Paavo Porkka

RFID-lukijan prototyypin kehitys

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tietotekniikan koulutusohjelma
Insinöörityö
30.5.2011

Tekijä Otsikko Sivumäärä Aika	Paavo Porkka RFID-lukijan prototyypin kehitystyö 39 sivua + 16 liitettä 30.5.2011
Tutkinto	insinööri
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	sulautettu tietotekniikka
Ohjaaja	lehtori Anssi Ikonen

Opinnäytetyön aiheena oli toteuttaa kannettava RFID-lukija Metropolia Ammattikorkeakoulun laboratoriotuntien seurantalaitteeksi. Lukijalaitteen tarkoitus on kerätä opiskelijoiden saapumis- ja poistumistietoja laboratoriotunneilta. Lukijaan oli myös tarkoitus saada USB-väylän PC-liitettävyys, jotta tiedonkeruu olisi mahdollista viedä edelleen käsiteltäväksi ja esimerkiksi tilastoitavaksi.

Ainakin vielä toistaiseksi nykykäytäntönä laboratoriotunneilla on vanhanaikainen kynä-paperimenetelmä, jolle haluttiin lähteä toteuttamaan vaihtoehtoista ja käytännöllisempää sähköistä ratkaisua. Laboratoriotuntien pakollisuuden ja toisaalta ajankäyttöön liittyvien erikoisuuksien takia tällaiselle sähköiselle mallille voisi olla käyttöä.

RFID-lukijalaitteen prototyypin toteutus koostui seuraavista komponenteista: Parallax RFID Card Readerista, PSoC-mikro-ohjaimen kehitysalustasta, LCD-näytöstä, reaaliaikakellopiiristä (RTC), ulkoisesta muistista (EEPROM) sekä numeronäppäimistöstä. Lisäksi kehitysalustalla olevan mikro-ohjaimen ohjelmointiin käytettiin PC-tietokoneeseen USB-liitettävää ohjelmointilaitetta. Ohjelmistokehitys tehtiin PC-tietokoneella PSoC Programmer -kehitysympäristössä.

Kehitystyön lopuksi sain valmiiksi testatusti toimivan prototyypin, joka sisälsi lähes kaikki alkuperäiset määritellyt toiminnallisuudet. Käyttäjä pystyy valitsemaan halutun kurssin, lisäämään kurssin, muuttamaan reaaliaikakellon aikaa, kirjautumaan kurssille sisään tai ulos sekä liittämään laitteen USB-väylään virran syöttöä varten. USB-väylän kommunikaatio PC-tietokoneelle jäi osittain vielä toteuttamatta.

Avainsanat	kannettava RFID-lukija, prototyyppi, PSoC, USB, UART, I2C, RTC, EEPROM

Author	Paavo Porkka
Title	Developing an RFID reader prototype
Number of Pages	39 pages + 16 appendices
Date	30 May 2010
Degree	Information Technology
Degree Programme	Bachelor of Engineering
Specialisation option	Embedded Systems
Instructor	Anssi Ikonen, Lecturer

The purpose of this bachelor's thesis was to design and implement a portable RFID reader prototype for the Embedded Systems laboratory classes of Metropolia University of Applied Sciences. The main goal was to build a reader device which can collect students' arrival and exit information from laboratory classes. The reader was also to be provided with USB port connectivity to PC, so that it would be possible to handle and export data for further processing, for example, for statistical purposes.

The current practice in laboratory classes is the old-fashioned so-called pen-paper method and we wanted to build a better and more practical solution for this kind of bookkeeping. As these laboratory classes are mandatory and on the other hand also contains some time-related deviations, we thought that this electronic solution could be much more useful.

The implementation of the RFID reader prototype consists of the following components: Parallax RFID Card Reader, PSoC microcontroller's development platform, an LCD screen, a real-time clock circuit (RTC), an external memory (EEPROM) and a keypad. In addition PSoC's development platform includes the USB connectivity that was successfully designed and programmed on PC with PSoC Programmer development software.

The outcome of the development work was a duly tested prototype, which included almost all the original set of planned functions. With the prototype the user can select or add a desired course, change the real-time clock's time, log into or out of class, and the device can be connected to a USB port for power. Full USB bus communication for PC was not yet fully implemented.

Keywords	portable RFID reader, prototype, PSoC, USB, UART, I2C, RTC, EEPROM

Sisällys

Liitteet

1 Johdanto

Tässä insinöörityön loppuraportissa kerrotaan ideasta kehittää Metropolia Ammattikorkeakoulun laboratoriotunteja varten kannettava RFID-tekniikkaa hyödyntävä seurantalaite. Raportin tarkoitus on kertoa lukijalle mahdollisimman kattavasti siitä, millainen oli alkuperäinen idea, miten ja miksi sitä lähdettiin kehittämään sekä analysoida, millainen lopputulos syntyi.

Dokumentissa pyritään myös antamaan mahdollisimman tarkka kuvaus projektin etenemisestä vaihe vaiheelta alusta loppuun asti. Lukijalle pyritään selostamaan asiat mahdollisimman yksinkertaisesti ja ymmärrettävästi, jotta kuka tahansa tietotekniikkaan, käytettyihin sovelluksiin, tekniikoihin, ohjelmistoihin tai käytettyihin laitekomponentteihin perehtymätönkin voisi ymmärtää projektin keskeisimmät asiat.

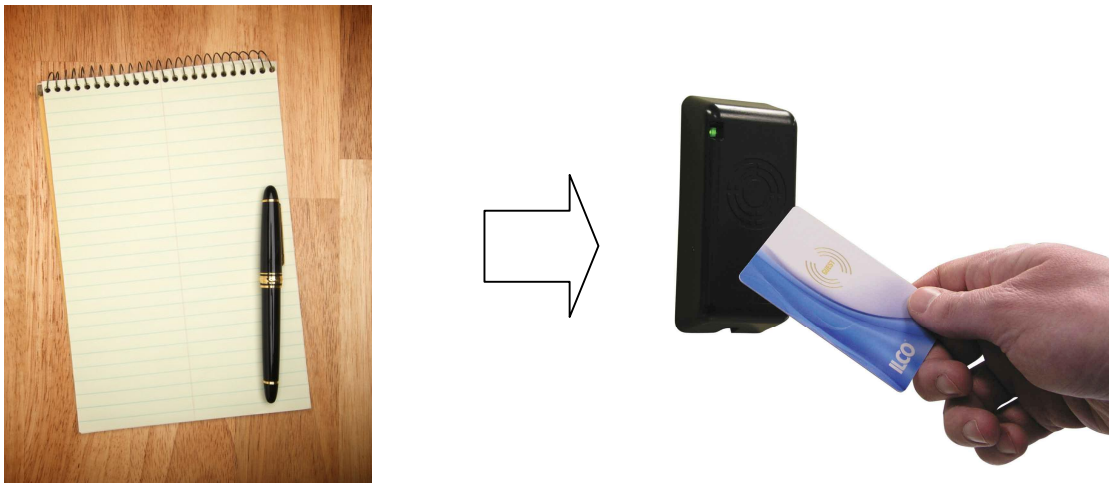
2 Kehitystyön tarkoitus ja tavoite sekä alkuperäinen määrittely

Insinöörityön tarkoituksena oli kehittää koulun laboratoriotyötunteja varten kannettava ja mahdollisimman kätevä sekä pienikokoinen RFID (Radio Frequency IDentification) -seurantalaite, jolla voitaisiin seurata laboratoriotunneille osallistuvia opiskelijoita ja nimenomaan heidän ajankäyttöään helposti ja vaivattomasti. Seurantalaitteen käytön tulisi olla nopeaa ja sujuvaa opiskelijoiden kannalta sekä helpottaa että nopeuttaa varsinkin opettajan työtä laboratoriotuntien läsnäolijoiden seuraamisessa ja listaamisessa.

2.1 Nykykäytännöt laboratoriotunneilla

Tällä hetkellä koulun laboratoriotunteja seurataan perinteisellä kynä-paperi-menetelmällä. Käytännössä tämä tarkoittaa sitä, että ennalta määräämättömällä ajanhetkellä opettaja ottaa osallistujalistan esille, tarkistaa, ketkä opiskelijoista ovat laboratoriossa läsnä, ja merkitsee heidän osallistumisensa listaan. Menettely on huono ennen kaikkea laboratoriotuntien vaihtelevuuden sekä opiskelijoiden omien kykyjen ja varsinkin työskentelytapojen vaihtelevuuden takia. Merkitsemistapa lisää myös turhaa paperin kulutusta ja aiheuttaa opettajalle vaivalloista lisätyötä tuntien ulkopuolella.

Laboratoriotunneilla on käytännössä läsnäolopakko, mutta opiskelijat käyttävät aikaansa eri tavalla tehtävistä suoriutumiseen. Tunneilla on myös, hieman opettajasta riippuen, yleensä sallittu vapaamuotoiset työskentelytavat, joten opiskelijat pitävät taukojaan kuka milloinkin. Näin ollen yksi opiskelija saattaa suoriutua annetusta tehtävästä 15 minuutissa, kun taas toisella työhön voi kulua helposti tunti ja 15 minuuttia tai jopa kauemmin. Näistä asioista johtuen juuri silloin, kun opettaja merkitsee läsnäolijat, saattaa osa opiskelijoista olla jo valmiita tai pitämässä parhaillaan taukoaan, eikä merkintää tällöin tule välttämättä tehtyä, ellei opiskelija itse sitä huomaa tarkistuttaa. Paperimerkintä ei myöskään ymmärrettävistä syistä kerro sen enempää, kuin että paikalla on oltu, mikäli merkintöjä ylipäätänsä on muistettu tehdä ja läsnäolijat ovat olleet oikeaan aikaan laboratoriossa paikalla. Kuvassa 1 kuvataan nykykäytäntöä sekä tavoiteltua RFID-lukijaa.



Kuva 1. Kuvassa nykyinen paperi-kynä-menetelmä sekä tavoiteltu toimintamalli: sähköinen RFID-lukija ja -tunniste [1, 2].

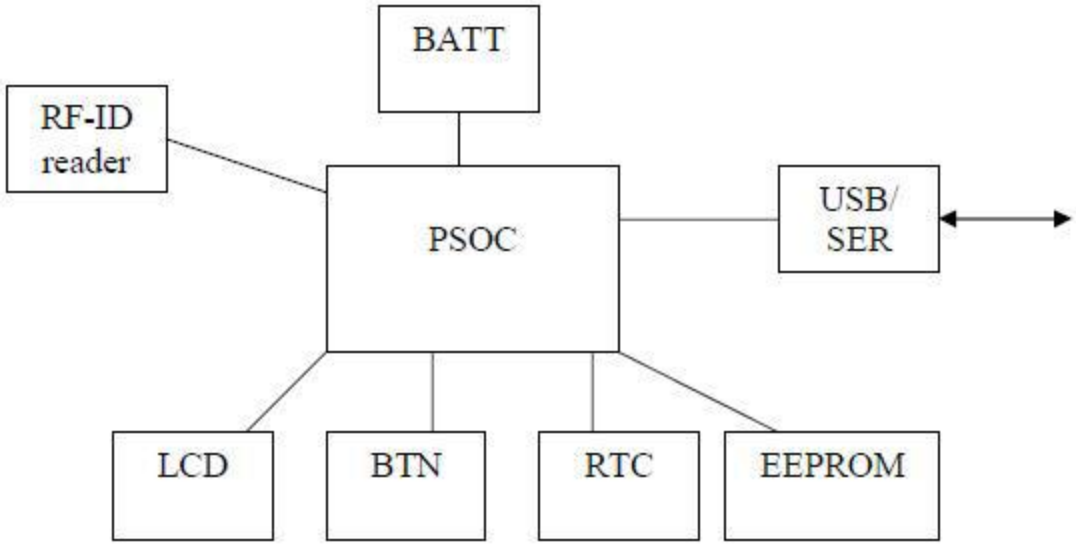
2.2 RFID-tekniikan hyödyntäminen

Projektin tarkoitus oli tietysti hyödyntää jo käytössä olevia koulun RFID-tunnisteita eli pieniä mustia avaimenperinäkin käytettäviä kulkulupatunnisteita, joilla opiskelijat voivat avata esimerkiksi luokkahuoneiden ovia näyttämällä tunnistetta oven lukijalle. Ensisijaisena projektin päämääränä oli siis ottaa tarkemmin selvää RFID-tekniikasta sekä olemassa olevien tunnisteiden soveltuvuudesta kyseiseen projektiin.

Sähköisen seurannan toivottiin myös antavan paljon lisätietoa siitä, minkä verran kukin opiskelija on käyttänyt aikaansa laboratoriotunteihin, sillä jokaiselta opiskelijalta saataisiin näin tarkka saapumis- sekä poistumisajankohta tallennettua jatkokäyttöä ja analysointia varten. Tätä tietoa voidaan hyödyntää esimerkiksi niiden opiskelijoiden kohdalla, jotka ovat kuitenkin viettäneet aikaansa laboratoriotunneilla yrittäen ja töitä tehden, mutta eivät kenties ole kuitenkaan onnistuneet tehtävissä täysin tavoitteiden mukaisesti, ja näin tehtävien lopullisessa arvioinnissa voitaisiin hyödyntää kerättyjä tarkempia tietoja ja kompensoida annettavia pisteitä tarpeen mukaan.

2.3 Alkuperäinen kehitettävän laitteen määrittely

Työtä varten hahmoteltiin halutusta laitteesta erittäin pelkistetty määrittelykuvaus, josta on nähtävillä tarkempi esitys lohkokaavion avulla kuvassa 2 sekä laitteen toiminnallisuus kirjallisesti selvitettynä jäljempänä. Näiden tietojen pohjalta projektia alettiin toteuttaa.



Kuva 2. Alkuperäinen tilaajan määrittely, lohkokaavio, jossa on kuvattu tarvittavat komponentit ja niiden väliset yhteydet.

Alkuperäiseen määrittelyyn kuului myös erillinen kirjallinen osuus (tekstinäyte 1), jossa pyrittiin mahdollisimman yksinkertaisella tasolla määrittelemään laitteen toiminnallisuus niin sanotussa ”Action mode” -tilassa eli akkuvirralla käytettävässä tilassa sekä ”Serial Command Protocol” -tilassa, jolloin laite olisi kytkettynä tietokoneeseen tietojen tuontia ja vientiä varten. ”Action mode” -tilan toteutus ja testaus sovittiin tärkeimmäksi ja

ensisijaiseksi tavoitteeksi. Alun perin määriteltyä kokonaisuutta pidettiin myös kokonaisuudessaan toteutuskelpoisena ainakin toimivaan ja testattavaan prototyyppiin asti, joten tämä asetettiin lopulliseksi tavoitteeksi.

Toimintamoodit:

- *"Serial Command Protocol" – yhdistetty USB-kaapelilla*
  - *Kurssien määritys*
    - *Ominaisuus voi olla rajoitettu esimerkiksi 10 kurssiin*
  - *Osallistujalistan tulostuskomennot*
    - *Yhdellä komennolla kaikkien kurssien osallistujalistat*
    - *Komento, jolla kaikki kurssit listataan*
    - *Komento, jolla listataan tietyn kurssin osallistujalistat*
- *"Action mode" – toiminta akuilla*
  - *Painonapeilla valitaan kurssi (BTN+LCD)*
  - *Avataan sessio*
  - *Luetaan RFID-tunnisteet*
  - *Suljetaan sessio*

Tarkennettu "Action mode" -tilan toiminnallisuus:

- *Luetaan aika Real Time Clockilta (RTC, DS1307)*
  - *I2C-väylässä, joka tarvitsee oman akun*
- *Aloitetaan sessio*
  - *Kurssi valitaan LCD-näytön ja näppäinten avulla*
  - *Session aloitusaika*
  - *Osallistujien tunnisteet*
    - *Tallennus jokaisen tunnisteen jälkeen EEPROM-muistille*
- *Suljetaan sessio, tai jos virta katkeaa, sessio sulkeutuu automaattisesti, eli informaatiota ei saa hävitä, jos esimerkiksi akuista loppuu puhti kesken kaiken*
- *"Action mode" mahdollinen myös, jos laite on USB-kaapelilla kytkettynä*
  - *Akkuja ei tarvitse ladata, joten kytkennästä tulee melko yksinkertainen*

Tekstinäyte 1. Prototyypin toiminnallisuuden määrittely.

Näiden määrittelyiden pohjalta aloin toteuttaa suunniteltua ja tehtäväksi sovittua prototyypin kehitystä. Kerron myöhemmin dokumentin ohjelmistokehitys- sekä käytännön työn etenemisosioissa tarkemmin siitä, miten alkuperäistä suunnitelmaa lähdettiin toteuttamaan ja miten se muuttui projektin edetessä.

3 RFID-tekniikan perusteet

Radio Frequency IDentification eli paremmin tunnettuna lyhenne RFID on laaja-alainen termi, joka viittaa erinäköisiin radiotaajuuksia ja langattomia ominaisuuksia hyödyntäviin tekniikoihin. Tässä yhteydessä radiotekniikkaa hyödyntävän tekniikan perusperiaate on jakaa kohdetietoja, paikkatietoja sekä hyödyntää taustajärjestelmiä. RF-termi viittaa RFID-tekniikassa tietysti siihen tosiasiaan, että kyseisessä tekniikassa hyödynnetään radioaaltoja. ID-termi taas viittaa tunnistetietoihin, jotka ovatkin RFID-tekniikan keskeisin asia.

