KYMENLAAKSON AMMATTIKORKEAKOULU
Elektroniikan koulutusohjelma / Elektroniikan suuntautumisvaihtoehto

Esa Heikkilä

UHF-ALUEEN RFID-LUKIJAN ALUSTAVA OHJAUS

Opinnäytetyö 2010

TIIVISTELMÄ

KYMENLAAKSON AMMATTIKORKEAKOULU
Elektroniikka

HEIKKILÄ, ESA	UHF-alueen RFID-lukijan alustava ohjaus
Insinöörityö	43 sivua + 18 liitesivua
Työn ohjaaja	Lehtori Marko Saxell
Toimeksiantaja	Kymenlaakson Ammattikorkeakoulu Oy
Toukokuu 2010	
Avainsanat	RFID, mikro-ohjain, Atmel AVR, EPC Class-1 Gen 2

Passiivisten UHF-alueen RFID-tunnisteiden käyttö tavaroiden automaattiseen tunnistukseen on lisääntynyt muutaman vuoden aikana huomattavasti. Yhtenä syynä tähän on internetin kautta vapaassa levityksessä oleva EPCglobal UHF Class-1 Gen 2 -standardi, joka määrittelee fyysiset ja loogiset vaatimukset UHF-alueella toimivalle passiivisten RFID-tunnisteiden järjestelmälle.

Tässä opinnäytetyössä oli tarkoitus toteuttaa mikro-ohjaimilla EPCglobal UHF Class-1 Gen 2 -standardin mukaisen RFID-tunnisteen lukusimulaatio. Työ toteutettiin kahdella Atmelin AVR-mikro-ohjaimella. Toisella mikro-ohjaimella mallinnettiin lukijaa ja toisella tunnistetta. Mikro-ohjaimia käytettiin ja ohjelmoitiin mikro-ohjainten testaukseen tarkoitetuilla mikro-ohjainkorteilla. Tunnisteen lukusimulaation sisältämien signalointien mittauksiin käytettiin PicoScope 2205-PC-oskilloskooppia ja tunnistetieto todennettiin PC:llä sarjaportin välityksellä.

Työn tuloksena valmistui standardin mukaisesti toimiva lukusimulaatio, joka täyttää standardin asettamat signaloinnin aikavaatimukset ja koodaukset. Työssä mikro-ohjaimilla tehty toteutus tarjoaa toimivat RFID-lukijan ohjauksen periaatteet jatkossa mahdollisesti kehitettävälle RFID-lukijan toteutukselle.

ABSTRACT

KYMENLAAKSON AMMATTIKORKEAKOULU
University of Applied Sciences
Electronics Engineering

HEIKKILÄ, ESA	Preliminary Control of an RFID Reader for UHF Frequencies
Bachelor's Thesis	43 pages + 18 pages of appendices
Supervisor	Marko Saxell, Senior lecturer
Commissioned by	Kymenlaakso University of Applied Sciences
May 2010	
Keywords	RFID, microcontroller, Atmel AVR, EPC Class 1 Gen 2

Passive RFID tags are commonly used for automatic item management. In the past few years, RFID systems for passive tags at UHF frequencies have increased rapidly. One reason for this is the EPCglobal UHF Class 1 Gen 2 standard. The Standard defines the physical and logical requirements for a passive-backscatter RFID system and is freely available via the Internet.

The goal of this work was to implement a simulated tag identification for a passive RFID tag as defined in the EPCglobal UHF Class 1 Gen 2 standard using microcontrollers. The implementation was made by using Atmel AVR microcontrollers, which were used and programmed with development boards. One was modelled as a reader and the other as a tag. The correct simulation signaling was verified by doing the needed measurements with a PicoScope 2205 PC-oscilloscope. The coding was verified with a PC-terminal program.

As a result, a functionally correct simulation meeting the requirements for timing and coding as set in the standard was achieved. The simulated tag identification made with microcontrollers offers functional principles for a later and more sophisticated application.

SISÄLLYS

LYHENTEET JA TERMIT

A/D-muunnin	*Analog to Digital*, elektroninen kytkentä analogisen tiedon muuttamiseksi digitaaliseen muotoon
ASK	*Amplitude-shift keying*, digitaalinen modulointimenetelmä
AVR	Atmelin mikro-ohjaimia kuvaava kirjainyhdistelmä
Binaariluku	binaarijärjestelmän luku, jossa lukujen merkintää käytetään numeroita 0 ja 1
BLF	*Backscatter-link frequency*, linkkitaajuuden merkintä
CISC	*Complex Instruction Set Computer*, mikroprosessorityyppi
CRC	*Cyclic Redundancy Check*, tiedonsiirrossa käytettävä virheentarkistusmenetelmä
Data-0	0-bitin koodaussymboli PIE-koodauksessa
Data-1	1-bitin koodaussymboli PIE-koodauksessa
Dipoliantenni	kahden eri suuntiin erkanevan johtimen muodostama antenni
DR	*Divide Ratio*, linkkitaajuuden määritykseen käytettävä jakosuhde
DSB-ASK	*Double-sideband amplitude-shift keying*, digitaalinen modulointimenetelmä
EEPROM	*Electrically Erasable Programmable Read Only Memory*, sähköisesti purettava, haihtumaton luku-kirjoitusmuisti
EPC	*Electronic Product Code*, sähköinen tuotekoodi
FM0	*Bi-phase space encoding*, koodausmenetelmä jossa koodattavat tiedot kuvataan erilaisilla signaalin tilanvaihdoilla
Half duplex	tiedonsiirtomenetelmä jossa tietoa siirretään yhteensuuntaan kerrallaan
HF	*High Frequencies*, radioaaltojen taajuusalue 3 – 30 MHz
IEC	*International Electrotechnical Commission*, kansainvälinen sähköalan standardointijärjestö
I/O-liitäntä	*Input/Output*, tulo/lähtö-liitäntä
ISM-taajuusalueet	*Industrial, Scientific and Medical*, maailmanlaajuiset lupavapaat taajuusalueet
ISO	*International Organization for Standardization*, kansainvälinen standardointijärjestö

ITF	*Interrogator-Talks-First*, RFID-lukijan ja tunnisteen välinen tunnistusmenettely, jossa lukija aloittaa tiedonsiirron
Kantoaalto	lukijalta lähetettävä toimintataajuinen, moduloimaton ja sinimuotoisesti vaihteleva radioaalto
LF	*Low Frequencies*, radioaaltojen taajuusalue 30 – 300 kHz
Linkkitaajuus	tiedonsiirtotaajuus tunnisteelta lukijalle
Modulointi	menetelmä jolla koodattu data sovitetaan siirtotielle
PC-bitit	*Protocol-control bits*, tunnisteelta lähetettävän datamäärän ilmoittavien protokollabittien merkintä
PIE	*Pulse-interval encoding*, koodausmenetelmä jossa koodattavat tiedot kuvataan eripituisilla pulsseilla
PR-ASK	*Phase-reversal amplitude-shift keying*, digitaalinen modulointimenetelmä
PSK	*Phase-shift keying*, digitaalinen modulointimenetelmä
Radioaalto	taajuusalueella 3 Hz - 3 GHz esiintyvän sähkö- ja magneettikentästä koostuvan sähkömagneettisen säteilyn nimitys
RFID	*Radio Frequency Identification*, radioaalloilla toteutettava etätunnistustekniikka
RISC	*Reduced Instruction Set Computer*, mikroprosessorityyppi
RN16	16-bittinen satunnaisluku
RTcal	*Interrogator-to-Tag calibration symbol*, lukijan lähettämä kalibrointisymboli 0- ja 1-bitin tulkintaan
SPI	*Serial Peripheral Interface*, sarjaliikenneliitäntä jota Atmelin AVR-mikro-ohjaimissa käytetään ohjelmointiliitäntänä
SRAM	*Static Random Access Memory*, mikro-ohjaimissa käyttömuistina käytettävä muistityyppi
SSB-ASK	*Single-sideband amplitude-shift keying*, digitaalinen modulointimenetelmä
Tari	*Type A Reference Interval*, PIE-koodauksessa käytettävä vertailuaikaväli
TID	*Tag ID*, tunnisteen muistialueen merkintä
TRcal	*Tag-to-Interrogator calibration symbol*, lukijan lähettämä kalibrointisymboli linkkitaajuuden määritykseen
UHF	*Ultra High Frequencies*, radioaaltojen taajuusalue 300 – 3000 MHz

1 JOHDANTO

Tässä opinnäytetyössä tehtiin alustava UHF-alueen RFID-lukijan ohjausmenettely. Työ tehtiin Kymenlaakson ammattikorkeakoululle ja se on jatkoa aiemmin tehdyille RFID-tekniikkaa käsitteleville opinnäytetöille. Työn tavoitteena oli tehdä EPCglobal UHF Class 1 Gen 2 -standardin mukainen RFID-tunnisteen lukusimulaatio. Kyseinen standardi määrittelee 860 - 960 MHz:n taajuusalueella toimivien passiivisten RFID-tunnisteiden ja näiden lukijan välisen yhteyskäytännön.

Atmelin AVR-mikro-ohjaimilla toteutetun simuloinnin toimivuus todennettiin Pico-Scope-PC-oskilloskoopin avulla. Simuloinnissa toinen mikro-ohjain mallinsi lukijaa ja toinen tunnistetta. Työn keskeisin osa oli suunnitella ja toteuttaa mikro-ohjaimien ominaisuuksilla standardin määräämät koodaukset. Mikro-ohjaimien täytyi kyetä tekemään koodaus ja dekoodaus PIE- ja FM0-koodausten mukaisesti. Lisäksi lukijan tuli toteuttaa määritetyt ohjauskäytännöt. Tässä työn dokumentaatiossa esitetään toteutuksen johdannoksi lyhyt katsaus RFID-tekniikkaan ja sen sisältämiin standardeihin, mutta pääpaino on työn kohteena olevassa EPCglobal UHF Class 1 Gen 2 -standardissa. Työn toteutusosa koostuu suurelta osin tehdyistä mittauksista. Työssä simulointiin tehdyt mikro-ohjainten ohjelmakoodit esitetään selkeyden vuoksi liitteinä.

2 RFID-TEKNIIKKA

Lyhenteellä RFID (Radio Frequency Identification) tarkoitetaan yleisesti radioaaltojen avulla toteutettua etätunnistustekniikkaa. RFID-tekniikka kattaa laajan joukon eri järjestelmiä, joiden tunnusomaisena piirteenä tiedon sisältävän tunnisteen ja tiedon lukijan välillä ei ole fyysistä kontaktia, vaan viestintä tapahtuu radioaaltojen välityksellä.

RFID-järjestelmä koostuu yksinkertaisimmillaan lukijasta ja tunnisteista sekä tietokoneesta. Järjestelmässä RFID-tunniste on liitettynä tunnistettavaan kohteeseen, sisältäen kohteeseen liittyvää informaatiota. Informaatio voidaan lukea tai kirjoittaa tunnisteeseen RFID-lukijan synnyttämien radioaaltojen avulla. Lukija on yhteydessä tietokoneeseen, joka on varustettuna tunnistetiedon keräävällä RFID-ohjelmistolla. RFID-ohjelmistolla hallinnoidaan lukijan ja tunnisteen välistä tiedonsiirtoa sekä suoritetaan

tiedonkäsittelyä luetulle datalle. Tiedonkäsittelyssä tehdään varsinainen kohteen tunnistus, jossa tunnisteesta luettu koodi liitetään vastaamaan määrättyä kohdetta.

RFID-tekniikka kuuluu viivakooditunnistuksen tavoin automaattisiin tunnistustekniikoihin. Tekniikka tarjoaa kuitenkin monia etuja viivakoodiin verrattuna. Esimerkkeinä tunnisteen sisältämää tietoa voidaan lukea ilman suoraa näköyhteyttä tunnisteeseen ja tunnisteeseen voidaan tallentaa viivakoodia suurempi määrä tietoa. Lisäksi tunnisteeseen tallennettu tieto voidaan vaihtaa uuteen tarvittaessa, toisin kuin tulostettuihin viivakoodeihin. RFID-tunnisteen heikkoutena viivakoodiin nähden on korkeampi valmistuskustannus, minkä vuoksi RFID-tekniikka ei kaikissa sovelluskohteissa ole kannattava viivakoodin korvaavana tekniikkana. RFID-tekniikkaa sovelletaan nykyään jo todella moniin eri tarkoituksiin. Tekniikkaa käytetään mm. tilaus- ja toimituslogistiikkaan, teollisuuden valmistusprosessien seurantaan, kauppojen vähittäismyyntiin, kulunvalvontaan, eläinten merkintään, maksujärjestelmiin ja varkauden estoon. (1.)

2.1 Lukija ja tunniste

RFID-lukija on laite, jonka avulla voidaan langattomasti sekä lukea tietoa RFID-tunnisteesta että tallentaa tietoa tunnisteeseen. Luku- sekä kirjoitustoiminnon lisäksi lukija toimii myös virtalähteenä passiivisille RFID-tunnisteille. Pystyäkseen kommunikoimaan tunnisteiden kanssa lukijan ja tunnisteiden täytyy olla yhteensopivia keskenään. Tämä tarkoittaa sitä, että lukija ja tunniste toimivat samalla taajuusalueella ja käyttävät tiedonsiirtoon samaa tiedonsiirtoprotokollaa. Tästä vuoksi eri taajuusalueella toimivat ja eri tiedonsiirtoprotokollaa käyttävät tunnisteet tarvitsevat omat lukijansa. RFID-lukija sisältää kuitenkin samat pääpiirteet käyttötaajuudesta ja lukutavasta riippumatta. Lukija voidaan jakaa tominnallisesti ohjausosaan ja suurtaajuusosaan. Ohjausosa on yhteydessä RFID-lukijaa ohjaavan tietokoneen RFID-ohjelmistoon ja toteuttaa sen antamia käskyjä. Lukijan ohjausosa ohjaa lukijan ja tunnisteen välistä tiedonsiirtoa ja tekee lähetettävän tiedon koodauksen ja vastaanotetun tiedon dekoodauksen käytetyn tiedonsiirtoprotokollan mukaisesti. Suurtaajuusosan tehtäviin kuuluu suurtaajuisen lähetystehon muodostus sekä koodatun signaalin modulointi ja vastaavasti vastaanotetun signaalin demodulointi. (1.)

RFID-tunniste koostuu yksinkertaisimmillaan muistin sisältävästä mikrosirusta ja siihen liitetystä radioaallon kytkentäelementistä. Kytkentäelementti riippuu käytetyn järjestelmän taajuusalueelle soveltuvasta kytkeytymistavasta ollen yleensä joko käämitty

johdin tai dipoliantenni. Tunnisteen fyysinen koko määräytyy suurelta osin kytkentäelementistä ja tunnistetyypistä. Tunnisteita on montaa eri muotoa ja kokoa. Se voi esimerkiksi olla tunnistettavaan kohteeseen liitettävä kortti, liimattava tarra, nappi tai jopa ihon alle sijoitettava implantti. (2.)

RFID-tunnisteet voidaan jakaa niiden toimintaperiaatteen mukaisesti aktiivisiin, passiivisiin ja puolipassiivisiin tunnisteisiin. Aktiiviset tunnisteet sisältävät mikrosirun lisäksi oman lähettimen ja virtalähteenä toimivan pariston. Aktiiviset tunnisteet saavat herätesignaalin lukijalta, mutta käyttävät omaa virtalähdettä tunnistetiedon lähettämiseen. Tunnisteiden sisältämä lähetin ja virtalähde mahdollistavat aktiivisille tunnisteille useiden kymmenien metrien lukuetäisyyden. Samalla ne kuitenkin tekevät tunnistetyypistä hinnaltaan kaikkein kalleimman, ja lisäksi virtalähteen ehtyminen rajoittaa tunnisteen käyttöikää. Passiiviset tunnisteet eivät sisällä omaa virtalähdettä, vaan käyttävät tiedonsiirtoon lukijan lähettämää radioaaltoa, lähettämällä osan radioaallosta lukijalle takaisin. Samasta lukijan radioaallosta tunnisteet saavat myös käyttöjännitteensä. Passiivinen tunniste onkin tästä ominaisuudesta johtuen tunnistetyypeistä edullisin. Tunnisteen virtalähteettömyys rajoittaa passiivisten tunnisteiden lukuetäisyyden alle kymmeneen metriin. Puolipassiiviset tunnisteet sijoittuvat aktiivisten ja passiivisten tunnisteiden väliin. Ne sisältävät oman virtalähteen, kuten aktiiviset tunnisteet, mutta käyttävät passiivisten tunnisteiden tavoin tiedonsiirtoon lukijan radioaaltoja. Puolipassiivisissa tunnisteissa paristo toimii käyttöjännitteenä tunnisteen mikropiirille, mutta sitä ei käytetä tiedonsiirtoon. (2.)

2.2 Taajuusalueet ja standardit

Radioaaltojen taajuusalueilla katsottuna erilaiset RFID-järjestelmät toimivat LF-, HF- ja UHF-alueella. Lisäksi virallisesta taajuusaluejaosta poiketen käytetään nimitystä mikroaaltoalueen RFID-järjestelmä yli gigahertsin taajuudella toimivasta järjestelmästä.

Radioaalto koostuu kahdesta komponentista: sähkö- ja magneettikentästä. Tiedonsiirron kannalta radioaallon sisältämä informaatio voidaan vastaanottaa valinnaisesti joko sähkö- tai magneettikentästä, koska ne sisältävät saman informaation. RFID-tekniikassa tiedonsiirtoon käytetty radioaallon komponentti vaihtelee taajuusalueittain. Radioaallon magneettikentän välityksellä tapahtuvaa tiedonsiirtoa käytetään yleisesti LF- ja HF-taajuusalueilla. Magneettikentän välityksellä tapahtuvasta kytkeytymisestä

käytetään nimitystä induktiivinen kytkentä ja se tunnetaan myös ns. lähikenttä-tekniikkana. Radioaallon magneettikentän avulla lukijan kanssa kommunikoiva tunniste sisältää esimerkiksi metallisilmukoita tai käämityn kelan, johon tiedonsiirtoon käytettävä energia magneettikentästä kytkeytyy. UHF-ja mikroaaltoalueilla tietoa siirretään yleisesti radioaallon sähkökentän avulla. Tällöin tunnisteen tiedonsiirtoon käytettävä energia kytkeytyy tunnisteen dipoliantenniin. UHF-alueelle on kehitetty myös ns. lähikenttä-tekniikkaa hyväksi käyttävä tunnistustekniikka. (3.)

Standardit ovat olennainen osa RFID-teknologiaa. Erityisesti logistiikassa standardit näyttelevät suurta osaa, kun eri toimitusketjun jäsenten täytyy pystyä lukemaan samoja tunnisteita. Kansainväliset standardit tarjoavat lisäksi valmistajariippumattomuuden, jolloin yritys ei ole sidoksissa mihinkään tiettyyn lukijoiden ja tunnisteiden valmistajaan. Kansainvälinen standardointijärjestö ISO on julkaissut jokaiselle RFID-taajuusalueelle omat tiedonsiirtostandardinsa. ISOn standardien lisäksi elektronisen tuotekoodin käyttöönottoa ajava yritys EPCglobal Inc. on julkaissut vapaan tiedonsiirtostandardin passiivisille UHF-alueen RFID-tunnisteille. (4.)

LF-taajuusalueella RFID-tekniikan hyödyntämät taajuudet keskittyvät 135 kHz:iä pienemmille taajuuksille. ISO-standardi ISO/IEC 18000-2 määrittelee tiedonsiirtoprotokollan 135 kHz:iä pienempiä taajuuksia käyttäville tunnisteille. LF-taajuusalueen tunnisteiden yhtenä sovelluskohteena on eläimiin liitettävä tunniste, jota koskien ISO on julkaissut standardit ISO 11784/11785:n ja ISO 14223:n. LF-aluetta käytetään myös eri kulunvalvontajärjestelmissä, joissa yleinen käytetty taajuus on 125 kHz. (4; 5.).

