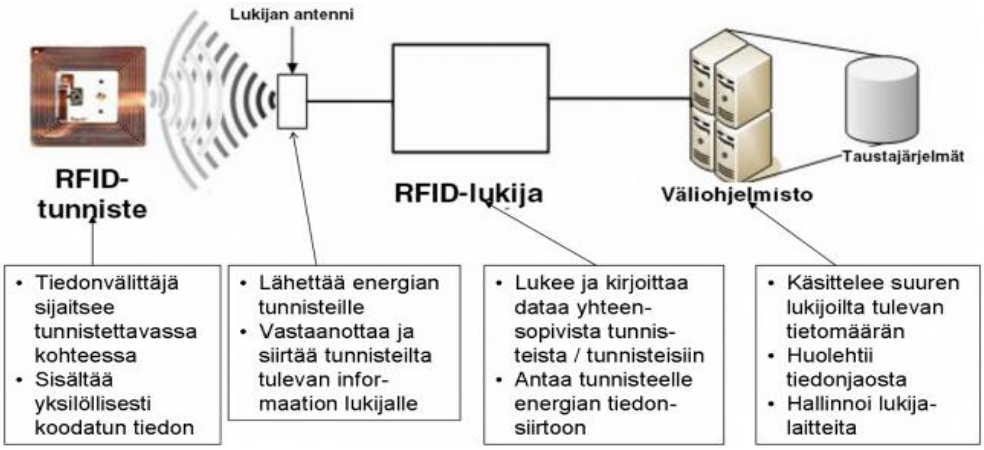
lukee seurattavaan kohteeseen tallennetun informaation. Usein RFID-järjestelmiin kuuluu myös taustajärjestelmä, johon lukijan saama tieto lähetetään, sekä väliohjelmisto lukijoiden hallintaa varten [5.]

RFID-teknologia on hyvä valinta lähes kaikille, jotka haluavat seurata fyysistä omaisuuttaan. Tuotteiden valmistajat käyttävät RFID-teknologiaa toimitusketjun suunnitteluun ja toteutukseen. Jälleenmyyjät käyttävät RFID:tä varkauksien hallintaan, toimitusketjun tehostamiseen ja tuotteiden kysynnän seuraamiseen. Lääkevalmistajat käyttävät RFID-järjestelmiä lääkeväärennösten ehkäisemiseen. RFID-tekniikkaa käytetään myös muun muassa lemmikki- ja tuotantoeläimien merkitsemiseen, kulunvalvontaan ja ajoneuvojen seurantaan. [5.]

RFID:tä verrataan usein vanhempaan, mutta vielä laajasti käytössä olevaan viivakooditekniikkaan. Viivakoodeihin verrattuna RFID-teknologiassa on kuitenkin useita etuja. Iso rajoite viivakoodeissa on se, että niiden lukeminen vaatii suoran näköyhteyden. Tunnistettava esine tulee asettaa lukijaa oikeassa asennossa, mihin kuluu kallisarvoista aikaa. RFID-tekniikka tarjoaa mahdollisuuden esineiden tunnistamiseen etäältä, eikä se ole yhtä herkkä esineen oikean suunnan suhteen. Lukija pystyy lukemaan tunnisteen jopa esineiden läpi. RFID-tunnisteita voidaan lukea useita kerrallaan, mikä lisää järjestelmän nopeutta entisestään. Lisäksi joidenkin RFID-tunnisteiden muistiin voidaan lisätä informaatiota, jota on mahdollista muokata ja lukea useita kertoja uudelleen. [5.]

3.1 RFID-järjestelmän osat

RFID-järjestelmä koostuu tunnisteista, lukijalaitteista antenneineen, lukijoiden hallinnoinnista huolehtivasta ja tiedon taustajärjestelmään lähettävästä väliohjelmistosta sekä taustajärjestelmästä. RFID-järjestelmän osat on esitelty kuvassa 4.



Kuva 4. RFID järjestelmän osat [6].

RFID-tunniste on tunnistattavaan objektiin kiinnitettävä tarra, ranneke, kortti, implantti, tms., joka sisältää antennin ja tiedon säilytykseen käytettävän sirun. Tunnistettavia objekteja voivat olla esimerkiksi seurantaa vaativa lähetys tai lemmikkieläin. Tunnisteiden muisti sisältää yksilöllisen kiinteän sarjanumeron ja sirun tyypistä riippuen kerran tai useamman kertaa ohjelmoitavaa muistia. Useissa sovelluksissa tunnisteista käytetään vain sarjanumeroa ja varsinainen tieto haetaan taustajärjestelmästä.

Lukijalaite havaitsee riittävän lähellä olevat tunnisteet. Sen avulla tunnisteiden sisältöä voidaan lukea ja kirjoittaa ilman fyysistä kontaktia. Lukijalaite antaa tunnisteelle sen vaatiman energian ja huolehtii niiden välisen kommunikoinnin aloittamisesta. Lukijalaite koostuu varsinaisesta lukijasta ja yhdestä tai useammasta antennista. Useissa sovelluksissa lukija lähettää tunnisteelta luetun tiedon taustajärjestelmän käsiteltäväksi, mutta laitteesta riippuen lukija voi toimia myös itsenäisesti.

Miljoonien tunnisteiden liikkuminen suuren toimitusketjun läpi tuottaa valtavan määrän dataa. Väliohjelmisto standardoi tavan, jolla tunnisteiden tuottama tietomäärä käsitellään. Se käsitelee tunnisteiden tiedot ja lähettää varsinaiselle sovelluksella vain merkityksellisen informaation. Lisäksi väliohjelmisto huolehtii järjestelmän fyysisten laitteiden konfiguroinnista ja ohjaamisesta.

On hyvä huomioida, että RFID-tunnisteen sisältämä tunnistenumero on vain yksilöllinen tunniste, eikä sinänsä sisällä tietoa itse tuotteesta. Tunnistettavan objektin tietojen saamista ja niiden muokkaamista varten RFID-järjestelmä tarvitsee taustajärjestelmän. Tunnisteella merkityn tuotteen liikkuessa toimitusketjun läpi on toimitusketjun osallisten

voitava päivittää järjestelmään tuotteen sijaintitiedot ja tuotteen omistajan on pystyttävä seuraamaan omia tuotteitaan. Kulunvalvonnan sovelluksissa taustajärjestelmän tehtävänä taas voi olla esimerkiksi käyttäjän oikeuksien tarkistaminen ja niiden salliessa ovien avaaminen.

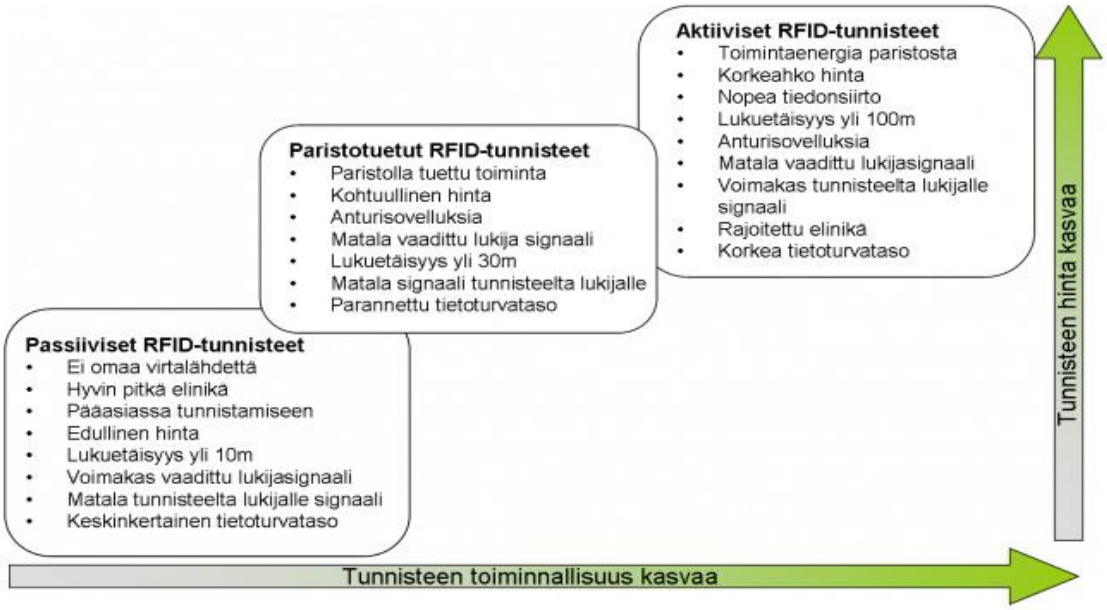
3.2 RFID-tekniikan teoreettinen toiminta

Passiivinen tunniste, millaisia tässäkin projektissa käytetään, koostuu antennista sekä tunnisteen informaation sisältävästä sirusta. LF- ja HF-taajuusalueilla toimivassa tunnisteessa on kuparisia silmukoita, jotka muodostavat käämin ja toimivat tunnisteen antennina. Lukijassa on vastaavanlainen antenni. Lukija johtaa käytetyn protokollan määrittämällä taajuudella vaihtovirtaa antennisilmukkaan oskilloivan magneettikentän luomiseksi. Luotu magneettikenttä indusoi vastaavan vaihtovirran tunnisteen käämiin tunnisteen sijaitessa riittävän lähellä. Tunnisteen siru saa virtansa indusoituneesta virrasta. Sirun EEPROM-muistissa olevaa dataa käytetään moduloimaan tunnisteen käämin virtaa. Tämä ilmenee magneettikentän välityksellä lukijan antennin jännitteen muutoksena, minkä lukija tulkitsee informaatioksi. [6.]

UHF- ja mikroaaltotaajuuksilla toimiva tunniste ja lukija kommunikoivat radioaaltojen välityksellä. Lukija lähettää antenninsa kautta radioaaltoja, jotka tunnisteen antenni vastaanottaa ja heijastaa ne takaisin lukijalle. Informaation lähettämiseksi tunniste muokkaa heijastetun signaalin ominaisuuksia. Jotkut tunnisteet muokkaavat radioaaltoa omilla fyysisillä ominaisuuksillaan, ja toiset tekevät tämän kytkemällä rinnan antennin kanssa olevaa kuormaa, mikä aiheutaa heijastuneen aallon amplitudin vaihtelua. Informaatio lähetetään tyypillisesti vuoroissa, sillä lukija ja tunniste käyttävät samaa taajuutta. [5; 6.]

UHF-taajuusalueella toimiva RFID-tekniikka toimii siis eri tavalla kuin HF- ja LF-taajuusalueella toimivat. UHF-taajuusalueella tapahtuvassa RFID-tunnistuksessa tunniste ja lukija kommunikoivat sähkömagneettista säteilyä, tässä tapauksessa radioaaltoja lähettämällä, kuten esimerkiksi radiossa ja matkapuhelimessa. HF- ja LF-taajuusaluella tapahtuvassa RFID-tunnistuksessa on taas kysymys induktiivisesta kytkennästä, jossa tunniste reagoi lukijan luomaan oskilloivaan magneettikenttään. LF- ja HF-tekniikan toimintaa voisi verrata muuntajan toimintaan. [6.]

Aktiivitunnisteet sisältävät antennin ja sirun lisäksi oman virtalähteen. Aktiivinen tunniste ja lukija keskustelevat keskenään kuin kaksi radiota tai matkapuhelinta. Aktiivisen tunnisteen lukuetäisyys voi oman virtalähteen ansiosta olla jopa yli sata metriä. Toisaalta aktiivisen tunnisteen tuotantokustannukset ovat passiivista tunnistetta huomattavasti korkeammat. Aktiiviset tunnisteet ovat lisäksi usein eliniältään passiivisia tunnisteita lyhytikäisempiä. Kuvassa 5 vertaillaan passivisten, paristotuettujen ja aktiivisten tunnisteiden ominaisuuksia ja niiden hintaa. [6.]



Kuva 5. Erityyppisten RFID-tunnisteiden ominaisuuksien vertailua [6].

3.3 Taajuusalueet ja standardit

Jotkut RFID-sovellukset on tarkoitettu vain yhden yhtiön tarpeisiin, kun taas toiset järjestelmät jakavat informaatiota maailmanlaajuisten yhteistyökumppaneiden kesken. Sovelluksen koosta riippumatta laajasti käytössä olevan standardin tunnisteiden käyttäminen on järkevää jo pelkästään sen seikan vuoksi, että suuret tuotantomäärät laskevat tunnisteiden hintaa. Lisäksi suuremmissa, usean toimijan järjestelmissä tulee pystyä käyttämään samoja tunnisteita. Myös standardien takaamassa valmistajariippumattomuudesta on hyötyä. Se takaa, että suunniteltuun järjestelmään sopivia laitteita voi ostaa myöhemminkin vapaasti, sitoutumatta tiettyyn toimittajaan. Vapaan kilpailun takaamiseksi osa standardeista on kuitenkin ns. vapaita standardeja, joiden mukaisia

laitteita kuka tahansa voi valmistaa. Tärkeimmät standardit määrittävät käytettävän tiedonsiirtoprotokollan sekä tunnisteen tietosisällön. [5; 8.]

LF-taajuusaluetta (Low Frequency) käytetään yleisesti eläinten tunnistuksessa, autojen käynnistyksenestojärjestelmissä ja kulunvalvonnassa. Kulunvalvontajärjestelmät on usein toteutettu suljettuina järjestelminä 125 kHz:n taajuudella. Karjan tunnistukseen käytettävän tunnisteen tietosisältö on määritelty standardissa ISO11784 ja tiedonsiirtoprotokolla 134 kHz taajuudella on määritetty standardissa ISO11785. [8.]

HF-taajuusalueen (High Frequency) käytännon standarditaajuus on 13,56 MHz, joka on kansainvälisesti vapaa taajuus. HF-taajuusalueella on olemassa sovittuja standardeja, joista standardi ISO14443 ei kuitenkaan takaa valmistajariippumatonta tunnisteiden ja lukijoiden yhteensopivuutta, eikä siten tuo kaikkia standardoinnin etuja. Kuitenkin Philips Mifare -tekniikka on saavuttanut käytännössä standardin aseman. Mifarea käytetään muun muassa erilaisissa maksusovelluksissa. Standardi ISO15693 on valmistajariippumaton. Tämän standardin mukaisia tunnisteita käytetään esimerkiksi kirjastoissa aineiston tunnistamiseen ja jäljittämiseen. Joitain HF-tekniikan etuja UHF-tekniikkaan verrattuna ovat sen parempi läpäisykyky vettä sisältävissä aineissa, häiriösietoisuus teollisuusympäristöissä ja ongelmattomuus heijastusten suhteen. [8; 9.]

UHF-tekniikkaa (Ultra High Frequency) käytetään erityisesti logistiikan sovelluksissa. Tätä taajuusaluetta hyödyntäviä logistiikkajärjestelmiä ovat pystyttäneet muun muassa sellaiset suuryritykset kuin Wal-Mart ja Tesco. UHF-taajuusalueella toimivien RFID-järjestelmien taajuudet ovat hieman erilaiset ympäri maailmaa. Esimerkiksi Yhdysvalloissa RFID-järjestelmien käyttämä UHF-taajuusalue on 902–928 MHz, kun taas Euroopassa sallittu taajuusalue on 869 MHz:n tietämillä. Tällä hetkellä merkittävin standardi UHF-taajuusalueella on ISO18000-6C eli Gen2. Sen myötä tunnistus on saatu varmemmaksi ja erityisesti toiminta monilukijaympäristössä on parantunut. [8; 9.]

3.4 Esimerkkisovelluksia

Ensimmäiset kaupalliset RFID-tekniikkaa hyödyntävät sovellukset tulivat markkinoille 1980-luvun puolivälissä. Yhdysvaltain hallituksen tutkimusohjelmassa Los Alamosissa kehitettiin tuolloin RFID-aktiivitunniste tietullien automaattista tunnistusta varten. Los Alamosissa kehitettiin myös RFID-tunniste lehmien ruokinnan ja lääkinnän seurantaa

varten. Lehmien ihon alle kiinnitettiin 125 kHz:n taajuusalueella toimiva passiivinen tunniste, jonka avulla jokaiselle lehmälle voitiin antaa oikea määrä lääkettä. Järjestelmä on edelleen käytössä. [10.]

Nykyään RFID-sovellusten kirjo on laaja, mutta ne voidaan jakaa viiteen pääkategoriaan, jotka ovat kulunvalvonta, esineiden merkitseminen, lavojen ja kartongin seuranta, jäljittäminen sekä älyhyllysovellukset. Luonnollisesti kaikkia sovelluksia ei voida sovittaa näihin kategorioihin, mutta ne ovat riittävät antamaan yleiskuvan RFID-tekniikan mahdollisuuksista. Seuraavaksi kerrotaan yksityiskohtaisemmin muutamista Suomessa toteutetuista RFID-tekniikaa hyödyntävistä sovelluksista. [5.]

Tavaravirran hallinta ABB:llä

ABB Oy käyttää RFID-tekniikkaa ulospäin suuntautuvan tavaravirran hallinnassa. Vilant Systems Oy:n kehittämä järjestelmä estää lähetysten virhelastauksia sekä kirjaa tavarasiirrot automaattisesti varastokirjanpitoon. Tavaroita ei enää tarvitse kerätä konsolidointialueelle vaan rekkojen perävaunuja voidaan käyttää varastoina. Näin säästetään merkittävästi lattiatilaa. [11.]

Järjestelmä perustuu lastauslaiturilla sijaitseviin RFID-lukijaportteihin, joiden läpi kuljetetaan kaikki lähetykseen kuuluvat kuljetusyksiköt. Kuljetusyksiköt on merkattu UPM Raflatacin RFID-tunnistetarroilla. RFID-ohjelmisto on integroitu ABB:n omaan SAP-järjestelmään. [11.]

SAP-järjestelmään kirjataan lastaamaan tulevan auton rekisterinumero, minkä jälkeen lähetyksen etenemistä valvotaan automaattisesti. Järjestelmässä on tieto kyseisen lähetyksen sisällöstä. Jos rekkaan yritetään lastata lähetykseen kuulumatonta tavaraa, antaa portti virheilmoituksen. Porttia ei myöskään pysty sulkemaan ennen kuin kaikki lähetykseen kuuluva tavara on lastattu rekkaan. Näin ollen virheellisten lähetysten tekeminen on käytännössä mahdotonta. [11.]