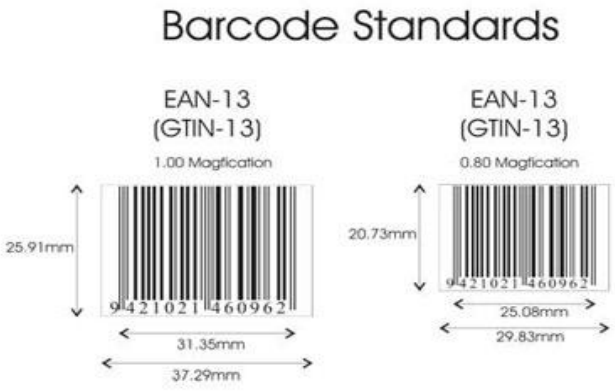
RFID-tekniikkaa voisi verrata vanhaan ja edelleen suuressa käytössä olevaan viivakooditekniikkaan. Suurin eroavaisuus viivakooditekniikkaan on nimenomaan langattomuus, joka mahdollistaa sen, että tunnisteita voidaan lukea ilman ”suoraa katsekontaktia” ja jopa joidenkin esineiden läpi sekä tietysti myös paljon suuremmilta etäisyyksiltä, kun perinteisiä viivakoodeja. Varsinkin, mikäli kyse on aktiivisesta tunnisteesta, jolloin myös tunnisteella on oma virtalähteensä, voi tunnisteen luku onnistua yllättävän suurelta etäisyydeltä. [8, s. 1 ; 10.]

RFID-tekniikan perusperiaate kiteytyy siihen, että on olemassa ”RFID-tageja” eli -tunnisteita, joihin on sisällytetty tietynlainen ID-tunnistetieto, kuten viivakoodeissa esimerkiksi ”ABC0123456”, tätä tietoa voidaan hyödyntää taustajärjestelmien tai kyseisen lukijalaitteen muistijärjestelmän avulla. Tunnistetta luettaessa RFID-lukijalla tunniste palauttaa lukijalle tämän ID-tiedon. Lukija voisi esimerkiksi tilanteessa, jossa käyttäjän yrittää avata RFID-lukittua ovea, kysyä taustajärjestelmältä tai tarkistaa lukijan muistista, onko juuri kyseinen ID:n arvo oikeutettu oven avaamiseen.

3.1 Tunnisteiden historia lyhyesti

Sähköisten tunnisteiden historia yltää aina vuoteen 1974 asti, jolloin ensimmäiset viivakoodit, esimerkki kuvassa 3, löysivät päivittäistavarakauppoihin. Viivakoodit yleistyivät eritoten halvan tuotantonsa takia. Käytännössä kauppojen tarvitsi vain tulostaa viivakoodi tarralapulle ja liittää se tuotteeseen. Viivakoodin lukemiseen käytetään yleensä infrapunalukijaa, joka havaitsee viivakoodin viivojen osoittaman numerosarjan eli ID-tunnisteen, mutta nykyään jopa matkapuhelimiin on tehty

viivakoodin lukijasovelluksia, joilla voi muun muassa hakea Internetistä lisätietoa tuotteista. [13, s. 9.]



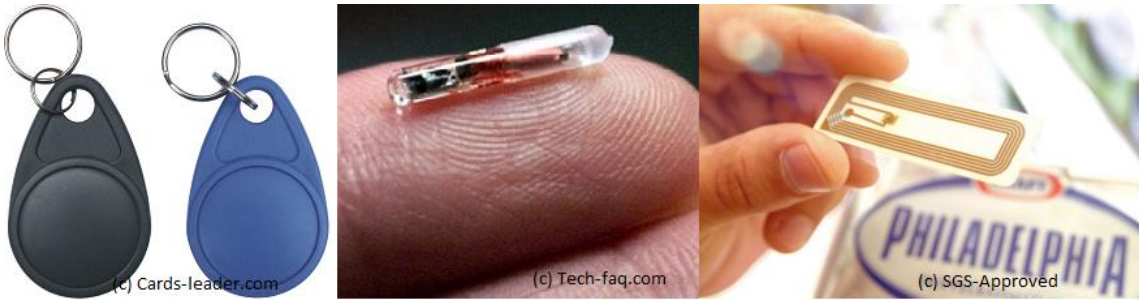
Kuva 3. Esimerkki viivakoodista [12].

Viivakoodien jälkeen käyttöön yleistyivät myös magneettinauhalla varustetut kortit. Yleisimmin käytettyjä magneettinauhallisia kortteja ovat erilaiset maksukortit, esimerkiksi luottokortit. Kortin toimintaidea on verrattavissa vanhaan magneettinauhalla toimivaan VHS- sekä C-kasetti-tekniikkaan. Lukija siis lukee magneettinauhalta fyysisessä kontaktissa ID-tunnisteen tiedot. Yleensä magneettinauhalla on 210 bittiä tietoa jokaista tuumaa kohti (8,27 bittiä/mm) tai vaihtoehtoisesti 75 bittiä tuumaa kohti (2,95 bittiä/mm). Tämä tieto voi sisältää esimerkiksi kortin numeron, voimassaolon päättymispäivän ja käyttäjän muut tiedot. [11, s. 17.]

Magneettinauhallisia kortteja seurasivat niin sanotut ”älykortit”. Näitä ”Smart Card” -kortteja on sittemmin 1990-luvulta lähtien käytetty muun muassa puhelimien SIM-kortteina ja ne mahdollistavat myös jonkun verran tiedon tallennusta EEPROM-muistiin. Kuvassa 4 on esimerkkejä erilaisista älykorteista. [11, s. 17–18.]

Kuva 4. SIM-kortti, sirullinen luottokortti sekä magneettinauhallinen kortti [13, 14, 15].

Nykyaikaisten RFID-laitteiden ensimmäiset käytännön testaukset ajoittuvat jo sähköisten kulkutunnisteiden alkuajoille asti, jolloin ensimmäisiä kertoja luettiin langattomasti passiivitunnisteita jopa useiden metrien päästä radiotekniikkaa hyväksi käyttäen. Aina 70–80-luvuilta lähtien RFID:n kehitys on jatkanut nousujohteista vauhtia ja nykypäivänä uusia innovaatioita kehitetään tekniikkaa hyväksi käyttäen jatkuvasti. RFID:n kehitystyön suurimpiin meriitteihin voidaan laskea vuoden 2005 ensimmäinen globaali standardisointi EPC (Electronic Product Code). Kuvassa 5 on esimerkkejä erilaisista RFID-tunnisteista. [11, s. 19-20.]



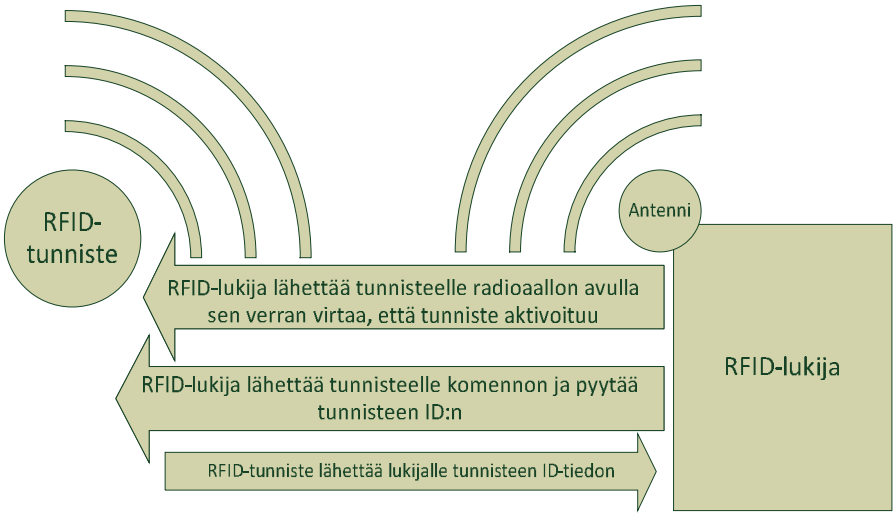
Kuva 5. Kuvassa muutamia erilaisia RFID-tunnisteita kulkutunnisteista tuotetunnisteisiin [19, 20, 21].

3.2 RFID-tekniikan käytännön toimintaperiaate

RFID:n perustoiminnallisuus on yksinkertainen. Tunnisteen lukuoperaatio käy käytännössä kolmivaiheisesti:

1. Lukija lähettää tunnisteelle radiosignaalin mukana tarpeeksi virtaa, jotta tunniste aktivoituu ja voi lähettää lukijalle tietoa.
2. Lukija käskee tunnisteen antamaan tunnistetiedon, eli ID:n arvon.
3. Tunniste lähettää lukijalle tunnisteen ID-tiedon.

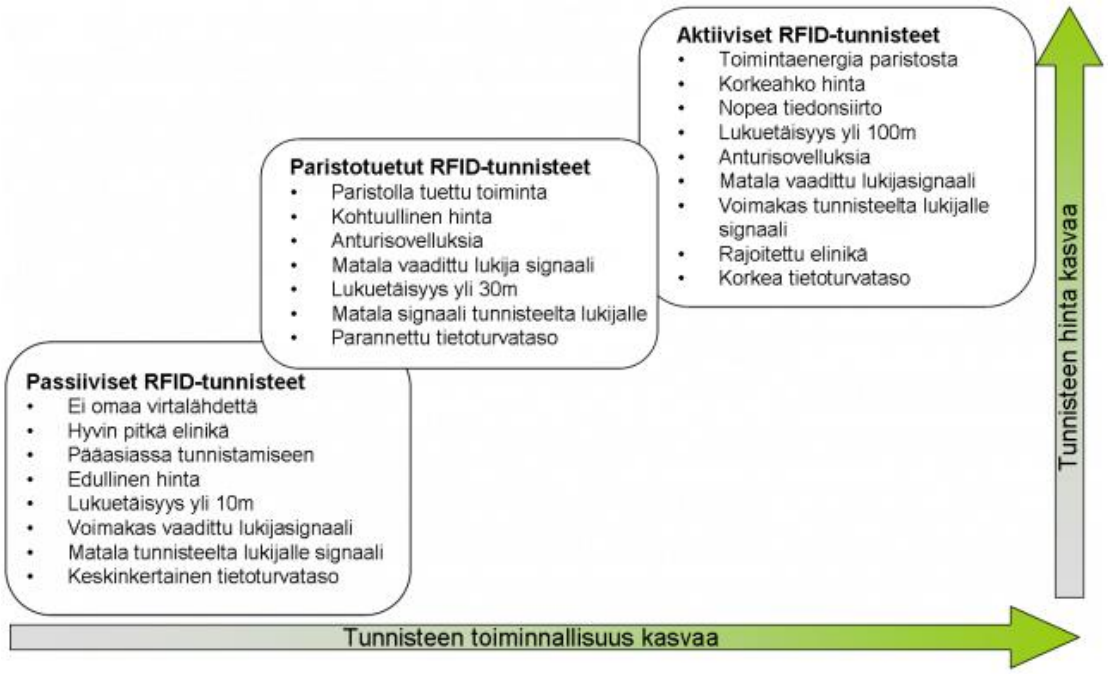
Tiedonsiirron teoria on havainnollistettu tarkemmin seuraavalla sivulla kuvassa 6.



Kuva 6. Kuvassa näkyy yksinkertaisimmalla tasolla kuvattuna, miten tiedonkulku välittyy lukijalta tunnisteelle ja tunnisteelta takaisin lukijalle.

Edellä kuvatun toimenpiteen jälkeen lukijaan liitetty, esimerkiksi päivittäistavarakaupan kassajärjestelmä voisi tunnistetietojen nojalla tehdä halutut tapahtumat. Järjestelmä voisi esimerkiksi veloittaa kyseistä asiakasta hänen tekemistään ostoksista.

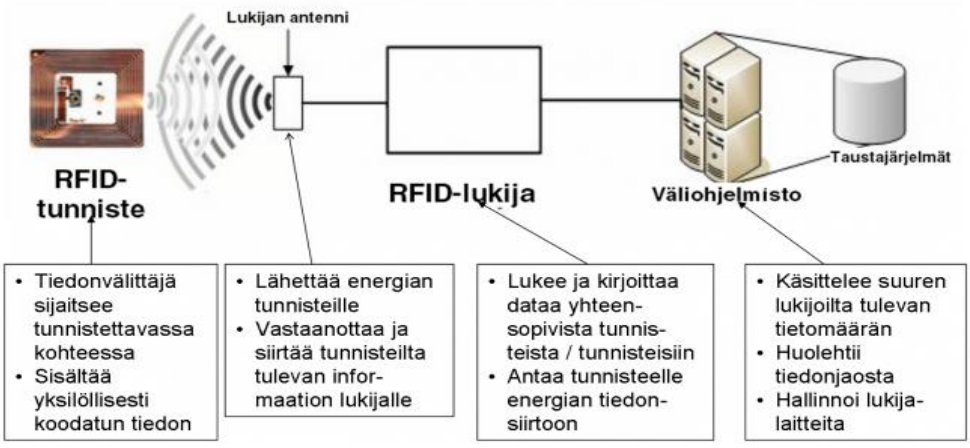
Tunnisteita on erilaisia, kuvassa 7 on kuvattu passiivisten, semi-passiivisten sekä aktiivisten tunnisteiden toiminnallisuuksien määrän suhdetta valmistushintaan.



Kuva 7. Kuvassa käydään läpi erilaisten tunnisteiden toiminnallisuuksia suhteessa hintaan [9].

RFID-tekniikassa käytetään niin passiivisia kuin aktiivisiakin tunnisteita. Tunnisteiden suurin ero on siinä, että passiivinen tunniste saa virtansa lukijalta, kun taas aktiivisessa tunnisteessa on mukana myös oma virtalähde. Aktiivisessa tunnisteessa lukuetäisyys voi kasvaa jopa yli 100 metriin, mutta myös tunnisteen tuotantohinta on passiivista tunnistetta korkeampi. Passiiviset tunnisteet ovat yleensä myös eliniältään pitkäkestoisempia. Myös passiivisille tunnisteille lukuetäisyys voi yltää jopa 10 metrin luokkaan, mikäli sekä lukijan että tunnisteen ominaisuudet tämän mahdollistavat. Ehkä tärkein huomio eroavaisuuksissa liittyy kuitenkin tietoturvaan. Aktiivinen tunniste on toki helpompi tietoturvasuojata ”älykkyytensä” ansiosta, kun taas passiivinen tunniste ”yksinkertaisuutensa” takia on paljon alttiimpi tietomurroille.

RFID-lukijat ovat yleensä liitettynä erinäiseen taustajärjestelmään ja sitä kautta tietokantaan. Esimerkiksi kouluni, Metropolian Leppävaaran yksikön, kaikkien luokkahuoneiden ovet ovat RFID-lukijoilla varustetuilla lukoilla lukittuja. Mikäli tämäkin lukijajärjestelmä olisi kytkettynä taustajärjestelmään, voisi siitä tulla huomattavia etuja. Esimerkiksi, kun näyttäisin koulun tunnisteen oven lukijalle, lukija lukisi tiedon tunnisteesta ja voisi tämän jälkeen tarkistaa taustajärjestelmästä, onko minulla juuri kyseiseen luokkahuoneeseen oikeus ja avaisi lopuksi oven, mikäli oikeus löytyisi ja olisi voimassa. Järjestelmä voisi antaa myös esimerkiksi hälytyksen, mikäli kyse olisi esimerkiksi pankkiholvista tai jostain muusta tarkoin varjellusta huoneesta, johon ei ulkopuolisilla olisi asiaa. Seuraavassa kuvassa 8 on kuvattu yleisellä tasolla RFID-järjestelmän toimintaperiaate.



Kuva 8. Kaaviokuva siitä, miten järjestelmä toimii, kun RFID-lukijaan on liitettynä taustajärjestelmä [9].

Edellä kuvatun kaltaisen taustajärjestelmän ansiosta on myös mahdollisuus seurata, ketkä koulussamme luokkahuoneita käyttävät ja milloin. Mikäli tapahtuisi varkaus tai vaikkapa irtaimistolle tehtäisiin ilkivaltaa, voitaisiin helposti lähteä taustajärjestelmän keräämiä tietoja hyväksi käyttäen rajaamaan kyseisen ajanhetken käyttäjät, jotka ovat huoneessa asioineet. Järjestelmät yleensä tallentavat tunnisteen historiatietoihin sekä paikan että ajan, mikä helpottaisi erityisesti tällaisten ongelmien ratkaisuissa.