HF-alueella käytetään ISM-taajuusalueelle kuuluvaa 13,56 MHz:iä, joka on vakiintunut yleisesti käytetyksi taajuudeksi. ISO-standardi ISO/IEC 18000-3 on yleinen tiedonsiirtostandardi 13,56 MHz:n taajuudella toimiville RFID-järjestelmille. Taajuutta 13,56 MHz käytetään lisäksi erilaisiin langattomiin älykortteihin, joiden oleelliset standardit ovat ISO/IEC 14443 ja ISO/IEC 15693. Erilaisissa maksusovelluksissa käytettävä MIFARE-tekniikka pohjautuu juuri ISO/IEC 14443-standardiin. (4; 5.)

UHF-alueella käytetyt taajuudet ovat 433 MHz ja 860 - 960 MHz:n välinen taajuusalue. Taajuutta 433 MHz käytetään lähinnä oman virtalähteen sisältävissä aktiivisissa RFID-tunnisteissa, joiden tiedonsiirtoon liittyvät ominaisuudet määritetään ISO/IEC 18000-7-standardissa. Taajuusaluetta 860 - 960 MHz käytetään yleisesti passiivisissa tunnisteissa, joiden tiedonsiirron määrittelevät standardit ISO/IEC 18000-6 ja EPC-

global Class-1 Gen 2. EPCglobal Class-1 Gen 2 -standardi on vapaassa levityksessä oleva standardi ja on yhteensopiva standardin ISO/IEC 18000-6C kanssa. (4; 5.)

Mikroaaltoalueen käytetyin taajuus on ISM-taajuuksiin kuuluva 2,45 GHz:n taajuus. Kyseisellä taajuudella käytetään sekä passiivisia että aktiivisia tunnisteita, joiden molempien toimintaparametrit määrittelee standardi ISO/IEC 18000-4. (1; 5.)

3 EPCGLOBAL CLASS-1 GEN 2 -STANDARDI

Viralliselta nimeltään EPCglobal Class 1 Generation 2 UHF Air Interface Protocol Standard on 860 - 960 MHz:n väliselle taajuusalueelle kehitetyn RFID-järjestelmän standardi. Standardin määrittelemä järjestelmä käyttää passiivisia tunnisteita, joiden tunnistusmekanismi perustuu radioaaltojen heijastumiseen. Standardi perustuu tutkimusohjelma MIT Auto-ID Centerin kehittämään EPC-teknologiaan. Yritysten GS1 ja GS1 US muodostama yhteisyritys EPCglobal sai marraskuussa 2003 hallintaansa EPC-teknologian sillä ehdolla, että se olisi kaikille ilmaiseksi saatavilla. Samalla MIT Auto-ID Centerin EPC-teknologiaan liittyvä tutkimustyö jaettiin maailmanlaajuisesti eri Auto-ID laboratorioihin, jotka toimivat yhteistyössä EPCglobalin kanssa. Auto-ID keskuksen valmiiksi kehittämien tunnisteluokkien EPC Class-0 ja Class-1 tiedonsiirtoprotokollat eivät olleet keskenään yhteensopivia ja ne eivät soveltuneet kansainvälisiin radiotaajuussääntöihin. Tästä johtuen EPCglobal alkoi vuonna 2004 kehittämään toisen sukupolven tunnistestandardia, joka korjaisi aiemmissa tunnisteluokissa havaitut ongelmat. Kehitystyönä syntynyt toisen sukupolven tunnistestandardi Class-1 Generation-2 -protokolla hyväksyttiin joulukuussa 2004 ja standardin versio 1.1.0 on yhteensopiva standardin ISO/IEC 18000-6C kanssa. Standardi tunnetaan yleisesti nimellä EPCglobal Class-1 Gen 2. (2; 6.)

EPCglobal Class 1 Gen2 -standardin määrittelemän järjestelmän rakentuessa passiivisille RFID-tunnisteille tiedonsiirtoon käytetään ITF-menetelmää, jolloin lukija aloittaa kommunikoinnin. Standardi määrittää lukijan lähettämään tietoa tunnisteelle 860 - 960 MHz:n väliseltä alueelta.Tästä tunnisteen lähetteestä passiivinen tunniste saa paitsi informaation myös käyttöjännitteensä. Tieto tunnisteesta saadaan lähettämällä lukijalta kantoaaltoa, josta tunniste heijastaa osan takaisin lukijalle.Tunniste heijastaa lukijan kantoaaltoa koodatun tiedon mukaisesti, yhteydelle määrätyn nopeuden ns. linkkitaajuuden tahdissa. (6.)

3.1 Standardin ominaisuudet

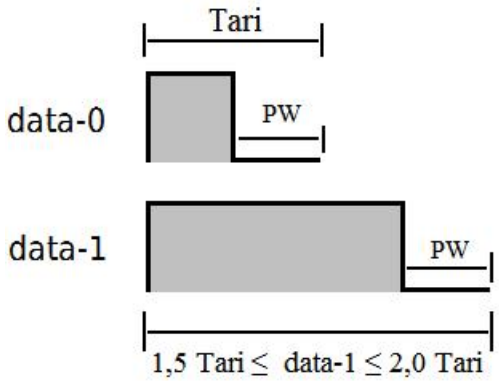
EPCglobal Class-1 Gen 2 -standardi määrittelee standardia tukevien UHF-alueen RFID-lukijoiden ja -tunnisteiden väliseen tiedonsiirtoon käytettävän yhteyskäytännön eli protokollan. Standardi määrittelee sekä fyysiset että loogiset ominaisuudet. Oleellisimmat määritellyt ominaisuudet ovat käytettävä taajuusalue, lukijan ja tunnisteen käyttämät modulointi- ja datan koodausmenetelmät, lähetettävän ja vastaanotetun tiedon tiedonsiirtonopeudet, tunnisteen muistin ja käytettävien komentojen rakenne, siirretyn datan virheenhavaitseminen ja tiedon törmäyksenestomenetelmä. (6.)

Standardi määrittelee käytettäväksi toimintataajuusalueeksi 860 - 960 MHz, jonka rajoissa standardia tukevien lukijoiden ja tunnisteiden täytyy kyetä toimimaan keskenään. RFID-lukijan toimintataajuuden valintaan vaikuttaa olennaisesti maakohtaiset radiotaajuuksia koskevat määräykset. Suomessa UHF-alueen RFID-käyttöön varatut taajuudet ja niille sallitut tehotasot on määrätty Viestintäviraston luvasta vapaiden radiolähettimien yhteistaajuuksista ja käytöstä koskevassa määräyksessä Viestintävirasto 15 Z/2009 M.

Standardin järjestelmä käyttää tiedonsiirtoon half duplex-menetelmää, jolloin tunniste saa käyttöjännitteestä lukijan kentästä koko viestinnän ajan, mutta tietoa siirretään vain yhteen suuntaan kerrallaan. Tiedonsiirrossa lukijan ja tunnisteen välillä lähetys aloitetaan lähetettävän binaariluvun eniten merkitsevästä bitistä. (6.)

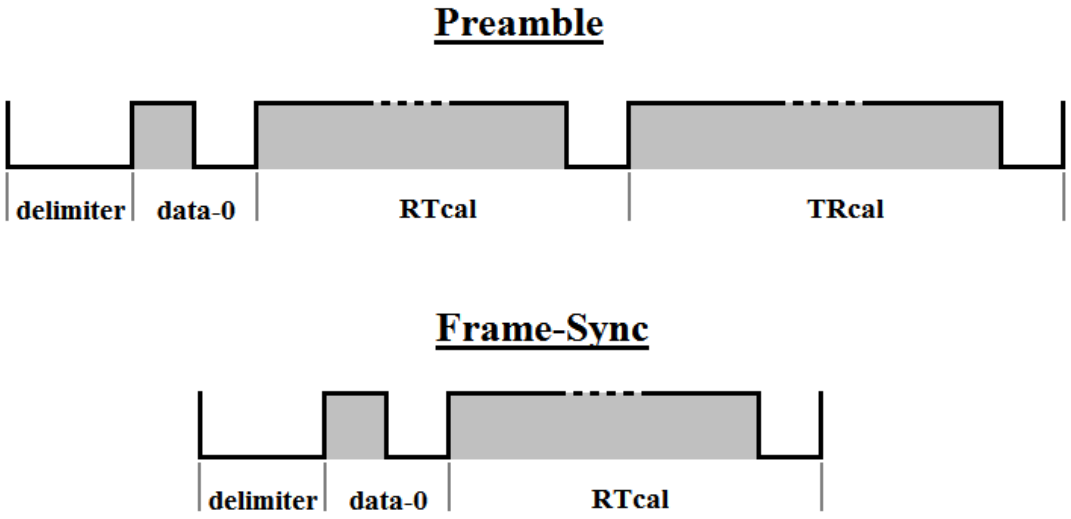
Koodaukset

Lukijan lähettämän binaarisen tiedon koodaukseen käytetään PIE-koodausta. PIE-koodauksessa binaarisen tiedon 0 ja 1 kuvataan eri pituisilla pulsseilla. Tällöin kuvatut symbolit ovat eri mittaisen ajan ylätilassa, mutta vakioajan alatilassa. PIE-koodauksen symbolit on esitetty kuvassa 1. Standardin mukaisessa koodauksessa käytetään vertailuaikaväliä, jota merkitään Tari-termillä. Tari-termi tulee ISOn standardista ISO/IEC 18000-6 part A, jossa Tari on lyhennys sanoista Type A Reference Interval. Tari määrittelee ensisijaisesti 0-bitin kuvaukseen käytettävän data-0-symbolin ajan. Muut symboliajat määritetään suhteessa Tariin. Lukijan käyttämä Tari-arvo tulee olla väliltä 6,25…25 μs. (6.)



Kuva 1. *PIE-koodaus. Tari-merkintä kuvaa data-0-symbolin ajan ja asettaa rajat data-1-symbolille. Kuvan merkintä PW kuvaa alatilassa olevaa signaalia, jonka kestoaika on sama kaikilla PIE-koodatuilla symboleilla.* (6.)

Lukijalta tunnisteelle lähetettäviin komentoihin tulee liittää alkutahdistus, jolla määrätään yhteyden tiedonsiirron ominaisuudet. Tahdistuksilla kerrotaan lukijan käyttämät tietosymboliajat data-0 ja data-1 sekä määrätään yhteydessä käytettävä linkkitaajuus. Alkutahdistuksia on kaksi eri tyyppiä: Preamble- ja Frame-Sync-tahdistus. Tahdistuksen käyttö riippuu lähetettävästä komennosta. Preamle-tahdistusta käytetään edeltämään Query-komentoa, jolla tunnistus aloitetaan. Muita komentoja tulee edeltää Frame-Sync-tahdistus. Tahdistusjaksot on esitetty kuvassa 2. (6.)



Kuva 2. *Lukijan tahdistusjaksot. Kuvassa lukijan lähettämiin komentoihin liitettävät alkutahdistukset, joista Query-komentoon liitetään Preamble-tahdistus ja muihin komentoihin Frame-Sync-tahdistus.* (6.)

Preamble koostuu peräkkäisessä järjestyksessä seuraavista neljästä eri osasta: delimiter, data-0, RTcal ja TRcal. Delimiter-osassa signaali on 12,5 μs vakioajan alatilassa, josta tunniste osaa varautua lähetetyn tiedon vastaanottoon. Data-0-osa on 0-bitin ku-

vaukseen käytettävä jakso, jonka aika on käytetty Tari. RTcal on lukijalta tunnisteelle lähetettävän tiedon kalibrointisymboli, jonka aika on data-0 ja data-1 symbolien yhteenlaskettu aika. RTcal-symbolista tunniste laskee keskiarvoajan, jonka perusteella tunniste tulkitsee keskiarvoaikaa lyhyemmät symbolit 0-biteiksi ja pidemmät symbolit 1-biteiksi. TRcal on tunnisteelta lukijalle heijastettavan tiedon kalibrointisymboli, jota käytetään tunnisteelta takaisinheijastuvan tiedon linkkitaajuuden määritykseen. Tunnisteelta takaisinheijastettavan tiedon taajuus määräytyy paitsi TRcal-symbolin ajasta, myös valitusta jakosuhteesta DR, joka lähetetään tunnisteelle Query-komennolla. Jakosuhde voi olla joko 8 tai 64/3. TRcal-symbolin kestoajalle määräytyy ehto $1.1 \cdot RTcal \leq TRcal \leq 3 \cdot RTcal$. Linkkitaajuus määräytyy Kaavasta 1. Linkkitaajuus voi vaihdella 40 kHz…640 kHz:iin. (6.)

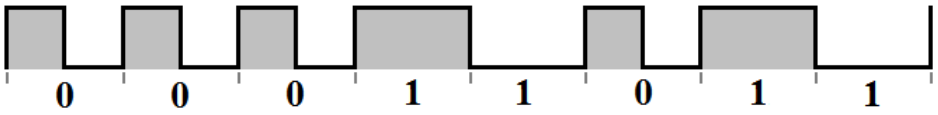
$$BLF = \frac{DR}{TRcal} \quad (1)$$

Kaava 1 määrää käytettävän linkkitaajuuden. Kaavassa BLF tarkoittaa linkkitaajuutta Hertzeinä ja TRcal on Preamble-tahdistuksessa lähettävän samannimisen symbolin aika sekunteina. DR on käytettävä jakosuhde (8 tai 64/3).

Frame-Sync-tahdistus on lyhennetty Preamble-tahdistus, jolloin se sisältää Preamble-tahdistuksen osat delimiter, data-0 ja RTcal, mutta ei TRcal-symbolia. Lukijan tulee käyttää samaa RTcal arvoa Frame-Sync-tahdistuksessa kuin Preamble-tahdistuksessa. (6.)

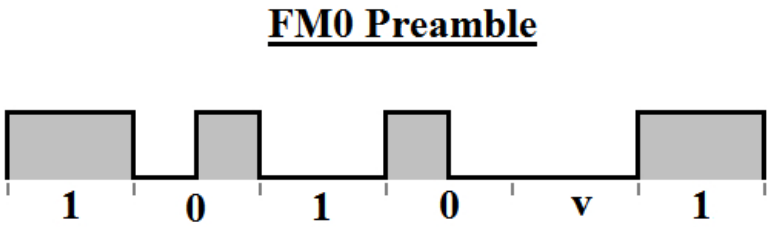
Standardin määrittämässä tunnisteessa tiedon koodaukseen voidaan käyttää joko FM0-koodausta tai Miller-modulated subcarrier-koodausta. Koodaustapa valitaan lukijan Query-komennossa lähettämän M-parametrin mukaisesti. Sama parametri määrää myös käytettävän datanopeuden. (6.)

FM0-koodauksessa signaalin tila vaihtuu alatilasta ylätilaan tai päinvastoin joka symbolilla. Koodauksessa 0-bitin kuvaukseen käytettävä symboli vaihtaa lisäksi tilaansa linkkitaajuuden jakson puolivälissä, kun taas 1-bitin kuvaava symboli pysyy samassa tilassa jakson ajan. Signaalin tila vaihtuu siis paitsi 0-bitti-symbolin keskellä niin myös joka symbolin välissä. FM0-koodauksen esimerkkikoodaus on esitetty kuvassa 3. (6.)



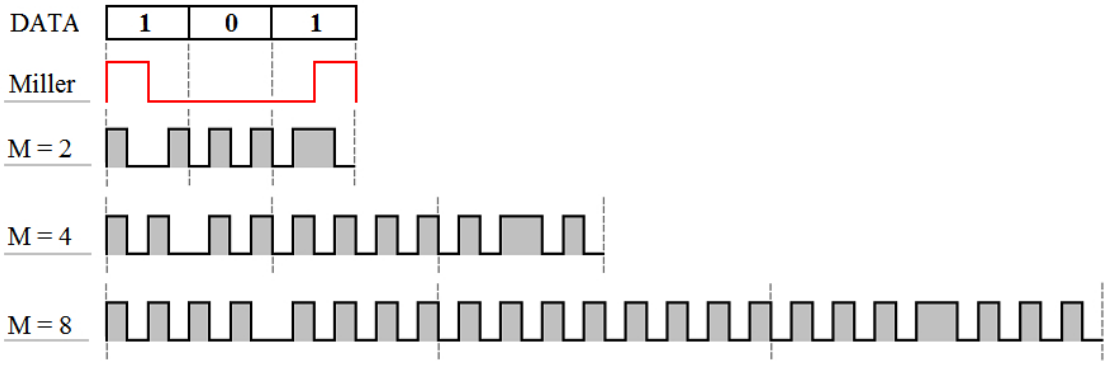
Kuva 3. *FM0-koodaus esimerkki. Kuvassa esitetty esimerkki 8-bitin FM0-koodauksesta.* (6.)

FM0-koodatun tiedon lähetys aloitetaan voidaan aloittaa kahdella eri FM0 Preamble-alkutahdistuksella. Käytettävissä on joko normaali tai pidennetty alkutahdistus, joka määräytyy lukijan lähettämän Query-komennon TRext-parametrin mukaan. Normaali FM0 preamble-tahdistus esitetty kuvassa 4. Tunnisteen muistiin kirjoitettaessa tunniste lähettää TRext-parametrin arvosta huolimatta pidennetyn alkutahdistuksen, jolloin tahdistuksen alkuun lisätään 12 linkkitaajuuden jaksoa. FM0-koodattu tiedon loppuun liitetään ylimääräinen 1-bitin symboli ns. dummy-päätebitti merkiksi lähetettävän tiedon loppumisesta. (6.)



Kuva 4. *FM0 Preamble-tahdistus. Signalointijakso jonka tunniste lisää jokaisen lähetteensä alkuun.* (6.)

Miller-modulated subcarrier-koodauksessa symbolin tila vaihtuu 1-bitin kuvaavan symbolin puolivälissä mikä on päinvastoin kuin FM0-koodauksessa. Miller-koodauksessa tila vaihtuu samojen peräkkäisten bittien välillä, mutta ei peräkkäisten 0 ja 1 bittien välillä. Lopullinen Miller-koodaus tehdään käyttämällä joko 2, 4 tai 8 linkkitaajuuden jaksoa kantataajuisen Miller-koodauksen kuvaamiseen. Koodauksessa käytettävä jaksomäärä valitaan lukijan Query-komennon M-parametrilla. Kuvassa 5 on kuvattu esimerkkidatan Miller-koodaus. Miller-koodattu lähete alkaa Miller-preamble-alkutahdistuksella, joka on valittavissa Query-komennon TRext-parametrillä joko normaalina tai pidennettynä versiona. Miller-koodattu lähete tulee päättyä FM0-koodauksen tavoin 1-bitin symbolia vastaavaan päätebittiin. (6.)



Kuva 5. *Miller-koodaus esimerkki. Kuvassa on 3-bittisen datan esimerkkikoodaus Miller-modulated subcarrier-koodauksella. Koodaustulos riippuu valitusta M-parametrista.* (6.)

Modulointi

Lukijan koodatulla datalla tehtävä kantoaallon modulointi voidaan tehdä kolmella eri menetelmällä. Lukijan on mahdollista käyttää joko DSB-ASK-, SSB-ASK- tai PR-ASK-modulointimenetelmää. Tunnisteen tulee kyetä demoduloimaan lukijan lähetettä kaikilla kolmella modulointimenetelmillä. Tunniste voi käyttää heijastettavan kantoaallon modulointiin ASK-tai PSK-modulointimenetelmää. Lukijan täytyy vastaavasti kyetä demoduloimaan vastaanotettavaa lähetettä kyseisillä menetelmillä. (6.)