VR Transpoint: Tavaravaunujen seuraaminen

Vuoden 2009 lopussa VR Transpoint käynnisti RFID-hankkeen, jossa toteutettiin täysimittainen EPC/RFID-järjestelmä 10000 tavaravaunun seuraamista varten. Jokaiseen vaunuun on kiinnitetty UHF-taajuusalueella toimiva passiivinen Confidex Ironside Gen2 -RFID-tunniste. Rautatiepihan työtekijät voivat kävellä junan rinnalla ja käyttää käsilukijaa jokaisen vaunun tunnisteen lukemiseen. Tiedot siirretään automaattisesti logistiikkajärjestelmään. Näin ollen manuaalista tietojen kirjaamista ei enää tarvita. Lisäksi useita kiinteitä RFID-lukijoita asennettiin ympäri Suomen rautatieverkon. Tämä paransi automaattista tiedonsiirtoa VR Transpointille ja sen asiakkaille. Suomen liikennevirasto on myös vuonna 2012 tilannut RFID-lukijat liitettäväksi rautatieverkoston sensoreihin. Tällä tavoin on mahdollista saada automatisoitua tietoa esimerkiksi vaunujen vaurioituneista osista. [12.]

RFID-järjestelmän tärkein etu on se, että henkilöstö voi tunnistaa vaunut käyttämällä käsilukijaa. Tämä tehostaa vaunujen hallinnointia ja lisää käyttöastetta. Automaatinen prosessi vähentää myös inhimillisiä virheitä ja nopeuttaa vaunujen vaihtotyön ohjausta ratapihoilla. Asiakkaille tämä näkyy vaunujen parempana saatavuutena. Asiakkaat saattavat hyötyä myös vaunujen paremmasta jäljitettävyydestä. Lisäksi vaunujen huolto parani ja korjausten läpimenoaika lyheni. [12.]

Kauhajoen kirjasto

Kirjastoissa RFID-tekniikalla parannetaan lainattavien nimikkeiden yksilöllistä tunnistamista sekä pyritään minimoimaan hävikkiä. Jokaista nidettä ei tarvitse tunnistaa yksi kerrallaan vaan ne voidaan palauttaa pinona, mikä nopeuttaa asiakaspalvelua huomattavasti. [13.]

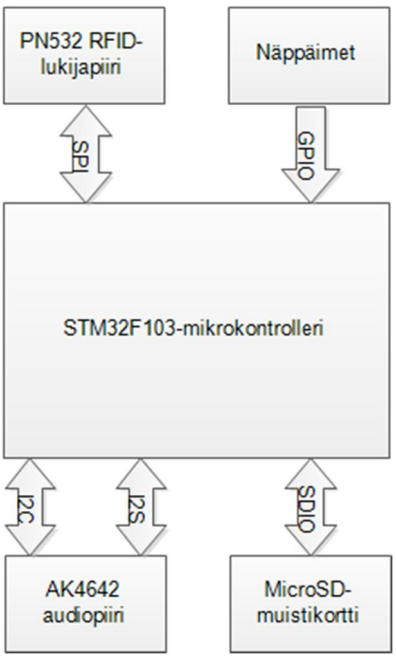
Kauhajoen kaupunginkirjasto oli ensimmäinen kirjasto Suomessa, joka otti RFID-tekniikan käyttöön kirjastojärjestelmään. Onnistuneesta pilotoinnista alkanut prosessi on osoittautunut toimivaksi, ja nykyään RFID-järjestelmä on käytössä useassa kirjastossa ympäri Suomea. Kauhajoen kirjastolle RFID-tunnisteet ja laitteet toimitti ToP Tunniste. Kauhajoen kirjaston laitteet ja tunnisteet toimivat HF-taajuusalueella. [13.]

Etäluettava RFID-tarra korvaa kirjastossa viivakoodin ja perinteisen sähkömagneettisen hävikinestotarran. Niteeseen liitetty tunnistetarra sisältää mikrosirun, johon on tal-

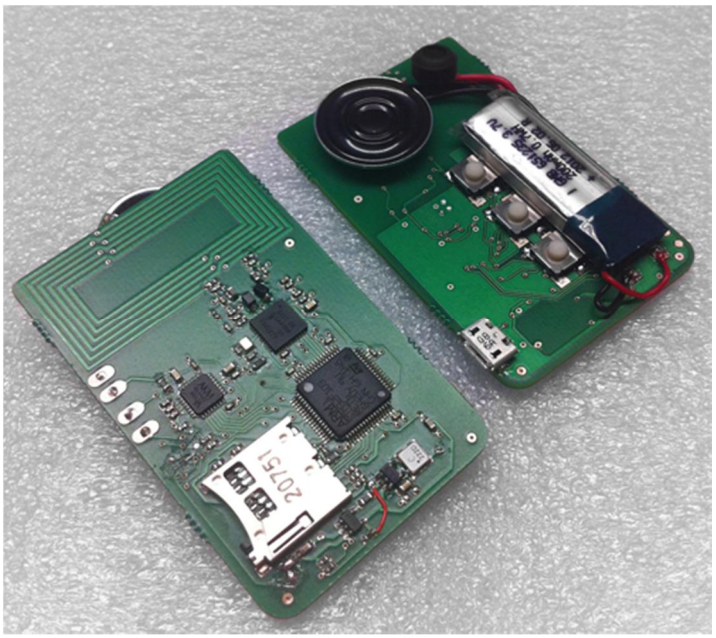
lennettu aineistokappaletta koskevaa tietoa. Lainauksen ja palautuksen yhteydessä mikrosiru kytkee automaattisesti hävikineston päälle ja pois. Tämä lisää oleellisesti suojauksen luotettavuutta. [13.]

4 Laitteen fyysiset komponentit ja käytetyt kommunikointiväylät

Tässä luvussa kerrotaan perustiedot ohjelmoityön kannalta olennaisista komponenteista ja niiden hallintaan käytetyistä kommunikointiväylistä. Nämä komponentit ovat RFID-tunnisteiden lukemiseen käytettävä PN532-RFID-lukijapiiri, audiotiedostojen toistamiseen ja tallentamiseen käytettävä AK4642-audiopiiri, audiotiedostojen tallentamiseen käytettävä microSD-muistikortti sekä STM32F103-mikrokontrolleri. Edellä mainittujen komponenttien lisäksi elektroniikkaan kuuluu myös muun muassa akku, akun latauspiiri, regulaattori sekä piirilevyllä oleva liuska-antenni. Näiden toimintaa ei ole kuitenkaan kuvattu, sillä ne eivät ole merkityksellisiä laitteen ohjelmoinnin näkökulmasta ja näin ollen ne jäävät työn rajauksen ulkopuolelle. Kuvassa 6 esitetään laitteen elektroniikan lohkokaavio, josta selviää laitteen olennaisimmat komponentit sekä niiden ohjaamiseen käytetyt kommunikointiväylät. Kuvassa 7 on Puhuvan avaimenperän lopullinen piirilevy komponentteineen.



Kuva 6. Puhuvan avaimenperän elektroniikan lohkokaavio.



Kuva 7. Puhuvan avaimenperän lopullinen piirilevy komponentteineen.

4.1 STM32F103RET6-mikrokontrolleri

Mikrokontrelleriksi laitteeseen valittiin STMicroelectronicsin suunnittelema ja valmistama STM32F103RET6. Valinta tehtiin lähinnä käytännön syistä. Electrialla, missä laite suunniteltiin ja toteutettiin, oli valmiina vastaavalla mikrokontrollerilla varustettu STM3210E-EVAL-kehityskortti. Kehityskortti sisälsi suurimman osan lopullisessa laitteessa käytettävistä oheislaitteista. Lisäksi Electrian henkilökunnalla oli edellisistä projekteista kertynyttä kokemusta STMicroelectronicsin mikrokontrollereiden ohjelmoinnista. Näin ollen kehitystyö päästiin aloittamaan välittömästi, ja tukea mahdollisiin ongelmatilanteisiin oli saatavilla. Tästä oli hyötyä erityisesti projektin alkuvaiheessa.

STM32F103RET6-mikrokontrollerin korkein mahdollinen kellotaajuus on 72 MHz. 36 MHz:n kellotaajuus osottautui kehitystyön aikana riittäväksi. Virrankulutuksen kannalta on järkevää käyttää mahdollisimman matalaa kellotaajuutta, joten mikrokontrollerin kellotaajuudeksi valittiin 36 MHz. Järjestelmäkellon luomiseen on mahdollista käyttää joko kontrollerin sisäistä tai ulkoista kellokidettä. Testeissä sisäinen kellokide osottautui riittävän tarkaksi. Niinpä mikrokontrollerin järjestelmäkellon lähteeksi valittiin sisäinen kellokide, sillä ulkoisen kellokiteen käyttäminen olisi vaatinut tilaa piirilevyltä, minkä lisäksi ulkoinen kellokide lisäisi laitteen tuotantokustannuksia. [14.]

Puhuva avaimenperä suunniteltiin käynnistettäväksi käyttöpainikkeesta, olemaan toiminnassa tietyn ajan ja tämän jälkeen menevän automaattisesti virransäästötilaan. Käytetyssä mikrokontrollerissa on useita eri virransäästötiloja, joista kaksi olivat potentiaalisia laitteen virrankulutuksen hallintaa varten. STANDBY-tilassa mikroprosessorin virrankulutus on noin 3 µA. Tässä tilassa mikrokontrolleri ei kuitenkaan säilytä pinnien tiloja muistissa. Näin ollen piirilevyn komponentit, jotka on mahdollista sammuttaa reset-pinnillä, saattavat käynnistyä mikrokontrollerin ollessa lepotilassa. Sen estämiseksi näiden oheislaitteiden reset-pinneille olisi tullut laittaa ylös- tai alasvetovastukset komponentista riippuen. Tätä ei kuitenkaan huomioitu piirilevyä suunniteltaessa. Tästä syystä päädyttiin käyttämään STOP-virransäästötilaa. Tässä tilassa mikroprosessorin virrankulutus on noin 35 µA. Mikrokontrolleri säilyttää kuitenkin pinnien tilat muistissa, joten oheislaitteet eivät mene tahattomasti päälle. [14.]

Mikrokontrollerin ohjelmointiin käytetään JTAG-liitäntää. JTAG-liitäntää on mahdollista käyttää myös laitteen ohjelmiston virheiden etsimiseen ja korjaamiseen. Äänitiedostojen toistamista ja tallentamista varten tulee osa mikrokontrollerin JTAG-väylän käyttämistä pinneistä uudelleenohjata. JTAG-väylää ei voi silloin käyttää laitteen vianetsintään. [14.]

MicroSD-muistikortin ja mikrokontrollerin välinen tiedonsiirto tapahtuu SDIO-väylän kautta. Audiopiirin ja mikrokontrollerin välinen tiedonsiirto hoidetaan kahdella I2S-väylällä, joista toinen on äänitiedostojen toistamista ja toinen äänittämistä varten. I2S-väylien kello- (SCK) ja kanavavalitsinlinjat (WS) on kytketty yhteen. Käyttämättömän I2S-väylän SCK- ja WS-pinni tulee määrittää kelluvaan tilaan, jolloin ne eivät aiheuta häiriöitä aktiiviselle I2S-väylälle. Audiopiirin hallintarekisterien kirjoittamista ja lukemista varten on käytössä I2C-väylä. RFID-lukijapiiriä kontrolloidaan SPI-väylän avulla. Mikrokontrollerissa on tuki USB 2.0 -liitäntää varten. Väylää on tulevaisuudessa mahdollista käyttää esimerkiksi audiotiedostojen varmuuskopiointia varten. Tätä toiminnallisuutta ei ole laitteeseen toistaiseksi ohjelmoitu. Ohjelmointityön lisäksi se vaatisi pieniä muutoksia piirilevylle. [14.]

STM32F103RET6:ssa on sisäistä SRAM-muistia 64 kB. Saatavilla on myös pienemmällä muistilla varustettuja vastaavia mikrokontrollereita. Suunniteltuun laitteeseen ne eivät sovellu, sillä audiotiedostojen äänittämiseen käytetty puskurointi vaatii runsaasti muistia toimiakseen kunnolla. Äänittämistä kokeiltiin myös käyttämällä vähemmän

muistia puskurointiin. Tällöin laite toimi kuitenkin epävakaasti ja saattoi jättää osan datasta tallentamatta, koska microSD-kortille kirjoittaminen ei ollut riittävän nopeaa. [14.]

4.2 PN532-RFID-lukijapiiri

RFID-lukijapiiriksi valittiin NXP:n valmistama PN532. Se on käytössä useissa matkapuhelimissa ja muissa NFC-tekniikkaa hyödyntävissä laitteissa. Piirin avulla voidaan muun muassa lukea ja kirjoittaa tunnistetarroille ja -korteille sekä kommunikoida matkapuhelimien kanssa. Lisäksi sitä voidaan käyttää NFC-tunnisteena. PN532-piirin tukemat tunnisteprotokollat ovat ISO14443A, ISO14443B ja NFCIP-1. Puhuvassa avaimenperässä käytetään Mifaren ISO14443A:n mukaisia tunnistetarroja, johtuen niiden hyvästä saatavuudesta. Laite tukee tällä hetkellä Mifare Classic- ja Mifare Ultralight -tunnisteita. [15; 16.]

PN532:ta voidaan kontrolloida SPI-, I2C- tai USART-väylää käyttäen. Käytettävä väylä valitaan kahdella pinnillä (I0 ja I1). Puhuvassa avaimenperässä piiriä hallitaan SPI-väylällä, jolloin I0:n arvon tulee olla 0 ja I1:n arvon tulee olla 1. Piiri on mahdollista laittaa virransäästötilaan yhden RSTPDN-pinnin avulla. Pinnin ollessa 0 piiri on virransäästötilassa. Pinnin nouseva reuna käynnistää piirin alustustoimenpiteet. [16.]

Puhuvan avaimenperän piirilevyn pinnalla on liuska-antenni, jonka avulla tunnisteet luetaan. Tunnisteiden lukuetäisyyden optimoimiseksi tulee antenni sovittaa laitteeseen sopivaksi. Antennin sovituspiiri ja ohjeet komponenttien arvojen laskemiseksi on annettu NXP:n laatimassa dokumentissa AN1445. [17.]

PN532:n virrankulutus sen ollessa toiminnassa on tyypillisesti 25 mA, mutta piirin lähettäessä virrankulutus nousee 60 mA:iin. Jos antenni on huonosti sovitettu, lähetyksen aikainen virrankulutus saattaa olla jopa 150 mA. Lepotilassa piirin virrankulutus on 1 µA. [16.]

4.3 AK4642-audiopiiri

Puhuva avaimenperä on tarkoitettu heikkonäköisten henkilöiden apuvälineeksi, joten toistettavien äänitiedostojen äänenlaatu on ensisijaisen tärkeää. Tästä syystä äänen

tuottamiseen ja tallentamiseen haluttiin käyttää laadukasta audiopiiriä. Toisena vaihtoehtona olisi ollut käyttää mikrokontrollerin omaa ADC-yksikköä tai ulkoista korkeampi resoluutioista ADC-piiriä äänittämistä ja DAC-muunninta äänitiedoston toistamista varten. Tämä ratkaisu olisi monimutkaistanut piirilevyn suunnittelua huomattavasti. Muuntimet olisivat esimerkiksi vaatineet erilliset vahvistimet, jotta äänen signaali olisi saatu riittävän voimakkaaksi näytteistämistä ja toistamista varten. Lisäksi äänen suodattaminen ohjelmallisesti olisi kuluttaneet runsaasti laskentatehoa ja monimutkaistanut laitteen ohjelmointia. Todennäköisesti äänenlaatu olisi joka tapauksessa ollut huomattavasti heikompi kuin käyttämällä ulkoista audiopiiriä.

Kehitystyössä käytetty STM3210E-EVAL-kehityskortti sisälsi Asahi Kasein valmistaman AK4343-äänisirun. Testikäytössä piiri tuottaman äänen laatu osottautui hyväksi. AK4343 on suunniteltu ainoastaan PCM-muotoisten äänitiedostojen toistamiseen. Tästä syystä tarvittiin vastaava äänipiiri, joka soveltuu myös äänitiedostojen nauhoittamista varten. AK4343:n datalehdessä esitellään samalla pinnijärjestyksellä ja samoilla rekisterimäärityksillä olevaa laajennettua, myös äänen nauhoittamiseen soveltuvaa audiopiiriä tuotenimeltään AK4642. Piiri on myös kohtuuhintainen, joten se valittiin laitteeseen.

AK4642-piirissä on liitäntämahdollisuudet kahdelle stereomikrofonille. Puhuvan avaimenperän nykyisessä versiossa käytetään ainoastaan yhtä sisäistä monomikrofonia, sillä stereoääntä ei koettu tarpeeliseksi laitteen kaiuttimen pienen koon ja laitteen käyttötarkoituksen vuoksi. Toista sisäänmenoa on laitteen tulevissa versioissa mahdollista käyttää laadukkaan ulkoisen mikrofonin liittämiseen. Sisäänmenoille on oma vahvistin, jonka vahvistuksen voimakkuus voidaan valita neljästä eri arvosta (0 dB, +20 dB, +26 dB tai +32 dB). Äänityksen vahvistustasoa valitetessa tulee huomioida, että äänen voimakas vahvistaminen heikentää äänen signaali-kohinasuhdetta. Äänityksen tasoa voidaan hienosäätää piirin sisäisellä digitaalisella voimakkuussäätimellä. Piirissä on lisäksi mahdollisuus äänitason automaattiseen hallintaan piirin ALC-yksiköllä (Automatic Level Control). Ominaisuus osottautui hyödylliseksi, sillä ALC:tä säätämällä saatiin vähennettyä kohinan osuutta äänityksestä. [18.]

Äänipiirissä on rekisterimäärityksillä säädettävä ylipäästösuodin ja alipäästösuodin, joita voidaan käyttää sekä sisäänmenoille että ulostuloille. Äänen korkeimpien ja matalimpien taajuuksien suodattaminen on tärkeää laitteessa käytettyjen pienikokoisten kaiuttimien kanssa, sillä erityisesti matalien taajuuksien toistaminen aiheuttaa äänen säröytymistä. [18.]

AK4642:ssa on erilliset ulostulot kaiuttimelle ja kuulokeliitännälle. Kuulokeliitäntää ei lisätty laitteeseen, koska laitteen koko haluttiin säilyttää mahdollisimman pienenä. Osa testikäyttäjistä olisi halunnut laitteeseen kuulokeliitännän, jotta laitetta voisi käyttää muita häiritsemättä. Lisäksi heikkokuuloinen käyttäjä voisi liittää kuulokeliitäntään kuulolaitteen. [18.]

Audiopiiriä hallitaan sirun sisäisten rekisterien avulla. Rekisteriarvojen kirjoitus ja lukeminen tapahtuu I2C-väylän avulla. Audiotiedostojen näytteiden siirtäminen piirille ja piiriltä mikrokontrollerille hoidetaan I2S-väylän kautta. Nauhoittaessa jokainen näyte on 16-bittinen ja kahden komplementin muodossa. Näytteet siirretään mikrokontrollerille eniten merkitsevä bitti ensin. Äänitiedostoa siirrettäessä audiopiirille sen näytteet voidaan lähettää joko eniten tai vähiten merkitsevä bitti ensin. [18.]