3.3 RFID-tekniikan teoreettinen toiminta ja standardit

Passiivinen RFID-tunniste, joka työssänikin on käytössä, sisältää mikrosirun sekä antennin. Toiminnan kannalta keskeisessä osassa oleva antenni on käytännössä kuparisilmukka tai -silmukoita, riippuen määritetystä taajuudesta. Kun silmukka eli tunnisteen antenni, tuodaan lukijan antennin läheisyyteen, muodostavat ne yhdessä induktiivisen kytkennän sähkömuuntajan tapaan. Induktiivinen kytkentä mahdollistaa tiedon siirron oskilloivaa eli värähtelevää magneettikenttää moduloimalla, eli muuttamalla. Lukija luo värähtelevän magneettikentän vaihtovirralla antenniinsa, joten mikäli RFID-tunniste tuodaan tarpeeksi lähelle lukijan luomaa magneettikenttää, siirtyy virtaa tunnisteen kuparilangan muodostamaan käämiin, joka taas muodostaa tunnisteelle indusoituneella virralla virtalähteen. Virtalähde herättää mikrosirulta löytyvän EEPROM-muistin, josta löytyy tunnisteen sisältämä ID-tieto. Tunnisteelle siirtyneellä virralla tunniste myös pystyy lähettämään, eli moduloimaan magneettikenttää niin, että ID-tieto siirtyy lukijalle. Induktiivisessa kytkennässä myös tunnisteen asennolla suhteessa lukijaan voi olla merkitystä, koska magneettikentän modulaation täytyy päästä myös tunnisteelta lukijalle päin lukijan muodostaman magneettikentän lävitse. Käytännössä lukija pystyy yleensä lukemaan tunnisteen kuitenkin monestakin eri asennosta täysin moitteitta. [9.]

RFID-tekniikkaa on pyritty myös standardisoimaan lähinnä käyttäjäystävällisyyden takia. On ensiarvoisen tärkeää, että esimerkiksi suuressa logistiikkaketjussa järjestelmä toimii alihankkijalta toiselle aina samalla tavalla, vaikka käytössä olisi esimerkiksi eri laitevalmistajilta ostetut RFID-tunnisteiden lukijat. Laitevalmistajien kilpailun takaamiseksi RFID:stä löytyy myös vapaita standardeja, joita noudattaen kuka tahansa voi alkaa laitevalmistajaksi ja näin ollen muiden kilpailijaksi. Vuonna 1999 on myös perustettu Auto-ID Center -järjestö, jonka tarkoitus on kehittää ja ylläpitää

kansainvälisiin avoimiin logistiikkaketjuihin EPC (Electronic Product Code) -standardia sekä siihen läheisesti liittyvää teknologiaa. [17.]

RFID:n taajuusalueet

Matalan taajuusalueen, LF (Low Frequency), standardit ovat kaikki suojattuja, eivätkä näin ollen täysin vapaassa käytössä. Useimmat RFID-sovellukset ovat kulunvalvonnan järjestelmiä, jotka toteutetaan 125 kHz taajuudella. Esimerkiksi karjan tunnistukseen on määritelty ISO-standardit: ISO11784, joka määrittelee tunnisteen tietosisällön sekä ISO11785, joka määrittelee tiedonsiirtoprotokollan 134 kHz taajuudella. [17.]

Korkeammilla taajuuksilla, HF (High Frequency), on olemassa sovittuja standardeja taajuusalueella 13,56 MHz. Valitettavasti ISO14443-standardi ei takaa valmistajariippumatonta lukijoiden eikä tunnisteiden yhteensopivuutta eikä näin ollen tuo täysin standardisoinnin kaikkia etuja. Philips Mifare -tekniikka on kuitenkin kasvanut niin sanottuun ”de facto” -standardin asemaan, joten sen määrittelyjä pyritään aina ensisijaisesti noudattamaan. Mifarea käytetään esimerkiksi erilaisissa maksusovelluksissa ja muun muassa sen lukuetäisyys on rajattu 3–4 senttimetriin, joka itsestään selvästi ehkäisee tunnisteiden salaa lukemista. ISO15693-standardi taas on täysin laitevalmistajasta riippumaton ja takaa näin ollen yhteensopivuuden eri laitevalmistajien laitteiden ristiinkäytössä. Suomessa tunnetuin ja käytetyin ISO15693-standardia noudattava siru on Philips I-CODE SLI. [17.]

Erittäin korkealla taajuudella, UHF (Ultra High Frequency), on tällä hetkellä vain yksi olennainen standardi ISO18000-6C eli Gen2. Tämä standardi on EPC Global -järjestön kehittämä, ja sen ansiosta UHF-taajuusalueen toimintaa on pystytty parantamaan merkittävästi. Erityisesti niin sanottua monilukijaympäristön toimintavarmuutta on saatu parannettua. [17.]

3.4 RFID-tekniikan esimerkkisovelluksia

RFID-tekniikkaa hyödyntäviä sovelluksia on tällä hetkellä erittäin paljon erilaisia ja jatkuvasti kehitteillä on varmasti satoja. Seuraavassa esittelen lyhyesti pari käytännön esimerkkiä: elektroniset passit (e-Passports) ja matkakortit.

ICAO e-Passports

Toukokuussa 2004 International Civil Aviation Organization (ICAO) hyväksyi niin sanotun koneellisesti luettavan matkadokumenttien määrittelyn (”MRTDs” – Machine-Readable Travel Documents). MRTDs käyttää standardoitua RFID-tekniikkaa keräämään tietoja passeista, viisumeista ja matkakorteista. Seuraavassa kuvassa 9 on kaksi esimerkkiä erilaisista elektronisen passin lukijoista.

Kuva 9. Kuvassa ePassin pöytälukija sekä käsilukija ePassin kanssa [22, 23].

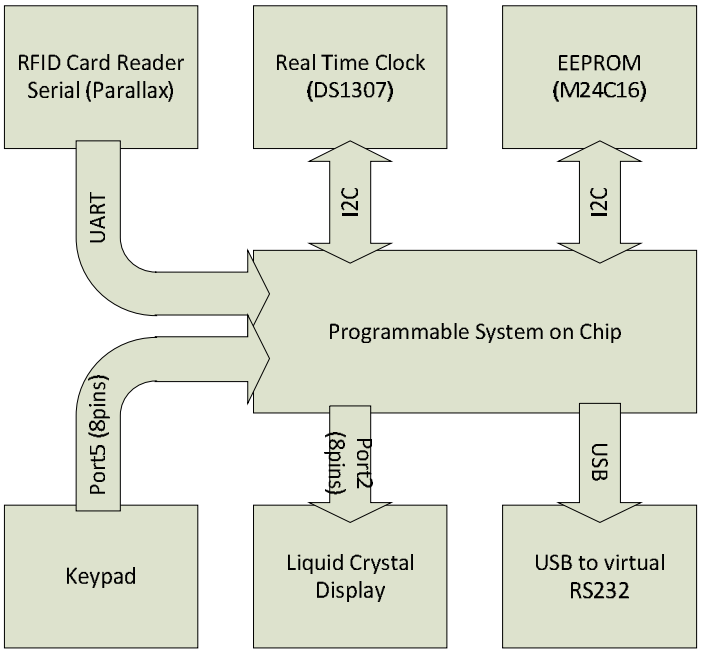
Matkakortit

Varmaankin lähes jokaisessa kaupungissa, tai ainakin suurimmissa, ympäri maailmaa pyritään esimerkiksi julkisen liikenteen maksujärjestelmät hoitamaan mahdollisimman pitkälle RFID-tekniikka hyväksi käyttäen. Näin järjestelmä on toteutettu myös Suomessa. Pääkaupunkiseudulla tuttu Helsingin seudun liikenteen (HSL) matkakorttijärjestelmäkin on tällainen. Kuten jokainen matkakortteja käyttänyt varmasti tietää, voidaan matkakorttiin tarvittaessa ladata latauspisteillä rahallista arvoa tai aikamäärätty kausiluontoinen lippu, jonka jälkeen esimerkiksi junassa tai bussissa korttia lukijalle näyttämällä voi leimata lippunsa. Kuvassa 10 on HKL:n lukijalaite.

Kuva 10. HSL:n matkakortti sekä lukija [16].

4 Toteutuksen fyysiset komponentit sekä käytetyt väylät

Työssä kehitetyn laitteen fyysinen toteutus koostuu pääosin viidestä eri komponentista: PSoC-mikro-ohjaimesta (Programmable System on Chip), RFID-lukijasta (Parallax: RFID Card Reader Serial), reaaliaikakellopiiristä (Real Time Clock, DS1307), EEPROM-muistista (M24C16) sekä LCD-näytöstä. Kuvan 11 lohkokaaviossa on esitetty myös toteutuksessa käytetyt väylät: UART (Universal Asynchronous Receiver and Transmitter), I2C (Inter-Integrated Circuit) sekä USB (Universal Serial Bus).



Kuva 11. RFID-lukijan lohkokaavio, jossa on kuvattu käytetyt komponentit sekä niiden väliset liitäntäväylät.

4.1 PSoC-mikro-ohjain

Fyysisen toteutuksen keskeisimmässä osassa oli C-kielellä ohjelmoitava Cypress'in valmistama mikro-ohjain mallia: Programmable System on Chip CY8C24894-24LFXI. Tämä M8C-prosessorilla varustettu mikro-ohjain valittiin lähinnä käytännön syistä, koska koululla oli valmiina kyseisellä piirillä varustettu testaus- ja kehitysalusta, josta löytyi myös laitteeseen suunniteltu USB-liitettävyys. Mikro-ohjaimen keskeisimpiä ominaisuuksia aina 24 MHz kellontaajuuteen yltävän prosessorin lisäksi on jopa 16 kilotavun ohjelmoitava FLASH-muisti. Lisäksi mikro-ohjaimessa on tarvittava määrä, testialustalla jopa 42, analogista tai digitaalista sisään- ja ulostulopinnejä, jotka ovat kaikki tarpeen mukaan käytettävissä ja ohjelmoitavissa. [18.]

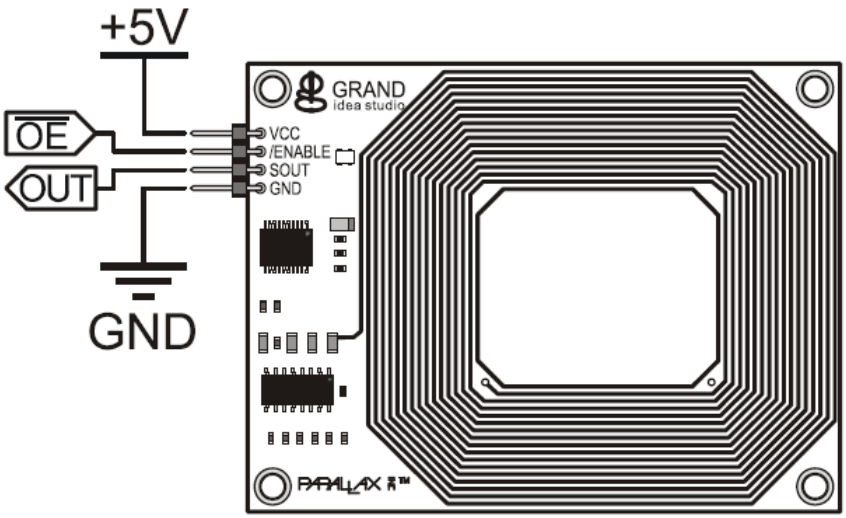
PSoC-mikro-ohjain tukee myös suuren määrän erilaisia valmiita toiminnallisuuksia, joita hallitaan PSoC Designer -ohjelmistolla suunnitteluvaiheessa. Mikro-ohjaimelle voi aktivoida halutun määrän erikseen määriteltyjä toiminnallisuuksia, esimerkiksi PWM-moduuleja (Pulse-Width Modulation), I2C-väylän, SPI-väylän (Serial Peripheral Interface), UART-väylän, erilaisia ajastimia ja laskureita sekä infrapunamoduuli. Valittavissa on kymmeniä erilaisia toiminnallisuuksia tarpeen mukaan. [18.]

4.2 Parallax RFID-lukija

RFID-tunnisteiden lukua varten valittiin lopulta huokea, alle 40 dollarin hintainen Parallaxin valmistama RFID-lukija (Parallax RFID Card Reader Serial), joka on tähän käyttöön erittäin soveltuvainen yksinkertaisen toiminnallisuutensa ansiosta. Käytännössä Parallaxin lukija vaatii toimiakseen vain käyttöjännitteen +5 volttia. Kun lukija on "enable"- eli aktiivitilassa, joka voidaan määrittää yhdellä pinnillä, antaa lukija automaattisesti sarjaliikenneväylän kautta (UART) lukemansa RFID-tunnisteen ID-numeron, mikäli tunniste tuodaan lukijan antennin välittömään läheisyyteen. Empiiristen tutkimusten mukaan lukija havaitsee tunnisteen jopa muutamien senttimetrien päästä.

Kuvasta 12 käy ilmi laitteen yksinkertainen toteutus. Kyseisestä laitteesta on saatavilla myös USB-liittimellä varustettu versio, mutta valittu UART-sarjaliikennettä tukeva versio oli tähän tarkoitukseen paremmin soveltuva, koska tarkoitus on lähettää haluttu ID-

tunnistetieto nimenomaan PSoC-mikro-ohjaimen käsiteltäväksi, jolloin kommunikointi UART-väylän kautta on järkevintä ja helpoin toteuttaa valmiiden PSoC-moduulien avulla.



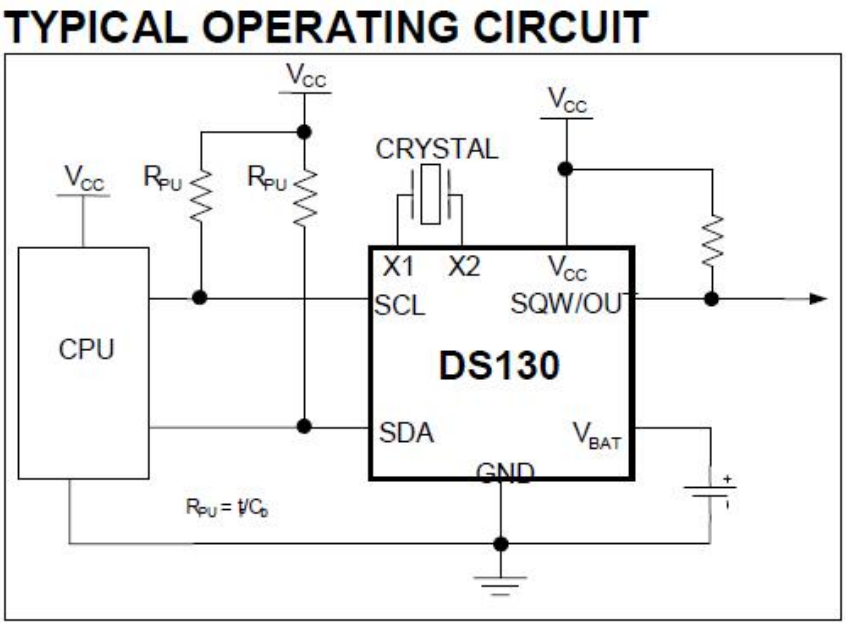
Kuva 12. Parallax RFID Card Reader Serial -piirikortti, jossa näkyy piirille integroitu antenni ja piirin komponentit sekä kuvaus kortin kytkennöistä [3].

4.3 Reaaliaikakello

Reaaliaikakellon olisi toki voinut toteuttaa myös PSoC-mikro-ohjaimesta löytyvien ominaisuuksien avulla, mutta päätimme yhdessä työn tilaajan kanssa jo määrittelyvaiheessa, että on viisaampaa lisätä erillinen RTC-piiri (Real Time Clock DS1307), joka voi mahdollisessa lopputuotteessa ylläpitää reaaliaikakelloa myös silloin, kun laite ei ole aktiivisessa käytössä. Erillinen RTC-piiri kuluttaa myös PSoC-mikro-ohjaimeen verrattuna vähemmän virtaa pelkän kellotoiminnon ylläpitämiseksi. Näin ollen valitsimme käyttöön DS1307-piirin, jonka kanssa mikro-ohjain voi yksinkertaisesti kommunikoida I2C-väylän kautta.