Virheentarkistus

Lukijan ja tunnisteen välisen tiedonsiirron siirtovirheen havaitsemiseen käytetään komennon yksilöllistä pituutta ja CRC-tarkistusmenetelmää. CRC-tarkistusmenetelmässä lähetettävään dataan lisätään tarkisteeksi jakojäännös, joka saadaan jakamalla data jakajapolynomilla. Binaariselle datalle jakojäännös saadaan lisäämällä datan loppuun käytetyn CRC-tarkisteen mukainen määrä nollia ja jakamalla se käytetyllä CRC-polynomilla. Jakaminen tapahtuu suorittamalla bittien välillä looginen XOR-operaatio. Liite 1 havainnollistaa CRC-tarkistuksen laskentaa. Vastaanotossa jakojäännöksen sisältämä data jaetaan samalla sovitulla jakajalla kuin lähetyksessä. Virheettömän datan jako tulee vastaanotossa mennä tasan, jolloin jakojäännös on nolla. Käytetyt CRC-tarkistusmenetelmät ovat usein muunnelmia tästä suoraviivaisesta menetelmästä. Tällöin laskennassa voidaan lisäksi käyttää alkuarvoa ja bittioperaatiota. Alkuarvo on ns. esijakaja, jolla tehdään ensimmäinen XOR-operaatio ennen varsinaista jakajapolynomia. CRC-tarkistuksen bittioperaatio on yleensä laskennan lopuksi jakojäännökselle tehtävä invertointi. Standardissa lukijan ja tunnisteen välisessä tiedonsiirrossa käytetään CRC-16- ja CRC-5-tarkistusta, jonka laskenta on esitetty liitteessä 1. CRC-16-laskennassa käytetään CRC-16-CITT-jakajaa, joka on on polynomimuodossa $x^{16} + x^{12} + x^5 + 1$ ja binaarisena 1 0001 0000 0010 0001. Alkuarvona

käytetään lukua $FFFF_h$ ja laskennan lopuksi tehdään jakojäännöksen invertointi, josta saadaan lähetteeseen lisättävä CRC-16-tarkiste. Standardissa käytetyn CRC-5-tarkistuksen jakaja on polynomina $x^5 + x^3 + 1$ ja binaarisena 101001. CRC-5-laskennassa käytetään alkuarvona binaarilukua 01001. Standardissa esitetään piirikytkennät CRC-16- ja CRC-5-tarkisteiden laskentaan. (6; 7; 8.)

Törmäyksen hallinta

Usean tunnisteen samanaikaisesta lähetteestä syntyy törmäys, jolloin lukijan on vaikea erottaa yksittäistä tunnisteen lähetettä. Standardissa törmäyksenesto toteutetaan todennäköisyyteen perustuvalla menetelmällä. Menetelmän toteuttamiseksi tunnisteessa käytetään alaspäin laskevaa 15-bittinen lähetysvuorolaskuria ja satunnaislukugeneraattoria. Laskurille arvotaan lukijan käskystä alkuarvo, jota laskuri vähentää sitä käskettäessä. Kun laskuri saa arvon 0, tunniste saa lähetysvuoron. Tunnisteiden törmäysten estämiseksi käytetään lukijan Query-komennolla lähettämää Q-parametria, jolla saadaan tunnisteen laskurille alkuarvo väliltä 0…32767. Q-parametri mahdollistaa lukijalla tehtävän tunnisteiden vastaustodennäköisyyden säätelyn. Parametrin arvon mukaan tunnisteen vastauksen todennäköisyys vaihtelee välillä 0,000031…1. (6.)

Tunnisteen muisti

Tunnisteen muisti jaetaan neljään eri muistialueeseen, joiden sisältämät muistipaikat ovat 16-bitin pituisia. Muistialueet on nimetty seuraavasti:

- *Reserved*
- *EPC*
- *TID*
- *User*

Reserved-muistialue on vapaavalintainen ja voi sisältää 32-bitin pituiset Kill- ja Access-salasanat. Kill-salasanaa käytetään tunnisteen toiminnan lakkauttamiseen ja Access-salasanaa siirryttäessä tunnisteen turvattuun Secured-tilaan. Tunnisteen, joka ei toteuta kyseisiä salasanoja, tulee toimia kuten salasanat olisivat arvoltaan nollia ja pysyvästi lukittuina. Tällöin tunnisteen ei tarvitse fyysisesti toteuttaa Reserved-muistialuetta. (6.)

EPC-muistialue on pakollinen ja sen tulee sisältää EPC-koodi, joka on tunnistuskoodi tunnisteen sisältävälle tuotteelle. Erilaiset EPC-koodirakenteet määritellään EPCglobal Tag Data Standards-spesifikaatiossa. Tunnistekoodi lisäksi EPC-muistialue sisäl-

tää 16 PC-bittiä ja 16-bittisen CRC-16-tarkistusluvun joka lasketaan PC-bittien ja koodin tarkisteeksi. PC-bitit sisältävät tiedon lähettävästä datamäärästä. Ensimmäiset 5 PC-bittiä määrittelevät lähetettävän PC-bittien ja EPC-koodin yhteispituuden, mahdollistaen 32:n muistipaikan osoituksen, jolloin EPC-muistin koko voi maksimissaan olla 512 bittiä. (6.)

TID-muistialue on pakollinen muistialue, joka sisältää tunnisteen valmistukseen liittyvää tietoa. Muistialue voi sisältää mm. valmistajatunnisteen ja tunnisteen sarjanumeron. User-muistialue on valinnainen muistialue, joka sallii käyttäjäkohtaisen tiedon tallennuksen. (6.)

Tunnisteen tilat

Tunniste toimii tilakoneen tavoin, jolloin tunnisteen toiminta määräytyy paitsi vastaanotetusta komennosta myös tunnisteen aiemmasta tilasta. Standardi määrittelee tunnisteelle seuraavat 7 eri tilaa: *Ready, Arbitrate, Reply, Acknowledged, Open, Secured ja Killed*. (6.)

Tunnistekoodin EPC-koodin lukuprosessissa kierretään järjestyksessä tilat Ready, Reply ja Acknowledged, josta palataan takaisin Ready-tilaan. Mahdolliset siirtymiset tilasta toiseen on standardissa kuvattu tilakohtaisilla siirtymä-taulukoilla. Ready-tila on nimensä mukaisesti alkutila, johon tunnisteen tulee siirtyä saadessaan käyttöjännitteen lukijan kantoaallosta. (6.)

Arbitrate-tila on eräänlainen odotustila, johon tunnistuskierrokseen osallistuva tunniste siirtyy Ready-tilasta arpoessaan lähetysvuorolaskurin alkuarvoksi nollaa suuremman luvun. Tunniste pysyy Arbitrate-tilassa niin kauan kun lähetysvuorolaskurin arvo on suurempi kuin nolla. (6.)

Reply-tila on ensimmäinen tunnisteen vastaustila, johon tunniste siirtyy Arbitrate-tilasta tai suoraan Ready-tilasta saadessaan lähetysvuorolaskurin arvoksi nolla. Tällöin tunniste lähettää 16-bittinen satunnaisluvun. (6.)

Acknowledged-tila on tunnistekoodin lukemisen päätetila, johon tunniste siirtyy Reply-tilasta saatuaan lukijalta oikeanmuotoisen ACK-komennon, joka sisältää tunnisteen aiemmin Reply-tilasta lähettämän 16-bittisen satunnaisluvun. Acknowledged tilaan siirtyessä tunniste lähettää tunnistuskoodiosat: PC + EPC + CRC-16. (6.)

Open-tila on tila johon tunniste voi siirtyä Acknowledged-tilasta. Open-tilassa tunniste voi suorittaa Lock-komentoa lukuunottamatta kaikki muut lukijan lähettämät access-komennot. (6.)

Secured-tila on Open-tilaa muistuttava, mutta ominaisuuksiltaan laajempi mahdollistaen kaikkien access-komentojen toteutuksen. Secured-tilaan siirrytään tunnisteen ominaisuuksien perusteella kahta eri tietä. Tunniste, jonka accesss-salasana on nolla, siirtyy Acknowledge-tilasta Secured-tilaan saatuaan Req_RN-komennon lukijalta. Tunniste, jonka access-salasana on erisuuri kuin nolla, siirtyy Open-tilasta Secured-tilaan saatuaan oikeanmuotoisen Access-komentosarjan. (6.)

Killed-tilaan tunniste voi siirtyä joko Open- tai Secured-tilasta saatuaan oikeanmuotoisen Kill-komentosarjan. Killed-tilaan siirryttyään tunniste ei enää toteuta mitään lukijan lähettämiä komentoja. Killed-tila on pysyvä, jolloin Killed-tilaan menneen tunnisteen tulee myöhemminkin käyttöjännitteen saatuaan siirtyä suoraan Killed-tilaan. Mikäli tunnisteen Kill-salasana on nolla Killed-tilaa ei suoriteta, vaan tunniste lähettää Kill-komentosarjan saatuaan lukijalle virhesanoman. (6.)

Tunnistusvalinnat

Standardi tarjoaa tunnisteiden lukemiseen tunnistusvalintoja. Tunnisteen tulee toteuttaa 4 eri istuntotilaa S0, S1, S2 ja S3. Jokainen istuntotila voi olla kahdessa eri tilassa, joka ilmoitetaan ns. istunnon tunnistuslipulla. Tunnistuslipun tilaa merkitään arvoilla A ja B. Tunnisteen tunnistekoodin luku voidaan suorittaan vain yhdessä istuntotilassa kerrallaan, jolloin valitun istuntotilan tunnistuslippu-arvo täytyy olla lukijan valinnan mukainen. Istuntotilojen ja niiden tunnistuslippujen lisäksi tunnisteen tulee toteuttaa erillinen SL-valintalippu, jota voidaan käyttää tunnistuskierrokseen osallistuvien tunnisteiden valintaan. Valintalippu voidaan asettaa päälle tai pois. (6.)

Komennot

Standardissa määritellään tunnisteen toteutettavaksi kaikkiaan 14 eri komentoa, joista 11 on pakollista. Komennot jaetaan tehtävän mukaisesti kolmeen eri ryhmään:

- *Select*
- *Inventory*
- *Access*.

Select-ryhmä tehtävänä on nimensä mukaan tehdä tunnisteeseen muutoksia, jotka vaikuttavat valintaan tunnistuskierrokselle. Ryhmä sisältää ainoastaan Select-komennon. Select-komennolla on kaksi pääasiallista tehtävää. Sillä voi joko vaihtaa tunnisteen istuntotilojen tunnistuslippu-arvoa tai asettaa tunnisteen SL-valintalippu päälle tai pois. Yhdellä Select-komennolla voidaan tehdä vain toinen kuvatuista toiminnoista. (6.)

Inventory-ryhmän komennot koskevat ns. tunnistuskierrosta, jossa luetaan tunnisteen tunnistekoodi. Inventory-komennot ovat Query, QueryRep, QueryAdjust, ACK ja NAK. Query-komento on standardin oleellisin komento, jolla aloitetaan tunnistekoodien kysely. Komennolla lähetetyistä parametreista määräytyy tunnisteiden linkkitaajuus, datan koodaus, datanopeus, tunnisteen lähettämä alkutahdistus ja tunnisteen vastauksen todennäköisyys. Lisäksi voidaan rajata tunnistettavien tunnisteiden määrää valintakriteerien avulla. (6.)

QueryRep-komento vähentää tunnisteen arpomaa vastausvuoronumeroa yhdellä. Tunnisteen lähetysvuoronumeron saadessa arvon nolla tunniste vastaa lukijalle 16-bittisellä satunnaisluvulla. QueryAdjust-komennolla muutetaan tunnisteiden vastaustodennäköisyyttä muuttaen Query-komennolla lähetettyä Q-parametria. (6.)

ACK-komento on kuittauskomento, jonka lukija lähettää tunnisteelle saatuaan tunnisteelta vastauksena 16-bittisen satunnaisluvun. Komennossa lukija lähettää kuittauksena saamansa satunnaisluvun takaisin tunnisteelle. Komennon sisällön ollessa oikea, tunniste vastaa lukijalle lähettämällä tunnistuskoodiosat: PC + EPC + CRC-16. (6.)

NAK-komento aiheuttaa tunnisteiden siirtymisen arbitrate-tilaan, muuttamatta tunnistuslippujaan. Tunnisteen tulee toteuttaa NAK-komento kaikista muista tiloista paitsi ready ja killed. (6.)

Access-ryhmän komennot liittyvät tunnisteen kokonaisvaltaiseen hallintaan. Ryhmän komennot mahdollistavat tunnisteen tiedon lukemisen, kirjoittamisen ja lukitsemisen sekä tunnisteen toiminnan lakkauttamisen. Access-ryhmän pakolliset toteutettavat komennot ovat: Req_RN, Read, Write, Lock ja Kill. Tunniste voi valmistajavalinnaisesti toteuttaa myös access-komennot: Access, BlockWrite ja BlockErase. (6.)

Req_RN-komennolla tunniste määrätään lähettämään lukijalle uusi 16-bittinen satunnaisluku, jota käytetään access-komentojen parametrina. Read-komento mahdollistaa

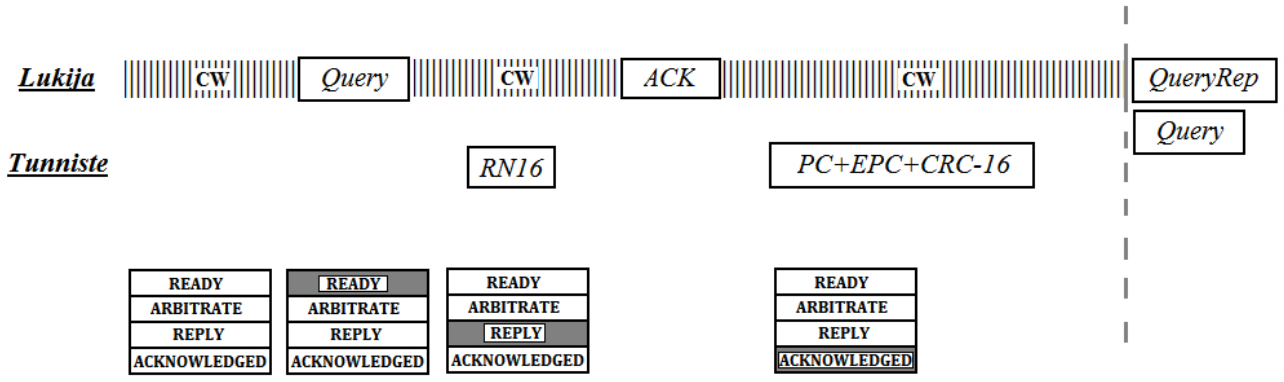
määrätyn tietomäärän lukemisen valitusta tunnisteen muistialueesta. Write-komento mahdollistaa tiedon tallentamisen haluttuun muistialueeseen. Write-komennolla voidaan tallentaa tietoa 16-bittiä kerrallaan. (6.)

Lock-komento mahdollistaa tunnisteen sisältämän tiedon lukituksen. Komennolla voidaan estää tai sallia mahdollisten salasanojen lukeminen/tallennus. Komennolla tehtävä lukitus voi olla joko muutettavissa tai pysyvä. Kill-komentoa käytetään moniosaisessa Kill-operaatiossa, joka lakkauttaa tunnisteen toiminnan pysyvästi. (6.)

Valmistavalinnaista Access-komentoa käytetään access-proseduurissa siirryttäessä tunnisteen Open-tilasta Secured-tilaan. BlockWrite-komento mahdollistaa yli 16 bitin tallennuksen tunnisteelle yhdellä komennolla. BlockErase-komennolla voidaan tunnisteen muistista poistaa tietoa yli 16 bittiä kerralla. (6.)

3.2 Tunnistekoodin lukuproseduuri

Tunnisteen sisältämän tunnistekoodin eli EPC-koodin lukeminen tapahtuu kyselykomennoilla, jotka lähetetään PIE-koodattuna, lähetysmodulaation ollessa DSB-ASK, SSB-ASK tai PR-ASK. Tiedon vastaanotto tunnisteelta tapahtuu lähettämällä komennon jälkeen kantoaaltoa vastaanottaen samalla tunnisteen takaisinheijastuksen sisältämää tietoa. Tunniste heijastama lähete on joko ASK-tai PSK-moduloitua. Lukija määrää vastaanotettavan tiedon koodauksen, joksi voidaan valita joko FM0- tai Miller-koodaus. Seuraavaksi on selostettu suoraviivainen tunnisteen EPC-koodin lukuproseduuri, joka on esitetty kuvassa 6. (6.)



Kuva 6. *Tunnistekoodin lukuproseduuri. Kuvassa esitetään lukijan ja tunnisteen lähetteet, sekä tunnisteen tilat lähetteiden aikana. Merkintä RN16 kuvaa tunnisteelta lähetettävää satunnaislukua ja merkintä CW lukijan lähettämää kantoaaltoa.*(6.)

Proseduuri käynnistyy lukijan lähettäessä kantoaaltoa vähintään 1,5 ms:n ajan. Tunniste muodostaa tässä ajassa itselleen käyttöjännitteen ja siirtyy Ready-tilaan. Seuraavaksi lukija lähettää tunnisteelle Query-komennon, joka määrää tunnisteen lähetysvuorollaan lähettämään 16-bittisen satunnaisluvun. Lähetettyään Query-komennon lukija lähettää kantoaaltoa odottaen vastausta tunnisteelta. Tunniste lähettää lukijalle vastausvuorollaan 16-bittisen satunnaisluvun ja siirtyy Reply-tilaan. Mikäli tunniste ei heti saa vastausvuoroa, se jää Arbitrate-tilaan odottamaan QueryRep-komentoa vastausvuoron saamiseksi. Vastaanotettuaan tunnisteen 16-bittisen satunnaisluvun lukija lähettää tunnisteelle kuittauksena ACK-komennon, joka sisältää tunnisteen lähettämän satunnaisluvun. Komennon lähetettyään lukija lähettää jälleen kantoaaltoa ja odottaa tunnisteen vastausta. Tunnisteen vastaanottaessa oikeanmuotoisen ACK-komennon se siirtyy Acknowledged-tilaan ja lähettää lukijalle tunnistuskooditiedot sisältäen PC-bitit, EPC-koodin ja CRC-16-virheentarkistusbitit. Tunnistekoodin noudon jälkeen voidaan luku uusia Query-komennolla tai lähettää QueryRep-komento, jolloin luettu tunniste siirtyy Ready-tilaan, aiheuttaen samalla mahdollisen toisen tunnisteen satunnaisluvun lähetyksen lukijalle. (6.)

4 TOTEUTUSLAITTEISTO

RFID-lukijan alustavan ohjauksen toteutus päädyttiin tekemään ohjelmoitavilla 8-bittisillä mikro-ohjaimilla. Käytetyiksi mikro-ohjaimiksi valikoituivat puolijohdevalmistaja Atmelin valmistamat ATtiny2313 ja ATmega32. Kyseisten mikro-ohjaimien ohjelmointiin ja kytkentöjen tekemiseen käytettiin Atmelin valmistamille mikro-ohjaimille tarkoitettuja EXB2313- ja PROJ32/644-ohjelmointi-/testausalustoja. Toteutetun ohjauksen mittauksiin käytettiin PicoScope 2205-PC-oskilloskooppia, jonka ohjaus ja mittaukset tehtiin PC-tietokoneen sovellusohjelmalla.

4.1 Mikro-ohjain

Mikro-ohjain (*engl. microcontroller*) on pienoiskoossa oleva mikrotietokone. Mikro-ohjain on keskeinen komponentti erilaisissa ns. sulautetuissa sovelluksissa, laitteissa joissa mikrotietokone on osana muuta elektroniikkajärjestelmää. Mikro-ohjain sisältää mikroprosessorin, muistit, liitäntäpiirit sekä muita oheispiirejä yhdistettynä samaan koteloon. (9.)