Piirin tyypillinen virrankulutus audiotiedostoa toistettaessa on 8 mA, kun käytetään kaiutinta, ja 5 mA, kun kuulokeliitännän on käytössä. Tyypillinen virrankulutus piirin ollessa sammutettuna on 10 µA. Piiri sammuu asettamalla reset-pinni alas. [18.]

4.4 MicroSD-muistikortti

Äänitiedostojen tallentamista varten laitteessa on microSD-muistikortti. Piirilevylle on muistikorttiliitin muistikorttia varten. Muistikortin käyttämiseen päädyttiin niiden edullisen hinnan vuoksi. Lisäksi se on tarvittaessa helppo irrottaa laitteesta ja kytkeä tietokoneeseen. Helposta ja nopeasta tiedostojen siirtämisestä oli suurta hyötyä erityisesti äänitiedostojen tallentamisen suodatusta suunniteltaessa. Tällä hetkellä käytetyn muistikortin kapasiteetti on 4 GB. Tällä kapasiteetilla muistikortille mahtuu äänitiedostoja noin 50 tunnin verran näytteenottotaajuuden ollessa 11,25 kHz. Audiotiedostojen käsittelyä varten muistikortilla on käytetty FAT-tiedostojärjestelmää.

4.5 Kommunikointiväylät

Kommunikointiväylien tuntemus on tärkeää onnistuneen ohjelmoityön kannalta, jotta pystytään tekemään paras mahdollinen valinta jokaisen komponentin hallitsemiseksi eri tilanteissa. Seuraavaksi kerrotaan perustiedot laitteen komponenttien hallintaan käytetyistä kommunikointiväylistä.

4.5.1 I2C-väylä

I2C-väylä (Inter-Integrated Circuit) on Philipsin 80-luvun alussa kehittämä sarjamuotoinen kommunikointiväylä, joka mahdollistaa helpon kommunikoinnin laitteiden välillä. Alunperin I2C on suuniteltu kommunikointiin samalla piirilevyllä sijaitsevien laitteiden välillä, mutta nykyisin sen avulla on mahdollista yhdistää laitteita myös kaapelin välityksellä. Nykyään I2C-väylän kehityksestä vastaa NXP Semiconductors. [19.]

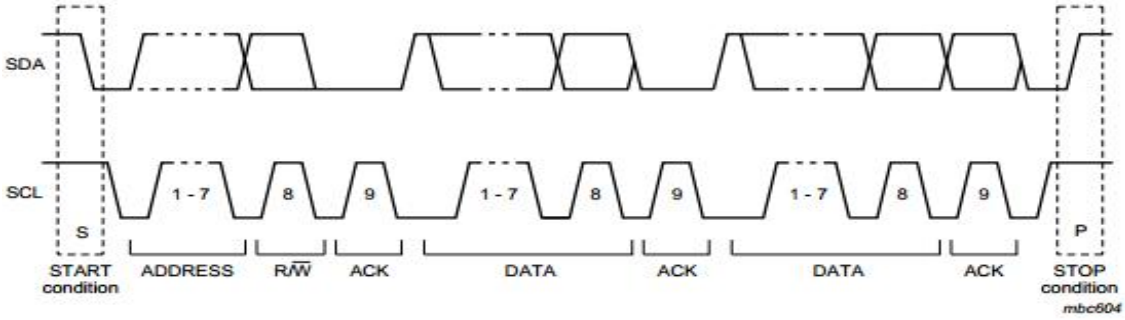
I2C-väylän yksinkertaisuus ja joustavuus tekevät siitä houkuttelevan ratkaisun moniin sovelluksiin. I2C tarvitsee vain kaksi linjaa toimiakseen, SDA-linjan (Serial Data Line) tiedonsiirtoa varten sekä SCL-linjan (Serial Clock Line) väylän kellotusta varten. I2C:llä ei ole myöskään tiukkoja siirtonopeuden vaatimuksia, kuten esimerkiksi RS232:lla, sillä isäntälaite huolehtii väylän kellotuksesta. Lisäksi jokaisella I2C-laitteella on yksilöllinen osoite, jotta samaan väylään kytketyt laitteet pystyvät erottamaan niille tarkoitetut viestit. [19.]

Alunperin I2C-väylä oli rajoitettu toimintanopeuteen 100 kbit/s. Vuosien saatossa tiedonsiirron nopeusvaatimukset ovat kuitenkin kasvaneet, ja nykyään I2C-väylälle on viisi eri nopeuskategoriaa. Standard-mode on rajoitettu 100 kbit:iin/s, Fast-mode 400 kbit:iin/s, Fast-mode Plus 1 Mbit:iin/s ja nopein kaksisuuntainen I2C-tyyppi, High-speed mode, 3,4 Mbit:iin/s. Lisäksi on olemassa yksisuuntainen Ultra Fast-mode, jonka suurin sallittu bittinopeus on 5 Mbit/s. Jokainen laite voi myös käyttää hitaampaa väylänopeutta. Ultra Fast-mode -laitteet eivät ole yhteensopivia aiempien versioiden kanssa, sillä muista poiketen ne ovat yksisuuntaisia. [20.]

Jokainen I2C-väylällä oleva laite voi toimia sekä isäntänä että orjana. Isäntä on laite, joka aloittaa tiedonsiirron väylällä ja tuottaa siirtoon vaadittavan kellosignaalin. Tällöin kaikki muut väylällä olevat laitteet ovat orjia. Isäntänä toimii usein mikrokontrolleri. Kaikki tiedonsiirtotapahtumat alkavat isännän tuottamalla START-ehdolla ja päättyvät STOP-ehtoon. Jokainen SDA-linjalla lähetetty tavu on 8 bitin mittainen. Lähetettävien tavujen lukumäärää ei ole rajoitettu. Jokaista tavua tulee seurata kuittausbitti merkkinä lähetyksen onnistuneesta vastaanottamisesta. [20.]

Kuvassa 8 esitetään I2C-standardin mukainen tiedonsiirtotapahtuma. Tiedonsiirto alkaa START-ehdolla. Tämän jälkeen isäntä lähettää osoitteen, jolla haluttu orjalaite valitaan. Osoite on seitsemän bitin mittainen, joita seuraa kahdeksas bitti, jolla määritetään

tiedonsiirron suunta (R/W). 0 ilmaisee datan lähettämistä (WRITE) ja 1 ilmaisee datapyyntöä (READ). Tiedonsiirto lopetetaan aina isännän tuottamaan STOP-ehtoon. [20.]



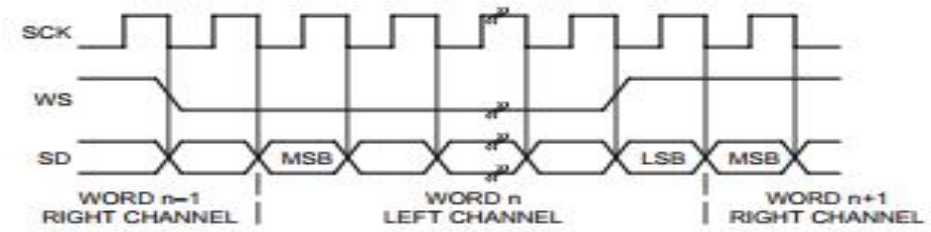
Kuva 8. I2C-väylän tiedonsiirto [20].

I2C-väylää käyttäessä tulee huomioida, että sen molemmat kytkentälinjat vaativat ylösvetovastukset toimiakseen. Ylösvetovastuksen maksimiresistantti riippuu väylän kokonaiskapasitanssista, joka on johtimen, liitäntöjen ja pinnien yhteenlaskeettu kapasitanssi. Ylösvetovastuksen minimiresistanssi riippuu käytetyn syöttöjännitteen suuruudesta. [20.]

4.5.2 I2S-väylä

I2S-väylä (Integrated Interchip Sound) on Philipsin vuonna 1986 kehittämä sarjamuotoinen väyläliitäntästandardi, jota käytetään digitaalisten audiolaitteiden yhteenliittämiseen. Väylä on suunniteltu PCM-muotoisen audiodatan siirtämiseen elektronisten laitteiden integroiduissa piireissä. Väylä on tarkoitettu ainoastaan audiodatan käsittelyyn, joten laitteiden väliset ohjaussignaalit tulee lähettää erikseen. [21.]

I2D-väylä koostuu minimissään kolmesta signaalilinjasta. Jatkuvasti toimivan kellosignaalin, SCK:n (Continuous Serial Clock), avulla vastaanottava laite ajoittaa lähetettävät audiodatan näytteistämisen bitti kerrallaan. Kanavavalitsinlinja, WS (Word Select), osoittaa parhaillaan käsiteltävän kanavan. WS:n ollessa 0 lähetettävä kanava on 1 (vasen) ja WS:n ollessa 1 lähetettävä kanava on 2 (oikea). Datalinjaa, SD:tä (Serial Data), käytetään varsinaisen audiodatan lähettämiseen. I2S-väylän tiedonsiirtoa havainnollistetaan kuvalla 9. [21.]



Kuva 9. I2S-väylän signaalilinjojen toiminta.

Sarjamuotoinen data lähetetään kahden komplementin muodossa eniten merkitsevä bitti ensin. Eniten merkitsevä bitti lähetetään ensin, koska lähettimen ja vastaanottimen sananpituus saattavat poiketa toisistaan. Lähettimen ei tarvitse tietää, kuinka monta bittiä vastaanotin pystyy käsittelemään. Eikä vastaanottimen tarvitse tietää, kuinka monta bittiä lähetetään. Jos vastaanottimelle on lähetetty enemmän bittejä kuin sen sananpituus, vähiten merkitsevät bitit jätetään huomiotta. Toisaalta, jos vastaanotin lähettää vähemmän bittejä kuin sen sananpituus on, puuttuvat bitit nollataan sisäisesti. [21.]

4.5.3 SDIO-väylä

SDIO (Secure Digital Input and Output) pohjautuu SD-spesifikaatioon, joka määrittelee kriteerit nykyään käytetyille SD-korteille. SD-kortteja käytetään matkapuhelimissa, kameroissa ja lukuisissa muissa kulutuselektroniikan laitteissa. Korttien yhteensopivuus sisältää mekaanisen, elektronisen, signaloinnin ja ohjelmiston yhteensopivuuden. Ensisijainen tavoite on, että laitteeseen tai sen ohjelmistoon ei aiheudu fyysistä vahinkoa eikä häiriötä, kun SD-kortti asetetaan ei-SDIO-tietoiseen isäntään. [22; 23.]

Normaalin isännän kortille suorittaman alustuksen ja tarkastuksen aikana kortti ilmaisee itsensä SDIO-laitteena. Seuraavaksi isäntäohjelmisto saa kortin tiedot linkitetyn listan muodossa ja päättää, ovatko kortin I/O-toiminnot hyväksyttäviä aktivoitaviksi. Tämä päätös perustuu muuttujiin, kuten tehovaatimuksiin tai vaadittavien ajureiden saatavuuteen. Jos kortti on hyväksyttävä, sen sallitaan käynnistyä kokonaan ja aloittaa sisäänrakennetut I/O-toiminnot. [23.]

SDIO-kortit saattavat tukea useita eri väylätyyppejä ja tiedonsiirtotiloja. SPI-tiedonsiirtotila ja yhden bitin tiedonsiirtotila ovat pakollisia kaikille SD-korteille. SPI-tiedonsiirtotila on suunniteltu kommunikoimaan SPI-kanavaa käyttäen. Kuten missä

tahansa SPI-laitteessa, SD-muistikortin SPI-liitäntä koostuu seuraavista neljästä signaalista:

- CS: Sirunvalinta
- CLK: Kello
- DI: Datan sisäänmeno
- DO: Datan ulostulo

Kello-, käyttöjännite- ja maasignaalit ovat yhtenevät kaikille tiedonsiirtotiloille. Yhden bitin tiedonsiirtotilassa data siirretään käyttämällä ainoastaan DAT[0]-pinniä. Pinni numero 8 on varattu keskeytyspinniksi. Neljän bitin tiedonsiirtotilassa käytetään kaikkia neljää datapinniä (DAT[3:0]). Tässä tilassa keskeytyspinni ei ole käytössä, sillä pinni on varattu tiedonsiirtoon. Neljän bitin tiedonsiirtotila tarjoaa korkeimman mahdollisen siirtonopeuden aina 100 Mb:iin/s asti. Taulukossa 1 on määritelty tarkemmin SDIO-väylän pinnien käyttö eri tiedonsiirtotiloissa.[23; 24.]

Taulukko 1. SDIO-väylän pinnien määritykset eri tiedonsiirtotiloissa.

Pin	SD 4-bit mode		SD 1-bit mode		SPI mode	
1	CD/DAT[3]	Data line 3	N/C	Not Used	CS	Card select
2	CMD	Command line	CMD	Command line	DI	Data input
3	VSS1	Ground	VSS1	Ground	VSS1	Ground
4	VDD	Supply voltage	VDD	Supply voltage	VDD	Supply voltage
5	CLK	Clock	CLK	Clock	SCLK	Clock
6	VSS2	Ground	VSS2	Ground	VSS2	Ground
7	DAT[0]	Data line 0	DAT[0]	Data line	DO	Data output
8	DAT[1]	Data line 1	IRQ	Interrupt	IRQ	Interrupt
9	DAT[2]	Data line 2	RW	Read Wait (optional)	NC	Not Used

Kommunikointi SDIO-väylässä perustuu komentojen ja datan välittämiseen eri linjoilla. Normaali tapahtuma SD/SDIO-väylällä on komento-vastaustapahtuma. Tämän tyyppisessä väylätapahtumassa tarvittavat tiedot lähetetään komento- tai vastausrakenteen sisällä. Komento on isännän lähettämä merkki, joka aloittaa operaation. Isäntä voi lähettää komennot joko yhdelle muistikortille tai kaikille liitetyille korteille. SD-kortti vastaa komentoon lähettämällä vastauspaketin käyttäen samaa linjaa. Datan siirtäminen SD/SDIO-muistikortin ja isännän välillä suoritetaan datalohkoissa. [25.]

4.5.4 SPI-väylä

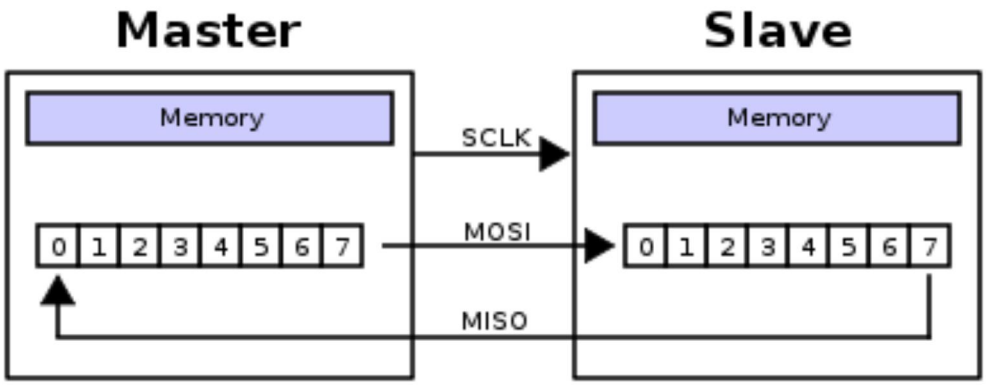
SPI-väylä (Serial Peripheral Interface) on Motorolan nimeämä synkroninen sarjamuotoinen datalinkki. Se soveltuu tilanteisiin, joissa vain vähän sisäänmeno- ja ulostulolinjoja on käytössä, mutta kommunikointi kahden tai useamman laitteen välillä tulee olla nopea ja helppo toteuttaa. Kahden laitteen välisessä kommunikoinnissa toinen laitteista on isäntälaite ja toinen orjalaite. Isännän kontrolloima kellosignaali huolehtii datan lähetyksen ja lukemisen ajoituksesta. [26; 27.]

SPI-väylä käyttää toimintaansa kahta kontrollilinjaa (SCLK ja CS) ja kahta datalinjaa (SDO ja SDI). Motorola käyttää datalinjoista nimityksiä MISO (Master-In-Slave-Out) ja MOSI (Master-Out-Slave-In). Samaan SPI-väylään voidaan liittää useampia orjalaitteita. Isäntälaite huolehtii halutun orjalaitteen valinnasta CS-linjan (Chip Select) avulla. Tämä pinni on useimmiten nolla-aktiivinen. Valitsemattomien orjalaitteiden SDO-linjat ovat korkeaimpedanssisessa tilassa, jolloin ne eivät aiheuta häiriötä dalalinjalle. Tiedonsiirron synkronoinnista huolehtiva kellolinja, SCLK, tulee kaikille orjalaitteille, olivat ne valittuja tai eivät. [27.]

Jokaisen kellojakson aikana tapahtuva SPI-väylän tiedonsiirto toteutetaan seuraavalla tavalla:

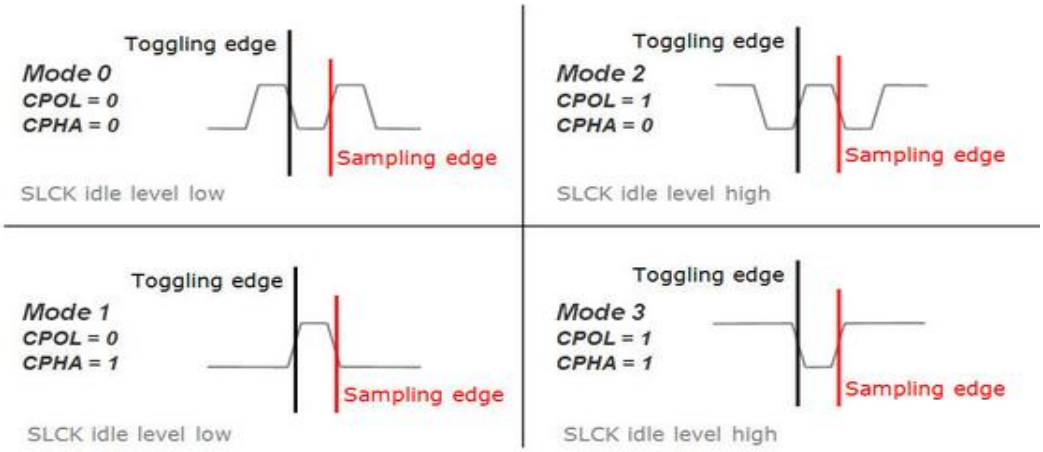
- Isäntä lähettää yhden bitin SDO-linjalla – orja lukee sen samalta linjalta.
- Orja lähettää yhden bitin SDI-linjalla – isäntä lukee sen samalta linjalta.