Sarjaliikennettä I2C-väylän ylitse tukeva DS1307-piiri on Maxim Dallas Semiconductor -yhtiön valmistama. Piiri ylläpitää BCD (Binary-coded decimal) -koodatusti yllä vuodenaikaa, kuukautta, kuunpäivää, viikonpäivää, tunteja, minuutteja ja sekunteja. Piiri on saatavilla usealla eri kotelointivaihtoehdolla, joista valitsimme prototyyppikehitykseen soveltuvimman eli 8-jalkaisen PDIP-koteloinnin, jota on helppo myös kytkentäalustalla testata. Kuvasta 13 selviää DS1307-piirin tyypillinen

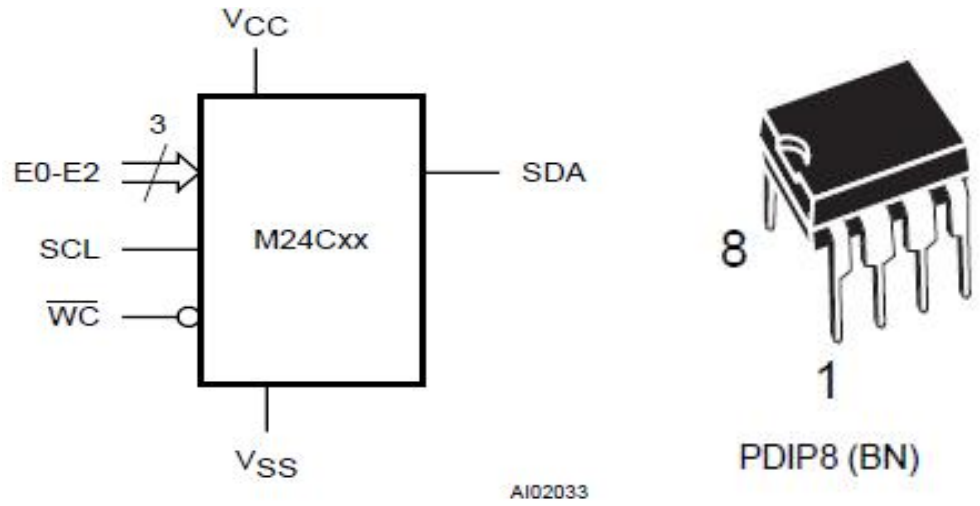
kytkentäkaavio. Toimiva kytkentä vaatii vain käyttöjännitteen (V_{CC}), kaksi ylösvetovastusta (R_{PU}) I2C -väylälle sekä ulkoisen oskillaattorikiteen (Crystal). [4.]



Kuva 13. DS1307-piirin tyypillinen kytkentäkaavio [4].

4.4 EEPROM-muistipiiri

Kehitettävän prototyypin määrittelyssä oli yhtenä tärkeimpänä kohtana tiedon säilyminen. Tämän takia valittiin erillinen EEPROM-muistipiiri, jolle tiedon tallennus I2C-väylän kautta oli mahdollista. Kaiken lisäksi kyseiseen kommunikointiväylään voi rinnastaa laitteita tarvittaessa vaikka kuinka paljon. Eri ”slave”- eli lisälaitteita samassa I2C-väylässä kutsutaan osoitteiden mukaan. Esimerkiksi EEPROM-muistipiirin osoite on valittavissa kolmesta eri vaihtoehdosta, joten sekä RTC-piiri että EEPROM-muistipiiri oli mahdollista liittää yhteen ja samaan I2C-kommunikointiväylään, mikä helpotti myös olennaisesti kytkentää. Itse kytkentä oli I2C-väylän EEPROM-muistipiirille yhtä yksinkertainen kuin samaan väylään kytketyn RTC-piirinkin. Käytännössä piiri tarvitsee vain käyttöjännitteen lisäksi ”$\overline{\text{WC}}$” (Write Control) -pinnin eli kirjoitusoikeuden asetuksen. Kirjoitusoikeuspinni on siis nolla-aktiivinen, ja mikäli halutaan, että muistipiirille voidaan kirjoittaa, täytyy tämä $\overline{\text{WC}}$ -pinni kytkeä jännitelähteen maapisteeseen. Kuvasta 14 selviää tarkemmin muistipiirin kytkentälogiikka. [5.]



Kuva 14. EEPROM-muistipiirin kytkentälogiikan kuvaus sekä esimerkki PDIP8-koteloidusta mikropiiristä [5].

4.5 Numeronäppäimistö

Laitteen toiminnallisuuden kannalta olennainen osa on myös numeronäppäimistö, jolla laitetta ”Action mode” -tilassa hallitaan. Koulun tarvikkeista löytyi tähän ratkaisuun sopiva ”matriisi-numeronäppäimistö”, jonka avulla sain toteutettua tarvittavia toiminnallisuuksia laitteeseen.

Numeronäppäimistön toiminta perustuu yksinkertaiseen matriisirakenteeseen. Mikro-ohjain skannaa ohjelman pyytäessä tietoja portilta 5, johon numeronäppäimistö on kytkettynä, ja pystyy havaitsemaan näin ollen, mikä näppäin milloinkin on painettuna. Kyseisestä prosessista on tarkempi kuvaus tämän dokumentin ohjelmointiosuuden esimerkissä.

4.6 Toteutuksessa käytetyt kommunikointiväylät

Toteutuksessa käytettiin kolmea eri kommunikointiväylää. Seuraavissa luvuissa avataan lyhyesti käytetyiden kommunikointiväylien historiaa, perusperiaatteita ja toiminnallisuuksia.

4.6.1 I2C-väylä

I2C-väylä (Inter-Integrated Circuit) on Philipsin jo 1980-luvulla kehittämä standardikommunikointiväylä erilaisille piireille. Nykyään kehityksestä vastaa NXP Semiconductors. (Viralliset kotisivut: http://www.i2c-bus.org/) I2C-väylän toiminnallisuus on erittäin yksinkertainen. Väylä vaatii toimiakseen vain kaksi johtoa laitteiden välille, ”SDA” (Serial Data Line) -johdon tiedon siirtoa varten sekä ”SCL” (Serial Clock Line) -johdon väylän kellotusta varten. I2C-väylä vaatii toimiakseen yhden ”Master”- eli hallintalaitteen, joka määrää linjalle kellojaksot ja pyytää ”Slave”-lisälaitteilta uniikkien osoitteiden ja muistipaikkojen avulla tietoa tai kirjoittaa niihin tietoa. Väylä myös tukee niin sanottua ”Multi-master” -toimintoa, eli väylässä voi olla monta isäntää, jotka hallitsevat väylää. [6.]

Sarjaliikenne on standardisoitu 8-bittiseksi kahden suunnan dataliikenteeksi, jolla on piirien ominaisuuksista riippuen valittavissa neljä eri nopeusluokkaa. ”Standard-mode” eli niin sanottu perusmoodi on nopeudella 100 kbit/s toimiva, ”Fast-mode” eli nopeampi toimii nopeudella 400 kbit/s, ”Fast-mode Plus” toimii nopeudella 1 Mbit/s nopeudella ja nopein eli ”High-speed mode” jopa nopeudella 3,4 Mbit/s. [6.]

Tärkeintä I2C-kytkennässä on muistaa ylösvetovastukset molemmille kytkentäjohdoille, sillä ilman niitä väylä ei toimi ollenkaan. Väylän käytännön toiminta perustuu siihen, että isäntälaite kirjoittaa aina ensin, myös lukuoperaatiossa, väylälle toisen laitteen osoitteen, jonka jälkeen kyseinen laite osaa vastata seuraavaan pyyntöön, joko kirjoitus- tai lukuoperaatioon. [6.]

Kuvassa 15 näkyy, miten I2C-väylän standardin mukainen kommunikointi toimii, kun hallintalaite kirjoittaa lisälaitteelle tietoa. Aloitusbitin jälkeen seuraa seitsemän bitin mittainen osoitetiedon kirjoitus sekä kirjoituskomento, johon lisälaite vastaa ”ACK”-kuittauksella olevansa valmis. Seuraavaksi hallintalaite siirtää tiedon lisälaitteelle. Tämän jälkeen lisälaite vastaa jälleen ”ACK” ja lopuksi hallintalaite lähettää väylälle lopetusbitin tiedoksi tiedonvälityksen valmistumisesta.

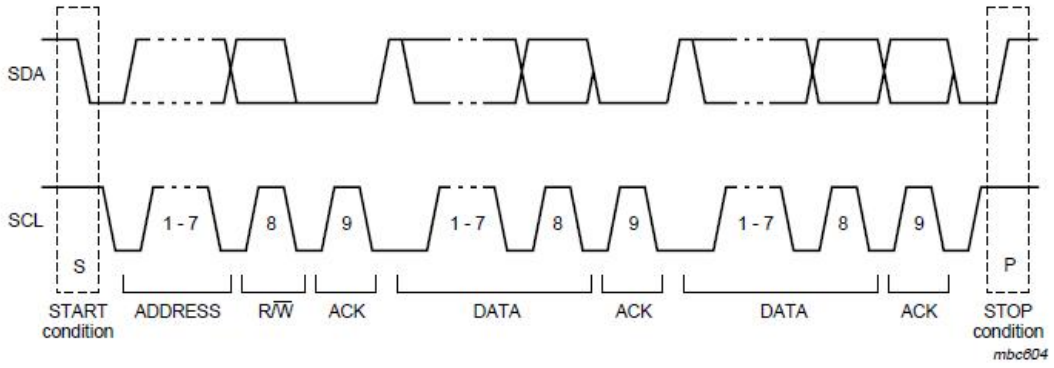


Fig 9. A complete data transfer

Kuva 15. I2C-väylän toiminta kellopulsseittain kuvattuna [6].

4.6.2 UART-väylä

UART-väylä (Universal Asynchronous Receiver and Transmitter) sopii myös RS-232-standardia noudattavien laitteiden kommunikointiväylän käsittelyyn, mikäli lähdelaitteessa, esimerkiksi juuri tämä Parallax RFID Card Reader Serial, on käytössä vain ”RX”, ”TX” ja ”CLK” -johtimet, eli vastaanotto-, lähetys- sekä kellojohdin.

4.6.3 USB-väylä

USB-väylätekniikka (Universal Serial Bus) on kohtuullisen uusi, vasta vuonna 1995 ensimmäistä kertaa julkaistu, standardisoitu tekniikka. Sen alkuperäiseen kehittäjäkokoonpanoon kuului ainakin nämä viisi yritystä: Compaq, DEC, IBM, Intel ja Microsoft. USB-laitteiden hienous piilee siinä, että ne eivät vaadi käyttöjärjestelmän uudelleenkäynnistystä, vaan asentavat ajurinsa ”plug-and-play”-menetelmällä, eli käytännössä itsestään heti laitteen kytkennän jälkeen, kunhan laite noudattaa standardia. USB -laitteet ovat myös niin sanotusti ”hot-swappable”, eli ne voidaan ottaa myös konetta sammuttamatta irti ja pois käytöstä. [7.]

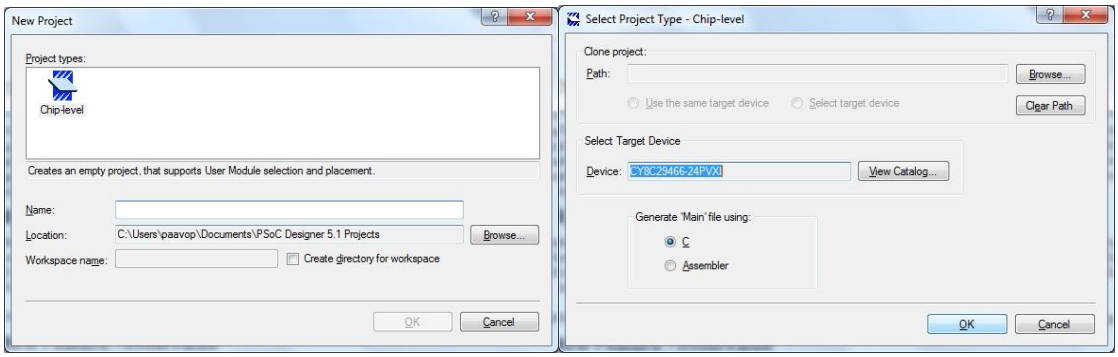
USB-väylätekniikka on tuonut myös aivan uudet siirtonopeudet. Ennen USB-tekniikkaa RS-232 standardin kautta nopeudet olivat parhaimmillaan luokkaa 115 kbit/s – 300 kbit/s ja jo ensimmäisellä laajalti käyttöön tullulla 1.1 USB -versiolla päästiin jopa 12 Mbit/s nopeuksiin. Nykyään USB 2.0 standardia noudattavat laitteet yltävät jopa 480Mbit/s nopeuteen. [7.]

5 Ohjelmistokehitys ja kehitysympäristö

Kuten jokainen sulautettu tietotekniikkasovellus, myös tämä kehitysprojekti vaati toimiakseen ohjelmallista kehitystyötä. Kaikki ohjelmointi tapahtui PSoC-mikro-ohjaimelle, jota ohjelmoitiin helppokäyttöisellä PSoC Programmer -ohjelmistolla. Graafisella käyttöliittymällä voitiin valita mikro-ohjaimelle halutut moduulit käyttöön, valittiin halutut kehitysalustan kytkentäpaikat liitäntälaitteille, kuten LCD-näytölle ja RFID-lukijalle sekä toteutettiin mikro-ohjaimelle ladattava pääohjelma.

5.1 PSoC Programmer -kehitysympäristö

PSoC Programmerin -kehitysympäristön käyttö on yksinkertaista ja sen voi halutessaan vapaasti ladata valmistajan sivuilta osoitteesta http://www.cypress.com/. Käyttäjä luo projektin automaattisen ohjauksen mukaisesti (liite 1), valitsee kohdeaseman, vaikkapa muistitikun, tiedon tallennukseen sekä käytettävän mikro-ohjaimen erillisestä listasta. Kun nämä on valittu, työstää ohjelmisto automaattisesti projektin ja luo sinne tarvittavat tiedostot sekä avaa tämän jälkeen projektityönäkymän, josta käyttäjä pääsee työstämään projektiin toiminnallisuuksia.



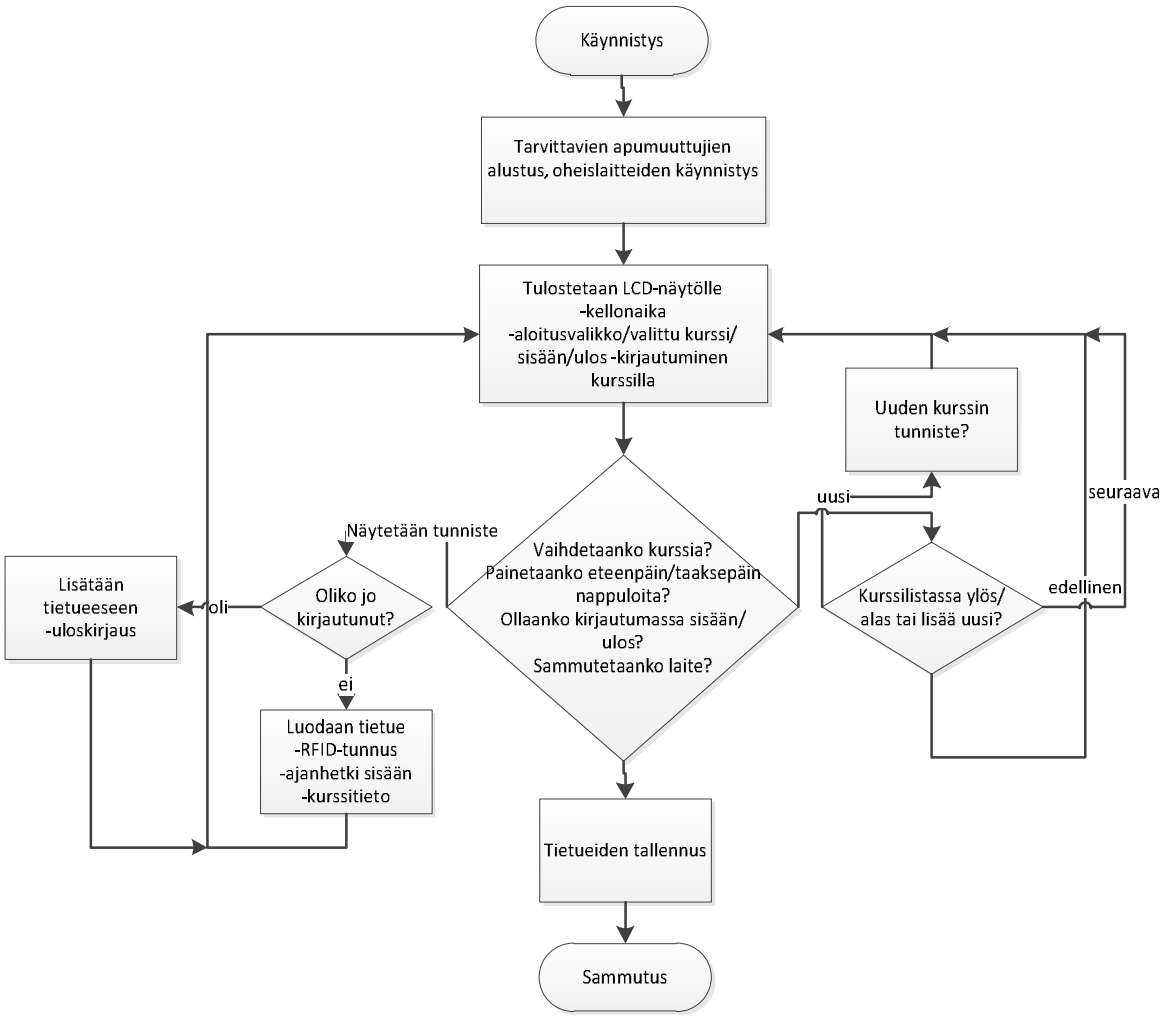
Kuva 16. Vasemmalla näkymä tallennuspaikan valinnasta ja oikealla mikro-ohjain mallin valinnasta, jossa valitaan myös käytettävä ohjelmointikieli, joko C-kieli tai Assembler, eli konekielinen kieli.