Mikro-ohjaimen muisti jakautuu käytön kannalta kolmeen muistityyppiin: ohjelmamuistiin (*engl. program memory*), käyttömuistiin (*engl. data memory*) ja haihtumat-

tomaan käyttömuistiin (*engl. non-volatile data memory*). Ohjelmamuisti on haihtumatonta muistia, jolloin muistin sisältö säilyy ohjelmoinnin jälkeen muuttumattomana ilman käyttöjännitettä. Ohjelmamuistiin on nimensä mukaan tallennettu mikro-ohjaimen ohjelma, eli suoritettavat käskyt. Ohjelmamuistina käytetään yleisesti Flash-muistia, joka voidaan ohjelmoida useita kertoja uudelleen. (9.)

Käyttömuistia käytetään ohjelman suorituksessa tarvittavien muuttujien arvojen tallentamiseen. Käyttömuisti on ns. haihtuvaa muistia, jolloin sen sisältämä tieto häviää käyttöjännitteen katketessa. Yleisesti mikro-ohjaimissa on käyttömuistina SRAM-tyyppistä muistia. Ohjelmamuisti ja käyttömuisti ovat mikro-ohjaimen toiminnan kannalta välttämättömiä. Mikro-ohjain sisältää kuitenkin usein myös haihtumatonta käyttömuistia. Haihtumatonta käyttömuistia tarvitaan, kun halutaan käyttöjännitteen katkettuakin säilyttää joitakin aseteltuja tietoja tai tiedonkäsittelyssä syntyneitä tuloksia. Haihtumattoman käyttömuistia ei voida käyttää SRAM-muistin tavoin nopeaan väliaikaiseen tallentamiseen, koska tiedon kirjoitus muistiin saattaa kestää jopa 100 kertaa kauemmin kuin tiedon lukeminen. Haihtumaton käyttömuisti on mikro-ohjaimissa tyypillisesti EEPROM-muistia. (9.)

Mikro-ohjaimen sisältämä mikroprosessori suorittaa kaikki piirin tiedonkäsittelyt ja hoitaa tiedonsiirron muistin ja muiden oheispiirien välillä. Mikroprosessori sisältää ohjausyksikön, aritmeettis-loogisen yksikön sekä sisäiset rekisterit, jotka ovat keskenään yhteydessä prosessorin sisäisellä tietoväylällä. Ohjausyksikössä suoritetaan ohjelmamuistin sisältämien käskyjen käsittely ja tuotetaan ohjaussignaalit prosessorin muille lohkoille ja ulkoisille väylille. Mikro-ohjaimen tiedon käsittelyn suorittaa aritmeettis-looginen yksikkö, jossa tehdään kaikki piirin matemaattiset laskutoimitukset ja lukujen vertailut. Mikroprosessorin sisäiset rekisterit ovat muistipaikkoja, joiden kautta kaikki prosessorin laskutoimitukset tehdään. Rekisterit myös säilyttävät tilapäistä tietoa tehdyistä operaatioista. (9.)

Toimiakseen mikroprosessori tarvitsee kellosignaalin sekä nollaus- keskeytystulot. Kellosignaalin tehtävän on tahdittaa mikroprosessoria ja tämän johdosta koko mikro-ohjainta. Mikro-ohjaimessa kellosignaali luodaan kello-oskillaattorilla. Kello-oskillaattorin taajuus voidaan määrätä ulkoisesti liitettävällä kiteellä. Mikroprosessori tarvitsee nollaustulon mikro-ohjaimen hallittuun käynnistykseen. Kun nollaustulo käy aktiivisessa tilassa prosessori pysäytetään ja sen sisäiset rekisterit asetetaan alkuarvoihin. Nollauksen palautuessa lepotilaan prosessori käynnistyy ja aloittaa ohjelma-

muistissa olevan ohjelman suorituksen. Mikroprosessorin keskeytystuloa käytetään esimerkiksi tulosignaalin muutoksen havaitsemiseen. Keskeytyksessä ohjelma hyppää keskeytyspalveluohjelmaan, jossa suoritetaan halutut toimenpiteet, ja palataan takaisin siihen kohtaan ohjelmassa, jossa keskeytys tapahtui. (9.)

Mikroprosessorit voidaan jakaa käskykantansa ja toimintansa mukaan RISC- ja CISC-prosessoreihin. Karkeasti eritellen RISC-prosessori suorittaa lyhyitä ja yksinkertaisia käskyjä, kun taas CISC-prosessori suorittaa pitkiä ja monimutkaisia käskyjä. Nykyisissä mikro-ohjaimissa käytetään pääsääntöisesti RISC-tyyppistä prosessoria. (9.)

Mikro-ohjaimen mikroprosessoria ja muisteja yhdistää kolme väylää: tietoväylä (*engl. data bus*), osoiteväylä (*engl. address bus*) ja ohjausväylä (*engl. control bus*). Väylällä tarkoitetaan samaan toiminnalliseen ryhmään kuuluvia rinnakkaisia johtimia. Tietoväylää eli dataväylää käytetään tiedon siirtämiseen prosessorin ja muistien välillä. Väylä on kaksisuuntainen ja sen leveys on tavallisesti 8, 16, 32 tai 64 bittiä. Kun mikro-ohjaimen dataväylänä on 8 rinnakkaista johdinta, sanotaan mikro-ohjaimen olevan 8-bittinen. Tämä tarkoittaa myös sitä että tietoa käsitellään 8-bittisillä rekistereillä. Osoiteväylällä osoitetaan muistipaikka, johon tiedonsiirto kohdistuu. Osoiteväylä on yksisuuntainen, ja sen leveys määrää osoitettavien muistipaikkojen määrän. Mikro-ohjaimen ohjausväylä ei ole dataväylän ja osoiteväylän mukainen väylä, vaan koostuu erilaisista ohjaavista johtimista, joilla ohjataan esimerkiksi muistin luku- ja kirjoitus. (9.)

Mikro-ohjaimen mikroprosessorin ja muistien liittämiseen käytetään kahta eri menetelmää. Liittämiseen käytetään joko von Neuman- tai Harvard-arkkitehtuuria. Arkkitehtuuria jossa ohjelma- ja käyttömuistilla on yhteiset väylät kutsutaan von Neuman-arkkitehtuuriksi. Ohjelma- ja käyttömuistilla ollessa erilliset tieto-, osoite- ja ohjausväylät on kyseessä Harvard-arkkitehtuuri. Mikro-ohjaimissa käytetään nykyään yleisesti Harvard-arkkitehtuuria. (9.)

Mikro-ohjaimen sisältämät liitäntäpiirit yhdistävät mikro-ohjaimen ulkoiseen elektroniikkaan. Ulkoinen elektroniikka voi olla mm. loistediodeja, kytkimiä, transistoreja ja releitä. Liitäntäpiirit voidaan jakaa tulo- ja lähtöpiiriin, jonka johdosta niitä kutsutaankin I/O-liitännöiksi. Liitännät ovat pääsääntöisesti kaksisuuntaisia eli jokainen piirin I/O-liitäntä voi toimia joko tulona tai lähtönä. Liitännällä voi olla myös aseteltavia erityistoimintoja, kuten esimerkiksi toiminta keskeytystulona. Mikro-ohjaimen sisältää

lisäksi integroituja eli piiriin sisäisesti liitettyjä oheispiirejä. Tyypillisiä mikro-ohjaimeen liitettyjä yksiköitä ovat erilaiset sarjaliikennepiirit, ajastin-laskurit, A/D-muunnin, analoginen jännitevertailija ja reaaliaikakello. Ohjauksen kannalta liitäntäpiirit näkyvät mikroprosessorille vain muutamina I/O-rekisterien muistiosoitteina. Näiden I/O-rekisterien arvot määräävät eri liitäntöjen ja yksiköiden käyttötarkoituksen ja toiminnan. (9.)

Mikro-ohjaimen ohjelmointi voidaan toteutettu laitteesta riippuen joko yleiskäyttöisellä tai valmistajakohtaisella ohjelmointilaitteella. Ohjelmointilaitteen käyttöliittymänä toimii PC-tietokoneen ohjelma ja ohjelmointiliitäntänä sarja-, rinnakkais- tai USB-portti. Osa nykyisistä mikro-ohjaimista ei tarvitse mitään erillistä ohjelmointilaitetta vaan pelkästään piirille soveltuvan ohjelmointijohdon PC:n ja mikro-ohjaimen liittämiseen. (9.)

4.2 Atmel AVR -mikro-ohjaimet

Atmel AVR -mikro-ohjaimella on tyypilliset tarkoitettu 8-bittisen AVR-mikro-ohjainperheeseen kuuluvia mikro-ohjaimia. Mikro-ohjainten ominaisuudet ovat kuitenkin kasvaneet valtavasti, ja nykyään Atmel valmistaa myös 16- ja 32-bittisiä AVR-mikro-ohjaimia. Atmelin valmistamat 8-bittiset AVR-mikro-ohjaimet jaetaan kolmeen eri sarjaan:

- Classic AVR
- tinyAVR
- megaAVR

Ensimmäisenä valmistetusta AVR-ohjainsarjasta käytetään nimitystä Classic AVR tai vain AVR-sarja. Classic AVR-sarjan mikro-ohjaimet ovat merkinnältään AT90S-alkuisia, kuten esimerkiksi sarjaan kuuluva AT90S2313. Sarjan valmistus on nykyään lopetettu, ja sarjaan kuuluneet piirit ovat nykyään korvautuneet vastaavien ominaisuuksien omaavilla tinyAVR- ja megaAVR-ohjaimilla. Esimerkkinä Classic AVR -sarjan ohjain AT90S2313 on korvattu tinyAVR-sarjan ATtiny2313-ohjaimella, joka on uudistettu versio aiemmasta ohjaimesta. Vastaavanlainen päivitys on tehty ATmega8515-ohjaimen korvatessa AT90S8515-ohjaimen. Nykyään valmistettavia sarjoja verratessa tinyAVR-sarja on ominaisuuksiensa puolesta karsittu versio megaAVR-sarjasta, jonka ohjaimet sisältävät laajan valikoiman eri ominaisuuksia. Suoritusno-

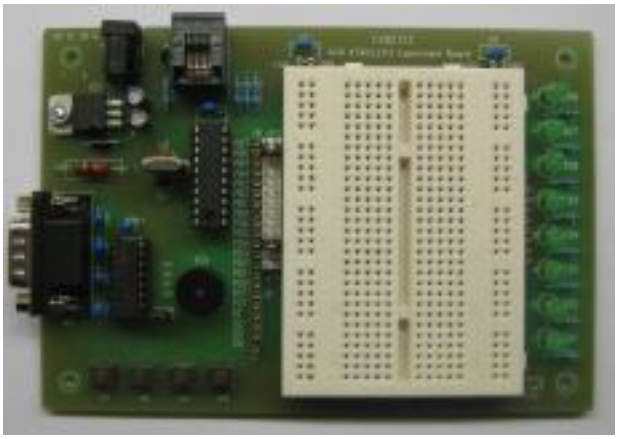
peuden perusteella piirejä ei voida sarjojen kesken jakaa, sillä kaikki AVR-mikro-ohjaimet sisältävät samanlaisen RISC-arkkitehtuurisen prosessoriytimen ja muistirakenteen. Prosessorirakenteen ollessa sama mahdollistuu vaivaton ohjelmiston siirto eri mikro-ohjainten välillä. Ohjelmaa siirrettäessä tulee kuitenkin huomioida ohjelmissa käytettyjen yksiköiden ohjausrekisterien yhteensopivuudet. (9.)

AVR-mikro-ohjaimet eroavat tyypillisesti toisistaan muistikapasiteetissa, piirin liitäntöjen lukumäärissä sekä eri ohjainten erikoisominaisuuksissa. Kaikissa mikro-ohjaimissa on kuitenkin vähintään yksi 8-bittinen ajastin-laskuri-yksikkö ja analoginen jännitevertailija. Paljon eri ominaisuuksia sisältävät megaAVR-ohjaimet sisältävät useita ajastin-laskuri-yksiköitä, erilaisia sarjaliitäntöjä sekä A/D-muuntimen. (9.)

AVR-mikro-ohjaimissa käytetään muistiarkkitehtuurina Harvard-arkkitehtuuria, jolloin mikro-ohjaimen ohjelma- ja käyttömuistilla on omat väylänsä. Ohjelmamuistin dataväylän leveys on 16-bittiä ja käyttömuistin 8-bittiä. Ohjelmamuisti on tyypiltään Flash-muistia ja käyttömuisti SRAM-muistia. Ohjelmamuistin koko vaihtelee tiny-AVR-sarjan puolesta kilotavusta aina megaAVR-sarjan 256 kilotavuun. Mikro-ohjaimien I/O-liitäntöjen lukumäärä vaihtelee vastaavasti välillä 4 - 86. Nämä I/O-liitännät ovat ohjaimissa jaettuna 8-bittisiksi rinnakkaisporteiksi, jotka on nimetty PORTA, PORTB, PORTC, PORTD jne. Kaikkien AVR-mikro-ohjaimet soveltuvat hyvin C-kielellä ohjelmoitaviksi ja ne on suunniteltukin niin, että C-kielestä konekieleen käännetty ohjelma saataisiin toteutettua mahdollisimman tehokkaasti. Atmelin AVR-mikro-ohjaimet ovat helpoiten ohjelmoitavissa mikro-ohjainten sisältämän SPI-liitännän avulla, jolloin tarvitaan vain ohjelmointijohto mikro-ohjaimen ja tietokoneen välille. SPI-liitännän kautta tapahtuvaan ohjelmointiin on myös saatavilla erillisiä ohjelmointilaitteita. (9.)

4.3 EXB2313-alusta

Kuvassa 7 esitetty EXB2313-alusta on Tietomyrsky Oy:n valmistama mikro-ohjainkortti Atmelin AT90S2313- ja ATtiny2313-mikro-ohjaimille. Alusta soveltuu hyvin mikro-ohjainten opiskeluun sekä erilaisten sovelluksien testaukseen. Virtalähteeksi alustalle soveltuu 12 V:n tasajännitettä syöttävä muuntaja. Alustalla tehtävään mikro-ohjaimen ohjelmointiin soveltuu Tietomyrsky Oy:n valmistama STK200/300-tyyppinen aktiivinen ohjelmointikaapeli.



Kuva 7. *EXB2313-mikro-ohjainkortti*

EXB2313-alusta sisältää seuraavat käyttöominaisuudet:

- syöttöjännite 8…18 V DC
- ohjelmointiliitäntä, RJ11-liitin
- RS-232-sarjaliikennepiiri
- 4 painonappia
- 8 kpl jumppereilla irtikytkettävää LEDiä
- piezo-summeri
- 14 x 29 pisteen kytkentäalusta
- ohjaimen portit vapaasti kytkettävissä

ATtiny2313

ATtiny2313 on Atmelin tinyAVR-mikro-ohjainsarjaan kuuluva 8-bittinen mikro-ohjain, joka tarjoaa tehokkaan ja joustavan ratkaisun erilaisiin sulautettuihin ohjausso-velluksiin. Toimiakseen ATtiny2313 ei tarvitse mitään ulkoisia komponentteja, koska se sisältää kello-oskillaattorin sekä nollauspiirin sisäänrakennettuina.

ATtiny2313-mikro-ohjaimen keskeiset ominaisuudet:

- ohjelmamuisti (Flash) 2 KB
- käyttömuisti (SRAM) 128 B
- EEPROM-muisti 128 B
- I/O-liitännät 18 kpl
- 8-bittinen ajastin-laskuri
- 16-bittinen ajastin-laskuri
- 4 PWM-lähtöä
- analoginen vertailija
- USI-liitäntä
- USART-yksikkö
- vahtiajastin
- ulkoiset ja sisäiset keskeytyslähteet
- sisäinen RC-oskillaattori (10.)

4.4 PROJ32/644-alusta

Kuvassa 8 esitetty PROJ32/644-alusta on Tietopetri Oy:n valmistama Atmelin ATmega32- ja ATmega644-mikro-ohjaimille soveltuva mikro-ohjainkortti, piirien testaukseen ja ohjelmointiin. Käytettävät piirit voidaan helposti ohjelmoida Atmelin AVRISP mkII-ohjelmointilaitteella piirien ollessa kiinni alustalla.

Kuva 8. *PROJ34/644-mikro-ohjainkortti*

PROJ32/644-alusta on varustettu seuraavilla ominaisuuksilla:

- ohjelmointiliitäntä, IDC-liitin
- sarjaliitäntä
- käyttöjänniteliitäntä 9 ... 18 V (AC tai DC)
- mikro-ohjaimen portit vapaasti käytettävissä
- mitat: 133 mm x 83 mm, mitoitettu koteloon TB-1.7 tai TB-2.9

ATmega32

Piirivalmistaja Atmelin megaAVR-mikro-ohjainsarjaan kuuluva ATmega32 on tehokas mikro-ohjain, joka tarjoaa kustannustehokkaan ratkaisun moniin sulautettuihin ohjaussovelluksiin. ATmega32-mikro-ohjaimen koteloina käytetään 40-nastaista PDIP-koteloa sekä 44-nastaisia TQFP- ja QFN/MFN-koteloita. Ohjaimen liitäntänastoista jää 32-nastaa yleiskäyttöisiksi I/O-liitännöiksi, jotka muodostavat neljä 8-bittistä rinnakkaisporttia: PORTA, PORTB, PORTC ja PORTD.

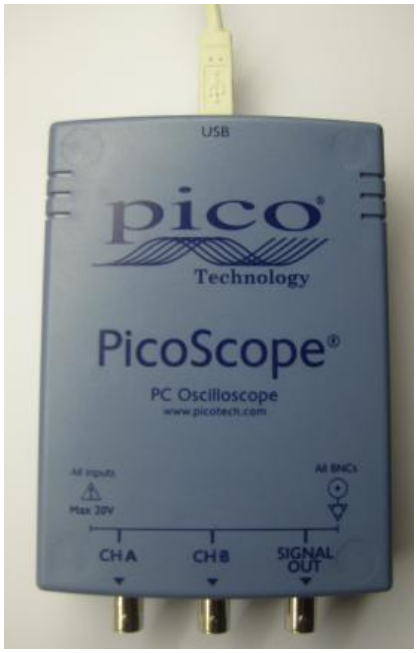
ATmega32-mikro-ohjaimen keskeiset ominaisuudet:

- ohjelmamuisti (Flash) 32 KB
- käyttömuisti (SRAM) 2 KB
- EEPROM-muisti 1024 B
- I/O-liitännät 32 kpl
- 8-bittinen ajastin-laskuri 2 kpl
- 16-bittinen ajastin-laskuri
- reaaliaikakello

- 4 PWM-lähtöä
- analoginen vertailija
- 10-bittinen A/D-muunnin, 8 kanavaa
- USART-yksikkö
- vahtiajastin
- JTAG-liitäntä
- SPI-liitäntä
- two-wire-liitäntä
- ulkoiset ja sisäiset keskeytyslähteet
- sisäinen RC-oskillaattori (11.)