Normaalisti tiedonsiirtoon käytetään kahta renkaaksi kytkettyä siirtorekisteriä, joista toinen sijaitsee isäntälaitteessa ja toinen orjalaitteessa. Data siirretään usein ulos eniten merkitsevä bitti ensin, samalla kun samaan rekisteriin siirretään uusi vähiten merkitsevä bitti. Samaa toistetaan, kunnes koko rekisteriarvo on siirretty. Tämän jälkeen laitteet käsittelevät saamansa datan. Jos halutaan jatkaa datan siirtämistä, tallennetaan siirtorekisteriin uusi arvo ja prosessi toistetaan. Kuvassa 10 kuvataan siirtorekisterien käyttöä SPI-väylän tiedonsiirrossa. [27.]



Kuva 10. SPI-väylälle tyypillinen tiedonsiirtojärjestely rengaspuskuria käyttäen [28].

SPI-väylää käyttäessä tulee huomioida, että kellon vaiheen ja polaarisuuden tulee olla identtiset isäntä- ja orjalaitteessa. Joissain tapauksissa vaihe ja polaarisuus joudutaan vaihtamaan datasiirtojen välissä, jotta isäntälaite pystyy kommunikoimaan eri ominaisuuksilla olevien orjalaitteiden kanssa. Näistä kahdesta vaihtoehdosta käytetään useimmiten nimityksiä CPOL ja CPHA. Polaarisuuden (CPOL) ollessa 0 kellosignaalin perusarvo on 0 ja polaarisuuden ollessa 1 kellosignaalin perusarvo on 1. Vaiheen (CPHA) ollessa 0 ensimmäistä SCLK:n reunaa käytetään siirtorekisterin ensimmäisen bitin tallentamiseen siirtorekisteriin, ja puoli kellonjaksoa myöhemmin olevaa reunaa käytetään seuraavan siirrettävän arvon asettamiseen. Tässä tulee huomioida, että ensimmäisen noudettavan bitin arvo tulee asettaa ennen ensimmäistä SCLK:n reunaa. CPHA:n ollessa 1 ensimmäinen vastaanotettava bitti noudetaan toisella SCLK:n reunalla. SCLK:n vaihde- ja polaarisuusvaihtoehtoja selvennetään kuvassa 11. [29.]



Kuva 11. SCLK:n vaihe- ja polaarisuusvaihtoehdot SPI-väylässä [30].

5 Kehitysympäristö

Ennen kuin varsinainen ohjelmointi voidaan aloittaa, tulee ohjelmointiin käytettävä kehitysympäristö valita ja pystyttää. Tässä työssä käytetty kehitysympäristö rakennettiin ilmaisista ja toimivista avoimen lähdekoodin ohjelmista. Käyttöjärjestelmänä oli Ubuntu 11.10, niin ikään avoimen lähdekoodin Linux-käyttöjärjestelmä.

5.1 CodeSourceryn asennus

Kääntäjäksi kehitysympäristöön valittiin Mentor Graphicsin Sourcery G++ Lite for ARM EABI, joka on jo itsessään kokonainen kehitysympäristö sulautettuun C/C++-ohjelmointiin muun muassa ARM- Power- ja ColdFire- arkkitehtuureille. Se sisältää muun muassa C- ja C++-kääntäjät, linkittimen sekä vianetsintätyökalun. Mentor Graphics tarjoaa Getting Started -oppaan, jossa käydään yksityiskohtaisesti läpi kääntäjän asennuksen vaiheet.

Ubuntun käyttämä komentotulkki ei ole yhteensopiva CodeSourceryn asennustiedoston kanssa. Helpoin keino saada ohjelma toimimaan on käyttää sen valmiiksi käännettyjä binääritiedostoja. Ensimmäinen vaihe on ladata CodeSourceryn IA32 GNU/Linux TAR -versio Mentor Graphicsin internetsivujen (https://sourcery. mentor.com/GNUToolchain/release1802) Advanced Packages -osiosta. Työssä käytetty versio on Sourcery G++ Lite 2011.03-42. Ladattu tiedostopaketti puretaan kotihakemistoon, josta se on helposti löydettävissä.

Ubuntu ei automaattisesti tiedä, missä juuri ladatun ohjelmiston suoritettavat komponentit sijaitsevat, vaan niille on ensin määritettävä PATH-asetukset. Tämän saa tehtyä lisäämällä kotihakemistossa sijaitsevaan .bashrc-tiedostoon rivin `export PATH=$HOME/arm-2011.3/bin:$PATH`. Seuraavalla komentorivin käynnistyskerralla ohjelmiston komponenttien tulisi olla löydettävissä. Tämän voi testata kirjoittamalla arm-none ja painelemalla tabulaattoria, jonka seurauksena lista arm-none-alkuisista ohjelmista pitäisi ilmestyä komentoriville.

5.2 Kääntämiseen tarvittavat tiedostot

STMicroelectronics tarjoaa sivuillaan (http://www.st.com/web/en/catalog/tools/PF257890) STM32F10x standard peripheral library -kirjaston ilmaiseksi. Paketti sisältää laiteajurit kaikille standardioheislaitteille, ja nämä ajurit sisältävät täyden toiminnallisuuden. C-kieliset lähdekoodit ovat dokumentoituja ja testattuja. Paketti sisältää kaikki määritykset ja rakenteet STM32-mikrokontrollerin ohjelmointia varten. Pakettiin kuuluu myös esimerkkiohjelmia. Kirjastosta sisältää esimerkiksi ohjaimen SD-kortin hallintaa varten SDIO-väylää käyttäen. SD-kortin käyttötila on valittavissa. Projektissa käytetään neljän bitin käyttötilaa maksimaalisen kirjoitus- ja lukunopeuden saavuttamiseksi.

Kaikki kirjaston tiedostot eivät ole tarpeellisia, vaan paketista kopioidaan vain oheislaitteiden lähdekoodit kansiosta Libraries/STM32F10x_StdPeriph_Driver/src ja header-tiedostot kansiosta Libraries/STM32F10x_StdPeriph_Driver/inc. Näitä varten projektiin luodaan kansio libstm32. Sen sisälle luodaan alikansiot src ja inc, joihin tiedostot sijoitetaan. Lisäksi STM:n tarjoaman kirjaston kansioista Libraries/CMSIS/CM3/CoreSupport tulee kopioida tiedostot core_cm3.c ja core_cm3.h sekä kansiosta Libraries/CMSIS/CM3/DeviceSupport/ST/STM32F10x tiedostot stm32f10x.h ja system_stm32f10x.h aiemmin luotuihin kansioihin.

STMicroelectonicsin tarjoaman oheislaitekirjastojen mukana tulee Libraries/CMSIS/CM3/DeviceSupport/ST/STM32F10x/startuo/gcc_ride7 kansiossa sijaitseva tiedosto nimeltään startup_stm32f10x_hd.s, joka tulee liittää projektiin. Sen tehtävänä on mikrokontrollerin alustus. Se kopioi staattiset muuttuja-arvot RAM-muistiin sekä alustaa muistin, keskeytysvektorit ja itse laitteen. Reset-käsittelijä alustetaan ja main()-funktiota, ohjelman tulokohtaa kutsutaan.

Projektin rakentaminen tapahtuu itsenäisesti ilman käyttöjärjestelmää, joten laitteisto ja muisti tulee alustaa manuaalisesti. Ohjelman tulokohdan tulee olla tietty osoite, jotta ohjelma voidaan suorittaa onnistuneesti. Tätä tarkoitusta varten projektissa tulee olla linkitinskripti, joka kertoo kääntäjälle, miten ohjelma täsmälleen rakennetaan. Tässä projektissa skripti on nimeltään linker.ld. Skriptissä määritetään eri muistien sijainnit mikroprosessorissa ja muistien koot. Nämä määritykset tulee muuttaa, jos mikroprosessori vaihdetaan muistiltaan erikokoiseen.

Projektin rakentamisprosessia, kääntämistä ja lähdekoodien linkittämistä voi helpottaa ja automatisoida käyttämällä Makefile-tiedostoja. Makefilen käyttämisessä on monia etuja:

- Makefilet sisältyvät projektiin.
- Niihin voi lisätä selventäviä kommentteja.
- Käännöksen tulokset ovat ennustettavia ja toistettavia.

Projektin päähakemistoon luotu Makefile voidaan sisällyttää kaikkiin alikansioiden Makefileihin, jolloin samoja määrityksiä ei tarvitse tehdä useampaan kertaan. Päähakemiston Makefileen määritetään käytettävä kääntäjä sekä sen käyttämät optimointiasetukset. Mikrokontrollerin ohjelmoimiseen käytettävä linkitintiedosto ja alustustiedosto liitetään osaksi laitteeseen siirrettävää binääritiedostoa. Kaikki projektiin käytettävät lähdetiedostot tulee kääntää ja sisällyttää pääohjelman käännökseen. Projektin saa käännettyä komennolla *make* ja siivottua komennolla *make clean*. Esimerkkikoodi 1:ssä on ote Puhuvassa avaimenperässä käytetystä Makefile-tiedostosta.

```
# Makefile for building  STM32 projects
# Using the CodeSourcery 2009q3 Lite ARM EABI tools
TOP=$(shell readlink -f "$(dir $(lastword $(MAKEFILE_LIST)))")
LIBSTM32=$(TOP)/libstm32/libstm32.a
LIBDIR=$(TOP)/libstm32
CROSS_COMPILE = arm-none-eabi-
CC = $(CROSS_COMPILE)gcc
AS = $(CROSS_COMPILE)as
OBJCOPY = $(CROSS_COMPILE)objcopy

INCLUDES=-I$(LIBDIR)/inc
INCLUDES+=-I$(LIBDIR)
INCLUDES+=-I$(TOP)

LINKER_SCRIPT=stm32.ld
STARTUP = startup_stm32f10x_hd.s

# Build for the Cortex-M3 with basic optimisations and # debug-
ging
CFLAGS = -mcpu=cortex-m3 -mthumb -g $(INCLUDES) -Wall

# List of all binaries to build
all: clean main.bin

# Create a raw binary file from the ELF version
main.bin: talkingkey.elf
           # $(CROSS_COMPILE)strip main.elf
           $(OBJCOPY) -O binary $< $@

# Create the ELF version by mixing together the
# startup file, application and linker file
talkingkey.elf: stmstuff.o startup.o usart.o talkingkey.o
STM32_PN532.o sdio_disk.o ff.o powerdowntimer.o battery.o
sounds.o interrupts.o recording.o playback.o
           $(CC) $(CFLAGS) -o $@ -nostartfiles -Wl,-
T$(LINKER_SCRIPT),-
Map=output.map $^ $(LIBSTM32) -lm

talkingkey.o:
           $(CC) $(CFLAGS) -o $@ talkingkey.c -c -O3

# compile peripherial stuffs
startup.o: $(STARTUP)
           $(CC) -o $@ $(CFLAGS) -c $<

playback.o: playback.c
           $(CC) -o $@ $(CFLAGS) -c $<

recording.o: recording.c
           $(CC) -o $@ $(CFLAGS) -c $<
           . . .
           . . .
# Remove the temporary files
clean:
           rm -f *.o *.elf *.bin *.map

cleanlibs:
           $(MAKE) -C libstm32 clean
```

Esimerkkikoodi 1. Ote pääohjelman Makefile-tiedostosta.

5.3 OpenOCD:n asennus ja käyttö

Open On-Chip Debugger on osa ohjelmaa, jota tarvitaan mikrokontrollerin flash-ohjelmointiin ja vianetsintään JTAG-laitteella. Se on myös ohjelmarajapinta GDB:hen (The GNU Project Debugger). OpenOCD tukemat JTAG-laitteet ovat usein melko edullisia. Tässä työssä oli käytössä Flyswatter Rev-B. OpenOCD:n asentaminen tapahtuu seuraavalla tavalla:

- Päivitetään Linuxin pakettilistaus uusimpien pakettitietojen saamiseksi komennolla `sudo apt-get update.`
- Asennetaan OpenOCD ja sen riippuvuudet komennolla `sudo apt-get install libftdi-dev libtool openocd.`
- Asennetun OpenOCD:n sijainti selviää komennolla `which openocd.`

OpenOCD käyttää openocd.cfg-nimistä konfigurointitiedostoa käynnistyksen yhteydessä, joka sisältää seuraavat tiedot:

- käytettävä portti
- JTAG:n määritykset
- alustan (mikrokontrolleri) määritykset
- kohteen (mikrokontrolleri) määritykset.

Tiedot saattavat vaihdella projektien välillä, joten tiedoston tulee sijaita jokaisen projektin kansiossa. Puhuvan avaimenperän openocd.cfg-tiedosto on yksinkertainen, sillä OpenOCD:sta löytyy valmiina määritykset käytetylle JTAG-laitteelle ja mikrokontrollerille. Näiden tiedostojen sijainti tulee kertoa OpenOCD:lle kirjoittamalla seuraavat komennot openocd.cfg-tiedostoon:

```
source /usr/share/openocd/scripts/interface/flyswatter.cfg
source /usr/share/openocd/scripts/board/stm3210e_eval.cfg
```

OpenOCD muodostaa yhteyden ohjelmoitavaan mikrokontrolleriin komennolla `sudo openocd –f openocd.cfg`. JTAG-laitteen tulee olla liitettynä tietokoneeseen ja mikrokontrolleriin. Seuraavaksi OpenOCD:n avaamaan porttiin 4444 otetaan telnet-yhteys, jota kautta mikrokontrolleri voidaan ohjelmoida. Tämä tapahtuu komennolla `telnet`

`localhost 4444`. Viimeinen vaihe ohjelman siirtämisessä mikrokontrollerin flash-muistiin tapahtuu seuraavilla komennoilla:

```
reset halt
flash probe 0
stm32f1x mass_erase 0
flash write_bank 0 main.bin 0
reset run
```

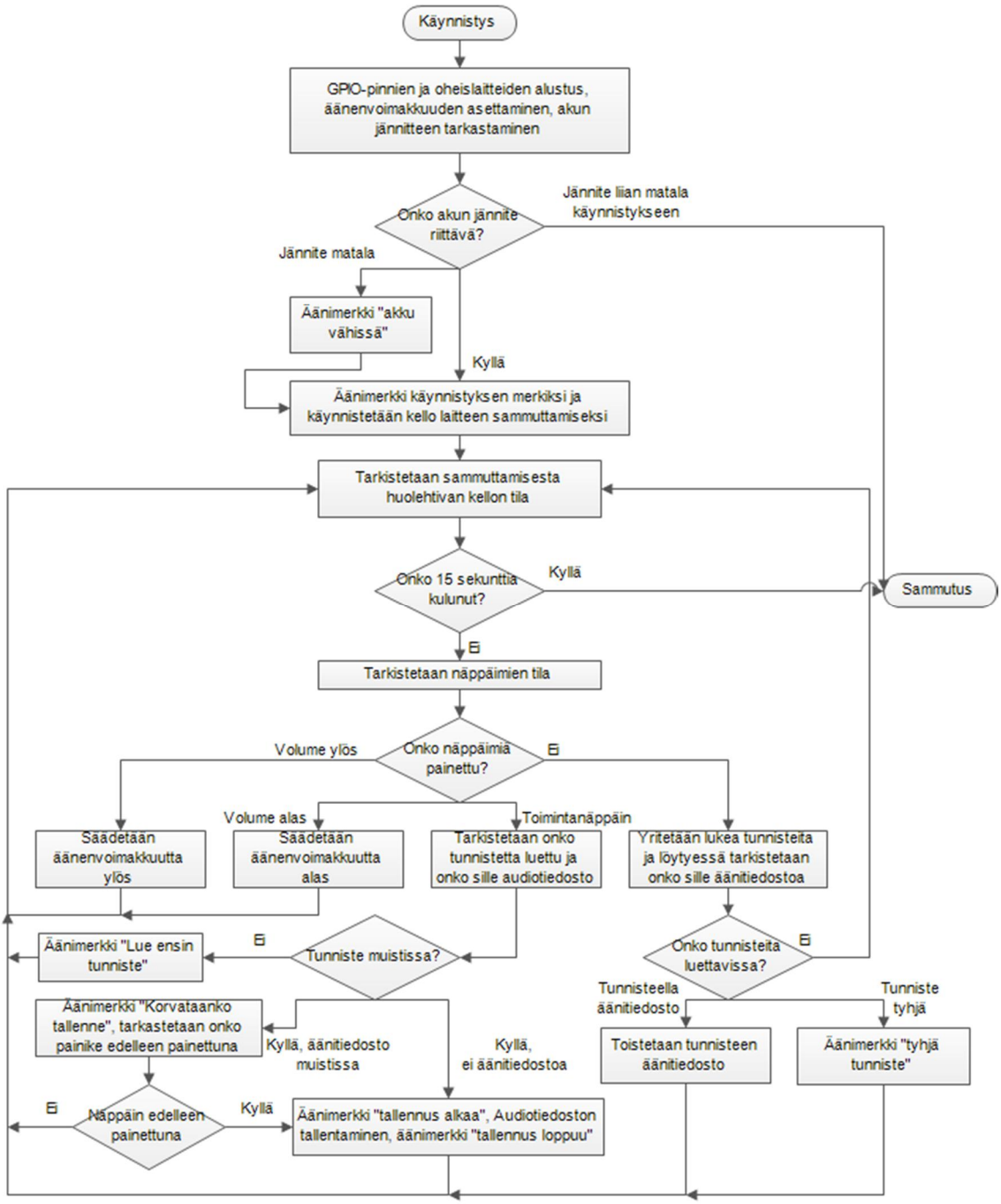
Näillä komennoilla mikrokontrolleri asetetaan alkutilaan sekä pysäytetään, flash-muisti pyyhitään kokonaan ja main.bin-tiedosto siirretään flash-muistiin. Tämän jälkeen mikrokontrolleri asetetaan alkutilaan ja käynnistetään. Jos laite halutaan ohjelmoida useamman kerran, voidaan kirjoittamista vähentää luomalla tiedosto program.ocd ja lisäämällä siihen käytetyt komennot. Mikrokontrolleri saadaan tämän jälkeen ohjelmoitua komennolla `script program.ocd`.

6 Laitteen ohjelmisto

6.1 Pääohjelma

Ennen Puhuvan avaimenperän ohjelmoinnin aloittamista ohjelmasta laadittiin kuvan 12 mukainen vuokaavio. Kaaviossa kuvataan laitteen toiminta vaiheittain. RFID-tunnisteiden lukemista sekä audiotiedostojen toistamista ja tallentamista ei ole kuvattu kaaviossa yksityiskohtaisesti. Näiden osioiden toimintaan palataan tarkemmin luvun myöhemmissä osissa.

Ensimmäinen tehtävä laitteen käynnistyessä on mikrokontrollerin kellotaajuuden asettaminen. Kellotaajuus asetetaan 36 MHz:iin, ja kellon lähteenä käytetään sisäistä kellokidettä. Mikrokontrollerin korkein mahdollinen kellotaajuus on 72 MHz, mutta matalampaa kellotaajuutta käyttämällä laitteen virrankulutusta saadaan vähennettyä.