Mikro-ohjaimelle voidaan valita erilaisia komponentteja graafisessa käyttöliittymässä. (liite 2) Tätä projektia varten valittiin käyttöön UART-, USB- sekä I2C-väylät ja LCD-moduuli. Numeronäppäimistön napin painalluksen tarkistusskannaus suoritettiin porttiskannauksella, jonka koodiesimerkki esitellään yksityiskohtaisemmin myöhemmin tässä osuudessa.

Ohjelmallinen toiminnallisuus määritetään mikro-ohjaimen pääohjelmaan ”main.c”:hen, jonne voidaan kirjoittaa ihan tyypillinen C-kielinen koodi (liite 3). Vaihtoehtoisesti käyttäjä voi muokata mitä tahansa projektista löytyvää Assembler- eli konekielistä koodia toiminnallisuuden toteuttamiseksi. Käyttäjä voi myös halutessaan projektin luontivaiheessa valita Assembler-kielen C-kielen sijaan.

5.2 Pääohjelman kehitys

Pääohjelmaa lähdin kehittämään kuvan 17 vuokaavion pohjalta, jossa on käyty läpi pääohjelmatasolla laitteen toiminnallisuus vaiheittain. Kaaviossa kuvataan, kuinka käynnistyksen jälkeen tehdään ensin tarvittavat liitäntälaitteiden alustukset, jonka jälkeen ohjelma alkaa suorittaa pääohjelman ”while”-rakennetta.



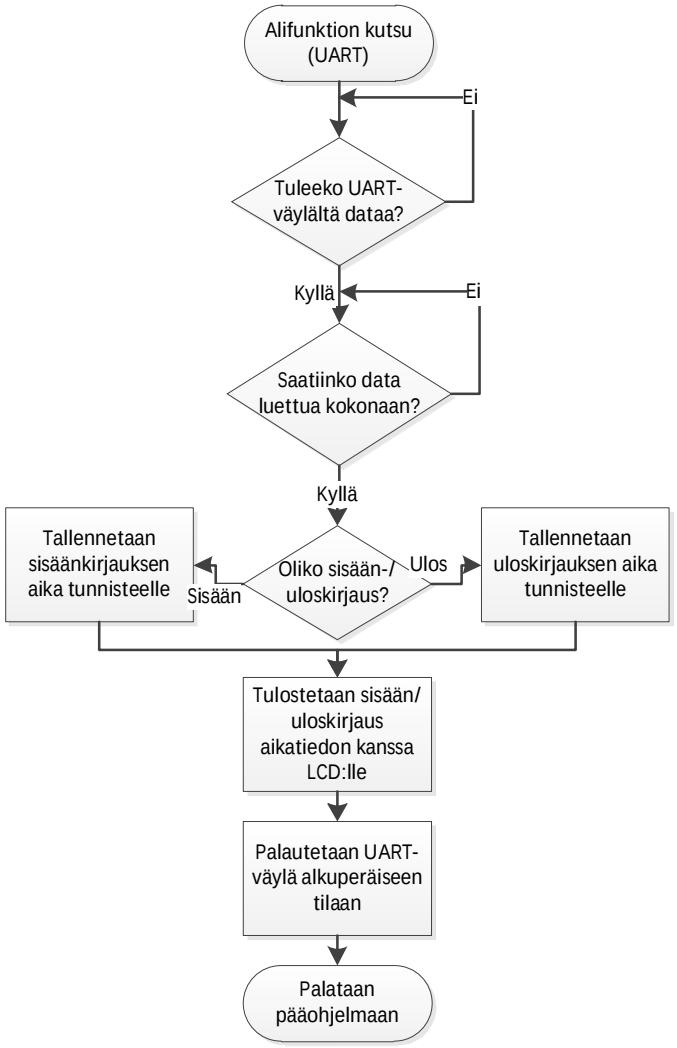
Kuva 17. Pääohjelman toiminnallisuus vuokaaviolla esitettynä.

Tarkoituksena oli luoda yksinkertainen toimintaperiaate edellä olevin ehdoin. Tärkeintä oli saada toteutettua toimiva valikkorakenne sekä tarvittavien tietueiden tallennus. Valikkorakenteen toteutin pääosin ”if”-, ”ifelse”- sekä ”switch-case” -rakenteita hyväksi käyttäen. Valikko on ikään kuin kaksiulotteinen matriisi, jossa pääsee liikkumaan valikosta seuraavaan, mikäli ensimmäinen ehto on toteutunut sallitusti. Esimerkiksi seuraavassa pääohjelman alifunktion lainauksessa (koodiesimerkki 1) tarkistetaan, minkä painikkeen käyttäjä on valinnut numeronäppäimistöltä sekä palautetaan tarkistettava arvo edelleen käsiteltäväksi. Lainauksessa koodi on mustalla tekstillä ja luettavuuden helpottamiseksi koodin kommentit ovat vihreällä. Esimerkkikoodi on myös keskeltä katkaistu, koska ”switch-case”-rakenteessa tässä kohtaa tehdään samanlaisia vertailuita monta, joissa vain palautettava muuttujan arvo vaihtuu ja jotka eivät tämän esimerkin kannalta ole tässä kohtaa oleellisia näyttää kokonaisuudessaan. Kokonainen pääohjelmakoodi on luettavissa ja vapaasti kopioitavissa sekä hyödynnettävissä kommentteineen liitteessä 4.

```
int check_keypad(void) // numeronäppäimistön tarkistusfunktio alkaa
{
PRT1DR|=0xF0; // toteutetaan matriisitarkistus numeronäppäimistölle, joten ajetaan 5 voltin jännite neljään ylimpään portin pinniin
PRT1DR&=0x0F;
prtBuf2=PRT1DR; // laitetaan vastinportin pinnien tulokset muistiin
PRT1DR|=0x0F; // tehdään matriisitarkistus numeronäppäimistölle, ajetaan 5v jännite neljään alimpaan porttiin
PRT1DR&=0xF0;
prtBuf = PRT1DR; // laitetaan vastinportin pinnien tulokset muistiin
keypad_result = prtBuf+prtBuf2; // tehdään aputulos vastinkappaleiden perusteella => arvo switch/case -rakenteeseen, josta palautetaan ko. napin arvo
switch (keypad_result)
                {
                case 0x22:
                return 0x01; // painonäppäin 1 painettuna, palautetaan arvo 0x01
                break;
                case 0x24:
                return 0x02; // painonäppäin 2 painettuna, palautetaan arvo 0x02
”pitkähkö koodin lainaus on katkaistu tältä kohtaa, koska samankaltaiset vertailut eivät ole olennaisia esimerkin osalta”
                default:
                return 0xFF; // oletuksena palautetaan 0xFF –arvo, mikäli näppäintä ei painettuna
                }
} // ------------ Numeronäppäimen tarkistusfunktio loppuu -----------------
```

Koodiesimerkki 1. Ote pääohjelmakoodista.

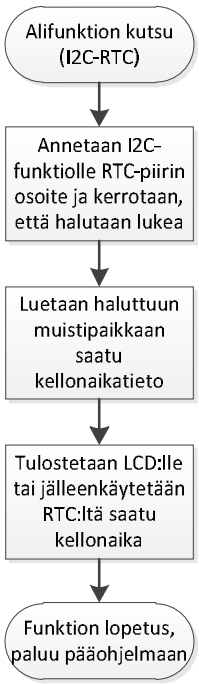
Numeronäppäimistön tarkistusalifunktion lisäksi ohjelmaan tehtiin muitakin tarpeellisia alifunktioita. Kehitetyt alifunktiot ovat UART-väylältä tulevan RFID-datan tarkistusfunktio, ajan tulostuksen funktio, ajan asetuksen funktio, uuden kurssin lisäyksen funktio, tietojen tallennusfunktio sekä USB-väylälle siirrettävän datan funktio. Seuraavaksi esittelen kaksi projektin kannalta tärkeintä ja olennaisinta alifunktiota vuokaavioittain. Laitteen toiminnallisuuden ja ylipäätään idean kannalta keskeisimmät osat ovat RFID-lukija sekä RTC-piiri, eli reaaliaikakelloa ylläpitävä piiri, jolta saadaan sisään- ja uloskirjauksissa tarkat aikaleimat.



Kuva 18. UART-alifunktion vuokaavio, josta käy ilmi funktion toiminnallisuus vaiheittain.

Kuten vuokaaviosta kuvassa 18 käy ilmi, mikäli laitteen pääohjelma kutsuu alifunktiota UART-väylän tarkistukseen, jää funktio aluksi odottamaan UART:lta, eli sarjaliikenteeltä tulevaa tietoa. Mikäli käyttäjä näyttää RFID-lukijalle tunnisteen, lähettää lukija automaattisesti tunnisteen ID-tiedon mikro-ohjaimelle sarjaliikenteellä, jonka tietoja funktio odottaa. Tämän jälkeen funktio tarkistaa, että varmasti koko tieto tulee

väylästä ja tämän jälkeen vertailee, oliko tunniste sisään- vai uloskirjautumassa. Lopuksi tieto tulostetaan LCD-näytölle aikaleiman kanssa.



Kuva 19. I2C-RTC-väyläkutsun vuokaavio.

PSoC-mikro-ohjain tarjoaa valmiin I2C-funktiokutsun I2C-väylälle, joten milloin ikinä ohjelmassa tarvitaan RTC-piiriltä haettua aikatietoa, annetaan vain tälle I2C-funktiolle RTC-piirin osoite ja käsky lukuoperaatioon tai kirjoitusoperaatioon. Esimerkiksi tässä kuvassa 19 esitetyssä vuokaaviossa ajan tulostusfunktiossa LCD:lle I2C-väylälle lähetetään ensin tarvittavat komennot, luetaan tieto tämän jälkeen muuttujaan ja lopuksi voidaan tulostaa aikatieto LCD:lle tämän muuttujan avulla.

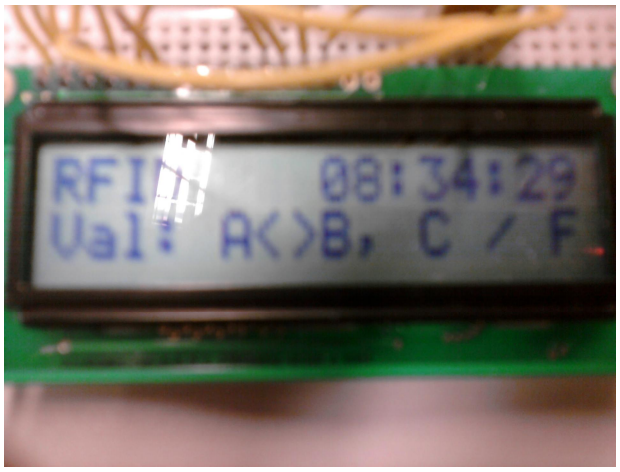
Muiden alifunktioiden tarkat toteutukset voi lukea tämän raportin lopussa olevasta liitteestä, jossa on luettavissa toteutettu pääohjelma kokonaisuudessaan kommentoituna. Pääohjelman käytännön kehitykseen perehdytään myös hieman lisää myöhemmin raportissa käytännön työn etenemisen osuudessa.

6 Käyttöohjeet

Tässä dokumentin osassa käyttäjä opastetaan käyttämään prototyypin käyttöliittymää, joka suppean LCD-näytön takia on erittäin lyhytsanainen ja suuntaa antava laitteen ”Action modessa”. Käyttöohjeiden päätarkoitus on kuvien ja selitysten avulla opastaa loppukäyttäjää käyttämään RIFD-lukijan käyttöliittymää. Käyttöliittymä on toteutettu matriisipohjaisesti sisäkkäisillä ehtolausekkeilla. Käyttäjä siis ikään kuin liikkuu valinnoissa joko oikealle tai vasemmalle, eteenpäin tai taaksepäin. Liikkumisessa käytetään näppäimiä "A", "B", "C", "D", "E", "F", jotka löytyvät numero-kirjain-näppäimistöltä.

6.1 Valikkorakenne

Alkuvalikossa käyttäjälle avataan typistetysti valintanäppäimet sekä tulostetaan reaaliaikakellopiirille asetettu kellonaika, joka päivittyy LCD:n yläreunaan reaaliajassa. Tämä kyseinen kellonajan tulostus näkyy myös monen muun valinnan aikana.



Kuva 20. RFID-lukijan aloitusruutu.

"Val: A<>B, C / F"-teksti aloitusruudulla (kuva 20) pyrkii kertomaan käyttäjälle, että valintanäppäiminä käytetään pääosin "A"-, "B"-, "C"- ja "F"-näppäimiä. "A"- ja "B"-näppäimiä käytetään lähinnä valikossa siirtymiseen sivuttain. "C"- ja "F"- näppäimet taas ovat ikään kuin taakse- ja eteenpäin -näppäimiä. Poikkeuksena kellonajan asetuksessa monimuotoisuudesta johtuen käytetään myös "D"- ja "E"-näppäimiä kellonajan määrittämiseen.

6.1.1 Kellonajan muuttaminen

Käynnistettäessä oletuksena RTC-piirin (Real Time Clock) kellonaika asettuu ennalta määrätyksi, mutta käyttäjä voi tarvittaessa muokata ajan asetusvalikosta. Käyttäjän tulee painaa "B"-näppäintä, kunnes LCD:llä lukee: "Aseta kellonaika?". Valinta aktivoidaan painamalla "F"-näppäintä. Valikkorakenteesta johtuen käyttäjältä kysytään vielä kertaalleen "C"- tai "F"-näppäimen painallusta, jonka jälkeen siirrytään itse kellonajan muokkaukseen.

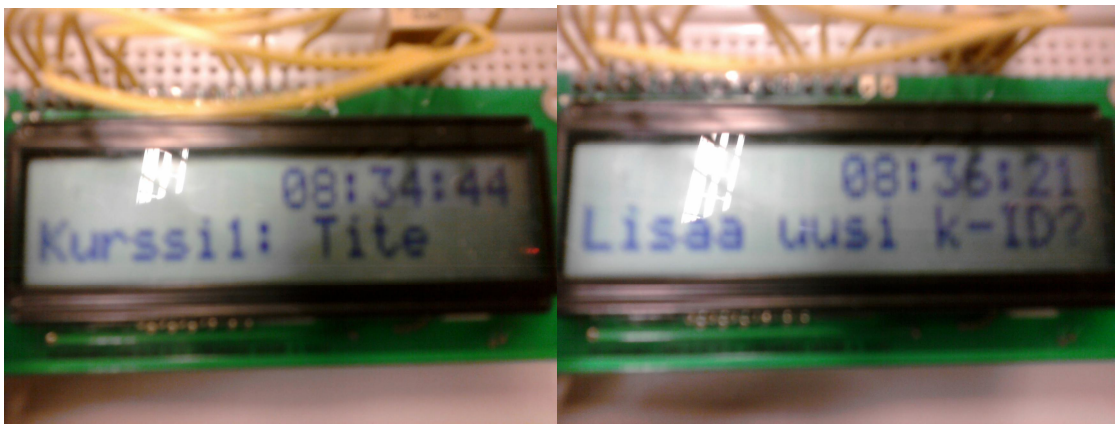
Kuva 21. Kellonajan muokkaus, valinta minuuttien kohdalla.

Kellonajan muokkauksessa (kuva 21) käyttäjä pääsee valitsemaan vuoden, kuukauden, viikonpäivän, tunnit, minuutit sekä sekunnit. Muokattavan valinnan kohdalle LCD:lle tulostuu aina kyseisen muuttujan lyhenne esimerkiksi vuoden kohdalla "YY" ja tuntien kohdalla "HH". Näin ollen käyttäjä pystyy havaitsemaan, mitä muuttujaa on kulloinkin muokkaamassa. Muuttuja valitaan "A"- ja "B"-näppäinten avulla vasemmalta oikealle. Muutettavaa muuttujaa joko lisätään "E"-näppäimellä tai vähennetään "D"-näppäimellä. Kun käyttäjä on mielestään valinnut oikean ajan, voi hän hyväksyä muokkaukset painamalla näppäintä "F". Mikäli käyttäjä haluaa perua muutokset, voi hän palata aikaa muuttamatta näppäimellä "C".