4.5 PicoScope 2205-PC-oskilloskooppi

PicoScope 2205-PC-oskilloskooppi on vuonna 1991 perustetun Pico Technology -nimisen yrityksen valmistama PC:ltä ohjattava oskilloskooppi. Kuvassa 9 esitetty PicoScope 2205 kuuluu 2-kanavaiseen PicoScope 2200-sarjaan sisältäen oskilloskooppitoiminnon lisäksi spektrianalysaattori-toiminnon sekä vapaan aaltomuodon generaattorin. Oskilloskoopin ohjaus ja mittaustulosten keruu tapahtuu PicoScope-PC-ohjelmistolla, USB-väylän välityksellä. Ohjelmiston avulla määritetään muun muassa mittauksessa käytettävä näytteenottonopeus sekä mittauksen liipaisuasetukset. Asetusten mukaiset oskilloskoopin mittaukset tallennetaan muistiin ja ne piirtyvät sovellusikkunaan tarkasteltavaksi. Picoscope 2205-oskilloskoopin sisältäessä kehittyneet mittausten liipaisuominaisuudet ja mittaustulosten jatkokäsittelyn se soveltuu hyvin digitaalisten signaalien mittaukseen ja tarkasteluun. Oskilloskoopin näytteenottotaajuus on maksimissaan 200 MS/s yhtä kanavaa käytettäessä ja 100 MS/s kahden kanavan ollessa käytössä. Analoginen kaistanleveys on 25 MHz ja jännitemittausalue –20…20 V, sisältäen ±100 V:n ylijännitesuojauksen. (12.)



Kuva 9. *PicoScope 2205-PC-oskilloskooppi*

5 TOTEUTUKSEN OHJELMISTOT

Työssä tehtyihin Atmelin mikro-ohjaimien ohjelmointiin käytettiin WinAVR- ja Atmel AVR Studio -ohjelmistoja. WinAVR on Atmelin mikro-ohjaimien ohjelmointiin ja testaukseen tarkoitettujen avoimen lähdekoodin sisältävien ohjelmistojen kokoelma. Sen keskeisimpinä ohjelmistoina ovat C-kielinen AVR-GCC-käännin ohjelmakoodin kääntämiseen, sekä AVRDUDE-latausohjelma mikro-ohjaimen ohjelmointiin. Toteutuksessa käytettiin WinAVR-pakettiversiota 20081205, joka on ladattavissa WinAVR-projektin internet-sivulta. (13.)

Toisena ohjelmointityökaluna käytetty Atmel AVR Studio on Atmelin internetin välityksellä vapaasti jakama kehitysympäristö Atmelin mikro-ohjaimille. Se sisältää editorin, assembler-kääntäjän, simulaattorin ja käyttöliittymät eri ohjelmointi- ja emulaattorilaitteille. AVR Studio ei sisällä C-kielen käännintä, mutta se linkittyy automaattisesti käyttämään tietokoneeseen asennettua WinAVR-paketin sisältämää AVR-GCC-C-käännintä. (14.)

Toteutuksen testaukseen ja toiminnan todennukseen käytettiin aiemmin esitellyn PicoScope 2205-oskilloskoopin PicoScope 6-ohjelmistoa. Lisäksi lukijalla vastaanotetut tiedot tarkastettiin PC:n ja mikro-ohjainalustan välisellä sarjaliikenne-yhteydellä, johon käytettiin Br@y++ Terminal 1.9b-pääteohjelmaa.

6 ALUSTAVAN RFID-LUKIJAN OHJAUKSEN TOTEUTUS

Alustava RFID-lukijan ohjaus toteutettiin tunnisteen lukuproseduurin simuloinnilla. Lukijan käyttämän EPCglobal Class-1 Gen2 -standardin mukainen lukusimulointi toteutettiin Atmelin ATtiny2313- ja ATmega32-mikro-ohjaimilla. Mikro-ohjaimista ATtiny2313 ohjelmoitiin mallintamaan tunnistetta ja ATmega32 lukijaa. Ohjauksessa tehtiin standardin määrittämä lukuproseduuri, joka etenee seuraavien toimenpiteiden mukaisesti:

- Lukija ohjataan lähettämään kantoaaltoa 1,5 ms:n ajan.
- Lukija lähettää PIE-koodatun kysely- eli Query-komennon, jota edeltää Preamble-tahdistusjakso.
- Tunniste vastaanottaa Query-komennon ja lähettää 16-bittisen satunnaisluvun FM0- tai Miller-koodattuna.

- Lukija vastaanottaa satunnaisluvun ja lähettää PIE-koodatun ACK-komennon, jota edeltää Frame-Sync-tahdistus. ACK-komennossa lähetetään takaisin tunnisteen lähettämä satunnaisluku.
- Tunniste vastaanottaa ACK-komennon ja vertaa vastaanotettua satunnaislukua lähetettyyn. Jos luvut ovat samat se lähettää sisältämänsä tunnistetiedot FM0- tai Miller-koodattuna.
- Lukija ohjataan aina komennon jälkeen lähettämään tunnisteelle kantoaaltoa.

RFID-lukijan ohjauksen toteutus aloitettiin valitsemalla lukijan PIE-koodauksessa käytettävä Tari-arvo ja data-1-symbolin aika. Lukijan PIE-koodauksessa käytettäväksi Tari-arvoksi valittiin 25 µs ja data-1-symbolille kaksinkertainen aika 50 µs. Näiden symbolien yhteispituuden ilmoittavalle RTcal-parametrille määräytyi arvoksi 75 µs. Koska toteutuksessa pyrittiin mahdollisimman varmaan toimivuuteen käytetyllä laitteistolla, valittiin linkkitaajuudeksi 40 kHz, joka on käytettävistä taajuuksista pienin. Valitusta linkkitaajuudesta laskettiin lukijalta lähetettävän TRcal-symbolin aika kaavan 1 avulla. Standardi määrittää 40 kHz:n taajuudelle jakosuhteen 8, jolloin TRcal-symbolin ajaksi saatiin 200 µs. Tunnisteen datan koodaukseksi toteutukseen valittiin FM0-koodaus. Koodauksen valinta määräsi myös tunnisteen datanopeudeksi 40 kbps. Toteutukseen valitut symboliajat ovat pulssin ja pulssivälin yhteisaikoja. Jokaista pulssia seuraa pulssiväli PW, joka on puolet Tari-arvosta eli tässä toteutuksessa 12,5 µs. Taulukossa 1 on kuvattu ohjauksessa käytetyt parametriarvot ja niille määritetyt toleranssit.

Taulukko 1. *Toteutuksen parametriarvot. Taulukossa on ensin esitetty toteutukseen käytetyt PIE-koodauksen symboliajat, joille määritetään standardissa ±1 % toleranssi. FM0-koodauksesta esitetty valittu linkkitaajuus, jonka jaksonaika on FM0-koodauksen symbolien pituus. Linkkitaajuuden vaihteluksi standardi määrittää ±2.5 %.*

Koodaus	Parametri	Kestoaika	Pulssin pituus	Toleranssi
PIE	data-0 / Tari	25 µs	12,5 µs	±0,25 µs
	data-1	50 µs	37,5 µs	±0,5 µs
	RTcal	75 µs	62,5 µs	±0,75 µs
	TRcal	200 µs	187,5 µs	±2 µs
FM0	BLF	40 kHz / 25 µs		± 1 kHz

Lukijan ja tunnisteen välisen yhteyden liikennöintimääritysten jälkeen määritettiin käytettävän Query-komennon rakenne. Taulukossa 2 on esitetty toteutuksessa käytetty

Query-komento, standardin esitysmuodon mukaisesti. Taulukon Bittien lkm-otsikoitu rivi kuvaa komennon eri kenttien sisältämien bittien lukumäärän. Kuvaus-rivi kuvaa kullekin kentälle valittavien bittien arvoja ja niiden merkitystä. Toteutus-rivi esittää simuloinnissa käytetyn Query-komennon bitit. Query-komento aloitetaan aina Command-kentän biteillä, jotka toimivat komennon tunnisteena. Touteutuksen Query-komennon DR-kentän arvon määräsi valittu 40 kHz:n linkkitaajuus ja vastaavasti M-kentän arvon määräsi FM0-koodauksen käyttö. TRext-kentällä valittiin normaali tunnisteen alkutahdistus, jolloin tunniste käyttää lähetteissään kuvan 4 mukaista alkutahdistusta. Sel-kentällä valittiin tunnistus kaikille tunnisteille, jolloin jätettiin huomioimatta tunnisteen SL-valintalippu. Session-kentästä valittiin tunnisteen koodin lukemiseen käytettäväksi S0-istuntoa ja Target-kentälle valitun istuntotilan tunnistuslippu-arvoksi A. Tällä valinnalla vain tunnisteet, joiden S0-istuntotilan tunnistulippu-arvo on A huomioivat Query-komennon. Valittu Q-kentän arvo antaa tunnisteelle lähetysvuoronumeroksi nollan ja määrää tunnisteen välittömästi vastaamaan komentoon. CRC-5-kentässä oleva tarkistusluku on laskettu liitteessä 1 esitetyllä tavalla.

Taulukko 2. *Toteutuksen Query-komento. Taulukossa esitetään 22-bittiä sisältävä Query-komento, joka rakentuu 9 eri kentästä.Query-komentoa edeltää kuvan 2 mukainen Preamble-tahdistus.* (6.)

	Command	**DR**	**M**	**TRext**	**Sel**	**Session**	**Target**	**Q**	**CRC-5**
Bittien lkm	4	1	2	1	2	2	1	4	5
Kuvaus	1000	0:DR=8 1:DR=64/3	00:M=1 01:M=2 10:M=4 11:M=8	0:No pilot tone 1:Use pilot tone	00:All 01:All 10:~SL 11:SL	00:S0 01:S1 10:S2 11:S3	0:A 1:B	0-15	
Toteutus	1000	0	00	0	01	00	0	0000	01110

Query-komennon määrittelyn jälkeen tehtiin käytettävän ACK-komennon määritys. Komentoa varten tuli määrittää simuloinnissa tunnisteelta lähetettävä 16-bittinen satunnaisluku. Satunnaisluvuksi valittiin luku $ABCD_h$ = $1010\ 1011\ 1100\ 1101_2$ = 43981_{10}. Lukijan ACK-komento määräytyi tällöin taulukon 3 mukaiseksi.

Taulukko 3. *Toteutuksen ACK-komennon rakenne.Taulukossa esitetty 18-bittiä sisältävä ACK-komento Command-tunnistekentästä ja satunnaisluvun sisältävästä RN-kentästä. ACK-komentoa tulee edeltää kuvassa 2 esitetty Frame-Sync-tahdistus.* (6.)

	Command	**RN**
Bittien lkm	2	16
Kuvaus	01	Kaiutettava satunnaisluku
Toteutus	01	1010 1011 1100 1101 ($ABCD_h$)

Toteutuksen tunnisteelta lähetettäviksi tunnistetiedoiksi valittiin standardin tunnistetietojen esimerkkitaulukossa esitetyt tiedot. Taulukossa 4 esitetään standardin esimerkkitaulukko. Taulukossa on heksadesimaalilukuina esitettynä tunnisteen EPC-muistialueen esimerkkitiedot 0 - 96-bittiseen EPC-koodiin. Taulukkoon on lisäksi laskettu jokaisessa sarakkeessa olevien tunnistetietojen CRC-16-tarkistusluvut. EPC-koodin lukeminen testattiin jokaisella 7:llä eri EPC-muistialueen esimerkkitiedolla.

Taulukko 4. *Tunnistetietojen esimerkkitaulukko. Taulukko sisältää tunnisteen EPC-muistialueen esimerkkitiedot. Kullekin esimerkkitieto-sarakkeelle on laskettu CRC-16-CITT-tarkistussumma sarakkeen ylimmälle riville.* (6.)

EPC osoite	**EPC sisältö**	**EPC-arvot**						
00_h	CRC-16	$E2F0_h$	$CCAE_h$	$968F_h$	$78F6_h$	$C241_h$	$2A91_h$	1835_h
10_h	PC	0000_h	0800_h	1000_h	1800_h	2000_h	2800_h	3000_h
20_h	EPC 1	----	1111_h	1111_h	1111_h	1111_h	1111_h	1111_h
30_h	EPC 2	----	----	2222_h	2222_h	2222_h	2222_h	2222_h
40_h	EPC 3	----	----	----	3333_h	3333_h	3333_h	3333_h
50_h	EPC 4	----	**----**	----	----	4444_h	4444_h	4444_h
60_h	EPC 5	----	----	----	----	----	5555_h	5555_h
70_h	EPC 6	----	----	----	----	----	----	6666_h

Parametrien ja komentojen määrittelyn jälkeen suunniteltiin luvussa 3.1 kuvattujen PIE- ja FM0-koodauksen toteutusalgoritmit eli se kuinka koodaus ja dekoodaus kyseisille koodaustavoille tapahtuu ja miten se mikro-ohjaimilla toteutettaisiin. PIE-koodaus päätettiin toteuttaa asettamalla koodattavan tiedon mukaisesti 0- ja 1-tietosymboleille eri pituiset ajat, jolloin lähtö on ylätilassa koodattavan tietosymbolin mukaisen ajan ja vakioajan alatilassa. Tunnisteen tekemä PIE-koodauksen dekoodaus suunniteltiin toteutettavaksi mittaamalla tulevien pulssien pituuksia ja tulkitsemalla asetun raja-arvon alittavat pulssit data-0-symboleiksi ja ylittävät data-1-symboleiksi.

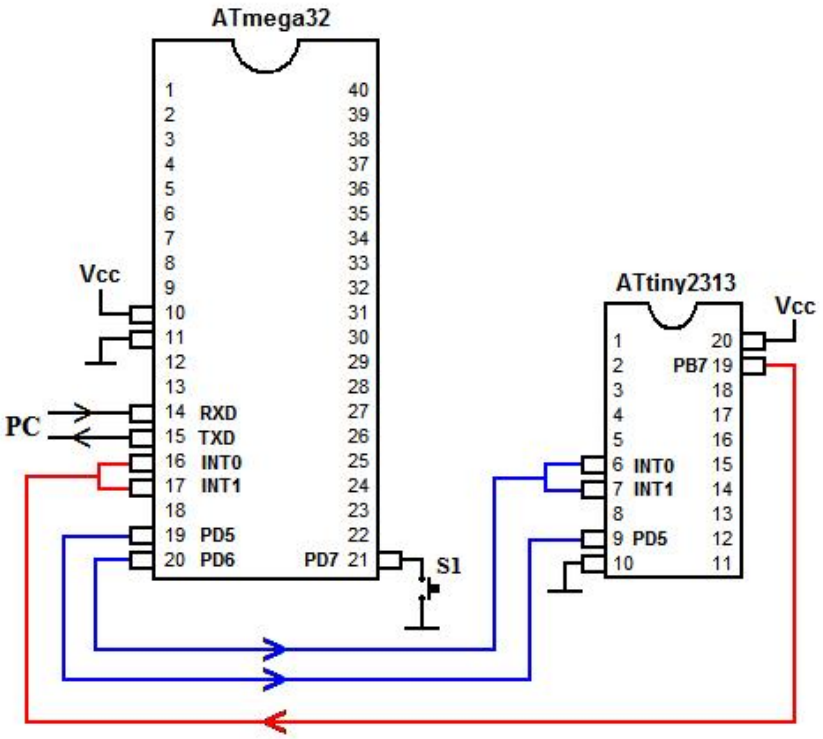
Tunnisteen sisältämän tiedon FM0-koodaus suunniteltiin tehtäväksi tunnistamalla koodattava tieto 0 tai 1 ja suorittaen tarvittavat lähdön tilan vaihdot. Lukijalla tehtävä FM0-dekoodaus suunniteltiin toteutettavaksi laskemalla ensin jokaisen tulossa tapahtuvan tilan muutoksen välinen aika, jonka jälkeen tehdään aikaväleille tarvittavat aikavertailut, joiden avulla saadaan tietosymbolit selvitettyä.

Koska työn tavoitteena oli lukijan standardinmukainen ohjaus, keskityttiin simuloinnissa lukijan oikeanlaisen toiminnan toteutukseen, mutta tunnisteen simuloinnissa jätettiin standardinmukainen tarkka sisäinen toiminta toteuttamatta. Simuloitu tunniste suunniteltiin ainoastaan varmistamaan koodauksen oikeellisuus ja tekemään lähettämilleen tiedoille standardin mukainen FM0-koodaus lukijan tulkittavaksi. Tunniste suunniteltiin todentamaan lukijan komennot oikeiksi vertaamalla niitä muistiin tallen-

nettuihin komentotietoihin. Simuloinnissa tunniste ei tällöin tee mitään päätelmiä komennon parametrien mukaan, vaan komennon ollessa oikean muotoinen lähetetään vastaus ohjelmoinnissa määrätyn nopeuden mukaan. Lukijalta tunnisteelle lähetettävää kantoaalto suunniteltiin mallinnettavaksi kytkemällä jännite erilliseen kantoaallon kytkevään nastaan standardin määrittelyjen mukaisesti. Mallinnettava kantoaalto suunniteltiin käytettäväksi tunnisteen vastaanoton käynnistämiseen. EPC-koodin lukusimulointi suunniteltiin käynnistettäväksi mikro-ohjainalustalla olevasta kytkimestä tai lähettämällä s-kirjain PC-tietokoneen sarjaportista lukijalle. Tunnisteelta luettava EPC-koodi suunniteltiin lähettäväksi lukijalta sarjaportin kautta PC-tietokoneelle.

Simuloinnin toteutuksen haasteeksi muodostui koodauksille sopivien vastaanottomenetelmien suunnitteleminen käytettyjen mikro-ohjaimien ominaisuuksista. Eri menetelmien suoritettujen testausten jälkeen käytetyksi toteutustavaksi valikoitui sekä lukijassa että tunnisteessa vastaanotettavan datan ohjaus mikro-ohjaimen kahteen ulkoiseen keskeytystuloon, merkitään INT0 ja INT1. Tällä ratkaisulla kyettiin ohjelmakeskeytyksiä hyväksikäyttäen havaitsemaan tulevan signaalin nouseva ja laskeva reuna. Tämä ominaisuus yhdistettynä mikro-ohjaimien ajastin-laskureiden käyttöön mahdollisti koodauksien tulkitsemisen.

Simuloinnissa käytetty mikro-ohjaimien periaatekytkentä on esitettynä kuvassa 10. Kuvassa esitetään mikro-ohjaimien simuloinnissa tarvittavat kytkennät. ATmega32-ohjaimelta lähetetään Query-komento vastaanottamalla s-kirjain nastan 14 sarjaporttivastaanotosta tai painettaessa PD7-merkittyyn D-portin 7.-liitäntään liitettyä painonappia. Lukijan PIE-koodattu data lähetetään ATmega32:n nastassa 20 olevasta D-portin 6.-liitännästä, merkitään PD6. Lukijan kantoaaltoa mallinnetaan piirin nastasta 19, joka on D-portin 5.-liitäntä, merkitään PD5. Tunnisteen lähettämä FM0-koodattu data vastaanotetaan nastoihin 16 ja 17, joita käytetään INT0- ja INT1-keskeytystuloina. Vastaanotetut tunnistetiedot lähetetään nastassa 15 olevasta sarjaporttilähdöstä. Tunnistetta simuloivan ATtiny2313:n nastasta 9 olevasta D-portin 5.-liitännästä todetaan lukijan kantoaalto. Lukijan PIE-koodattu data vastaanotetaan nastoista 6 ja 7, jotka toimivat INT0- ja INT1-keskeytystuloina. ATtiny2313:n nastassa 19 oleva PB7 merkitty B-portin 7.-liitäntä toimii FM0-koodatun tiedon lähetysnastana.