Kuva 12. Puhuvan avaimenperän pääohjelman vuokaavio.

Seuraavaksi asetetaan laitteen näppäimien määritykset. Laitteessa on kaksi painiketta äänenvoimakkuuden säätämistä varten ja lisäksi toiminatapainike. Painikkeet tulee määrittää sisääntuloiksi, ja niille asetetaan mikrokontrollerin sisäiset ylösvetovastukset. Ylösvetovastuksen tehtävänä on varmistaa, että pinnin tila on 1 aina kun painiketta ei paineta. Painikkeiden keskeytykset kytketään pois toiminnasta. Näppäinten keskeytyksiä käytetään vain laitteen toistaessa audiotiedostoa.

Tämän jälkeen määritetään RFID-lukijan pinnien asetukset, jonka jälkeen PN532:een yritetään ottaa yhteyttä. Yhteydenoton epäonnistuessa Puhuva avaimenperä asetetaan virransäästötilaan. Tämä toiminnallisuus on toteutettu lähinnä kehitystyön aikaista vianetsintää varten. Laitetta on ollut turha käynnistää, jollei RFID-lukija ole toiminnassa. Valmiissa versiossakaan siitä ei ole haittaa, sillä jollei RFID-tunnisteita voida lukea, on laite käytännössä käyttökelvoton tarkoitukseensa.

Seuraavana vuorossa on SD-kortin asetusten määrittäminen. SD-kortin hallintaan käytettävä SDIO-väylä alustetaan ja alustuksen onnistuessa SD-kortti liitetään mikrokontrollerin käyttöön. Tämän jälkeen tarkastetaan, onko tallennettaville audiotiedostoille luotu jo kansio, ja tarvittaessa luodaan se. Tämän jälkeen tutkitaan, onko laitteen asetuksia varten luotu kansio ja onko tämän kansio sisällä tiedosto laitteen äänenvoimakkuuden tallentamista varten. Tiedoston löytyessä ladataan tiedot laitteen talletetusta äänenvoimakkudesta ja siirretään tieto muuttujaan. Jos asetusten kansiota tai vaadittavaa tiedostoa ei löydy, niin ne luodaan.

Tämän jälkeen tarkistetaan laitteen akun varauksen taso. Tätä varten laitteen piirilevyllä on kaksi samansuuruista vastusta, joiden välisen pisteen varaus luetaan ADC-muuntimen avulla. Nämä vastukset sijaitsevat akun ja regulaattorin välissä, sillä muussa tapauksessa mitattava arvo pysyisi vakiona. Vertailupisteen jännitteen ja ADC:n referenssijännitteen suhde ei enää muutu akun jännitteen laskiessa 3,3 V:iin tai sen alle. ADC:n avulla luettua jännitettä verrataan oletettuun referenssijännitteeseen (3,3 V), ja tämän vertailun perusteella lasketaan akun jännite. Akun jännitteen laskiessa alle 3,4 V:n laitetta ei käynnistetä vaan se laitetaan virransäästötilaan. Jos akun jännite on alle 3,6 V, soitetaan äänimerkki matalan jännitteen merkiksi. Muussa tapauksessa laite käynnistetään normaalisti. Käynnistyksen yhteydessä soitetaan aina äänimerkki, jotta käyttäjä tietää laitteen käynnistyneen. Käynnistysäänen jälkeen käynnistetään kello, jonka tehtävänä on huolehtia laitteen automaattisesta sammuttamisesta. Kello sammutetaan ja käynnistetään uudelleen aina kun laitteen näppäimiä painetaan tai kun RFID-tunniste luetaan onnistuneesti.

Seuraavaksi siirrytään ohjelman pääsilmukkaan, jota toistetaan kunnes automaattisesta sammuttamisesta huolehtiva kello umpeutuu. Kellon umpeutuminen ilmaistaan muuttujan arvolla, joka tarkastetaan aina silmukan alussa.

Ensimmäinen asia pääsilmukassa on tarkistaa, onko laitteen näppäimiä painettu. Jos jompaa kumpaa äänenvoimakkuuden säätöpainikkeitta on painettu, säädetään laitteen äänenvoimakkuutta ylös tai alas. Jotta audiopiirin äänenvoimakkuutta voidaan säätää, tulee audiopiiri kytkeä päälle. Tämän jälkeen audiopiirin rekisteriin kirjoitetaan uusi äänenvoimakkuuden arvo, joka tallennetaan myös muuttujaan. Laite toistaa merkkiäänen uudella äänenvoimakkuudella. Äänenvoimakkuuden arvo tallennetaan SD-kortille, josta se on ladattavissa laitteen käynnistyksen yhteydessä.

Toimintapainikkeen ollessa painettuna laite tarkistaa, onko sillä luettu tunnisteita käynnistyksen jälkeen. Laite soittaa ”osoita tunnistetta” -äänimerkin, jos tunnistetta ei ole aiemmin luettu. Jos tunniste on luettu, mutta SD-kortin muistista ei löydy sille äänitiedostoa, aloitetaan uuden äänitiedoston nauhoitus. Ennen äänityksen aloittamista soitetaan ”äänitys alkaa” -merkkiääni. Äänittämistä jatketaan niin pitään kuin toimintapainike on painettuna tai kunnes äänitteen maksimipituus täyttyy. Äänityksen lopetuksen merkiksi soitetaan merkkiääni. Tunnisteen löytyessä jo muistista laite soittaa ”korvataanko tallenne” -merkkiäänen. Jos käyttäjä jatkaa toimintanäppäimen painamista vielä noin sekunnin verran, aloitetaan äänitys edellä kerrotulla tavalla. Tässä tapauksessa alkuperäinen äänitiedosto korvataan uudella. Käyttäjän vapauttaessa toimintapainikkeen ennen äänityksen alkamista säilytetään alkuperäinen äänitiedosto.

Painikkeiden toiminnan käsittelyn jälkeen yritetään lukea laitteen antennin lukuetäisyydellä oleva RFID-tunniste. Jos tunnistetta ei löydy, aloitetaan pääsilmukan alusta. Tunnisteiden löytyessä tarkistetaan löydetyn tunnisteen ID:n tavujen lukumäärä. Jos tavuja on neljä, on löydetty tunniste Mifare Classic. Jos tavuja taas on seitsemän, on kyseessä Mifare Ultralight. Tämän jälkeen tarkistetaan löytyykö SD-kortin muistista kyseisen tunnisteen äänitiedosto. Äänitiedoston löytyessä soitetaan kyseinen äänitiedosto. Jos äänitiedostoa ei löydy, soitetaan ”tyhjä tunniste” -äänimerkki. Löydetyn tunnisteen ollessa Mifare Ultralight, tarkistetaan löytyykö SD-kortin muistista tunnisteen ID:n kolmea ensimmäistä tavua vastaava alikansio, ja jos kansiota ei löydy, luodaan se. Alikansioita joudutaan käyttämään, sillä käytetyssä tiedostojärjestelmässä on mahdollista luoda kansio tai tiedosto, jonka nimi on maksimissaan kahdeksan kirjainta. Mifare Ultralightin ID on seitsemän tavua, joten sen tallentamiseen tiedostoon vaadittaisiin 14 merkin mittainen tiedostonimi.

RFID-tunnisteiden etsimisen ja löydetyn tunnisteen käsittelyn jälkeen pääsilmukka aloitetaan alusta. Jos sammuttamisesta huolehtiva kello on umpeutunut, laite asetetaan

virransäästötilaan. Ennen virransäästötilaan siirtymistä sammutetaan RFID-lukija ja audiopiiri. Laitteen sammuttamisen merkiksi soitetaan merkkiääni. Toimintapainikkeen keskeytykset kytketään, jotta laite herää näppäintä painettaessa. Virransäästötilasta palattaessa palataan samaan kohtaan ohjelmaa, mistä virransäästötilaan siirryttiin. Tämän jälkeen laite palautetaan alkutilaan, jolloin toiminta alkaa pääohjelman alusta.

Osa Puhuvan avaimenperän toiminnoista on toteutettu keskeytysten avulla. Laitteen sammuttamisesta huolehtivan ajastimen umpeutuminen aiheuttaa keskeytyksen, jonka käsittelijässä muutetaan ohjelman pääsilmukan alussa tarkistettavan muuttujan arvo, jolloin laite sammutetaan. Äänitteen toiston aikaiset näppäimien toiminnat on toteutettu keskeytysten avulla. Myös äänitiedoston äänityksen ja toiston datansiirto mikrokontrollerin ja äänipiirin välillä on toteutettu I2S-väylän keskeytysten avulla. Mahdollista ongelmatilanteista palautumista varten käytössä on watchdog-ajastin, jonka umpeutuminen aiheuttaa keskeytyksen, jolloin mikrokontrolleri palautetaan alkutilaan.

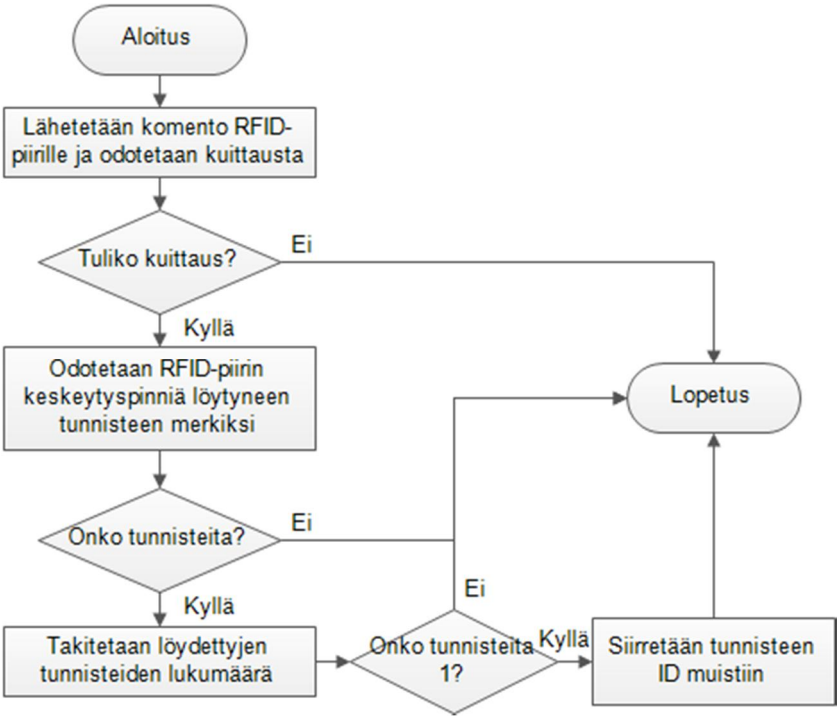
Keskeytysten käsittelyä varten STM32:ssa on NVIC-yksikkö (The Nested Vector Interrupt Controller). NVIC on suunniteltu tukemaan sisäkkäisiä keskeytyksiä, ja keskeytyksillä on 16 eri prioriteettitasoa. Oheislaitteen aiheuttaessa keskeytyksen NVIC käynnistää keskeytyksen käsittelyn Cortexin CPU:ssa. Parhaillaan käsiteltävän komennon osoite siirretään komentopinoon CPU:n mennessä keskeytystilaan. Tämän jälkeen keskeytyksenkäsittelijän alkuosoite noudetaan komentoväylälle. Keskeytyksen loputtua palataan komentopinoon laitetun komennon osoitteeseen automaattisesti. [7.]

NVIC:n käyttämiseksi tulee tehdä kolme asiaa. Ensimmäiseksi konfiguroidaan vektoritaulukko käytettäviä keskeytyksiä varten. Seuraavaksi määritetään NVIC-rekisterit, joissa sallitaan NVIC-keskeytykset ja asetetaan niille prioriteettitaso. Viimeiseksi konfiguroidaan oheislaite ja sallitaan sen keskeytykset. [7.]

STM32:n keskeytysten vektoritaulukko alkaa osoiteavaruuden alusta. Jokainen keskeytysvektorin määrittely on neljän tavun mittainen ja sisältää keskeytyksenkäsittelijän alkuosoitteen. Ensimmäiset 15 määrittelyä on STM32:n sisäisiä poikkeuksia varten. Oheislaitteiden keskeytysten määritttelyt alkavat 16:sta, ja ne on linkitetty oheislaitteisiin valmistajan määrittämällä tavalla. [7.]

6.2 RFID-tunnisteiden lukeminen

RFID-tunnisteiden lukeminen on kuvattu kuvan 13 vuokaaviossa. Tunnisteiden lukeminen alkaa lähettämällä komento lukemisesta RFID-lukijalle. Vastaanotettuaan komennon PN532 kuittaa sen nostamalla keskeytyspinnin ylös vastaanotettavan datan merkiksi. PN532:n keskeytyspinni on kytketty mikrokontrollerin GPIO-pinniin. Komennon lähettämisen jälkeen mikrokontrolleri jää odottamaan keskeytyspinnin nousevaa reunaa, jolloin siirrytään eteenpäin. Jos merkkiä keskeytyspinniltä ei tule tietyn aikarajan puitteissa, palataan pääohjelmaan ja palautetaan virhekoodi. Keskeytyspinnin nousevan reunan jälkeen data haetaan RFID-piiriltä ja tarkistetaan, että se vastaa datalehdessä määriteltyä ACK-kuittausta. Tunnisteiden lukeminen lopetetaan, jos vastaus on jotain muuta.



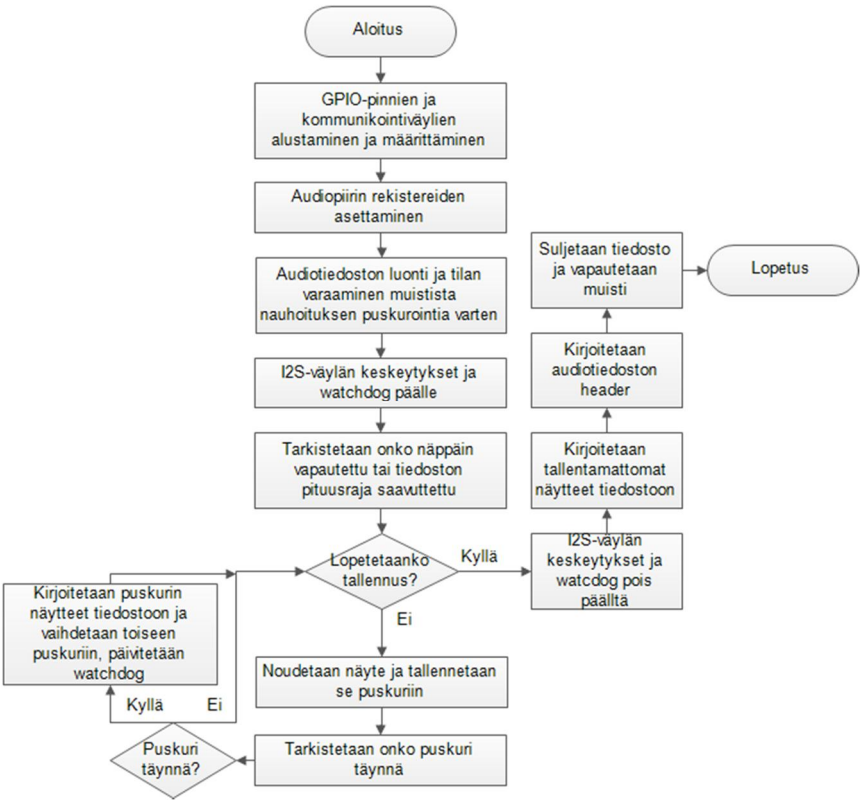
Kuva 13. Vuokaavio RFID-tunnisteiden lukemisesta.

Kuittauksen jälkeen odotetaan jälleen PN532:n keskeytyspinnin nousevaa reunaa – tällä kertaa löydetyn RFID-tunnisteen merkiksi. Jos sitä ei tule aikarajan puitteissa, keskeytetään tunnisteiden lukeminen. Jos taas RFID-piiri ilmoittaa löytyneestä tunnisteesta, haetaan tämän tunnisteen tiedot piiriltä ja tallennetaan ne muuttujiin. Tämän jälkeen tarkastetaan löytyneiden tunnisteiden lukumäärä. Lukumäärän ollessa jotain muuta kuin yksi lopetetaan tunnisteiden lukeminen. Tämä tehdään siitä syystä, että

halutaan varmistaa, että luettu tunniste on haluttu esimerkiksi tilanteessa, jossa useita tunnisteita on lähellä toisiaan. Seuraavaksi siirretään luetun tunnisteen pituus uid-length-muuttujan ilmoittamaan osoitteeseen ja pituuden määrittämä määrä tunnisteen ID:n tavuja uid-muuttujan ilmoittamasta alkuosoitteesta eteenpäin tavu kerrallaan. Nämä tiedot palautuvat pääohjelmalla.

6.3 Audiotiedoston nauhoittaminen

Audiotiedoston nauhoittaminen on esitelty kuvassa 14, jossa on vuokaavio nauhoittamisen etenemisestä. Ensimmäinen tehtävä on audiopiirin ohjaamiseen käytettävien pinnien ja väylien asetusten määrittäminen. Audiopiirin toiminnan ohjaamiseen käytettävät rekisteriarvot nollataan asettamalla audiopiirin reset-pinnin arvoksi 0 ja muutaman millisekunnin päästä nostamalla se arvoon 1. Muutaman millisekunnin viive on välttämätön, jotta audiopiirin rekistereiden arvot ehtivät nollaantua. Näytteet vastaanotetaan audiopiiriltä I2S-väylän avulla, joten myös se konfiguroidaan tässä vaiheessa. Tässä vaiheessa I2S-väylän kellotaajuus, ääniformaatti sekä näytteiden bittimäärä asetetaan halutuiksi. Audiotiedostojen käsittelyyn käytetään kahta I2S-väylää, joiden kellosignaalit on kytketty toisiinsa kiinni. Käyttämättömän I2S-väylän pinnit tulee määrittää kelluvaan tilaan, jolloin passiivinen väylä ei aiheuta häiriöitä aktiiviselle. Laitteen ohjelmointiin käytettävän JTAG-väylän pinnit ovat osittain samat kuin toisen I2S-väylän pinnit. Tästä syystä toisen I2S-väylän pinnit tulee uudelleenohjata mikrokontrollerissa. Samalla JTAG-väylä tulee kytkeä pois toiminnasta.



Kuva 14. Vuokaavio audiotioedoston nauhoittamisesta.