6.1.2 Kurssin valinta ja sisään- ja uloskirjautuminen

Käyttäjä voi "A"- ja "B"-näppäimiä painamalla valita valikosta joko valmiiksi asetetun kurssin tai lisätä haluamansa kurssi-ID:n, joka on maksimissaan nelinumeroinen luku. Mikäli halutaan valita olemassa oleva kurssi, esimerkiksi "Kurssi1: Tite" (kuva 22), siirrytään aloitusvalikosta kertaalleen "B"-näppäintä painamalla seuraavaan valikkoon

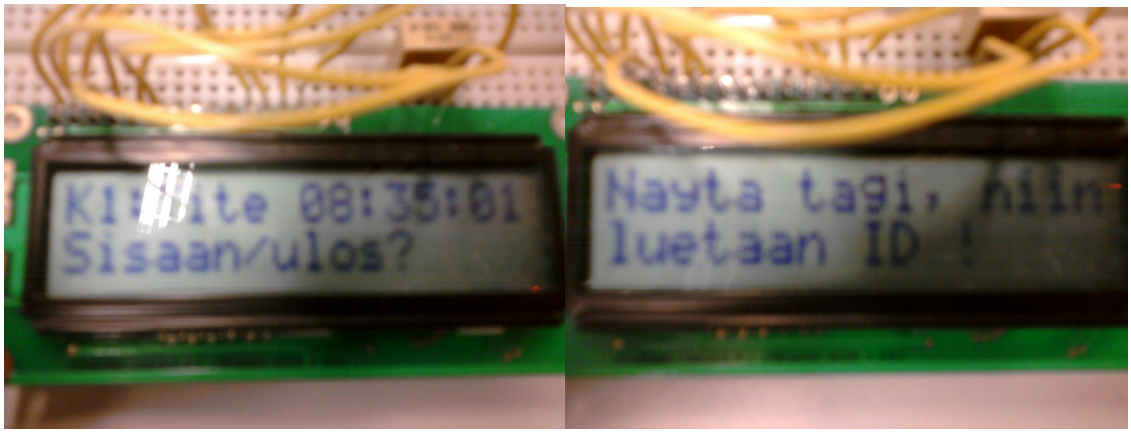
eli "Kurssi1: Tite":een. Tämän jälkeen valinnan vahvistamiseksi painetaan "F"-näppäintä.



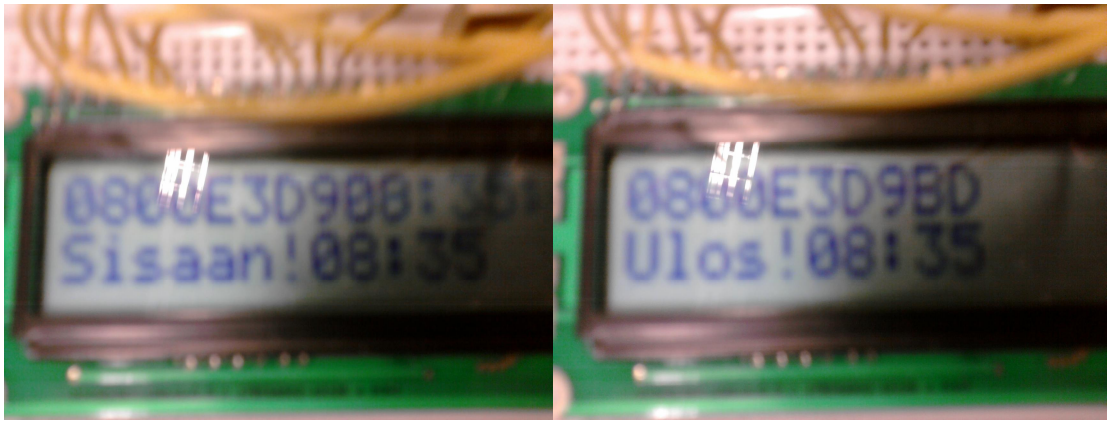
Kuva 22. Vasemmalla ennalta määritetyn kurssin valinta, oikealla uuden kurssi-ID:n lisäyksen valinta.

Mikäli käyttäjä valitsee lisättävän kurssi-ID:n, kysytään käyttäjältä seuraavassa vaiheessa uutta kurssi-ID:tä (kuva 22). Lopulta, kun käyttäjä on hyväksynyt kurssi-ID:n, käyttäjä palautuu automaattisesti kurssivalinnassa asetetun kurssin tilaan odottamaan sisään- tai uloskirjauksia.

Sisään- tai uloskirjaukset tapahtuvat "F"-näppäintä painamalla ja näyttämällä RFID-tunnistetta (kuva 23). Mikäli lukija havaitsee kyseisen tunnisteen ensimmäistä kertaa, olettaa se tulijan sisään kirjautujaksi. Jos taas lukija tietää jo ennestään tunnisteen kirjautuneen, luonnollisesti seuraava käyttäjän tunnistautuminen on tällöin uloskirjaus (kuva 24). Tämän takia sisään- tai uloskirjaus on toteutettu saman napin valinnalla.



Kuva 23. Vasemmalla laite odottaa mahdollista kirjausta, oikealla kirjausta on painettu ja laite odottaa luettavaa tunnistetta.



Kuva 24. Vasemmalla sisäänkirjauksen näkymä ja oikealla uloskirjauksen näkymä.

7 Yhteenveto

Dokumentin viimeisessä osiossa tehdään yhteenveto projektista, työtavoista, tavoitteista ja niiden toteutumisesta. Projektin yhteenvedon tarkoitus on myös avata lukijalle saavutetut tulokset sekä antaa viitteitä mahdollisista jatkotoimenpiteistä ja kehitysehdotuksista.

7.1 Käytännön työn eteneminen

Käytännön työt aloitettiin jo vuoden 2010 marras-joulukuun tienoilla. Työt alkoivat lähinnä ohjaavalta opettajalta saatuihin ohjeisiin tutustumisella sekä RFID-lukijan ominaisuuksia tutkimalla. Alkuperäinen käyttötarkoitukseen suunniteltu RFID-lukija oli Ib Technologyn valmistama Micro RWD Quad-Tag Reader, mutta muutaman viikon prototyyppitestauksen jälkeen kävi ilmi, että kyseinen piiri oli aivan liian monipuolinen ja vaikeahkosti hallittava lukija tähän projektiin, joten hylkäsimme tämän lukijan ja aloimme etsiä tilalle sopivampaa vaihtoehtoa.

Löysimme lopulta pienen etsiskelyn jälkeen Active Robots (http://www.active-robots.com) -verkkomyymälän kauppaaman Parallaxin valmistaman RFID Reader -lukijan, joka osoittautui kohtuuhintaiseksi ja juuri tähän projektiin soveltuvaksi lukijaksi.

Nopeasti projektin ja komponenttien testauksen edetessä kävi kuitenkin ilmi, että koulun käyttämät RFID-tunnisteet eivät sovellukaan juuri tämän RFID-lukijan kanssa kommunikoimaan, joten työ jäi näin ollen pääosin laitteen prototyypin kehitys- ja

tutkintatyöksi. Onneksi Parallaxin valmistaman RFID-lukijankin kanssa toimivia tunnisteita pystytään kuitenkin tilaamaan tarvittaessa lisää testikäyttöön esimerkiksi Active Robotsilta tai suoraan Parallaxilta. Tarkoituksena onkin testata lähitulevaisuudessa lukijan kanssa toimivilla tunnisteilla laboratoriotuntien seurantaa myös käytännössä.

Sopivan RFID-lukijalaitteen löydyttyä kehitystyö jatkui sopivan RTC-piirin sekä EEPROM-piirin etsinnällä. Tarkoitus oli hyödyntää PSoC-mikro-ohjaimen tukemaa I2C-väylää erillisen RTC-piirin ja EEPROM-piirin liittämiseen. RTC-piiri haluttiin erillisenä, vaikka mikro-ohjaimeen olisi toki voinut lisätä myös ohjelmallisen RTC-rutiinin, mutta haluttiin välttyä siltä, ettei reaaliaikakellon ylläpitäminen vie liikaa prosessorin laskenta-aikaa ja näin ollen hidasta tai estä muuta toiminnallisuutta. Näiden erillisen RTC-piirin sekä EEPROM-piirin lisääminen myös tavallaan yksinkertaisti ohjelmallista suunnittelua ja lopullisen prototyypin muistin käsittelyä. Esimerkiksi EEPROM-muistipiirille tietoa vietäessä kerrotaan I2C-väylälle väylältä löytyvän laitteen muistipaikka sekä muistille vietävän tiedon muistipaikka ja annetaan data, joka muistiin halutaan viedä. Näin ollen muun muassa muistiosoittimen hallinnasta ei tarvitse välittää, vaan EEPROM ylläpitää sitä automaattisesti. Lisäksi I2C-väylää hallitaan valmiilla PSoC-mikro-ohjaimelle suunnitellulla valmiilla funktiokutsulla, mikä helpotti ohjelman suunnittelua oleellisesti.

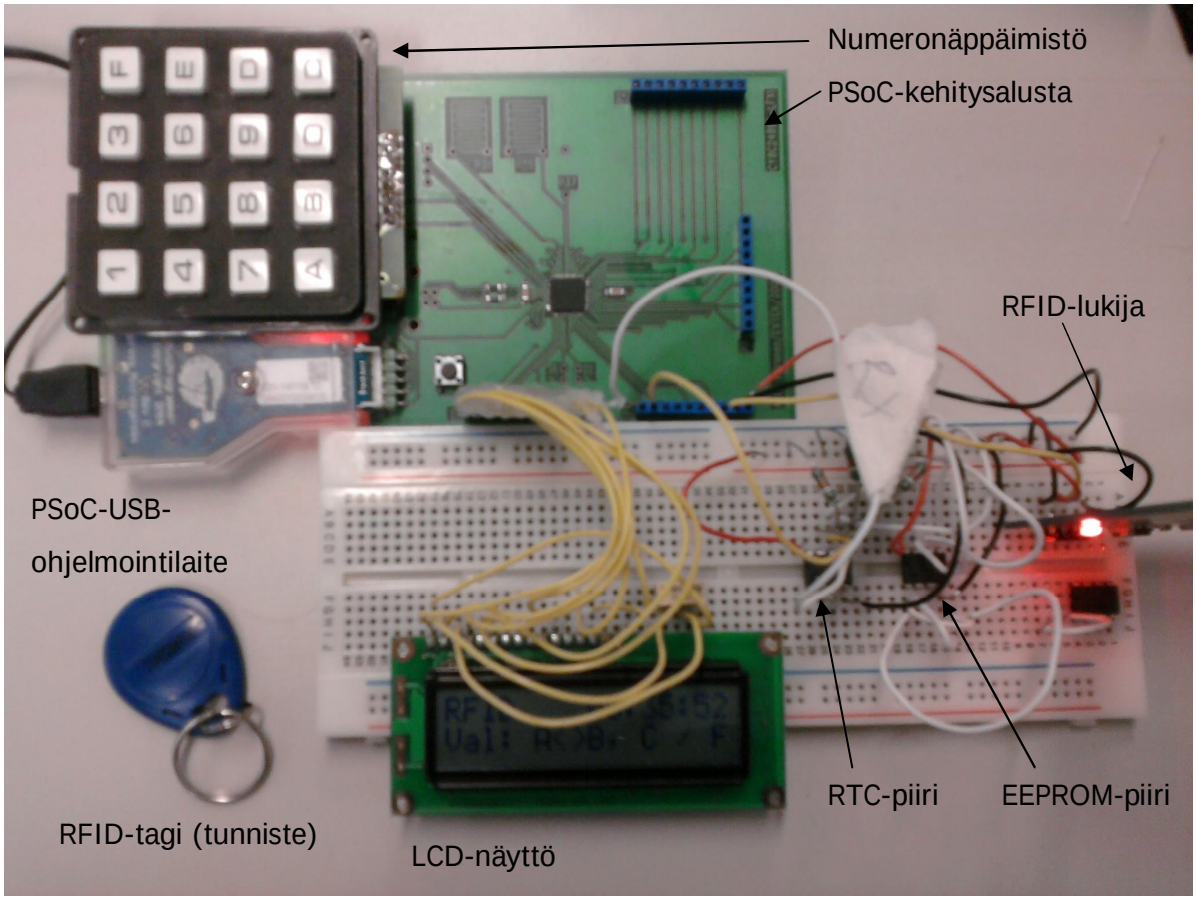
RFID-lukijan, RTC-piirin, EEPROM-piirin ja PSoC-mikro-ohjaimen kehitysalustan lisäksi koululta löytyi loput tarvittavat komponentit eli LCD-näyttö sekä numeronäppäimistö. Näillä komponenteilla voitiin rakentaa prototyyppitestaukseen sopiva laitteisto. Sopivan prototyyppialustan kasaamisen jälkeen jatkettiin ohjelmistopuolen kehityksellä ja testaamisella.

Pääohjelmaa kehitin yhden vaiheen kerrallaan. Aluksi testasin jokaisen komponentin perustoiminnan erikseen, eli esimerkiksi LCD-näytön toimivuuden lyhyellä esimerkkikoodilla, jossa tulostin tekstiä LCD-näytölle. Tämän jälkeen lisäsin mukaan RFID-lukijan, jolloin näin RFID-lukijalta UART-väylän kautta tulevan ID-tietueen LCD-näytöllä. Seuraavaksi tuli mukaan numeronäppäimistö, jonka avulla pystyin rakentamaan ”if”-, ”ifelse”- sekä ”swith-case” -rakenteilla toimivan valikon. Valikkorakenteen jälkeen mukaan tuli RTC-piirin ja I2C-väylän lisääminen. Vaihe

vaiheelta ja testaus testaukselta laite toiminnallisuuksineen kasvoi vastaamaan likimain alun perin määriteltyä ja työn tavoitteeksi asetettua prototyyppiä.

7.2 Työn lopputulos

Lopulta sain aikaan toimivan prototyypin, jossa on lähes tulkoon kaikki ne ominaisuudet, jotka olivat alkuperäisessä suunnitelmassakin mukana.



Kuva 25. Prototyyppi, PSoC-mikro-ohjaimen kehitysalusta, johon on liitetty ulkoiset komponentit: LCD-näyttö, RTC-piiri, EEPROM-piiri, RFID-lukija, numeronäppäimistö sekä ohjelmointiväline, jolla mikro-ohjaimelle viedään haluttu pääohjelma.

Lopullinen prototyyppi (kuva 25) sisältää käytännössä kaikki ”Action mode” -tilalle määritellyt ominaisuudet, jotka toimivat testatusti. Ainoastaan alun perin suunniteltu akkukäyttöisyys jäi toteutuksesta uupumaan. Laitetta voi kuitenkin käyttää, mikäli USB-johto on kytketty ja näin laite voi ottaa virran USB-kytkennän kautta. PC:lle kommunikoiva USB-väyläyhteys on likimain implementoitu, mutta sitä ei PC-tietokoneen ajurien puutteen vuoksi päästy käytännössä testaamaan.

7.3 Analyysi projektin onnistumisesta

Projektin kanssa työskentely on kokemuksena mielestäni ainutkertainen. Olen tuntuvasti kehittynyt ohjelmoijana sekä saanut paljon lisää ymmärrystä esimerkiksi erilaisten kommunikointiväylien käytännön toiminnasta. Lisäksi projektissa oppi niin sanotusti kantapään kautta, miten paljon suunnittelulla ja aikataulutuksella on tällaisessa projektissa merkitystä.

Positiivisimpia ja projektia edesauttavia asioita oli aikaisempi tuntemus ja käyttökokemus PSoC Designer -kehitysympäristöstä, mikä vauhditti ohjelman kehitystä todella paljon. PSoC Designerista löytyi myös paljon lyhyitä esimerkkikoodeja esimerkiksi väyläkommunikaation toteutuksesta, mistä sain erittäin suurelta osin apua ohjelman toteutukseen.

Lopulliseen prototyyppiin olen melko tyytyväinen, sillä tavoitteet saatiin lähes kaikilta osin saavutettua ja omiin henkilökohtaisiin tavoitteisiin kuului uusien asioiden oppiminen sekä työskentelytapojen parantaminen, joten myös näiden kannalta projekti oli mielestäni onnistunut.

7.4 Jatkokehitysideat

Projektin fyysistä toteutusta ja kehitettyä ohjelmakoodia saa puolestani kuka tahansa haluamallaan tavalla kehittää edelleen, mutta mitä todennäköisimmin prototyyppiä halutaan testata ainakin koulun laboratorio-olosuhteissa, mikäli halukkaita opiskelijoita löytyy laitetta ja käytäntöä testaamaan. Toki laite tulisi mielestäni vielä viedä lopulliseen kannettavaan ja käyttökelpoisempaan muotoon ennen testausta.

Prototyypin rinnalle olisi myös hyvä kehittää PC-tietokoneelle asennettava ohjelmisto, joka tukisi sarjaliikenneyhteyttä ja voisi tätä kautta kommunikoida laitteen kanssa. Olisi mielestäni erinomaista, jos PC-tietokoneelta voisi suoraan muokata sarjaliikenteen välityksellä laitteelta löytyviä kurssitietoja. Kursseja voisi lisätä, poistaa, asettaa kurssien ajankohtia ja niin edelleen. Lukijalta tulisi voida myös hakea osallistujalistoja esimerkiksi kurssien, käyttäjä-ID:n ja aikaperusteella.

Nykyaikaisiin RFID-lukijoihin on monesti liitettynä erilaisia taustajärjestelmiä, ja pohdin myös ideaa, että opiskelijat voisivat halutessaan seurata reaaliajassa Internetistä esimerkiksi omaa ajankäyttöään osallistumillaan kursseilla. Laite voisi siis hyvinkin olla myös Internet-yhteydellä, esimerkiksi matkapuhelimista tutulla 3G-yhteydellä varustettu. Mikäli laitteessa olisi aina käytettävissä Internet-yhteys, voisi se jatkuvasti päivittää tietonsa reaaliaikaisesti halutulle sivustolle, josta opiskelijat voisivat hakea ja tarkastella tietojaan. Tämä automaattisuus vähentäisi myös laitteelta tuotavien tietojen käsittelyyn menevää aikaa ja näin ollen helpottaisi tuntuvasti yhtä tämän projektin alkuperäistä tavoitetta, eli vähentää opettajien ylimääräistä manuaalista työtä.