Kuva 10. *Toteutuksen mikro-ohjainten kytkentä. Kuvassa sininen johdotus kuvaa lukijaa simuloivalta ATmega32-mikro-ohjaimelta lähteviä johtimia ja punainen johdotus kuvaa tunnistetta simuloivalta ATtiny2313-mikro-ohjaimelta lähtevää johdinta. Viivojen nuolet kuvaavat tiedonsiirtosuuntia.*

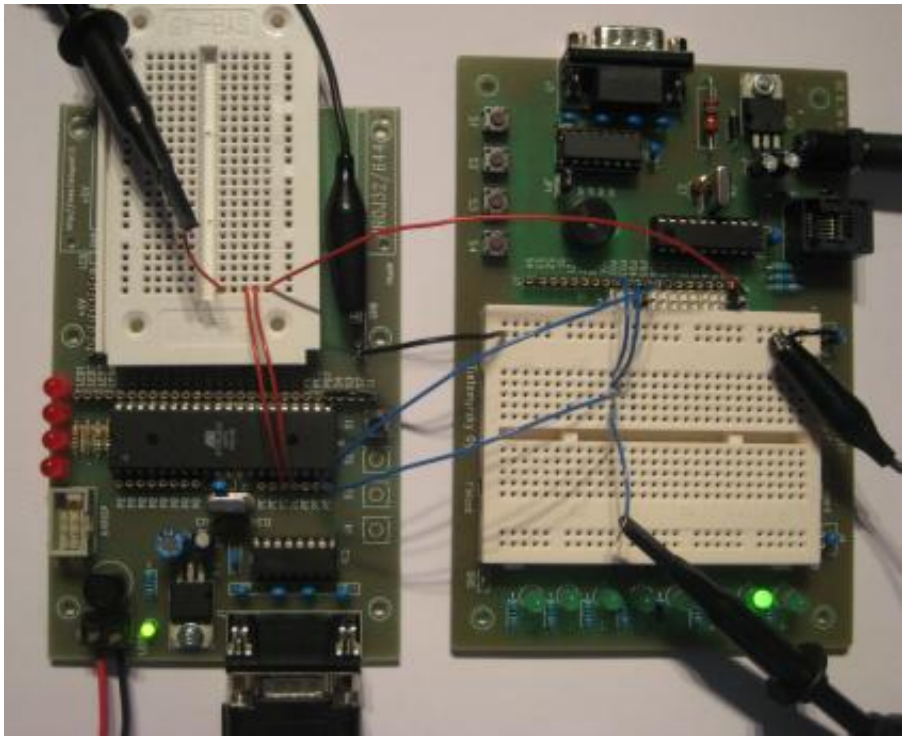
Ennen suunniteltujen algoritmien mukaisen signaloinnin toteutusta ohjelmoitiin ATtiny2313- ja ATmega32-mikro-ohjaimien toimintaa ohjaavat sulakebitit, jotta ohjaimet toimisivat niiden ulkoisten kiteiden määräämillä kellotaajuuksilla. ATtiny2313-ohjaimella käytettiin 12 MHz:n kidettä ja ATmega32-ohjaimessa 16 MHz:n kidettä. ATtiny2313-ohjaimen sulakebitit ohjelmoitiin WinAVR-paketin sisältämällä komentorivipohjaisella AVRDUDE-latausohjelmalla, jota käytettiin graafisen käyttöliittymän AVRDUDE GUI v0.2.0 avulla. Ohjelmointijohtona käytettiin STK200/300-tyyppistä ohjelmointijohtoa. PROJ32/644-alustalla käytetyn ATmega32-ohjaimen sulakebitit muutettiin käyttämällä AVR Studiota, sekä AVRISP mkII-ohjelmointilaitetta. Taulukossa 5 on esitetty käytetyt sulakebittiarvot heksadesimaalilukuina esitettyinä.

Taulukko 5. *Mikro-ohjaimien sulakebittien arvot. Taulukossa esitetyillä sulakebiteillä mikro-ohjaimet käyttävät kellotaajuuksinaan ulkoisten kiteiden määräämää taajuutta.*

Mikro-ohjain	High Fuse Byte	Low Fuse Byte
ATtiny2313	0xDF	0xFF
ATmega32	0x89	0xFF

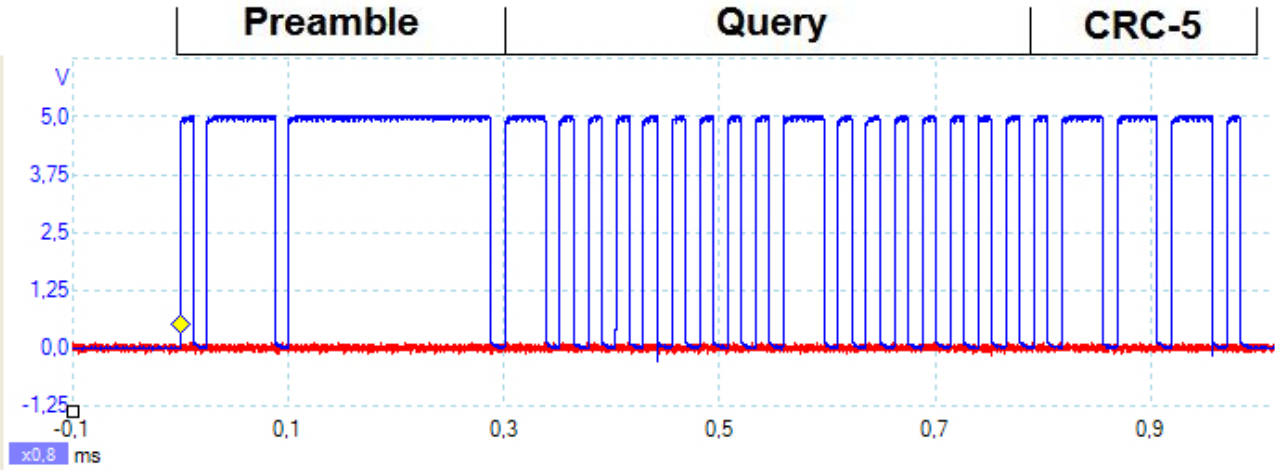
Mikro-ohjaimien ohjelmoinnit suoritettiin kuten sulakebittien ohjelmoinnissa ATtiny2313:lle WinAVR-paketin AVRDUDE-ohjelmalla ja ATmega32:lle AVR Studion kautta. Tunnistussimuloinnin testausmittaukset tehtiin PicoScope2205-PC-

oskilloskoopilla, PicoScope 6-mittausohjelman ohjaamana. Lisäksi ATmega32-mikro-ohjaimen ja PC-tietokoneen välillä käytettiin 9600 kbps:n nopeudella toimivaa sarjaliitäntää, jonka kautta lähetettiin vastaanotetut tunnistetiedot PC-tietokoneelle. Oskilloskooppimittauksissa ATmega32-ohjaimen nastasta 20 lähetetty lukijan signalointi mitattiin oskilloskoopin A-kanavasta. Oskilloskoopin B-kanavasta mitattiin ATtiny2313-ohjaimen nastasta 19 lähettämä signalointi. Kuvassa 11 on esitetty mikro-ohjainalustoilla toteutettu simuloinnin testauskytkentä. Kytkennässä on käytetty alustojen lisäksi pientä koekytkentäalustaa kytkentöjen tekemiseksi. Testauskytkennän johdotus ja johdinvärit vastaavat kuvan 10 periaatekytkentää.



Kuva 11. *Simuloinnin testauskytkentä. Kuvassa esitetään simulointikytkentä mittauspisteineen. Vasemmalla lukijaa mallintava ATmega32-mikro-ohjain PROJ32/644-alustalla. Oikealla tunnistetta mallintava ATtiny2313-mikro-ohjain EXB2313-alustalla.*

Mittauksissa Picoscope 2205-PC-oskilloskooppi asetettiin käyttämään suurinta mahdollista näytteenottonopeutta. Mittauksista mitattiin ensimmäisenä kuvassa 12 esitetty lukijan lähettämä PIE-koodattu Query-komento. Query-komennosta mitattiin lähetettävien datasymbolien pituudet. Aluksi symbolien ajoissa havaittiin ±0,3 µs:n vaihteluja ja noin 1 µs:n eroa taulukossa 1 esitettyihin ohjearvoihin. Mitattuihin aikoihin tehtiin ohjelmassa tarvittavat viiveaikojen korjaukset, jolloin symbolien ajat olivat taulukossa 1 esitetyissä toleransseissa. Taulukossa 6 on esitetty mitatut PIE-koodauksen symboliajat.

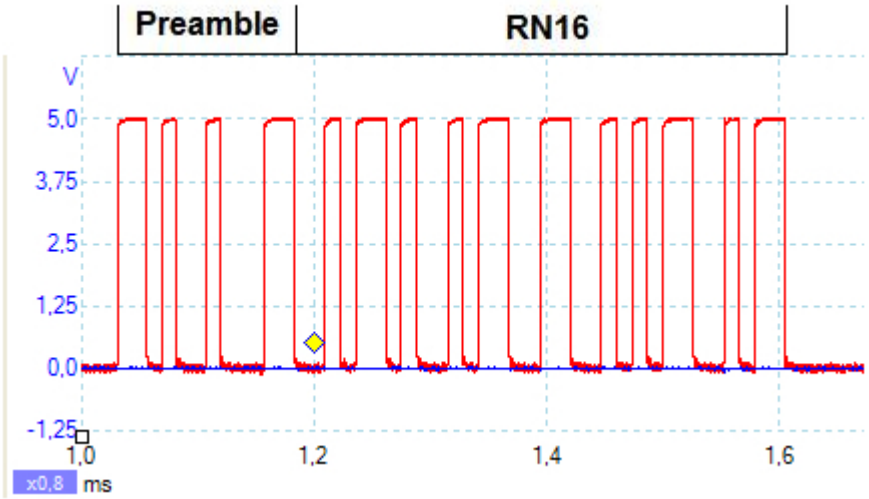


Kuva 12. *Lukijan PIE-koodattu Query-komento. Kuvassa taulukossa 2 esitetyn Query-komennon PIE-koodaus Preamble-tahdistuksella. Kuvan mittauksen aika-akselin sarakevälinä on 200 µs/sarake.*

Taulukko 6. *Mitatut PIE-koodauksen symboliajat*

Symboli	Mitatut ajat (µs)
Data-0	25,12
RTcal	75,1
TRcal	200,0
Data-1	50,25

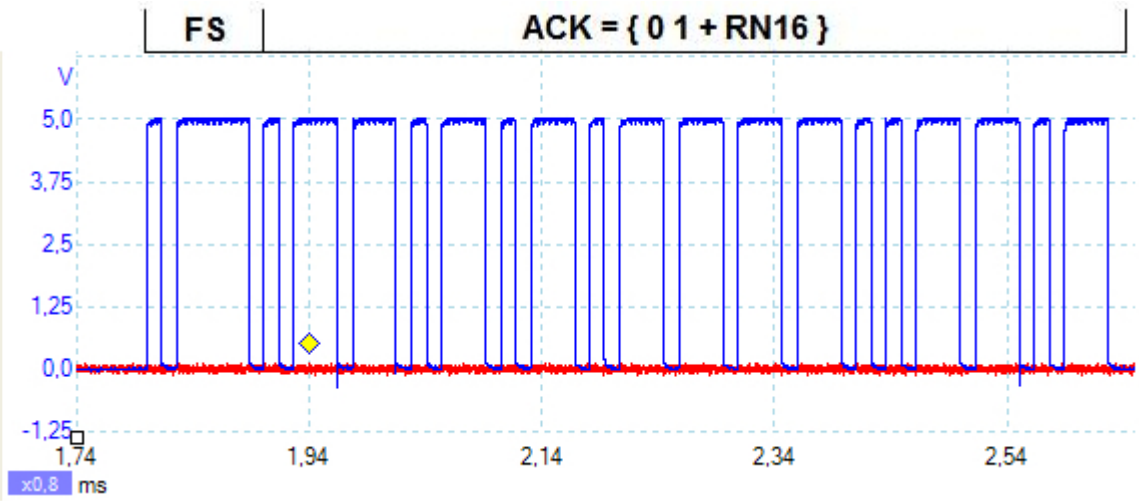
Peräkkäiset signaloinnit saatiin mitattua käyttämällä Picoscope 2205:n viivästystoimintoa. Kuvassa 13 on esitettynä tunnistetta mallintavan ATtiny2313-ohjaimen lähettämä FM0-koodattu 16-bittinen satunnaislukuvastaus, jota edeltää FM0 Preamble-tahdistus. Kuvan 13 mukainen mittauskuva on saatu viivästämällä näytteistystä 1,2 ms:a Query-komennosta.



Kuva 13. *Tunnisteen FM0-koodattu satunnaislukuvastaus. Kuvassa mitattu FM0-koodattu satunnaislukuna käytetty $ABCD_h$ jota edeltää kuvan 4 mukainen FM0-preamble. Kuvan mittauksen aika-akselin sarakevälinä on 200 µs/sarake.*

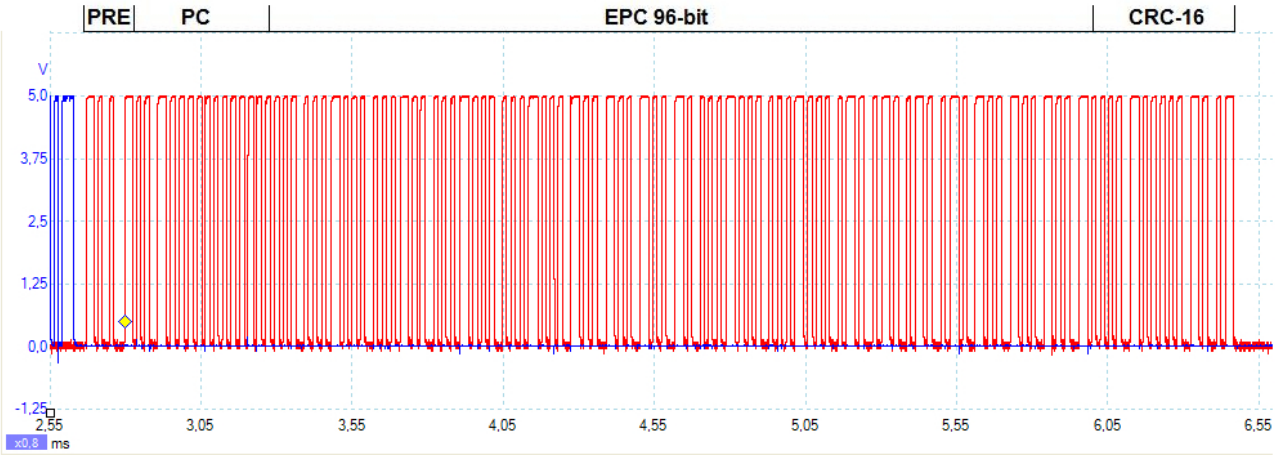
Kuvassa 14 on esitetty lukijan lähettämä ACK-komento, jossa lähetetään PIE-

koodattuna tunnisteen lähettämä RN16-satunnaisluku. Mittauskuvan signalointi on saatu viivästämällä näytteistystä 1,94 ms:a Query-komennosta.



Kuva 14. *Lukijan PIE-koodattu ACK-komento. Kuvassa taulukon 3 mukainen ACK-komento PIE-koodauttuna. Komentoa edeltää kuvan 2 mukainen Frame-Sync-tahdistus. Kuvan mittauksen aika-akselin sarakevälinä on 200 µs/sarake..*

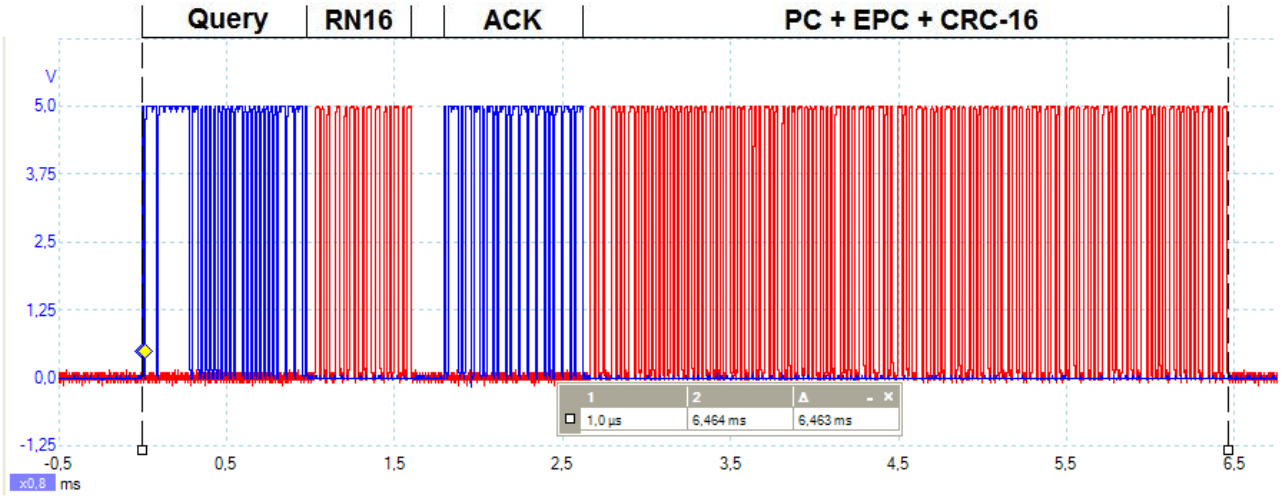
Mitatut tunnisteen FM0-koodatut tunnistetiedot on esitetty kuvassa 15. Jotta koodatut tunnistetiedot saatiin näkymää PicoScope 6-sovelluksen ruudulla kokonaisuudessaan käytettiin aika-akselin sarakevälinä 500 µs/sarake ja viivästettin näytteistystä 2,8 ms:lla. Tunnistetietojen lähetys sisältää PC-bitit, EPC-tunnistekoodin ja näiden yli lasketun CRC-16-tarkistusluvun.



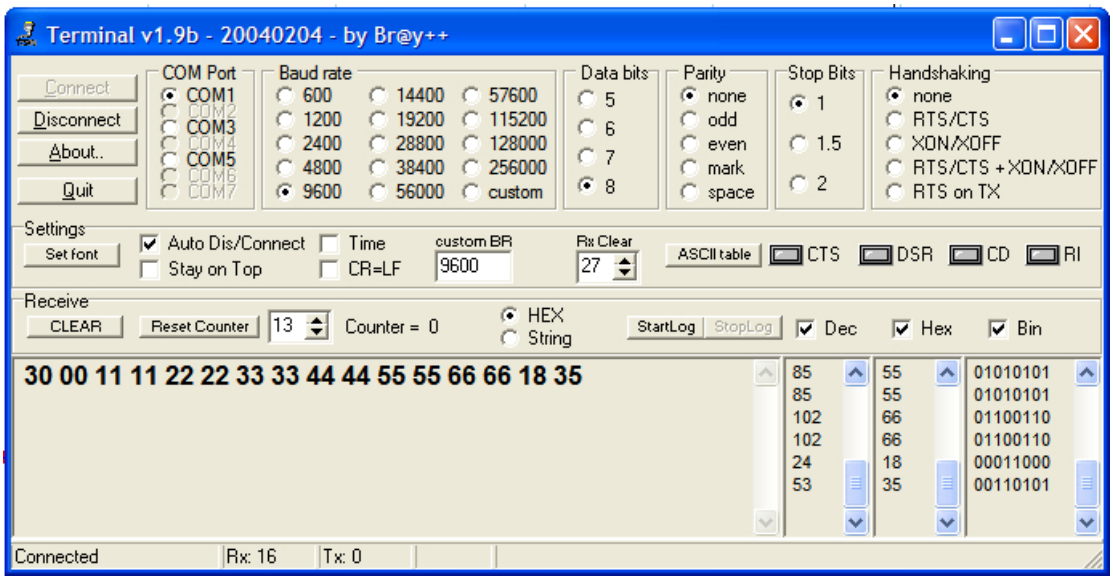
Kuva 15. *Tunnisteen FM0-koodatut tunnistetiedot. Kuvassa esitetty FM0-koodatun 96-bittisen EPC-koodin mittaus.*

Kuvassa 16 on esitetty tunnisteen lukuproseduurin signaloinnista tehty mittaus, josta nähdään signaloinnin eri osat. Lukuproseduurin mittaukseen käytettiin aika-akselin sarakevälinä 1 ms/sarake. Lukuproseduurin kestoajaksi mitattiin noin 6,46 ms. Kuvassa 17 on esitetty Br@y++ Terminal 1.9b-pääteohjelmalla tehty simuloinnin

testaus, jossa vastaanotettiin PC-tietokoneen sarjaportista ATmega32-ohjaimelta lähetetyt tunnistetiedot.