Seuraavaksi määritetään audiopiirin rekisterien arvot, joilla ohjataan audiopiirin toimintaa. Tässä vaiheessa määritetään muun muassa mikrofonin vahvistuksen taso ja herkkyys, käytetty audioprotokolla sekä äänen suodatus. Äänen suodattamista varten audiopiirissä on sekä ylipäästösuodin että alipäästösuodin. Näiden käyttäminen on välttämätöntä käytettyjen pienikokoisten kaiuttimien kanssa, jotta äänenlaadusta saadaan vaatimusten mukainen.

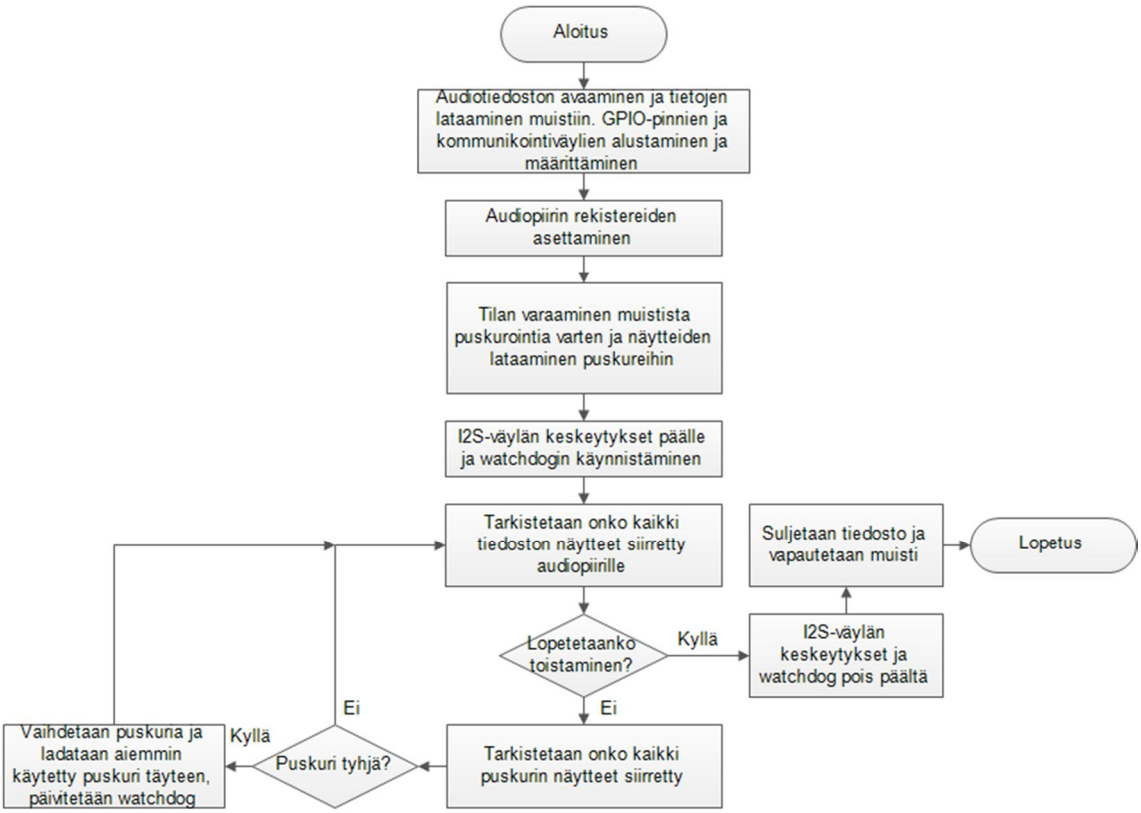
Uutta äänitiedostoa varten luodaan aina uusi tiedosto, jolloin vanha tiedosto korvataan automaattisesti uudella. Tiedoston luomisen jälkeen tiedoston osoitinta siirretään 44 tavua eteenpäin. Ensimmäiset 44 tavua on varattu audiotiedoston header-osiota varten, joka kirjoitetaan vasta, kun tiedoston lopullinen koko tiedetään. Nauhoittamista varten mikrokontrollerin muistista varataan kaksi 10240 tavun suuruista puskuria. Puskuroinnissa käytetään kahta puskuria, joista toiseen siirretään dataa audiopiiriltä samanaikaisesti, kun toista tallennetaan muistikortille. Kaksoispuskurointi on välttämätöntä, jotta SD-kortille ehditään tallentaa edellinen puskuri ennen toisen puskurin täyttymistä.

Puskureiden tilanvaraamisen jälkeen kytketään käytettävän I2S-väylän keskeytykset, jolloin äänittäminen alkaa. Myös watchdog-ajastin käynnistetään mahdollisista virhetilanteista toipumista varten. Näytteiden siirtäminen audiopiiriltä tapahtuu I2S-väylän keskeytysten avulla. Keskeytyksillä huolehditaan myös audiopiirin näytteistämisen ajoituksesta. Samanaikaisesti tapahtuvassa silmukassa tarkkaillaan toimintapainikkeen tilaa. Äänittäminen lopetetaan, jos painike vapautetaan. Tällöin I2S-väylän keskeytykset kytketään pois. Äänitys lopetetaan myös audiotiedoston maksimipituuden täyttyessä. Edellinen silmukka huolehtii myös puskureiden tallentamisesta SD-kortille ja watchdog-ajastimen tilan päivittämisestä.

Silmukasta eteenpäin siirtymisen jälkeen I2S-väylän keskeytykset ja watchdog-ajastin kytketään pois, minkä jälkeen tarkistetaan, onko puskureissa tallentamatonta dataa, ja tallennetaan data SD-kortille tarvittaessa. Tämän jälkeen tiedoston osoitin siirretään tiedoston alkuun, johon tallennetaan audiotiedoston header. Headeriin tallennetaan toistamisen kannalta olennaiset tiedot. Näitä tietoja ovat muun muassa audiotiedoston pituus, näytteenottotaajuus, näytteen bittikoko sekä tiedostoformaatti. Headerin tallentamisen jälkeen suljetaan audiotiedosto ja vapautetaan puskurointiin käytetty muisti.

6.4 Audiotiedoston toistaminen

Audiotiedoston toistamista havainnollistetaan kuvassa 15 olevassa vuokaaviossa. Ensimmäinen asia audiotiedostoa toistettaessa on tiedoston avaaminen ja tiedoston header-osion datan lataaminen SRAM-muistiin ja sen käsitteleminen. Header-osio sisältää tiedon muun muassa audiotiedoston pituudesta, näytteenottotaajuudesta, näytteiden bittien lukumäärästä ja kanavien lukumäärästä. Tämän jälkeen määritetään audiopiirin ohjaamiseen käytettävien pinnien ja väylien asetukset. Audiopiirin rekisterit nollataan asettamalla audiopiirin reset-pinnin tilaksi 0 ja tämän jälkeen arvoon 1. Tilan muutosten välissä tulee odottaa muutama millisekuntti, jotta rekisteriarvot ehtivät nollaantua. Rekisteriarvojen nollauksen jälkeen määritetään audiopiirin rekisteriarvot I2C-väylän avulla. Isona erona äänittämiseen verrattuna on, että audiopiirin ADC-muunninta ei käynnistetä, vaan sen sijaan käynnistetään audiopiirin DAC-muunninyksikkö. Audiopiirin äänenvoimakkuus säädetään SD-kortilta ladatun arvon mukaiseksi. Äänidatan siirtämiseen audiopiirille käytetään I2S-väylää, joten myös tämän väylän asetukset määritellään tässä vaiheessa. Väylän määritykset perustuvat audiotiedostosta luettuihin arvoihin.



Kuva 15. Vuokaavio audiotiedoston toistamisesta.

Seuraavaksi varataan muistista tilaa audiotiedoston toistamiseen vaadittavaa puskurointia varten. Sujuvan äänenlaadun takaamiseksi käytetään kahta puskuria. Puskureiden muisti varataan muistista dynaamisesti, jolloin muisti voidaan vapauttaa myöhemmin. Puskuroinnin ideana on, että samanaikaisesti kun toisen puskurin näytteitä lähetetään audiopiirille, voidaan tiedoston seuraavat näytteet ladata toiseen puskuriin. Muistin varaamisen jälkeen ladataan ensimmäinen puskuri täyteen SD-muistista.

Ensimmäisten näytteiden lataamisen jälkeen kytketään painikkeiden sekä SPI-väylän keskeytykset, jolloin aloitetaan näytteiden siirtäminen audiopiirille. Audiopiirille lähetetään näyte aina I2S-väylän keskeytyksen tapahtuessa. Samalla huolehditaan myös audiopiirin kellottamisesta. Painikkeiden keskeytysten avulla audiopiirin äänenvoimakkuutta voidaan säätää myös toistamisen aikana. Laitteen toimintapainikkeesta voidaan keskeyttää audiotiedoston soitto. Ennen äänen toistamisen alkamista kytketään myös watchdog-ajastin päälle.

Datan lähettäminen audiopiirille tapahtuu I2S-väylän keskeytysten avulla. Samanaikaisesti toimivassa silmukassa seurataan toistettavan näytteen indeksiä ja sitä onko toi-

mintapainikkeella keskeytetty äänitteen toistaminen. Tämä silmukka huolehtii myös datan lataamisesta puskureihin. Audiotiedoston toistaminen lopetetaan, kun kaikki näytteet on lähetetty audiopiirille.

Audiotiedoston toiston loputtua kytketään painikkeiden sekä I2S-väylän keskeytykset pois. JTAG-väylän toiminnan varmistamiseksi tulee I2S-väylän käyttämien pinnien uudelleenohjaus ottaa pois toiminnasta. Myös watchdog-ajastin otetaan pois toiminnasta. Lisäksi puskurointiin käytetty muisti vapautetaan ja audiotiedosto suljetaan. Äänenvoimakkuuden taso tallennetaan SD-kortille, jotta viimeisin käytetty äänenvoimakkuuden taso on ladattavissa laitteen käynnistyessä uudelleen.

7 Vianetsintä

Mikrokontrolleripohjaisen laitteen onnistunut suunnittelu vaatii useiden virheenkorjaus- ja testaustyönkalujen käyttötaitoja. Ohjelma saattaa olla käynnissä, mutta sen toiminta voi olla tehotonta tai se saattaa antaa vääriä tuloksia. Vianetsintä saattaa olla ainoa työkalu näiden ongelmien eliminoimiseksi.

Yksi tapa yksinkertaisten mikrokontrolleripohjaisten systeemien testaamista ja vianetsintää varten on käyttää mikrokontrollerin I/O-pinnien vaihtelua, sarjaportin kautta lähetettyä dataa, LCD:lle tulostettua dataa jne. kertomaan ohjelman toiminnasta. Tässä tavassa sovelluksen koodin joukkoon lisätään vianetsintään käytettäviä komentoja ja sovellus ohjelmoidaan kehitystyön kohteena olevaan laitteistoon. Huonona puolena tässä testitavassa on sen tehottomuus ja hitaus.

Seuraavaksi esitellään muutama edellä kuvattu vianetsintätapa, joita käytin Puhuvan avaimenperän kehitystyön aikana. LED:in sytyttämisellä ja sammuttamisella voidaan kertoa käyttäjälle, mitä koodi tekee. Huonona puolena on se, että sitä ei voi käyttää nopeissa aikakriittisissä sovelluksissa. Mikrokontrolleri voi myös lähettää tietokoneelle dataa käyttäen USART-väylää. Haittana on, että datan lähettämisaika on pitkä, minkä lisäksi testaustapaa varten tarvitaan ylimääräisiä laitteita. Logiikka-analysaattoria voidaan käyttää esimerkiksi oheislaitteiden kommunikointiväylän, kuten USB:n, I2C:n tai SPI:n toiminnan tutkimiseen.

Sulautetun sovelluksen käydessä monimutkaisemmaksi säästää OCD (On-Chip Debugger) huomattavasti aikaa laitteen ongelmia selvittäessä. OCD on työkalu tutkittavan ohjelman pysäyttämiseksi muuttujien arvojen tutkimista ja muuttamista varten ohjelman suorittamisen aikana. OCD:n käyttämistä varten tarvitaan ohjelmointilaite sekä JTAG-liitäntä mikrokontrollerin tietokoneeseen liittämistä varten. Useat kehitysympäristöt sisältävät tuen vianetsintää varten. Tällaisissa kehitysympäristöissä on mahdollista testata laitteistoa graaffisen käyttöliittymän avulla. Puhuvan avaimenperän testauksessa käytetty käyttöliittymä oli kuitenkin yksinkertainen komentoriviltä käytettävä työkalu.

OCD:n käyttäminen vianetsintään vaatii kolme asiaa: ohjelman, joka kontrolloi sirua vianetsintä-moodissa, rajapinnan sirun kanssa kommunikointiin sekä vianetsintäinformaation sisältävän ohjelman, joka on käännetty ja asennettu sirulle. Sirun ja tietokoneen välisenä rajapintana käytetään GDB:tä (GNU Debugger). Useimmiten tämä ohjelma sisältyy kääntäjän kanssa pakettiin. Puhuvan avaimenperän tapauksessa käytettiin OpenOCD:tä mikrokontrollerin vianetsintäominaisuuksien hallintaan. OpenOCD tarjoaa portin, jonka kautta GDB yhdistetään mikrokontrolleriin.

Vianetsintään käytettävän laitteiston tulee olla yhdistettynä tietokoneeseen ja mikrokontrolleriin ennen OpenOCD:n käynnistämistä. OpenOCD:n saa käynnistettyä komennolla `openocd –f openocd.cfg.` Seuraavaksi käynnistetään GDB-ohjelma. GDB:n käynnistyksen yhteydessä on hyvä kertoa, mitä tiedostoa vianetsinnässä käytetään. CodeSourceryn kääntäjän mukana tulleen GDB-ohjelman saa käynnistettyä komennolla `arm-none-eabi-gdb talkingkey.elf`. GDB:n käynnistymisen jälkeen yhdistetään kohteena olevaan mikrokontrolleriin komennolla `target remote :3333`. Tämän jälkeen GDB on valmis vianetsintää varten.

GDB:n avulla voidaan asettaa pysäytyspisteitä, jolloin ohjelman suoritus pysäytetään. Pysäytyksen jälkeen voidaan tutkia mikrokontrollerin muuttujien ja rekisterien arvoja. Jos esimerkiksi halutaan tutkia Puhuvan avaimenperän toimintaa onnistuneen RFID-tunnisteen lukemisen jälkeen, voidaan HandleFoundTagID-funktioon asettaa pysäytyspiste. Tämän saa tehtyä komennolla `break HandleFoundTagID`. Keskeytyspiste voidaan asettaa myös koodin rivinumeron perusteella. Ohjelman suorittamista voidaan jatkaa pysäytyspisteen jälkeen komennolla `continue`. Ohjelman ollessa pysäytettynä voidaan muuttujien arvoja tutkia komennolla `print`. Esimerkiksi pysähtyessä HandleFoundID-komennon kohdalla saatetaan haluta tietää muuttujan uidLength arvo. Tämä tieto saadaa komennolla `print uidLength`. Jos muuttujan arvo halutaan tietää jo-

kaisella kerralla, kun pysäytyspiste saavutetaan, saadaan tämä tieto esimerkiksi komennolla `display uidLength`.

8 Yhteenveto

Työn tavoitteena oli suunnitella ja toteuttaa vaikeasti toisistaan erotettavien esineiden tunnistamista helpottava apuväline heikkonäköisille henkilöille. Tarkkaa määrittelyä laitteen vaatimuksista ei alussa annettu. Kaikki mitä tiedettiin oli, että laitteen tulisi olla pienikokoinen, helppokäyttöinen, esineiden tunnistamiseen tulisi käyttää RFID-teknologiaa ja tieto tunnistetusta esineestä tulisi välittää nauhoitetun äänen avulla. Alusta asti oli siis selvää, että projektissa joutuisi itsenäisesti punnitsemaan laitteen toteutuksen eri vaihtoehtoja, mutta samalla pääsisi myös luomaan jotain uutta alusta alkaen.

Alku oli minulle haastava, sillä jouduin heti tutustumaan useisiin minulle uusiin asioihin ja poistumaan omalta mukavuusalueeltani. RFID-tekniikasta minulla ei ollut mitään tietoa, elektroniikantuntemukseni oli heikkoa, käytettävä mikrokontrolleri oli minulle uusi ja kehitystyö tapahtuisi Linux-ympäristössä, joka ei kuulunut vahvimpiin osaamisalueisiini. Alun haasteellisuutta lisäsi, että koko kehitystiimiä ei ollut vielä kasattu, vaan se rakentuisi pikku hiljaa projektin edetessä.

Ensimmäinen tehtävä oli uuteen mikrokontrolleriin tutustuminen ja kehitysympäristön rakentaminen sitä varten. Onnekseni Electrian ammattitaitoinen henkilökunta oli auttamassa, joten alkuun pääsi kivuttomasti. Alkua helpotti huomattavasti myös STMicroelectronicsin valmistama STM3210E-EVAL-kehityskortti. Siitä löytyi valmiina useita komponentteja, jotka olisivat lopullisessakin laitteessa tarpeellisia, kuten SD-muistikortinlukija ja AK4343-audiopiiri. Muutama ensimmäinen viikko kului mikrokontrolleriin tutustuessa ja sen ominaisuuksilla leikkiessä.

Laitteen ensimmäinen prototyyppi rakennettiin STM3210E-EVAL-kehityskortin ympärille, johon PN532-RFID-lukijapiiri ja AK4642-audiopiiri kytkettiin johtimilla. Lopullisen laitteen eri osat ohjelmoin ja testasin osa kerrallaan. Ensimmäisenä lopullisen laitteen komponenteista testasin PN532-RFID-lukijapiirin toiminnan. Tässä vaiheestaan viimeistään ymmärsin, että sulautettujen laitteiden kehittämisessä ja ohjelmoinnissa on paljon kysymys eri komponentteihin tutustumisesta datalehtien avulla ja niiden toimin-

nan ymmärtämisestä. Tämän jälkeen vain pieni osa kaikesta kulutetusta ajasta menee itse ohjelmointiin. Huomasin myös, että tietämykseni eri kommunikointiväylistä ei ollut riittävä, vaan niihinkin tulisi tutustua perusteellisemmin. Kun olin opiskellut PN532-piirin ja SPI-väylän toiminnan riittävän hyvin, sujui ensimmäisen testikoodin kirjoittaminen ja ensimmäisten RFID-tunnisteiden lukeminen helposti.

Seuraavaksi testasin tiedostojen lukemisen ja kirjoittamisen SD-muistikortille. Electriassa oli tehty edellisessä projektissa laite, jolla myös käsiteltiin SD-muistikortilla olevia tiedostoja. Tässä laitteessa käytettiin samaa mikrokontrolleria, joka Puhuvaan avaimenperään oli valittu. Näin ollen SD-muistikortin käsittelyltä vaadittujen ominaisuuksien lisääminen ohjelmaan oli melko helppoa.