Lähteet

1 Kuvalähde: Office 2010 Clip Art -kuva-arkisto. ”Pen and paper”. Lainattu 29.4.2010.

2 Kuvalähde: <http://www.kaba.co.nz/media/34150/v2/ImageFile/rac-rfid-reader.jpg>. Kuva RFID-lukijasta ja -tunnisteesta. Lainattu 29.4.2010.

3 Parallax RFID Reader documentation. 2010. Verkkodokumentti. <http://www.parallax.com/Portals/0/Downloads/docs/prod/audiovis/28140-28340-RFIDreader-v2.2.pdf>. Luettu 14.4.2011.

4 Maxim DS1307 Real Time Clock. 2008. Verkkodokumentti. <http://datasheets.maxim-ic.com/en/ds/DS1307.pdf>. Luettu 13.4.2011.

5 STMicroelectronics M24C EEPROM documentation. 2004. Verkkodokumentti. <http://pdf1.alldatasheet.com/datasheet-pdf/view/98272/STMICROELECTRONICS/M24C16-WMN6TP.html>. Luettu 16.4.2011.

6 I2C-bus Specification and User Manual. 2007. Verkkodokumentti. <http://www.nxp.com/documents/user_manual/UM10204.pdf>. Luettu 17.4.2011.

7 Standards and specs: The ins and outs of USB. 2005. Verkkodokumentti. <http://www.ibm.com/developerworks/power/library/pa-spec7.html>. Luettu 20.4.2011.

8 Roussos, George. 2008. Networked RFID: Systems, Software and Services. Springer-Verlag London Limited.

9 RFIDLab Finland Ry. RFID-tekniikan perusteet. Verkkodokumentti. <http://www.rfidlab.fi/rfid-tekniikan-perusteet>. Luettu 1.5.2011.

10 RFIDLab Finland Ry. RFID-tekniikan tietoutta. Verkkodokumentti. <http://www.rfidlab.fi/rfid-tietoutta>. Luettu 1.5.2011.

11 Rida Amin, Yang Li, Tentzeris Manos. 2010 RFID-Enabled Sensor Design and Applications. Artech House.

12 Kuvalähde: <http://www.amstoregroup.com.au/images/barcode_details_ean-13_470.jpg>. Kuva viivakoodista. Lainattu 3.5.2011.

13 Kuvalähde: <http://www.bristol.ac.uk/online-payments/help/images/cvv2.jpg>. Kuva luottokortista. Lainattu 3.5.2011.

14 Kuvalähde: <http://www.psdgraphics.com/file/credit-card.jpg>. Kuva luottokortista. Lainattu 3.5.2011.

15 Kuvalähde: <http://www.internationalphonecalls.org/images/nomi_sim_card_chip.jpg>. Kuva SIM-kortista. Lainattu 3.5.2011.

16 Kuvalähde: <http://fi.wikipedia.org/wiki/Tiedosto:Matkakortti_ja_kortinlukija.jpg>. Kuva matkakortista ja lukijasta. Lainattu 3.5.2011.

17 RFIDLab Finland Ry. RFID-tekniikan standardit. Verkkodokumentti. <http://www.rfidlab.fi/rfid-standardit>. Luettu 3.5.2011.

18 Cypress PSoC® Programmable System-on-Chip™. CY8C24894 documentation. Verkkodokumentti. <http://www.cypress.com/?docID=26676>. Luettu 29.4.2011.

19 Kuvalähde: <http://image.made-in-china.com/2f0j00TMlQEkrzvZoc/SGS-Approved-RFID-Tag.jpg>. RFID-tunniste. Lainattu 5.5.2011.

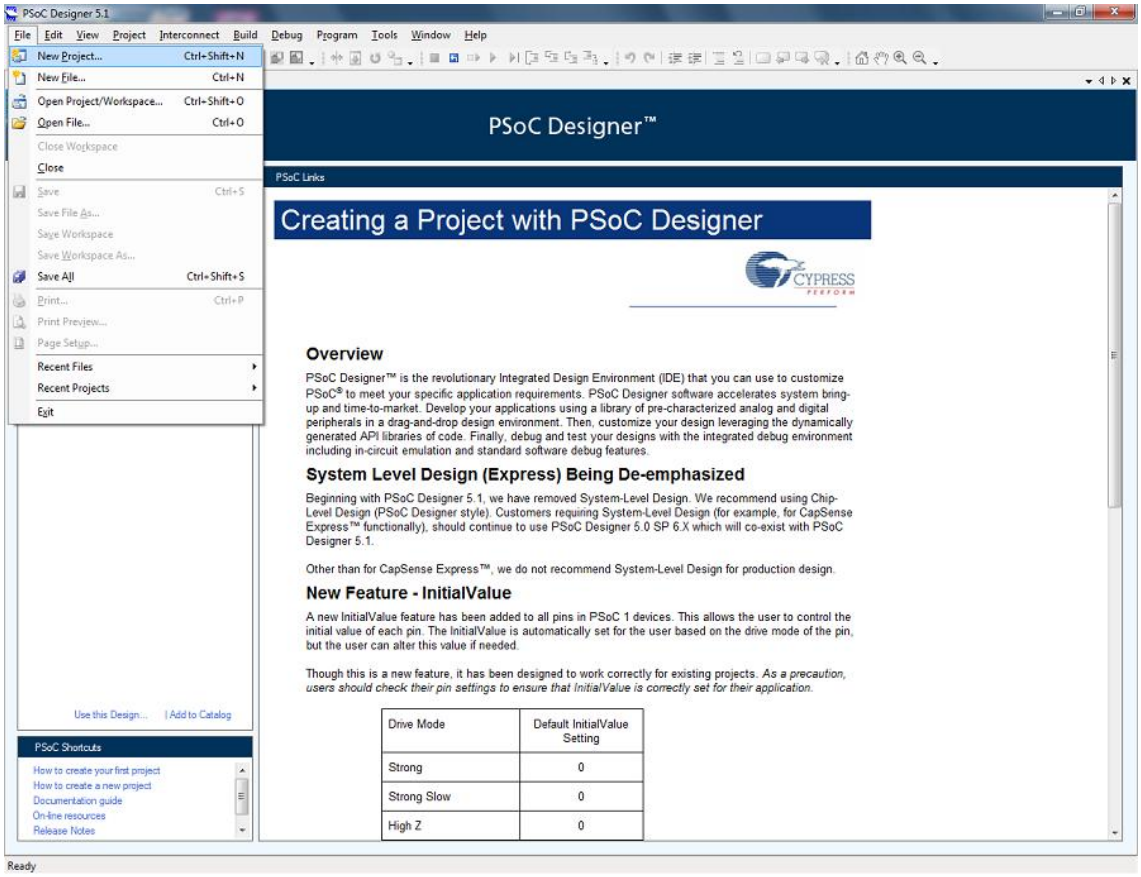
20 Kuvalähde: <http://www.cards-leader.com/UploadFiles/20101120104826179.jpg>. RFID-tunniste. Lainattu 5.5.2011.

21 Kuvalähde: < http://www.tech-faq.com/wp-content/uploads/images/Radio-Frequency-IDentification.jpg>. RFID-tunniste. Lainattu 5.5.2011.

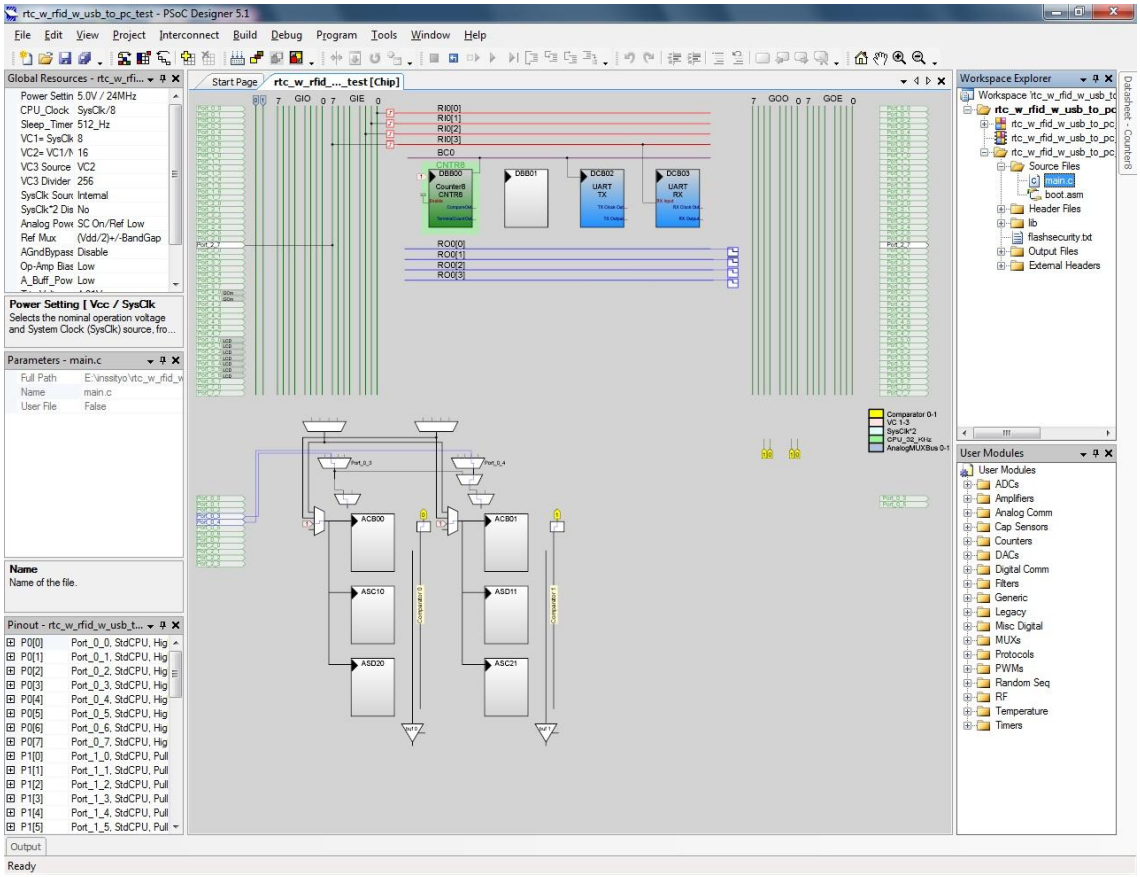
22 Kuvalähde: <http://i00.i.aliimg.com/photo/v1/338965314/IPT500_Handheld_e_Passport_Terminal.jpg>. RFID-lukija ja ePassport. Lainattu 5.5.2011.

23 Kuvalähde: < http://isecuretech.com/yahoo_site_admin/assets/images/SER100.15432854_std.jpg>. RFID-lukija. Lainattu 5.5.2011.