Kuva 16. *Tunnistusproseduurin signalointi. Kuvassa simuloidun EPC-koodin lukuproseduurin mittaus, jossa sininen väri kuvaa lukijan lähetettä ja punainen tunnisteen.*



Kuva 17. *Pääteohjelma-testaus. Kuvassa Br@y++ Terminal 1.9b-pääteohjelmalla vastaanotettu tunnistekooditiedot lukijaa mallintaneelta ATmega32-mikro-ohjaimelta. EPC-koodina käytettiin taulukossa 4 esitettyä 96-bittistä EPC-koodiesimerkkiä.*

6.1 Lukijan simulointikoodin analysointi

Tässä luvussa analysoitava tunnistussimuloinnissa käytetty C-kielinen lukijan ohjelmakoodi on esitetty liitteessä 2. Lukijaa simuloiva ATmega32-mikro-ohjain aloittaa tunnistussimuloinnin, kun ohjaimen PD7-liitäntään kytkettyä painonappia on painettu tai sarjaportista vastaanotettu s-kirjain. Lukijan komentojen sisältämät databitit on sijoitettuna taulukoihin, joista ne lähetetään PIE-koodattuna tunnisteelle kytkemällä 0-

ja 1-bittejä vastaavat pulssien pituudet PD6-lähtöliitäntään. Pulssien pituuksiin lisätään viiveaikakorjaukset CORR_1 ja CORR_2 symboliaikojen tarkennukseen. Tunnisteelta tuleva FM0-koodattu data vastaanotetaan ulkoisten keskeytystoimintojen INT0 ja INT1 avulla. Ohjelmassa INT0-keskeytys tapahtuu signaalin nousevalla reunalla ja INT1-keskeytys laskevalla reunalla. FM0-koodatun datan ilmaisu aloitetaan tallentamalla tulevassa signaalissa tapahtuvien muutosten aikavälit. Tähän tarkoitukseen käytetään apuna mikro-ohjaimen Timer/Counter0-ajastin-laskuria, joka ohjelmassa laskee 0…255:een kasvattaen arvoa 0,5 μs:n välein. Laskurin laskemilla lukuarvoilla voidaan ilmaista ajat 0,5…127,5 μs:iin. Tunnisteen datan vastaanotto-aliohjelmassa otetaan ulkoisten keskeytysten tapahtuessa laskurin aika talteen ja nollataan laskuri, jolloin saadaan signaalin muutosten aikavälit talteen. Muutosten aikaväleistä saadaan tehtyä FM0-dekoodaus tekemällä seuraavat ehtolauseet:

ONKO aikaväli n < 18,5 μs ?

ON → Onko aikaisempi aikaväli < 18,5μs ?

ON → Tulkitaan ’0’-tiedoksi → Otetaan aikaväli n+1 vertailuun

EI → Asetetaan muistilippu → Otetaan aikaväli n+1 vertailuun

EI → Onko aikaväli < 31 μs ?

ON→ tulkitaan ’1’-tiedoksi → Otetaan seuraava aikaväli n+1 vertailuun

EI →tulkitaan ’0’+virhe tai ’1’+virhe → Otetaan aikaväli n+2 vertailuun

Dekoodauksessa käytetään aikavälien tulkinnassa linkkitaajuuden neljännesjakson mittaista tulkintavaraa, ideaaliaikavälien ollessa ’0’-tiedolle 12,5 μs ja ’1’-tiedolle 25 μs.

Ulkoisten keskeytysten lisäksi käytetään ohjelmassa kahta ajastin-laskuri-keskeytystä: Timer/Counter1:n vertailuarvokeskeytystä sekä Timer/Counter0:n ylivuotokeskeytystä. Timer/Counter1:n vertailuarvokeskeytystä käytetään estämään ohjelman jumiutuminen tunnisteen datan vastaanottoon. Jumiutuminen estetään ajastin-laskurin aloittaessa laskennan vastaanoton alkaessa, ja mikäli yhtään tilanmuutosta ei havaita laskurin saavuttaessa vertailuaika 20 ms:ia, poistutaan vastaanotosta ja ohjelma palaa odottamaan uuden tunnistuksen käynnistystä. Lukijan vastaanottaessa dataa käytetään Timer/Counter0:n ylivuotokeskeytystä vastaanoton lopettamiseen. Jos tilanmuutosta ei havaita 127,5 μs:iin edellisestä tilan muutoksesta, lopetaan vastaanotto ja ryhdytään tulkitsemaan dataa.

Tunnistuksessa Query-komentoon liitettävä CRC-5 tarkistusluvun laskenta tehdään suoralla ohjelmakäännöksellä standardissa esitetystä CRC-5-laskennan kytkennästä. Tunnisteelta tulevan tunnistetiedon varmentava CRC-16-tarkistusluku tarkistetaan laskemalla CRC-16-CITT-tarkistusluku tunnistetiedoille ja vertaamalla laskettua arvoa vastaanotettuun. Lukuarvojen tulee olla samat virheettömässä vastaanotossa. CRC-16-CITT-tarkistuksen laskentaan käytetään WinAVR-paketin CRC-16-tarkistuksiin tehtyä kirjastotiedostoa. Kun vastaanotetut tunnistetiedot on ohjelmassa dekoodattu virheittä, lähetetään ne sarjaportin välityksellä tietokoneen pääteohjelmalla tarkasteltavaksi.

6.2 Tunnisteen simulointikoodin analysointi

Tässä luvussa analysoitava tunnisteen C-ohjelmointikielinen ohjelmakoodi on esitetty liitteessä 3. Tunnistetta simuloivan ATtiny2313-mikro-ohjaimen ohjelma alkaa vastaanottamaan lukijan komentoja havaittuaan kantoaaltoa kuvaavan signaalin PD5-liitännässä. Lukijan lähettämä PIE-koodattu komento tulkitaan käyttämällä ulkoisia keskeytyksiä INT0 ja INT1 sekä Timer/Counter1-ajastin-laskuria. INT0-keskeytys asetetaan nousevasta reunasta ja INT1-keskeytys laskevasta reunasta. PIE-koodatun datan vastaanotossa INT0-keskeytys nollaa laskurin ja INT1-keskeytys tallentaa laskurin ajan. Tällä menetelmällä saadaan tulevan signaalien pulssien pituudet selville. Lukijan lähettäessä 12,5 μs:n pulssin ’0’-tiedolle ja 37,5μs:n pulssin ’1’-tiedolle, määritetään tulkinnassa alle 25 μs pituiset pulssit ’0’:ksi ja yli 25μs:n pulssit ’1’:ksi.

Lukijalla lähetettävillä Query-komennon parametreilla ei ohjelmassa tehdä muutoksia tunnisteen asetuksiin. Vastaanotettujen datan oikeellisuus todetaan vertaamalla vastaanotettua dataa lukijan komentojen sisältöä vastaaviin taulukoihin. Mikäli vastaanotettu data on sama kuin taulukossa oleva, lähetetään lukijalle vastaus ennalta määrätyllä linkkitaajuudella. Lukijalle lähetettävän tiedon FM0-koodaus tehdään lukemalla lähettävä data taulukosta ja datasta riippuen vaihtamalla PD7-lähtöliitännän tilaa FM0-koodauksen mukaisesti.

7 YHTEENVETO

Työssä mikro-ohjaimilla tehty EPC-koodin tunnistussimuloinnin signalointi onnistui EPCglobal Class-1 Gen2-standardin asettamien määräysten mukaisesti. Suurin ongelma työssä oli standardin määräämien koodauksien tekeminen ja tulkitseminen käytetyllä laitteistolla. Toteutuksen ajoituksissa syntyi aluksi virhettä mutta ne saatiin korjattua tarvittavilla viiveaikakorjauksilla. Lukijalle tehty FM0-koodauksen tulkinta-algoritmi ei ole varmasti tehokkain ratkaisu, mutta sillä saatiin toteutettua kohtuullisen varmatoiminen vastaanotto. Työssä saatiin jatkokehittelyä varten tehtyä perusrakenne lukijan ohjaukselle.

Jatkokehittelynä voidaan tehtyihin ohjelmiin suhteellisen helposti lisätä muita standardin määrittämiä komentoja. Ohjauksen kehittelynä voitaisiin simulointiin lisätä esimerkiksi Select-komento, jolla asetettaisiin tunnisteen tunnistusvalinnat vastaamaan Query-komennon ehtoja. Lisäksi voitaisiin lisätä Access-komennot, jolloin EPC-koodia pystyttäisiin muuttamaan. Mikäli tehtyä toteutusta kehitetään lisäkomennoilla, niin tällöin tulee myös tunnisteen simulointi tehdä tarkemmin standardin mukaan toimivaksi. Tunnisteelle voitaisiin tehdä täysi mallinnus, jolloin mallintavan mikro-ohjaimen tulisi tehdä standardin mukaisia parametrimuutoksia lukijan lähettämien komentojen mukaan. Tunnisteelle ollessa täysi mallinnus, voitaisiin mm. simuloida linkkitaajuuden ja koodauksen muuttamista. Lisäkehittelyksi lukijan ohjaukseen sopisi myös erillinen PC-ohjelma, jolla lähetetään lukijalle sarjaportin välityksellä käytettävät tunnistusparametrit. Samalla ohjelmalla vastaanotettaisiin ja tallennettaisiin luetut tunnisteiden EPC-koodit vastaanottotiedostoon.

Työssä tehdyn lukijan ohjauksen käyttämistä oikeiden Gen 2 RFID-tunnisteiden EPC-koodin lukemiseen vaaditaan standardin mukaisen radio-osan rakentaminen. Vasta silloin saadaan täysi varmuus työssä tehtyjen koodausmenetelmien toimivuudesta. Ennen radio-osan rakentamista on tietysti hyvä testata simuloinnin toimivuus radiorajapinnan yli käyttäen valmiita radiopiirejä.

LÄHTEET

1. Finkenzeller, K. 2003. RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification 2nd ed. Chichester: Wiley.

2. Lahiri, S. 2006. RFID Sourcebook. Upper Saddle River, NJ: IBM Press.

3. RFID-tekniikan käyttämät taajuusalueet. Saatavissa: http://www.rfidlab.fi/rfid-tekniikan-käyttämät-taajuusalueet [viitattu 14.2.2010].

4. RFID-standardit. Saatavissa: http://www.rfidlab.fi/rfid-standardit [viitattu 14.2.2010].

5. ISO Standards. Saatavissa: http://www.iso.org/iso/iso_catalogue.htm [viitattu 26.10.2009].

6. UHF Class 1 Gen 2 Standard v. 1.1.0. Saatavissa: http://www.epcglobalinc.org/standards/uhfc1g2/uhfc1g2_1_1_0-standard-20071017.pdf [viitattu 26.10.2009].

7. Penttinen, J. 2006. Tietoliikennetekniikka: Perusverkot ja GSM. Helsinki: WSOY.

8. Ritter, T.1986. The Great CRC Mystery. Saatavissa: http://www.ciphersbyritter.com/ARTS/CRCMYST.HTM [viitattu 4.10.2009].

9. Koskinen, J. 2006. Mikrotietokonetekniikka: Sulautetut järjestelmät. Helsinki: Otava.

10. ATtiny2313 Datasheet. Saatavissa: http://www.atmel.com/dyn/resources/prod_documents/doc2543.pdf [viitattu 7.2.2010].

11. ATmega32 Datasheet. Saatavissa: http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf [viitattu 7.2.2010].

12. Picoscope 2203, 2204, 2205 User's Guide. Saatavissa: http://www.picotech.com/document/pdf/ps2203-en.pdf [viitattu 7.2.2010].

13. Toolchain Overview. Saatavissa: http://www.nongnu.org/avr-libc/user-manual/overview.html [viitattu 14.2.2010].

14. AVRStudio 4. Saatavissa: http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=2725 [viitattu 14.2.2010].

CRC-5-TARKISTUKSEN LASKENTA Liite 1

Varmennettava luku: 10000000010000000_2
CRC-5 jakaja: $x^5 + x^3 + 1 = 101001_2$ Esijakaja: 01001_2

```
          10000000010000000 00000
XOR       01001
          110010
XOR       101001
          0110110
XOR        101001
           0111110
XOR         101001
            0101110
XOR          101001
             000111100
XOR             101001
                0101010
XOR              101001
                 0000110000
XOR                 101001
                    0110010
XOR                  101001
                     0110110
XOR                   101001
                      0111110
XOR                    101001
                       0101100
XOR                     101001
                        0001110  → CRC-tarkiste 01110
```

TARKISTUS:

```
          10000000010000000 01110
XOR       01001
          110010
XOR       101001
          0110110
XOR        101001
           0111110
XOR         101001
            0101110
XOR          101001
             000111100
XOR             101001
                0101010
XOR              101001
                 0000110000
XOR                 101001
                    0110010
XOR                  101001
                     0110111
XOR                   101001
                      0111101
XOR                    101001
                       0101001
XOR                     101001
                        0000000  → tarkistus OK
```

LUKIJAN SIMULOINTIKOODI Liite 2/1

```
/*******************************************
Projekti    : Gen2 UHF RFID-lukijan ohjaus
Laitteisto  : PROJ32/644 + ATmega32-16PU 16MHz
Ohjelma     : WinAVR-20081205 + AVR Studio 4.14.589
Tekijä      : EH
Kommentit :

Ohjelma tekee EPCglobal Gen2-standardin
mukaisen lukuproseduurin, jossa tunnisteelta
luetaan 96-bittinen tunnistetieto.
Lukuproseduuri käynnistyy PD7-liitäntään
kytketystä kytkimestä tai lähettämällä pääteohjelmalla s-kirjain.

Tx: PD6
Rx: PD2 ja PD3 (INT0 ja INT1)
CW: PD5

*******************************************/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <util/crc16.h>


#define   TARI      25   // lähetysparametrit (us)
#define   DATA_0    12.5
#define   DATA_1    37.5
#define   PW        12.5
#define   RTCAL     62.5
#define   TRCAL     187.5

#define   DELIMITER 12.5

#define   CORR_1    0.1    // aikakorjaukset
#define   CORR_2    1

#define   CW_ON     PORTD |=  (1 << PD5) // PD5 -nasta '1'
#define   CW_OFF    PORTD &= ~(1 << PD5) // PD5 -nasta '0'

#define   OUT_1     PORTD |=  (1 << PD6) // PD6 -nasta '1'
#define   OUT_0     PORTD &= ~(1 << PD6) // PD6 -nasta '0'

#define   S1        !(PIND & 0x80) // Kytkin nastassa PD7, kun
                                   //  kytkintä painetaan lauseen arvo on tosi
#define   PRE       6
#define   QUE       17
#define   CRC5      5

#define   ACK       2
#define   RN16      16

#define   PC        16
#define   EPC       96
#define   CRC_16    16

#define   BITS      134 // vastaanotettavat bitit PRE+PC+EPC+CRC_16

#define   TRUE      1
#define   FALSE     0

#define   ZERO      0
#define   ONE       1

#define   BYTES     16
```