Ainoat puuttuvat ominaisuudet ensimmäisestä prototyypistä olivatkin enää audiotiedostojen toistaminen ja äänittäminen. STM3210E-EVAL-kehityskortissa on valmiina AK4343-audiopiiri, johon STMicroelectronic tarjoaa myös esimerkkikoodin. Tämä esimerkkikoodi toimikin hyvänä lähtökohtana ensimmäiselle varsinaiselle testikoodille, jossa soitin muistikortille valmiiksi tallennettuja WAV-tiedostoja. RFID-tunnisteiden lukemisen ja audiotiedostojen soittamisen yhdistäminen sujuikin tämän jälkeen vaivattomasti.

AK4343-piirin tarjoamat ominaisuudet eivät kuitenkaan olleet riittävät lopullisen laitteen vaatimuksiin nähden, vaan laitteella pitäisi myös pystyä nauhoittamaan uusia äänitiedostoja. Onneksi AK4343-piiristä on olemassa täsmälleen samalla pinnijärjestyksellä ja samoilla hallintarekistereillä varustettu laajennettu audiopiiri AK4642, joka mahdollistaa myös audiotiedostojen äänittämisen. Äänitysominaisuuksien ohjelmointi ei ollut mitenkään ylivoimainen haaste, sillä olihan osa piirin toiminnoista minulle jo entuudestaan tuttuja. Äänitysominaisuuksien ohjelmoinnin jälkeen olikin jo mahdollista rakentaa ensimmäinen toiminnallinen prototyyppi. Prototyyppi ei ollut kaunis eikä kovin pienikään, mutta se sisälsi kaikki lopullisen laitteen vaatimat ominaisuudet.

Tässä vaiheessa Puhuvan avaimenperän koko tiimi oli saatu koottua. Ryhmä sisälsi lisäkseni elektroniikkasuunnittelijan ja kaksi muotoilijaa. Projektin edetessä teimme yhteistyötä myös käytettävyystutkijoiden kanssa. Todellinen työnteko alkoi oikeastaan tästä eteenpäin, sillä vaikka laitteen perustoiminnallisuus oli kasassa, tulisi yksityiskohtiin kiinnittää entistä enemmän huomiota.

Laitteen käyttöliittymästä tehtiin useita versioita, joita toimivuutta testattiin yhteistyössä Näkövammaisten keskusliiton kanssa. Näissä testeissä sain paljon uutta tietoa siitä, miten eri tavalla eri henkilöt kokevat esimerkiksi laitteen käyttöliittymän. Laitteen ulkonäkö, muoto ja annetut äänipalautteet vaikuttavat suuresti ihmisten mielipiteeseen laitteen toimivuudesta. Käytettävyystestit lisäsivät omaa ymmärrystäni käyttäjälähtöisyyden merkityksestä tuotesuunnittelussa.

Laitteen elektroniikka tarjosi uusia haasteita myös ohjelmoinnin näkökulmasta. Kehityskortin ympärille rakennetussa prototyypissä äänityksen laatu oli moitteeton. Ensimmäisten piirilevylle koottujen prototyyppien äänenlaatu taas kärsi voimakkaasta kohinasta. Laitteen ensimmäiset versiot toimivat muutenkin huomattavasti epävakaammin kuin kehityskortin ympärille rakennetut versiot. Haastetta lisäsi vielä, ettei aina voinut olla varma, johtuvatko ongelmat elektroniikasta vai ohjelmasta. Kykyni etsiä elektronisista laitteista vikoja ja paikantaa niiden aiheuttajat kehittyivätkin projektin aikana huomattavasti. Tilanne myös pakotti tutkimaan audiopiirin toimintaa entistä tarkemmin ja pienin askelin äänenlaatua saatiinkin parannettua. Nyt piirilevyn viimeisimmällä versiolla laite toimii suhteellisen luotettavasti ja äänenlaatukin on kohtuullinen. Myös laitteessa käytetty pienikokoinen kaiutin oli haaste äänentoiston kannalta. Nauhoitettun ja toistettavan äänen ääritaajuuksia joutui suodattamaan voimakkaasti, jotta puhe kuullosti selvältä eikä säröytynyt. Halusin säilyttää inhimillisen äänensävyn, joten näiden kahden asian yhdistäminen vaati lukuisien tuntien testejä.

Kaiken kaikkiaan koen projektin onnistuneeksi. Laitteessa on kaikki ne ominaisuudet, joita siihen alun perin haluttiinkin. Testilaitteet ovat nyt olleen useamman kuukauden eri henkilöiden koekäytössä, ja ne toimivat vielä moitteettomasti. Laite siis vaikuttaisi olevan suhteellisen luotettava. Käytettävyystesteissä laite on saanut kiitosta muun muassa helppokäytöisyydestään, mukavasti käteen istuvasta muodostaan ja RFID-tunnisteiden nopeasta tunnistamisesta. Alkuperäiseen kokovaatimukseen ei päästy, mutta koekäyttäjän ovat siitä huolimatta pitäneet laitetta sopivan kokoisena. Laite on siis hyvä lähtökohta valmiin tuotteen kehittämistä varten. Kuvassa 16 on Puhuvan avaimenperän viimeisin versio.

Kuva 16. Puhuvan avaimenperän viimeisin prototyyppi.

Lähteet

1 Touch Memo -laitteen käyttöopas. Verkkodokumentti. Näkövammaisten keskusliitto. <http://aviris.nkl.fi/index.php?__file_display_id=8117>. Luettu 8.4.2013.

2 Touch memo -laite. 2011. Verkkodokumentti. MK Prosopsis. <http://www.mkprosopsis.com/Hardware/PersonalItems.htm >. Luettu 8.4.2013.

3 Penfriend-laitteen käyttöopas. 2012. Verkkodokumentti. < http://www.rnib.org.uk /Shop/Product%20instructions/DL7601.doc>. Luettu 8.4.2013.

4 Penfriend-laite. Verkkodokumentti. Perkins. <https://secure2.convio.net/psb/site/Ecommerce/1639969309?VIEW_PRODUCT=true&product_id=5501&store_id=1101>. Luettu 8.4.2013.

5 Himanshu Bhatt, Bill Glover. 2006. RFID Essentials. O'Reilly.

6 RFID-tekniikan fysikaaliset perusteet. Verkkodokumentti. RFIDLab Finland RY. <http://www.rfidlab.fi/rfid-tekniikan-perusteet>. Luettu 18.3.2013.

7 The insider's guide to the STM32 ARM based microcontroller. 2009. Verkkodokumentti. Hitex Ltd. <http://www.hitex.com/fileadmin/pdf/insiders-guides/stm32/isg-stm32-v18d-scr.pdf>. Luettu 10.5.2013.

8 RFID-standardit. Verkkodokumentti. RFIDLab Finland RY. <http://www.rfidlab.fi/rfid-standardit>. Luettu 19.3.2013.

9 RFID-tekniikan käyttämät taajuusalueet. Verkkodokumentti. RFIDLab Finland RY. <http://www.rfidlab.fi/rfid-tekniikan-käyttämät-taajuusalueet>. Luettu 19.3.2013.

10 RFID. 2013. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/RFID>. Luettu 19.3.2013.

11 Uusi RFID-tekniikan sovelluskohde ABB:lla. 2009. Verkkodokumentti. Vilant Systems Oy. <http://rfidlab.fi/system/files/ABB_outbound_RFID_press_FI_090417.pdf>. Luettu 19.3.2013.

12 The first full scale EPC/RFID system in rail. 2012. Verkkodokumentti. VR Transpoint. <http://rfidlab.fi/system/files/Case%20Study_VR%20Transpoint_ final.pdf>. Luettu 19.3.2013.

13 RFID:n hyödyntäminen kirjastoissa. 2006. Verkkodokumentti. ToP Tunniste. <http://www.toptunniste.fi/index.php?id=rfid-kirjasto>. Luettu 19.3.2013.

14 STM32F103RET6 datalehti. 2011. Verkkodokumentti. STMicroelectronics. <http://www.st.com/st-web-ui/static/active/en/resource/technical/document/ datasheet /CD00191185.pdf>. Luettu 18.3.2013.

15 PN532 NFC/RFID controller breakout board - v1.3. Verkkodokumentti. Adafruit. <http://www.adafruit.com/products/364>. Luettu 18.3.2013.

16 PN532 datalehti. 2012. Verkkodokumentti. NXP. <http://www.nxp.com/ documents/short_data_sheet/PN532_C1_SDS.pdf>. Luettu 18.3.2013.

17 AN1445 Antenna design guide for MFRC52x, PN51x, PN53x. 2010. Verkkodokumentti. NXP. <http://www.nxp.com/documents/application_note/ AN1445_An1444.zip>. Luettu 18.3.2013.

18 Asahi Kasei AK4642 datalehti. 2005. Verkkodokumentti. Asahi Kasei. <http://www.asahi-kasei.co.jp/akm/en/product/ak4642en/ak4642en_f00e.pdf>. Luettu 15.3.2013.

19 I2C-Bus: What's that?. Verkkodokumentti. <http://www.i2c-bus.org/>. Luettu 13.3.2013.

20 I2C-bus specification and user manual. 2012. Verkkodokumentti. <http://www.nxp.com/documents/user_manual/UM10204.pdf>. Luettu 13.3.2013.

21 I2S-bus specification. 1996. Verkkodokumentti. Philips Semiconductors. <http://www.sparkfun.com/datasheets/BreakoutBoards/I2SBUS.pdf>. Luettu 13.3.2013.

22 SDIO: Bridging the gap of modular hardware design today. 2009. Verkkodokumentti. Cypress Semiconductor Corp. <http://www.cypress.com/ ?docID=17406>. Luettu 12.3.2013.

23 Secure Digital Input/Output (SDIO) Card Specification. 2001. Verkkodokumentti. SD Association. <http://www.yumpu.com/en/document/view/3320687/sd-i-o-card-specification-sandisk>. Luettu 12.3.2013.

24 SD Memory Card Specifications. 2000. Verkkodokumentti. SD Group. <http://blog.ednchina.com/Upload/Blog/2007/3/26/46e97d13-85cb-4521-b500-2e3fdaf1fa0c.pdf >. Luettu 12.3.2013.

25 Reference manual - STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx. 2011. Verkkodokumentti. STMicroelectronics. <http://www.st.com/web/en/resource/technical/document/reference_manual/CD00171190.pdf>. Luettu 12.3.2013.

26 SPI - Overview and Use of the PICmicro Serial Peripheral Interface. Verkkodokumentti. Microchip. <http://ww1.microchip.com/downloads/en/DeviceDoc/spi.pdf> Luettu 14.2.2013.

27 SPI - Serial Peripheral Interface. 2006. Verkkodokumentti. <http://www.mct.net/faq/spi.html>. Luettu 14.3.2013.

28 SPI 8-bit circular transfer. 2007. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/File:SPI_8-bit_circular_transfer.svg>. Luettu 14.3.2013.

29 SPI - Block Guide. 2000. Verkkodokumentti. Motorola, Inc. <http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf>. Luettu 14.3.2013.

30 SPI-väylän kellon polaarisuus ja vaihe. 2013. Verkkodokumentti. Byte Paradigm. <http://www.byteparadigm.com/kb/admin/media_store/2/AA-00255/figure3.jpg>. Luettu 14.3.2013.

31 Audacityn lataussivu. 2013. Verkkodokumentti. Audacity. <http://audacity.sourceforge.net/download/>. Luettu 9.5.2013.

LIITE1: Pääohjelman koodi

```
#include <stdlib.h>
#include <stdio.h>
#include "math.h"
#include "string.h"
#include "main.h"
#include "stm32f10x.h"
#include "stmstuff.h"
#include "STM32_PN532.h"
#include "fat_sd/ff.h"
#include "sdio_disk.h"
#include "usart.h"
#include "powerdowntimer.h"
#include "battery.h"
#include "sounds.h"
#include "interrupts.h"
#include "recording.h"
#include "playback.h"
#include "hardware.h"

//#define USARTDEBUG                    //Uncomment to use USART for debugging
#ifdef USARTDEBUG
  #define debug(s)         serialconsole(s)
  #define debugHEX(HEX)    serialconsoleHEX(HEX)
#else
  #define debug(s)
  #define debugHEX(s)
#endif

/* Definitions */
#define BUTTON_NOT_PRESSED              0x00
#define UP_BUTTON_PRESSED               0x01
#define DOWN_BUTTON_PRESSED             0x02
#define FUNCTION_BUTTON_PRESSED         0x03
#define TAG_MAIN_FOLDER    "voice"      // Directory of the tag sound files
#define CONFIGURE_FOLDER   "Config"     // Directory of the volume.bin file
#define VOLUME_FILE        "volume.bin" // File of the volume level
#define POWERDOWNTIME      15000        // Powerdown time in milliseconds
#define NO_TAG                          0x00
#define EMPTY_TAG                       0x01
#define SOUND_TAG                       0x02

/* Global variables */
FATFS fatfs;
DIR dirobj;
FIL filobj;
char fullfilename[35];
__IO uint16_t ADC1ConvertedValue = 0;
uint8_t lasttag = NO_TAG;

/* Private functions */
static void RCC_SetSysClockTo36HSI(void);
static void TryToStartPN532(void);
static void InitializeSDcard(uint8_t *folder);
static void LoadPlaybackVolume();
static void SavePlaybackVolume();
static void CreateSubfolder(char *subdir);
static void AudioControlButtonsConfig(void);
static void DisableButtonInterrupts(void);
static void EnableButtonInterrupts(void);
static void EnableFunctionButtonInterrupt(void);
static uint8_t CheckButtonsState(void);
```

```
static void HandleButtonAction(uint8_t buttonpressed);
static void DownButtonAction(void);
static void FunctionButtonAction(void);
static void UpButtonAction(void);
static void HandleFoundTagID(void);
static void PlayLastTagSignal(void);
static void GoToStopMode(void);
static void HEXtoString(const uint8_t *HEX, char *string, uint8_t UIDlength);

/*************************************************************************
* Function Name  : main
* Description    : The main program
* Input          : None
* Output         : None
*************************************************************************/
int main(void) {
  uint8_t batterylevel;

  /* Change system clock to 36 MHz */
  RCC_SetSysClockTo36HSI();
  SystemCoreClockUpdate();

#ifdef USARTDEBUG
  /* Initialize usart for debugging */
  initserialconsole();
#endif
  debug("********************Talking key chain*********************");

  /* Configure buttons for the audio player*/
  AudioControlButtonsConfig();

  /* Try to get connection to PN532 and go to stop mode if it's not possible */
  debug("Testing PN532 ");
  TryToStart_PN532();

  /* Initialize SD-card and load volume setting for playback functionality */
  InitializeSDcard((uint8_t *)TAG_MAIN_FOLDER);
  LoadPlaybackVolume();

  /* Configure ADC for battery indicator and test if battery level is OK */
  Battery_Config();
  batterylevel = Battery_Batlevel();

  /* If battery level is under the minimum level go to standby mode */
  if(batterylevel==BATTERYLEVEL_CRITICALLOW) {
    debug("Going to stand-by mode\n");
    PN532_PowerDown(PN532_SPI);
    GoToStopMode();
  }
  /* If battery need to recharge system plays a sound signal */
  else if(batterylevel==BATTERYLEVEL_LOW) {
    debug("Play the warning signal sound");
    Play_BatteryTone();
    delay(200);
  }
  /* Plays a sound signal to tell to user that power is on  */
  debug("System power on.");
  Play_PowerOnTone();

  /* Start shutdown timer. Same timer is used for multiple purposes. That's why the
  flag */
  debug("Start shutdown timer\n");
  PowerDownTimer_Start(POWERDOWNTIME, POWERDOWNTIMER_TIMEOUT);
```

```
  /* Main loop */
  debug("Waiting for an ISO14443A Card ...");
  while(!(PowerDownTimer_TimerExpired())) {
    uint8_t readingSuccess;
    uint8_t buttonpressed = BUTTON_NOT_PRESSED;

    /* Handle status of the buttons and handle the actions if any of them is
    pressed */
    buttonpressed = CheckButtonsState();
    HandleButtonAction(buttonpressed);

    /* Try to read RFID tags */
    readingSuccess = PN532_readPassiveTargetID(PN532_SPI, PN532_MIFARE_ISO14443A,
             uid, &uidLength);
    if (readingSuccess) {
      HandleFoundTagID();
    }
  }

  PN532_PowerDown(PN532_SPI);
  Play_PowerOffTone();
  GoToStopMode();

  return 0;
}//end of main()

/*******************************************************************************
* Function Name  : RCC_SetSysClockTo36HSI
* Description    : Set system clock to 36 MHz by using internal oscillator
* Input          : None
* Output         : None
*******************************************************************************/
void RCC_SetSysClockTo36HSI(void) {
  uint32_t HSIStatus;
  uint32_t StartUpCounter = 0;

  /* Reset the RCC clock configuration to the default reset state(for debug
  purpose) */
  /* Set HSION bit */
  RCC->CR |= (uint32_t)0x00000001;
  /* Reset SW, HPRE, PPRE1, PPRE2, ADCPRE and MCO bits */
  RCC->CFGR &= (uint32_t)0xF0FF0000;
  /* Reset HSEON, CSSON and PLLON bits */
  RCC->CR &= (uint32_t)0xFEF6FFFF;
  /* Reset HSEBYP bit */
  RCC->CR &= (uint32_t)0xFFFBFFFF;
  /* Reset PLLSRC, PLLXTPRE, PLLMUL and USBPRE/OTGFSPRE bits */
  RCC->CFGR &= (uint32_t)0xFF80FFFF;
  /* Disable all interrupts and clear pending bits  */
  RCC->CIR = 0x009F0000;

  /* Enable HSI */
  RCC->CR |= ((uint32_t)RCC_CR_HSION);
  do {
    HSIStatus = RCC->CR & RCC_CR_HSIRDY;
    StartUpCounter++;
  }
  while((HSIStatus == 0) && (StartUpCounter != 0x0500));

  if ((RCC->CR & RCC_CR_HSIRDY) != RESET) {
    HSIStatus = (uint32_t)0x01;
  }
```

```
  else {
    HSIStatus = (uint32_t)0x00;
  }
  /* If that was successfull, we configure the PLL to use HSI as entry
  clock and to generate a system clock frequency of 36 MHz: */
  if (HSIStatus == (uint32_t)0x01) {
    /* Enable Prefetch Buffer */
    FLASH->ACR |= FLASH_ACR_PRFTBE;

    /* Flash 2 wait state */
    FLASH->ACR &= (uint32_t)((uint32_t)~FLASH_ACR_LATENCY);
    FLASH->ACR |= (uint32_t)FLASH_ACR_LATENCY_2;

    /* HCLK = SYSCLK */
    RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV1;

    /* PCLK2 = HCLK */
    RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV1;

    /* PCLK1 = HCLK */
    RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE1_DIV2;

    /*  PLL configuration: PLLCLK = HSI/2 * 9 = 36 MHz */
    /* Set Bits to 0 */
    RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_PLLSRC   |
                          RCC_CFGR_PLLXTPRE |
                          RCC_CFGR_PLLMULL ));
    /* Set Bits to 1 */
    RCC->CFGR |= (uint32_t)( RCC_CFGR_PLLSRC_HSI_Div2 | RCC_CFGR_PLLMULL9 );

    /* Enable PLL */
    RCC->CR |= RCC_CR_PLLON;
    /* Wait till PLL is ready */
    while((RCC->CR & RCC_CR_PLLRDY) == 0);

    /* Select PLL as system clock source */
    RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW));
    RCC->CFGR |= (uint32_t)RCC_CFGR_SW_PLL;
    /* Wait till PLL is used as system clock source */
    while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS) != (uint32_t)0x08);
  }
}

/*******************************************************************************
* Function Name  : PlayLastTagSignal
* Description    : Plays the signal tone of lastly identified tag. You can delete
*                : the sound file by holding function button down for 3 seconds
* Input          : None
* Output         : None
*******************************************************************************/
void PlayLastTagSignal(void) {
  Play_StartTone();
  if( !(Playback_InitForWave(fullfilename, RECORDED_AUDIO))) {
    EnableButtonInterrupts();
    Playback_PlayWave();
  }
  Playback_PowerDown(CodecPowerDown_HW);
  DisableButtonInterrupts();
}

/*******************************************************************************
* Function Name  : GoToStopMode
* Description    : Go to STOP mode and play signal tone
```

```
* Input          : None
* Output         : None
*******************************************************************************/
void GoToStopMode(void) {
  DisableButtonInterrupts();
  EnableFunctionButtonInterrupt();
  PWR_EnterSTOPMode(PWR_Regulator_LowPower, PWR_STOPEntry_WFI);
  // At this stage the system has resumed from STOP mode
  //Reset the system
  NVIC_SystemReset();
}

/*******************************************************************************
* Function Name  : HEXtoString
* Description    : Change hexadecimal number to a string
* Input          : HEX: Hexadecimal number you want to get as a string
*                : string: poiter to the returned string
*                : UIDlength: Length of the HEX number
* Output         : None
*******************************************************************************/
static void HEXtoString(const uint8_t *HEX, char *string, uint8_t UIDlength) {
  int i;
  char temp[(2*UIDlength)+1];
  for(i=0;i<UIDlength;i++) {
    if((HEX[i] >> 4) < 0x0A)
      temp[i*2] = ((HEX[i] >> 4) + 0x30);
    else
      temp[i*2] = ((HEX[i] >> 4) + 0x37);

    if((HEX[i] & 0x0F) < 0x0A)
      temp[i*2 + 1] = ((HEX[i] & 0x0F) + 0x30);
    else
      temp[i*2 + 1] = ((HEX[i] & 0x0F) + 0x37);
  }
  temp[2*UIDlength] = 0x00;
  strcpy(string, temp);
}

/*******************************************************************************
* Function Name  : InitializeSDcard
* Description    : Initialize SD-card and create(if doesn't exist) a folder for
*                  tags
* Input          : folder: main folder for the signal tone of the tags
* Output         : None
*******************************************************************************/
static void InitializeSDcard(uint8_t * folder) {
/*Initializing drive 0 of SD-card. Init status 1 if no disk and 0 if disk found*/
  debug("Initializing SD-card");
  if(disk_initialize(0)) {
    debug("No SD-card");
  }
  else {
    debug("SD-card found");
    if (f_mount(0, &fatfs)) {
      debug("Fail to mount SD-card");
    }
    else {$
      debug("Mounting successful");
      if (f_opendir(&dirobj, (XCHAR *)folder)) {
        debug("Can't open folder");
        if(f_mkdir((XCHAR *)folder)) {
          debug("Couldn't create directory");
        }
```

```
        else {
          debug("Directory created");
        }
      }
      else {
        debug("Folder opened");
      }
    }
  }
}

/*****************************************************************************
* Function Name  : LoadPlaybackVolume
* Description    : Load last used playback volume from SD-card
* Input          : None
* Output         : None
*****************************************************************************/
static void LoadPlaybackVolume() {
  uint8_t tempfulladdress[30];

  sprintf((char *)tempfulladdress,"%s/%s",CONFIGURE_FOLDER,VOLUME_FILE);
  if (f_opendir(&dirobj, CONFIGURE_FOLDER)) {
    if(!(f_mkdir(CONFIGURE_FOLDER))) {
      debug("Configure directory created");
      if(!(f_open(&filobj, (XCHAR *)tempfulladdress, FA_CREATE_NEW | FA_WRITE)) {
        debug("File created");
      }
      f_close(&filobj);
    }
  }
  else {
    debug("Configure folder opened");
    if(f_open(&filobj, (XCHAR *)tempfulladdress, FA_READ)) {
      if(!(f_open(&filobj, (XCHAR *)tempfulladdress, FA_CREATE_NEW | FA_WRITE))){
        debug("File created");
      }
    }
    else {
      uint8_t tempvolume = 0x00;
      uint8_t bytesread;
      debug("Volume setting file opened");
      f_read(&filobj, &tempvolume, 1, (UINT *)&bytesread);
      if(bytesread!=0) {
        debug("Setting volume");
        Playback_SetCurrentVolume(tempvolume);
      }
    }
    f_close(&filobj);
  }
}

/*****************************************************************************
* Function Name  : SavePlaybackVolume
* Description    : Save last used playback volume to the SD-card
* Input          : None
* Output         : None
*****************************************************************************/
void SavePlaybackVolume() {
  uint8_t tempfulladdress[30];
  uint8_t tempvolume[1];
  FIL volumefil;

  sprintf((char *)tempfulladdress,"%s/%s",CONFIGURE_FOLDER,VOLUME_FILE);
```

```
  tempvolume[0] =  Playback_GetCurrentVolume();

  /* At this point we assume that we have created Configure directory
  We also assume that we created volume.bin file to save last used volume */
  if(!(f_open(&volumefil, (XCHAR *)tempfulladdress, FA_WRITE))) {
    uint8_t byteswritten;
    debug("Volume setting file opened for write");
    if (!(f_write(&volumefil, tempvolume, 1, (UINT *)&byteswritten))) {
      debug("Volume file write OK");
      if (!(f_truncate(&volumefil))) {
        debug("truncate ok");
      }
    }
  }
  f_close(&volumefil);
}

/*****************************************************************************
* Function Name  : CreateSubfolder
* Description    : Created a subfolder for tags with 7 bytes long UID
* Input          : None
* Output         : None
*****************************************************************************/
void CreateSubfolder(char *subdir) {
  DIR subdirobj;
  debug(subdir);

  if(f_opendir(&subdirobj, subdir)) {
    if(!(f_mkdir(subdir))) {
      debug("Directory created");
    }
  }
  else {
    debug("Directory opened");
  }
}

/*****************************************************************************
* Function Name  : TryToStartPN532
* Description    : Try to initialize the PN532 and put the device to power saving
*                  mode if it wasn't possible
* Input          : None
* Output         : None
*****************************************************************************/
void TryToStartPN532(void) {
  int pn532timeout;

  PN532_SPIConfiguration();
  pn532timeout = 10;
  while(Initialize_PN532(PN532_SPI)&&(pn532timeout > 0)) {
    debug("Trying to initialize PN532:");
    delay(10);
    pn532timeout--;
  }
  if(pn532timeout == 0) {
    debug("Go to stop mode");
    GoToStopMode();
  }
}

/*****************************************************************************
* Function Name  : HandleFoundTagID
* Description    : Handle the actions based on the ID of the read tag
```

```
* Input          : None
* Output         : None
*******************************************************************************/
void HandleFoundTagID(void) {
  uint8_t uidLength;                                // Length of the UID number
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };          // UID as a HEX number
  char *filename;
  filename = (char *)malloc(9 * sizeof(char));

  PowerDownTimer_Stop();
  PN532_ShutDown();
  /* Mifare Classic card found */
  if(uidLength==0x04) {
    debug("Mifare Classic card found:");
    HEXtoString(uid, filename, 4);
    sprintf(fullfilename, "%s/%s.wav", TAG_MAIN_FOLDER, filename);
  }
  /* Mifare Ultralight card found */
  else if(uidLength==0x07) {
    char subdir[20];
    char *subfolder;
    subfolder = (char *)malloc(7 * sizeof(char));
    debug("Mifare Ultralight card found: ");
    HEXtoString(uid, subfolder, 3);
    HEXtoString(uid+3, filename, 4);
    sprintf(subdir, "%s/%s", TAG_MAIN_FOLDER, subfolder);
    CreateSubfolder(subdir);
    sprintf(fullfilename, "%s/%s/%s.wav", TAG_MAIN_FOLDER, subfolder, filename);
    free(subfolder);
  }
  debug(tagnumber);
  debug(fullfilename);
  /* Try to open file which name is same as ID of the tag */
  if(f_open(&filobj, (XCHAR *)fullfilename, FA_READ)) {
    debug("Empty Tag");
    Play_EmptyTagTone();
    lasttag = EMPTY_TAG;
  }
  else {
    debug("play tag");
    PlayLastTagSignal();
    while( !(GPIO_ReadInputDataBit(FUNCTION_GPIO, FUNCTION_PIN)) );
    lasttag = SOUND_TAG;
    SavePlaybackVolume();
  }
  free(filename);
  f_close(&filobj);

  /* Start shutdown timer to turn power off after specific time */
  debug("Start shutdown timer\n");
  PowerDownTimer_Start(POWERDOWNTIME, POWERDOWNTIMER_TIMEOUT);
  Start_PN532();
}

/*******************************************************************************
* Function Name  : AudioControlButtonsConfig
* Description    : Configures buttons for controlling the device
* Input          : None
* Output         : None
*******************************************************************************/
static void AudioControlButtonsConfig(void) {
  EXTI_InitTypeDef EXTI_InitStructure;
  GPIO_InitTypeDef GPIO_InitStructure;
```

```
  NVIC_InitTypeDef NVIC_InitStructure;

  /* Enable GPIOB, GPIOC and AFIO clock */
  RCC_APB2PeriphClockCmd( FUNCTION_GPIO_CLK
              | VOLDOWN_GPIO_CLK
              | VOLUP_GPIO_CLK
              | RCC_APB2Periph_AFIO, ENABLE);

  EXTI_DeInit();

  /* Configure PG.15 as input floating (Joystick UP)*/
  GPIO_InitStructure.GPIO_Pin = VOLUP_PIN;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
  GPIO_Init(VOLUP_GPIO, &GPIO_InitStructure);

  /* Configure PD.03 as input floating (Joystick DOWN) */
  GPIO_InitStructure.GPIO_Pin = VOLDOWN_PIN;
  GPIO_Init(GPIOD, &GPIO_InitStructure);

  /* Configure AD.00 as input floating (WAKE-UP BUTTON) */
  GPIO_InitStructure.GPIO_Pin = FUNCTION_PIN;
  GPIO_Init(FUNCTION_GPIO, &GPIO_InitStructure);

  GPIO_EXTILineConfig(VOLUP_EXTI_PORT_SOURCE, VOLUP_EXTI_PIN_SOURCE);
  GPIO_EXTILineConfig(VOLDOWN_EXTI_PORT_SOURCE, VOLDOWN_EXTI_PIN_SOURCE);
  GPIO_EXTILineConfig(FUNCTION_EXTI_PORT_SOURCE, FUNCTION_EXTI_PIN_SOURCE);

  EXTI_InitStructure.EXTI_Line =  VOLDOWN_EXTI_LINE | VOLUP_EXTI_LINE;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
  EXTI_InitStructure.EXTI_LineCmd = DISABLE;
  EXTI_Init(&EXTI_InitStructure);

  EXTI_InitStructure.EXTI_Line =  FUNCTION_EXTI_LINE;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
  EXTI_InitStructure.EXTI_LineCmd = DISABLE;
  EXTI_Init(&EXTI_InitStructure);

   /* Enable the EXTI0 Interrupt */
  NVIC_InitStructure.NVIC_IRQChannel = FUNCTION_EXTI_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
   /* Enable the EXTI3 Interrupt */
  NVIC_InitStructure.NVIC_IRQChannel = VOLDOWN_EXTI_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;

  NVIC_Init(&NVIC_InitStructure);
  /* Enable the EXTI15 Interrupt */
  NVIC_InitStructure.NVIC_IRQChannel = VOLUP_EXTI_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
  NVIC_Init(&NVIC_InitStructure);
}

/*******************************************************************************
* Function Name  : CheckButtonsState
* Description    : Check if is any of the buttons is pressed
* Input          : None
```

```
* Output         : Button pressed
*******************************************************************************/
uint8_t CheckButtonsState(void) {
  if(!(GPIO_ReadInputDataBit(FUNCTION_GPIO, FUNCTION_PIN))) {
    delay(10);
    if(!(GPIO_ReadInputDataBit(FUNCTION_GPIO, FUNCTION_PIN))) {
      return FUNCTION_BUTTON_PRESSED;
    }
  }
  else if(!(GPIO_ReadInputDataBit(VOLUP_GPIO, VOLUP_PIN))){
    delay(10);
    if(!(GPIO_ReadInputDataBit(VOLUP_GPIO, VOLUP_PIN))) {
      return UP_BUTTON_PRESSED;
    }
  }
  else if(!(GPIO_ReadInputDataBit(VOLDOWN_GPIO, VOLDOWN_PIN))) {
    delay(10);
    if(!(GPIO_ReadInputDataBit(VOLDOWN_GPIO, VOLDOWN_PIN))) {
      return DOWN_BUTTON_PRESSED;
    }
  }
  return BUTTON_NOT_PRESSED;
}

/*******************************************************************************
* Function Name  : HandleButtonActions
* Description    : Handle the actions of pressed button
* Input          : None
* Output         : None
*******************************************************************************/
void HandleButtonAction(uint8_t buttonpressed) {
  if(buttonpressed == DOWN_BUTTON_PRESSED){
    PN532_ShutDown();
    DownButtonAction();
    Start_PN532();
  }
  else if(buttonpressed == UP_BUTTON_PRESSED) {
    PN532_ShutDown();
    UpButtonAction();
    Start_PN532();
  }
  else if(buttonpressed == FUNCTION_BUTTON_PRESSED) {
    PN532_ShutDown();
    FunctionButtonAction();
    Start_PN532();
  }
}

/*******************************************************************************
* Function Name  : DownButtonAction
* Description    : Handle the action of down button
*                : Adjust volume level a step down
* Input          : None
* Output         : None
*******************************************************************************/
void DownButtonAction(void) {
  PowerDownTimer_Stop();
  Playback_PowerUp();
  if( Playback_ControlVolume(Volume_DOWN, VOLStep) ) {
    Play_VolumeSettingTone();
  }
  SavePlaybackVolume();
  debug("\nStart shutdown timer\n");
```

```
  delay(100);
  PowerDownTimer_Start(POWERDOWNTIME, POWERDOWNTIMER_TIMEOUT);
}

/*****************************************************************************
* Function Name  : UpButtonAction
* Description    : Handle the action of up button
*                : Adjust volume level a step up
* Input          : None
* Output         : None
*****************************************************************************/
void UpButtonAction(void) {
  PowerDownTimer_Stop();
  Playback_PowerUp();
  if( Playback_ControlVolume(Volume_UP, VOLStep) ) {
    Play_VolumeSettingTone();
  }
  SavePlaybackVolume();
  debug("Start shutdown timer\n");
  delay(100);
  PowerDownTimer_Start(POWERDOWNTIME, POWERDOWNTIMER_TIMEOUT);
}

/*****************************************************************************
* Function Name  : FunctionButtonAction
* Description    : Handle the action of function button
*                : Starts the recording of a new sound file
* Input          : None
* Output         : None
*****************************************************************************/
void FunctionButtonAction(void) {
  PowerDownTimer_Stop();
  if( (lasttag == SOUND_TAG) ) {
    uint32_t startrec = 700000;

    Play_DeleteWarningTone();
    while( ( !(GPIO_ReadInputDataBit(FUNCTION_GPIO, FUNCTION_PIN)) &
    (startrec > 0)) ) {
      startrec--;
    }
    if(startrec == 0) {
      Play_StartRecTone();
      Recording_InitializeAndRecord(fullfilename);
      Play_SavedTone();
    }
    while( !(GPIO_ReadInputDataBit(FUNCTION_GPIO, FUNCTION_PIN)) );
      lasttag = SOUND_TAG;
    }
  else if( (lasttag == EMPTY_TAG) ) {
    Play_StartRecTone();
    Recording_InitializeAndRecord(fullfilename);
    Play_SavedTone();
    while( !(GPIO_ReadInputDataBit(FUNCTION_GPIO, FUNCTION_PIN)) );
    lasttag = SOUND_TAG;
  }
  else if( (lasttag == NO_TAG) ) {
    Play_NoTagTone();
    while( !(GPIO_ReadInputDataBit(FUNCTION_GPIO, FUNCTION_PIN)) );
  }
  PowerDownTimer_Start(POWERDOWNTIME, POWERDOWNTIMER_TIMEOUT);
}

/*****************************************************************************
```

```
$* Function Name  : Disable_ButtonInterrupts
* Description    : Disable button interrupts
* Input          : None
* Output         : None
*******************************************************************************/
void Disable_ButtonInterrupts(void) {
  EXTI_InitTypeDef EXTI_InitStructure;

  EXTI_InitStructure.EXTI_Line = VOLDOWN_EXTI_LINE | VOLUP_EXTI_LINE;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
  EXTI_InitStructure.EXTI_LineCmd = DISABLE;
  EXTI_Init(&EXTI_InitStructure);

  EXTI_InitStructure.EXTI_Line = FUNCTION_EXTI_LINE;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
  EXTI_InitStructure.EXTI_LineCmd = DISABLE;
  EXTI_Init(&EXTI_InitStructure);
}

/*******************************************************************************
* Function Name  : EnableButtonInterrupts
* Description    : Enable button interrupts
* Input          : None
* Output         : None
*******************************************************************************/
void EnableButtonInterrupts(void) {
  EXTI_InitTypeDef EXTI_InitStructure;

  EXTI_InitStructure.EXTI_Line = VOLDOWN_EXTI_LINE | VOLUP_EXTI_LINE;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);

  EXTI_InitStructure.EXTI_Line = FUNCTION_EXTI_LINE;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);
}

/*******************************************************************************
* Function Name  : EnableFunctionButtonInterrupt
* Description    : Enable button interrupts
* Input          : None
* Output         : None
*******************************************************************************/
void EnableFunctionButtonInterrupt(void) {
  EXTI_InitTypeDef EXTI_InitStructure;

  EXTI_InitStructure.EXTI_Line = FUNCTION_EXTI_LINE;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);
}
```