Liite 2/2

```
/*************************************************/
/*     Globaalit muuttujat         */
/*************************************************/

volatile uint8_t edge_done = FALSE;
volatile uint8_t receive_stop = FALSE;
volatile uint8_t time=0;
volatile uint8_t error = FALSE;
volatile uint8_t start = 0;

const uint8_t tag_preamble[PRE]={1,0,1,0,'v',1};


/*************************************************/
/*         Aliohjelmien prototyypit              */
/*************************************************/

void USART_Init(void);
void USART_Transmit(uint8_t data);
void Crc5_Calc(uint8_t *query,uint8_t *crc_5);
void Send_Query(uint8_t *query,uint8_t *crc_5);
void Receive_Data(uint8_t *times);
void Decode_Data_FM0(uint8_t *times,uint8_t *data,uint8_t n_bits);
uint8_t Check(uint8_t *data);
void Send_ACK(uint8_t *ack, uint8_t *data);
uint8_t Crc16_Check(uint8_t *data_byte);

/*************************************************/
/*               Aliohjelmat                     */

/*************************************************/
/*             Sarjaportin alustus               */
/*************************************************/

void USART_Init(void)
 {
     UBRRH = 0x00;
     UBRRL = 0x67; // nopeus 9600 bps XTAL 16MHZ

     UCSRB = 1 << TXEN;
     UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);
 }


/*************************************************/
/*             Sarjaportti-lähetys               */
/*************************************************/

void USART_Transmit(uint8_t data)
 {
     while( !( UCSRA & (1<<UDRE)) )
     ; // odotetaan kunnes lähetyspuskuri on tyhjä

     UDR = data; //kirjoitetaan lähetettävä tavu datarekisteriin
 }

/*************************************** *******/
/*          CRC-5-tarkisteen laskenta         */
/*************************************************/

void Crc5_Calc(uint8_t *query,uint8_t *crc_5)
 {
     uint8_t i,j;
     uint8_t crc_pre[CRC5];

     for(i=0;i<CRC5;i++)*(crc_5+i)=0;
```

```
    crc_pre[0]=1;
    crc_pre[1]=0;
    crc_pre[2]=0;
    crc_pre[3]=1;
    crc_pre[4]=0;


    for(i=0;i<QUE;i++)
     {
         *(crc_5) = *(query+i) ^ crc_pre[4];
         *(crc_5+1)= crc_pre[0];
         *(crc_5+2) = crc_pre[1];
         *(crc_5+3) = *(query+i) ^ crc_pre[4] ^ crc_pre[2];
         *(crc_5+4) = crc_pre[3];

         for(j=0;j<CRC5;j++)
          {
              crc_pre[j] = *(crc_5+j);
          }

     }

 }   // Crc5_Calc lopetus

/*************************************************/
/*           Query-komennon lähetys              */
/*************************************************/

void Send_Query(uint8_t *query,uint8_t *crc_5)
 {
    uint8_t i;

    /************ Preamble *************/
    OUT_1;
    _delay_us(DATA_0);

    OUT_0;
    _delay_us(PW - CORR_1);

    OUT_1;
    _delay_us(RTCAL);

    OUT_0;
    _delay_us(PW - CORR_1);

    OUT_1;
    _delay_us(TRCAL);

    OUT_0;
    _delay_us(PW - CORR_2);

    /***********************************/

    for(i=0;i<QUE;i++)
     {
        if(*(query+i))
         {
           OUT_1;
           _delay_us(DATA_1 + CORR_1);
           OUT_0;
           _delay_us(PW - CORR_2);

         }
```

```
        else
         {
          OUT_1;
          _delay_us(DATA_0 + CORR_1);
          OUT_0;
          _delay_us(PW - CORR_2);
         }

     }

    for(i=0;i<CRC5;i++)
     {
        if(*(crc_5+i))
         {
            OUT_1;
            _delay_us(DATA_1);
            OUT_0;
            _delay_us(PW);

         }

        else
         {
            OUT_1;
            _delay_us(DATA_0);
            OUT_0;
            _delay_us(PW);
         }

     }
    _delay_us(CORR_2);
 }                      // Send_Query lopetus

/***********************************************/
/*      Tunnisteen datan vastaanotto           */
/***********************************************/

void Receive_Data(uint8_t *times)
 {
    SREG = 0x80;        // sallitaan keskeytykset

    uint8_t i=0,j=0;

    error = FALSE;

    TCCR1B = 0x03;      // Timer/Counter1 f_clk/8 -> f=250 kHz ,T=4us
    TIFR = 0x10;        // Timer/Counter1 OCR1A-lipun nollaus
    TIMSK = 0x10;       // Timer/Counter1 OCR1A vertailukeskeytys käyttöön
    TCNT1 = 0;          // Timer/Counter1:n nollaus

    while(receive_stop == FALSE)
     {
      if(edge_done == TRUE)
       {
          SREG = 0;     // estetään keskeytykset

          edge_done = FALSE;

          *(times+i)=time;

          i++;

          SREG = 0x80;

          TIFR = 0x01;     // ylivuotolipun nollaus
          TIMSK = 0x01; // Timer/Counter0 ylivuotokeskeytys käyttöön;

       }
     }
```

```
        SREG = 0;         // estetään keskeytykset

        TIMSK = 0;
        TCCR1B = 0;

        receive_stop = FALSE;

        for(j=0;j<7;j++)
         {
             if(*(times+j)==0)error=TRUE;
         }

    } // Receive_Data lopetus

/************************************************/
/*     Vastaanotetun datan FM0-dekoodaus        */
/************************************************/

void Decode_Data_FM0(uint8_t *times,uint8_t *data,uint8_t n_bits)
 {
     uint8_t d=0;
     uint16_t j=0;
     uint8_t temp=0;

     for(j=1;j<2*BITS;j++)
      {

       if(*(times+j) < 37)     // time < 18,5us (37 * 0,5us)
        {
            if(temp==1)
             {
              *(data+d)=ZERO;
              temp=0;
              d++;
             }
            else   temp=1;
        }

        else
         {
            if(*(times+j) < 62)  // time < 31us (62 * 0,5us)
             {
                 *(data+d)=ONE;
                 d++;
             }

            else
             {
              if(temp==1)
               {
                   *(data+d)=ZERO;
                   *(data+d+1)='v';
                   temp=0;
               }

              else
               {
                   *(data+d)=ONE;
                   *(data+d+1)='v';
               }

              d=d+2;
             }

         }

       if(d>=n_bits) break;
      }
 }// Decode_Data_FM0 lopetus
```

```
/*****************************************/
/*          Datan tarkistus              */
/*****************************************/

uint8_t Check(uint8_t *data)
 {
     uint8_t j;
     uint8_t check=0;

     for(j=0;j < PRE;j++)
      {
          if(tag_preamble[j] == *(data+j)) check++;
      }

     if(check == PRE) return 1;
     else return 0;
 }


/*****************************************/
/*          ACK-komennon lähetys         */
/*****************************************/

void Send_ACK(uint8_t *ack, uint8_t *data)
 {
     uint8_t i;

     /***** Frame-sync *****/

     OUT_1;
     _delay_us(DATA_0);

     OUT_0;
     _delay_us(PW - CORR_1);

     OUT_1;
     _delay_us(RTCAL);

     OUT_0;
     _delay_us(PW - CORR_2);


     /*********************/

     for(i=0;i<ACK;i++)
      {
       if(*(ack+i))
        {
            OUT_1;
            _delay_us(DATA_1 + CORR_1);
            OUT_0;
            _delay_us(PW - CORR_2);

        }

       else
        {
            OUT_1;
            _delay_us(DATA_0 + CORR_1);
            OUT_0;
            _delay_us(PW - CORR_2);
        }

      }
```

```
    /* Vastaanotettu sat.luku RN16 */

    for(i=0;i<RN16;i++)
     {
         if(*(data+PRE+i))
          {
              OUT_1;
              _delay_us(DATA_1 + CORR_1);
              OUT_0;
              _delay_us(PW - CORR_2);
          }

         else
          {
              OUT_1;
              _delay_us(DATA_0 + CORR_1);
              OUT_0;
              _delay_us(PW - CORR_2);
          }

     }

    _delay_us(CORR_2);
 }                      // Send_ACK lopetus


/*******************************************/
/*          CRC-16-tarkistus               */
/*******************************************/

uint8_t Crc16_Check(uint8_t *data_byte)
 {
    uint8_t i;
    uint16_t crc_16;
    uint8_t crc[2]={0,0};

    crc_16=0xffff;

    for (i = 0; i < (BYTES-2); i++)
     {
         crc_16 = _crc_xmodem_update(crc_16, *(data_byte+i));
     }

    crc_16 = ~crc_16;

    crc[0]=(uint8_t)(crc_16 >> 8); // ylätavu

    crc[1]=(uint8_t)crc_16;        // alatavu

    if(*(data_byte+BYTES-2)==crc[0] && *(data_byte+BYTES-1)==crc[1])
    return 1;

    else
    return 0;

 }
```

```
/*********************************************/
/*       Keskeytyspalveluohjelmat            */
/*********************************************/

ISR(INT0_vect) // keskeytyspalveluohjelma PD2-nastan nousevasta reunasta
 {
     time = TCNT0;
     TCNT0 = 0;
     edge_done = TRUE;
 }

ISR(INT1_vect)// keskeytyspalveluohjelma PD2-nastan laskevasta reunasta
 {
     time = TCNT0;
     TCNT0 = 0;
     edge_done = TRUE;
 }

ISR(TIMER0_OVF_vect) // Timer0 ylivuotokeskeytys
 {
     receive_stop = TRUE;
 }

ISR(TIMER1_COMPA_vect) // Timer1 vertailukeskeytys
 {
     receive_stop = TRUE;
     error = TRUE;
 }

ISR(USART_RXC_vect)  // sarjavastaanottokeskeytys
 {
     start = UDR;

 }


/*********************************************/
/*               Pääohjelma                  */
/*********************************************/

int main(void)
 {
     uint16_t i;
     uint8_t j,k;
     uint8_t check = 0;
     uint8_t crc_check=0;

     uint8_t query[QUE]={1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0};
     uint8_t *p_query = &query[0];

     uint8_t crc_5[CRC5]={0,0,0,0,0};
     uint8_t *p_crc_5 = &crc_5[0];

     uint8_t ack[ACK]={0,1};
     uint8_t *p_ack = &ack[0];

     uint8_t times[2*BITS];       // taulukko havaituille aikaväleille
     uint8_t *p_times = &times[0];

     uint8_t data[BITS+1];        // taulukko vastaanotetulle datalle
     uint8_t *p_data = &data[0];

     uint8_t data_byte[BYTES];
     uint8_t *p_data_byte = &data_byte[0];
```

```
/* porttialustukset */

DDRB = 0x01;  // 0000 0001 merkkivalo
PORTB = 0x00; // 0000 0000

DDRD = 0x60;  // 0110 0000 lähdöt: PD5 ja PD6
PORTD = 0x80; // 1000 0000 kytkin nastassa PD7

/* Käytettävien I/O -rekisterien alustukset */

MCUCR  = 0x0B;   // INT0 nousevasta INT laskevasta reunasta

GICR  = 0xC0;    // ulkoiset keskeytykset INT0 ja INT1 käyttöön

TCCR0 = 0x02;    // laskuri Timer/Counter0:n käynnistys
// clk_timer = f_cpu/8 -> f=2MHz, T=0,5us

OCR1AH = 0x13;   // laskuri Timer/Counter1:n vertailurekisterin asetus
OCR1AL = 0x88;   // OCR1A = 0x1388 -> 5000 des

USART_Init();    // USART-yksikön alustus

Crc5_Calc(p_query,p_crc_5); // lasketaan CRC-5 tarkistusluku

while(1)
 {

  for(i=0;i<(2*BITS);i++)times[i]=0;  // nollataan taulukot
  for(i=0;i<(BITS+1);i++)data[i]=0;
  for(i=0;i<BYTES;i++)data_byte[i]=0;

  UCSRB = 0x98; // asetetaan sarjaportin vastaanotto-keskeytys käyttöön
  SREG = 0x80;  // sallitaan keskeytykset

  if(S1 || start == 's')
   {
       start = 0;
       SREG = 0;      // estetään keskeytykset
       UCSRB = 0x08; // sarjaportin vastaanotto-keskeytys pois käytöstä

       _delay_ms(500);  // viive kytkimen painalluksesta

       CW_ON; // kantoaalto päälle

       _delay_us(1500);

       CW_OFF;// kantoaalto pois päältä

       _delay_us(DELIMITER);

       Send_Query(p_query,p_crc_5);// lähetetään Query-komento

       CW_ON; // kantoaalto päälle

       Receive_Data(p_times);  //vastaanotetaan RN16

       if(error==FALSE)
        {
            Decode_Data_FM0(p_times,p_data,(PRE+RN16));

            check=Check(p_data);

            if(check)
             {
                 CW_OFF;// kantoaalto pois päältä

                 _delay_us(DELIMITER);
```

```
                    Send_ACK(p_ack,p_data); // lähetetään ACK-komento

                    CW_ON; // kantoaalto päälle

                    Receive_Data(p_times);

                    CW_OFF;// kantoaalto pois päältä

                    if(error==FALSE)
                     {
                         Decode_Data_FM0(p_times,p_data,BITS);

                         check=Check(p_data);

                         if(check)
                          {
                              /*** muutetaan vastaanottobitit tavuiksi ***/

                              k = 0;

                              for(j=0;j<BYTES;j++)
                               {
                                   for(i=0;i<8;i++)
                                    {
                                      data_byte[j] |= data[6+k] << (7-i);
                                      k++;
                                      // bitti-> tavu-muunnos 6.bitistä
                                      // josta data alkaa
                                    }
                               }

                              // CRC-16 tarkistus
                              crc_check = Crc16_Check(p_data_byte);

                              // jos CRC-16 tarkistus oikein lähetetään
                              // data sarjaportin kautta tietokoneelle

                              if(crc_check)
                               {
                                   for(i=0;i<(PC/8);i++)
                                    {
                                        USART_Transmit(data_byte[i]);
                                        _delay_ms(5);
                                    }

                                   for(i=(PC/8);i<(PC+EPC)/8;i++)
                                    {
                                        USART_Transmit(data_byte[i]);
                                        _delay_ms(5);
                                    }

                                   for(i=(PC+EPC)/8;i<(PC+EPC+CRC_16)/8;i++)
                                    {
                                        USART_Transmit(data_byte[i]);
                                        _delay_ms(5);
                                    }
                               }
                          } // if(check) lopetus

                     }   // if(error==FALSE) lopetus

                }    // if(check) lopetus

            }   // if(error==FALSE) lopetus
          CW_OFF;// kantoaalto pois päältä

        }   // if(S1 || start == 's') lopetus
    }    // while(1)
}     // main(int)
```

```
/*************************************************
Projekti   : Gen2-tunnisteen simulointi
Laitteisto : EXB2313 + ATtiny2313-20PU 12MHz
Ohjelma    : WinAVR-20081205 + AVR Studio 4.14.589
Tekijä     : EH
Kommentit  :

Ohjelmassa simuloidaan EPCglobal Gen2-standardin
mukaisen tunnisteen toimintaa.
Lukijan komennot vastaanotetaan keskeytystuloista
INT0 ja INT1.
Tunnisteen vastaukset lähetetään nastasta 19 (PB7)
Lukijalle lähetetään 96-bittinen tunnistetieto.

*************************************************/

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>


#define   TRUE        1
#define   FALSE       0

#define   PRE         3
#define   QUE         22
#define   QUERY       25
#define   RN16        16
#define   FS          2
#define   ACK         2

#define   ACK_COMM    20 // FS+ACK+RN16

#define   T_BLF1      25
#define   T_BLF0      12.5

#define   INVERT_OUT  PORTB ^= 0x80 // PB7
#define   OUT_1       PORTB |=  (1 << PB7)
#define   OUT_0       PORTB &= ~(1 << PB7)

#define   CW          PIND & 0x20

#define   ZERO        0
#define   ONE         1

#define   PC          2 // lähetettävien tavujen lukumäärä
#define   EPC         12
#define   CRC_16      2


/************************************************/
/*              Globaalit muuttujat             */
/************************************************/

volatile uint8_t symbol_done = FALSE;
volatile uint16_t time=0;
volatile uint8_t flag0 = FALSE;

const uint8_t preamble_comp[PRE]={0,1,1};

const uint8_t query_comp[QUE]={1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,1,1,0};

const uint8_t rn16[RN16]={1,0,1,0,1,0,1,1,1,1,0,0,1,1,0,1}; // 0xAB,0xCD

const uint8_t ack_comp[FS+ACK]={0,1,0,1};

const uint8_t pc_bits[PC] = {0x30,0x00};
const uint8_t epc[EPC] = {0x11,0x11,0x22,0x22,0x33,0x33,
                          0x44,0x44,0x55,0x55,0x66,0x66};

const uint8_t crc_16[CRC_16] = {0x18,0x35};
```

```
/************************************************/
/*          Aliohjelmien prototyypit            */
/************************************************/


void Receive_command(uint8_t bits,uint8_t *data);
uint8_t Check_query(uint8_t *data);
void Send_RN16(void);
uint8_t Check_ack(uint8_t *data);
void Send_EPC(void);

/************************************************/
/*                 Aliohjelmat                  */


/************************************************/
/*            Komennon vastaanotto              */
/************************************************/

void Receive_command(uint8_t bits,uint8_t *data)
 {
   SREG = 0x80;

   uint8_t i=0;

   while(i<bits)
    {

     if(symbol_done == TRUE)
        {
          SREG = 0x00;

          symbol_done = FALSE;

          if(time > 0x004B)
           {
               if(time < 0x012C )
                {
                  *(data+i)=ZERO;  // t < 25us
                }

               else
                {
                  *(data+i) = ONE;
                }

               i++;
           }

          SREG = 0x80;
       }

    }


   SREG = 0x00;

 }


/************************************************/
/*          Query-komennon tarkistus            */
/************************************************/

uint8_t Check_query(uint8_t *data)
 {
   uint8_t j;
   uint8_t check=0;
```

```
    for(j=0;j < PRE;j++)
     {
      if(preamble_comp[j] == *(data+j)) check++;
     }

    for(j=0;j < QUE;j++)
     {
      if(query_comp[j] == *(data+PRE+j)) check++;
     }


    if(check == QUERY) return 1;   // jos ei virheitä
    else return 0;                 // jos virheitä

 }


/*************************************************/
/*                  RN16 -lähetys                */
/*************************************************/

void Send_RN16(void)
 {
  uint8_t i;

    /******** Preamble *******/

      OUT_1;
      _delay_us(T_BLF1);

      OUT_0;
      _delay_us(T_BLF0);

      OUT_1;
      _delay_us(T_BLF0);

      OUT_0;
      _delay_us(T_BLF1);

      OUT_1;
      _delay_us(T_BLF0);

      OUT_0;
      _delay_us(T_BLF0);

      _delay_us(T_BLF1); // ei tilanvaihtoa

      OUT_1;
      _delay_us(T_BLF1);


      for(i=0;i<RN16;i++)
       {
        if(rn16[i])
            {
             INVERT_OUT;         // '1' kääntää lähdön tilan kerran
             _delay_us(T_BLF1);

            }

           else
            {
             INVERT_OUT;         // '0' kääntää lähdön tilan kahdesti
             _delay_us(T_BLF0);

             INVERT_OUT;
             _delay_us(T_BLF0);
            }
       }
```

```
    /**** dummy-bitti ****/

       INVERT_OUT;
       _delay_us(T_BLF1);

       OUT_0;

 }

/*************************************************/
/*             ACK-komennon tarkistus            */
/*************************************************/

uint8_t Check_ack(uint8_t *data)
 {
  uint8_t j;
  uint8_t check=0;

     for(j=0;j < (FS+ACK);j++)
      {
       if(ack_comp[j] == *(data+j)) check++;
      }

     for(j=0;j < RN16;j++)
      {
       if(rn16[j] == *(data+FS+ACK+j)) check++;
      }


     if(check == ACK_COMM) return 1;  // jos ei virheitä
     else return 0;          // jos virheit


 }

/*************************************************/
/*               EPC-koodin lähetys              */
/*************************************************/
void Send_EPC(void)
 {
  uint8_t i,j;

     /******** Preamble *******/

       OUT_1;
       _delay_us(T_BLF1);

       OUT_0;
       _delay_us(T_BLF0);

       OUT_1;
       _delay_us(T_BLF0);

       OUT_0;
       _delay_us(T_BLF1);

       OUT_1;
       _delay_us(T_BLF0);

       OUT_0;
       _delay_us(T_BLF0);

       _delay_us(T_BLF1); // ei tilanvaihtoa

       OUT_1;
       _delay_us(T_BLF1);
```

```
/******** PC-bitit ********/

       for(j=0;j<PC;j++)
        {
         for(i=0;i<8;i++)
          {
           if(pc_bits[j] & (0x80 >> i))
            {
             INVERT_OUT;      // kääntää lähdön tilan
             _delay_us(T_BLF1);

            }

           else
            {
             INVERT_OUT;
             _delay_us(T_BLF0);

             INVERT_OUT;
             _delay_us(T_BLF0);
            }
          }
        }

     /******** EPC-koodi ********/

       for(j=0;j<EPC;j++)
        {
         for(i=0;i<8;i++)
          {
           if(epc[j] & (0x80 >> i))
            {
             INVERT_OUT;
             _delay_us(T_BLF1);

            }

           else
            {
             INVERT_OUT;
             _delay_us(T_BLF0);

             INVERT_OUT;
             _delay_us(T_BLF0);
            }
          }
        }


     /****** CRC-16 tarkiste ******/

       for(j=0;j<CRC_16;j++)
        {
         for(i=0;i<8;i++)
          {
           if(crc_16[j] & (0x80 >> i))
            {
             INVERT_OUT;
             _delay_us(T_BLF1);

            }

           else
            {
             INVERT_OUT;
             _delay_us(T_BLF0);

             INVERT_OUT;
             _delay_us(T_BLF0);
            }
```

```
        }
      }

    /**** dummy-bitti ****/

      INVERT_OUT;
      _delay_us(T_BLF1);

      OUT_0;


}


/************************************************/
/*          Keskeytyspalveluohjelmat            */
/************************************************/

ISR(INT0_vect) // keskeytyspalveluohjelma PD2-nastan nousevasta reunasta
 {
  TCNT1 = 0;

}


ISR(INT1_vect) // keskeytyspalveluohjelma PD3-nastan laskevasta reunasta
 {
  time = TCNT1;    // laskurin aika talteen

  symbol_done = TRUE;

}

/************************************************/
/*                  Pääohjelma                  */
/************************************************/

int main(void)
 {

  uint8_t j=0;
  volatile uint8_t check=0;

  uint8_t data[QUERY];
  uint8_t *p_data = &data[0];

  for(j=0;j < QUERY;j++)data[j]=0; // nollataan data-taulukko

  /* porttialustukset */

  DDRB   = 0xFF;   // 1111 1111
  PORTB  = 0x00;   // 0000 0000

  DDRD   = 0x00;
  PORTD  = 0x00;


  /* Käytettävien I/O -rekisterien alustukset */

  MCUCR  = 0x0B;     // INT0 nousevasta, INT1 laskevasta reunasta

  GIMSK  = 0xC0;     // ulkoisen keskeytykset INT0 ja INT1 käyttöön

  TCCR1B = 0x01;     // laskurin Timer/Counter1:n käynnistys
```

```
while(1)
    {
     if(CW)
      {
       Receive_command(QUERY,p_data);

       check=Check_query(p_data);

       if(check)
        {
         Send_RN16();

         Receive_command(ACK_COMM,p_data);

         check=Check_ack(p_data);

         if(check)
          {
           Send_EPC();

          }

        }

       for(j=0;j < QUERY;j++)data[j]=0;

     }

   }

 }
```