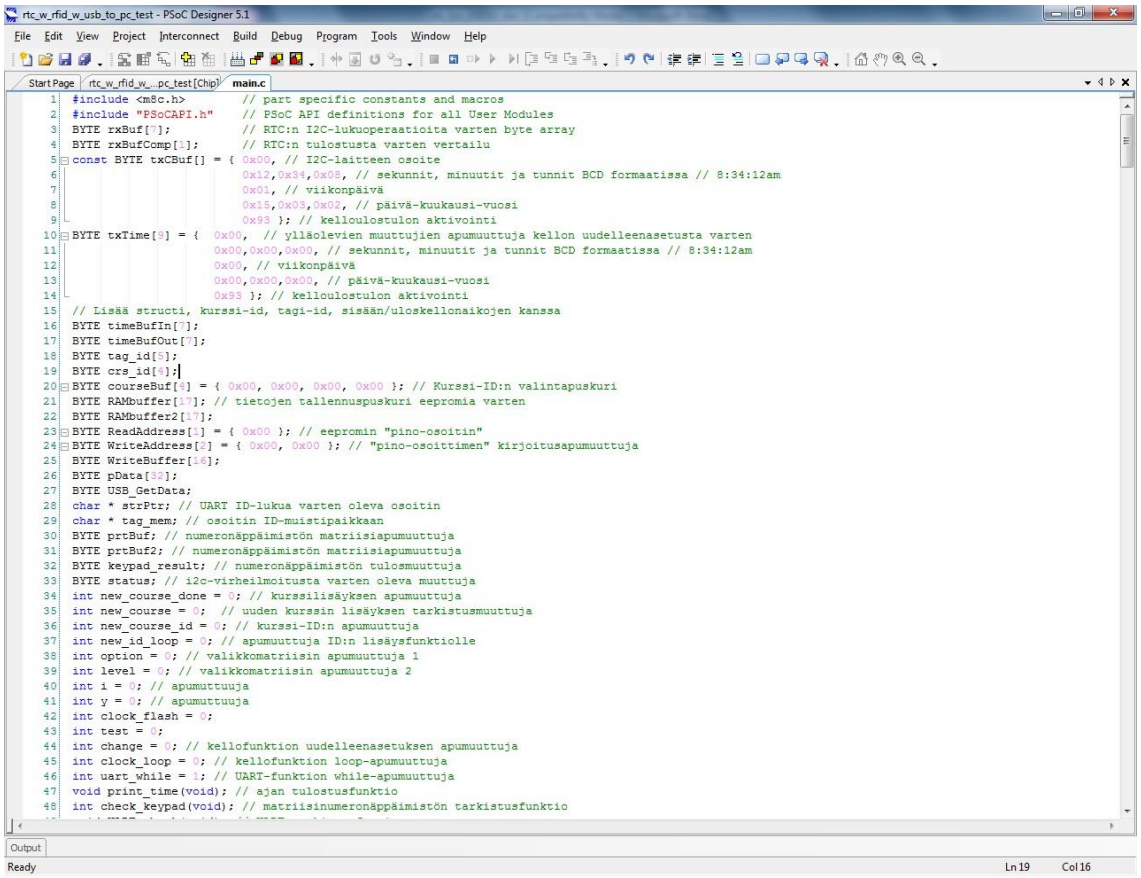
PsoC Designerin aloitussivu



PSoC Designer Chip Levelin näkymä



PSoC Designer pääohjelman näkymä



Pääohjelman koodi

```
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
BYTE rxBuf[7]; // RTC:n I2C-lukuoperaatioita varten byte array
BYTE rxBufComp[1];      // RTC:n tulostusta varten vertailu
const BYTE txCBuf[] = { 0x00, // I2C-laitteen osoite
0x12,0x34,0x08, // sekunnit, minuutit ja tunnit BCD formaatissa // 8:34:12am
0x01, // viikonpäivä
0x15,0x03,0x02, // päivä-kuukausi-vuosi
0x93 }; // kelloulostulon aktivointi
BYTE txTime[9] = {0x00,  // yllä olevien muuttujien apumuuttuja kellon uudelleenasetusta varten
0x00,0x00,0x00, // sekunnit, minuutit ja tunnit BCD formaatissa // 8:34:12am
0x00, // viikonpäivä
0x00,0x00,0x00, // päivä-kuukausi-vuosi
0x93 }; // kelloulostulon aktivointi
BYTE timeBufIn[7];
BYTE timeBufOut[7];
BYTE tag_id[5];
BYTE crs_id[4];
BYTE courseBuf[4] = { 0x00, 0x00, 0x00, 0x00 }; // Kurssi-ID:n valintapuskuri
BYTE RAMbuffer[17]; // tietojen tallennuspuskuri eepromia varten
BYTE RAMbuffer2[17];
BYTE ReadAddress[1] = { 0x00 }; // eepromin "pino-osoitin"
BYTE WriteAddress[2] = { 0x00, 0x00 }; // "pino-osoittimen" kirjoitusapumuuttuja
BYTE WriteBuffer[16];
BYTE pData[32];
BYTE USB_GetData;
char * strPtr; // UART ID-lukua varten oleva osoitin
char * tag_mem; // osoitin ID:n muistipaikkaan
BYTE prtBuf; // numeronäppäimistön matriisiapumuuttuja
BYTE prtBuf2; // numeronäppäimistön matriisiapumuuttuja
BYTE keypad_result; // numeronäppäimistön tulosmuuttuja
BYTE status; // i2c-virheilmoitusta varten oleva muuttuja
int new_course_done = 0; // kurssilisäyksen apumuuttuja
int new_course = 0;  // uuden kurssin lisäyksen tarkistusmuuttuja
int new_course_id = 0; // kurssi-ID:n apumuuttuja
int new_id_loop = 0; // apumuuttuja ID:n lisäysfunktiolle
int option = 0; // valikkomatriisin apumuuttuja 1
int level = 0; // valikkomatriisin apumuuttuja 2
int i = 0; // apumuttuuja
int y = 0; // apumuttuuja
int clock_flash = 0;
int test = 0;
int change = 0; // kellofunktion uudelleenasetuksen apumuuttuja
int clock_loop = 0; // kellofunktion loop-apumuuttuja
int uart_while = 1; // UART-funktion while-apumuuttuja
void print_time(void); // ajan tulostusfunktio
```

```
int check_keypad(void); // matriisinumeronäppäimistön tarkistusfunktio
void UART_check(void); // UART-tarkistusfuntio
void set_new_time(void); // uuden ajan määritysfunktio
void add_new_course(void); // uuden kurssin lisäysfunktio
void save_information(void); // tiedon eepromille tallennusfunktio
void USB_PrintData(void);
void main(void) // pääohjelma 1
{
// Alustukset
UART_CmdReset(); // Alustetaan UART puskuri
UART_IntCntl(UART_ENABLE_RX_INT); // Käynnistetään RX-keskeytykset
Counter8_WritePeriod(155); // Kellogeneraattorin tahdistus
Counter8_WriteCompareValue(77);
Counter8_Start(); // Kellogeneraattorin käynnistys
UART_Start(UART_PARITY_NONE); // Käynnistetään UART
M8C_EnableGInt ; // Keskeytykset päälle
SleepTimer_Start(); // Käynnistetään odotukset
SleepTimer_SetInterval(SleepTimer_8_HZ); // alustetaan torkkuaika
SleepTimer_EnableInt();
LCD_Start(); // alustetaan LCD-moduuli
   I2Cm_Start(); // Alustetaan I2C-moduuli
I2Cm_bWriteCBytes(0x68,txCBuf,9,I2Cm_CompleteXfer); // lähetetään I2C:n ylitse RTC-piirille aika
USBUART_Start(USBUART_5V_OPERATION); //Start USBUART 5V operation
PRT2DR&=0xFF;
//while(!USBUART_Init()); //Wait for Device to initialize
while(1)
{
print_time(); // tulostetaan LCD-näytölle RTC:n aika
SleepTimer_SyncWait(0x03, SleepTimer_WAIT_RELOAD); // option/level valikkomatriisi, jonka avulla hallitaan valintoja
if (check_keypad()!= 0xFF) // mikäli valikkomatriisilla on painettuna joku näppäin, tarkistetaan ehtojen mukaisesti
{
if (check_keypad() == 0x0A && option<=4 && option>0 && level==0) // tarkistetaan valikkorakenteen nykyhetki ja
painallus
{
option = option-1; // mikäli sallittua, siirrytään hierarkiassa alaspäin
}
else if (check_keypad() == 0x0B && option>=0 && option<4 && level==0) // tarkistetaan valikkorakenteen nykyhetki
ja painallus
{
option = option+1; // mikäli sallittua, siirrytään hierarkiassa ylöspäin
}
else if (check_keypad() == 0x0C && level<=2 && level>0) // tarkistetaan valikkorakenteen nykyhetki ja painallus
{
level = level-1; // mikäli sallittua, siirrytään hierarkiassa alaspäin
}
else if (check_keypad() == 0x0F && level>=0 && level<2) // tarkistetaan valikkorakenteen nykyhetki ja painallus
{
```

```
level = level+1; // mikäli sallittua, siirrytään hierarkiassa ylöspäin
}
}
else if (level==0)
{   // Tulostetaan valikkorakenne
switch (option)
{
case 0: // aloitusvalikko
LCD_Position(0,0);
LCD_PrCString("RFID              ");
LCD_Position(1,0);
LCD_PrCString("Val: A<>B, C / F  ");
break;
case 1: // kovakoodattu kurssi 1
LCD_Position(0,0);
LCD_PrCString("                  "); // tyhjennetään vanhat LCD-tulosteet
LCD_Position(1,0);
LCD_PrCString("Kurssi1: Tite     "); // tulostetaan kurssivalinta
break;
case 2: // kovakoodattu kurssi 2
LCD_Position(0,0);
LCD_PrCString("                  "); // tyhjennetään vanhat LCD-tulosteet
LCD_Position(1,0);
LCD_PrCString("Kurssi2: Titu     "); // tulostetaan kurssivalinta
break;
case 3: // uuden ID:n lisäysvalikko
LCD_Position(0,0);
LCD_PrCString("                  "); // tyhjennetään vanhat LCD-tulosteet
LCD_Position(1,0);
LCD_PrCString("Lisaa uusi k-ID?");
break;
case 4: // kellonajan uudelleenmääritysvalikko
LCD_Position(0,0);
LCD_PrCString("                  "); // tyhjennetään vanhat LCD-tulosteet
LCD_Position(1,0);
LCD_PrCString("Aseta kellonaika!?");
break;
default: // mikäli valikkorakenne hajoaa tulostetaan virhe ja poistutaan switch/case-rakenteesta
LCD_Position(0,0);
LCD_PrCString("Virrrrhe!");
break;
}
}
else if (level==1) // tarkistetaan hierarkinen taso
{
switch (option) // otetaan valinta huomioon
{
```

```
case 0: // ensimmäisessä valinnassa eteenpäin siirtyminen aiheuttaa virheen ja palauttaa käyttäjän valikon 0-levelille
LCD_Position(0,0);
LCD_PrCString("Virrrhe!");
LCD_Position(1,0);
LCD_PrCString("Takaisin!");
SleepTimer_SyncWait(0x10, SleepTimer_WAIT_RELOAD); // viivästystä, että käyttäjä huomaa virheen tulostuksen näytölle
level = level-1; // levelin palautus
break;
case 1: // valinta 1:ssä tulostetaan haluttu kurssi lyhyemmässä muodossa ja odotetaan kirjausta
LCD_Position(0,0);
LCD_PrCString("K1:Tite");
LCD_Position(1,0);
LCD_PrCString("Sisaan/ulos?  ");
crs_id[0]=0x01; // laitetaan kurssi-id ylös, mikäli valittuna ko. kurssi
crs_id[1]=0x02;
crs_id[2]=0x03;
crs_id[3]=0x04;
break;
case 2: // valinta 2:ssa tulostetaan haluttu kurssi lyhyemmässä muodossa ja odotetaan kirjausta
LCD_Position(0,0);
LCD_PrCString("K2:Titu");
LCD_Position(1,0);
LCD_PrCString("Sisaan/ulos?  ");
crs_id[0]=0x02; // laitetaan kurssi-id ylös, mikäli valittuna ko. kurssi
crs_id[1]=0x03;
crs_id[2]=0x04;
crs_id[3]=0x05;
break;
case 3: // valinta 3:ssa viedään käyttäjä ID:n lisäysfunktioon, mikäli ID:tä ei ole jo annettu
if (new_course)
{
LCD_Position(0,0);
LCD_PrCString("                "); // tyhjennetään vanhat LCD-tulostukset
LCD_Position(0,0);
LCD_PrCString("ID:"); // tulostetaan teksti "ID:", jonka perään tulee ID-numerosarja
if(new_id_loop < 3) // jos uusi ID on jo olemassa, tulostetaan 3krt "uusi lisätty!" tuloste
{
LCD_Position(1,0);
LCD_PrCString("Uusi lisatty!");
SleepTimer_SyncWait(0x10, SleepTimer_WAIT_RELOAD); // pieni viivästys, että keretään lukemaan näytöltä lisäys
SleepTimer_SyncWait(0x05, SleepTimer_WAIT_RELOAD);
new_id_loop++; // kierrätetään printtausta 3krt
}
LCD_Position(0,3); // kerrotaan tulosteen paikka, tulostetaan tekstin "ID:" perään kurssi-ID
LCD_PrHexInt(new_course_id); // uuden kurssi-ID:n muuttujan tulostus
LCD_Position(1,0);
```

```
LCD_PrCString("Sisaan/ulos?   "); // tulostetaan alapuolelle optio sisään/ulos-kirjaus
}
else add_new_course(); // jos ei jo uutta kurssi-ID:tä lisättynä, käydään uuden kurssin lisäysfunktiossa
break;
case 4: // mikäli valinta 4 valitaan, tulostetaan kellon asetusteksti ja odotetaan "F" = eteenpäin, "C" = takaisin
LCD_Position(1,0);
LCD_PrCString("Aseta klo F/C?     ");
break;
default: // mikäli ei optio 1-4 ole valittuna, tulostetaan virhe, koska tätä ei pitäisi tapahtua
LCD_Position(0,0);
LCD_PrCString("Virrrrhe!      ");
LCD_Position(1,0);
LCD_PrCString("Takaisin!      ");
break;
}
}
else if (level==2 && option <4) // tarkistetaan taso ja optio, mikäli kurssi valittuna, voidaan käydä lukemassa levelillä 2
UART
{
LCD_Position(0,0);
LCD_PrCString("                "); // tyhjennetään vanhat tulosteet näytöltä rivi 1
LCD_Position(1,0);
LCD_PrCString("                "); // tyhjennetään vanhat tulosteet näytöltä rivi 2
LCD_Position(0,0);
LCD_PrCString("Nayta tagi, niin   "); // pyydetään käyttäjää antamaan tagi lukualueelle
LCD_Position(1,0);
LCD_PrCString("luetaan ID !       "); // pyydetään käyttäjää antamaan tagi lukualueelle
UART_check(); // käydään lukemassa UARTin yli tuleva tagi_ID
level=level-1; // tiputetaan leveli takaisin
}
else if (level==2 && option==4) // mikäli ollaan levelillä 2 ja valinta "option 4" valittuna => kellon uudelleenasetus
{
LCD_Position(0,0);
LCD_PrCString("                "); // tyhjennetään LCD-tulosteet 1. riviltä
LCD_Position(1,0);
LCD_PrCString("                "); // tyhjennetään LCD-tulosteet 2. riviltä
clock_loop = 1;
set_new_time(); // ajastetaan kello omassa funktiossa
level=level-1;
}
save_information(); // tallennetaan mahd. annetut tagi-ID-informaatiot / kurssi / kellonaika
USB_GetData = USBUART_bGetRxCount();
if (USB_GetData)
{
USB_PrintData();
}
// Loop-testaus
```

```
/*LCD_Position(1,12);
LCD_PrHexByte(test);
test++;*/// ------------------Pääohjelman loop loppuu! -------------||
}
} // -------------------Pääohjelma loppuu! ------------- |||

int check_keypad(void) // numeronäppäimistön tarkistusfunktio
{
PRT1DR|=0xF0; // tehdään matriisitarkistus numeronäppäimistölle, ajetaan 5v jännite neljään ylimpään porttiin
PRT1DR&=0x0F;
prtBuf2=PRT1DR; // laitetaan vastinportin pinnien tulokset muistiin
PRT1DR|=0x0F; // tehdään matriisitarkistus numeronäppäimistölle, ajetaan 5v jännite neljään alimpaan porttiin
PRT1DR&=0xF0;
prtBuf = PRT1DR; // laitetaan vastinportin pinnien tulokset muistiin
keypad_result = prtBuf+prtBuf2; // tehdään aputulos vastinkappaleiden perusteella => arvo switch/case -rakenteeseen, josta palautetaan ko. napin arvo
switch (keypad_result)
{
case 0x22:
return 0x01; // painonäppäin 1
break;
case 0x24:
return 0x02; // painonäppäin 2
break;
case 0x28:
return 0x03; // painonäppäin 3
break;
case 0x30:
return 0x0F; // painonäppäin F
break;
case 0x42:
return 0x04; // painonäppäin 4
break;
case 0x44:
return 0x05; // painonäppäin 5
break;
case 0x48:
return 0x06; // painonäppäin 6
break;
case 0x50:
return 0x0E; // painonäppäin E
break;
case 0x82:
return 0x07; // painonäppäin 7
break;
case 0x84:
return 0x08; // painonäppäin 8
```

```
break;
case 0x88:
return 0x09; // painonäppäin 9
break;
case 0x90:
return 0x0D; // painonäppäin D
break;
case 0x02:
return 0x0A; // painonäppäin A
break;
case 0x04:
return 0x0B; // painonäppäin B
break;
case 0x08:
return 0x00; // painonäppäin 0
break;
case 0x10:
return 0x0C; // painonäppäin C
break;
default:
return 0xFF;
}
} // ------------ Numeronäppäimen tarkistusfunktio loppuu ----------------

void UART_check(void) // UART-tarkistusfunktio
{
while(uart_while) // UART-rutiini
{
if(UART_bCmdCheck())                    // Tarkistetaan tuleeko RFID:ltä rs232:n yli jotain?
{
if(strPtr = UART_szGetParam())        // Mitä saatiin RFID-lukijalta?
{
if (strPtr == tag_mem) // testataan onko tagi-ID jo tiedossa, jos niin kirjautuminen ulos
{
LCD_Position(0,0);
LCD_PrCString("                    "); // tyhjennetään LCD:ltä vanhat tulosteet
LCD_Position(0,0);
LCD_PrString(strPtr); // tulostetaan tagi-ID riville 1
I2Cm_bWriteCBytes(0x68,txCBuf,1,I2Cm_NoStop); // kirjotetaan I2C:n yli RTC-piirin osoite lukua varten
status = I2Cm_fReadBytes(0x68,rxBuf,7,I2Cm_RepStart); // luetaan kellonaika RTC:ltä
for(i=0;i<7;i++)
{
timeBufOut[i] = rxBuf[i];
}
LCD_Position(1,0);
LCD_PrCString("Ulos!           "); // tulostetaan uloskirjaus LCD:lle
LCD_Position(1,5);
```

```
LCD_PrHexByte(rxBuf[2]); // tulostetaan kellonaika tunnit
LCD_Position(1,7);
LCD_PrCString(":"); // tulostetaan välimerkki ":" kellonaikaan
LCD_Position(1,8);
LCD_PrHexByte(rxBuf[1]); // tulostetaan kellonaika minuutit
SleepTimer_SyncWait(0x10, SleepTimer_WAIT_RELOAD); // Viivästetään sen verran, että keretään lukea
SleepTimer_SyncWait(0x10, SleepTimer_WAIT_RELOAD);
uart_while = 0; // tullaan pois UART-while-loopista ja palataan odottamaan seuraavia käyttäjän valintoja
}
else
{
tag_mem = strPtr; // laitetaan tagi muistiin, jotta voidaan vertailla poistuessa
LCD_Position(0,0);
LCD_PrCString("                "); // tyhjennetään LCD:ltä vanhat tulosteet
LCD_Position(0,0);
LCD_PrString(strPtr); // tulostetaan tagi-ID riville 1
I2Cm_bWriteCBytes(0x68,txCBuf,1,I2Cm_NoStop); // kirjotetaan I2C:n yli RTC-piirin osoite lukua varten
status = I2Cm_fReadBytes(0x68,rxBuf,7,I2Cm_RepStart); // luetaan kellonaika RTC:ltä
for(i=0;i<7;i++)
{
timeBufIn[i] = rxBuf[i];
}
LCD_Position(1,0);
LCD_PrCString("Sisaan!          "); // tulostetaan sisäänkirjaus LCD:lle
LCD_Position(1,7);
LCD_PrHexByte(rxBuf[2]); // tulostetaan kellonaika tunnit
LCD_Position(1,9);
LCD_PrCString(":"); // tulostetaan välimerkki ":" kellonaikaan
LCD_Position(1,10);
LCD_PrHexByte(rxBuf[1]); // tulostetaan kellonaika minuutit
SleepTimer_SyncWait(0x10, SleepTimer_WAIT_RELOAD); // Viivästetään sen verran, että keretään lukea
SleepTimer_SyncWait(0x10, SleepTimer_WAIT_RELOAD);
uart_while = 0; // tullaan pois UART-while-loopista ja palataan odottamaan seuraavia käyttäjän valintoja
}
}
UART_CmdReset(); // nollataan UART
}
}
uart_while = 1; // asetetaan UART-apumuuttuja takaisin aktiiviseksi, mikäli palataan hakemaan UARTilta tietoja
} // ------------------ UART-tarkistusfunktio loppuu ----------------

void print_time(void) // ajan tulostusfunktio (LCD-näytön oikea ylänurkka)
{
I2Cm_bWriteCBytes(0x68,txCBuf,1,I2Cm_NoStop); // Annetaan i2c:lle RTC:n osoite
status = I2Cm_fReadBytes(0x68,rxBuf,7,I2Cm_RepStart); // luetaan kellonaika
LCD_Position(0,8);
LCD_PrHexByte(rxBuf[2]); // tulostetaan tunnit
```

```
if (clock_flash == 3)
{
LCD_Position(0,10);
LCD_PrCString(" "); // tulostetaan erotin ":"
LCD_Position(0,13);
LCD_PrCString(" "); // tulostetaan erotin ":"
}
LCD_Position(0,11);
LCD_PrHexByte(rxBuf[1]); // tulostetaan minuutit
LCD_Position(0,14);
LCD_PrHexByte(rxBuf[0]); // tulostetaan sekunnit
if (rxBuf[0] != rxBufComp[0])
{
LCD_Position(0,10);
LCD_PrCString(":"); // tulostetaan erotin ":"
LCD_Position(0,13);
LCD_PrCString(":"); // tulostetaan erotin ":"
clock_flash = 0;
}
clock_flash++;
rxBufComp[0] = rxBuf[0];
} // -------------------------- ajan tulostusfunktio loppuu ----------------------
void set_new_time(void) // uuden ajan asetusfunktio
{
i = 6;
while(clock_loop)
{
I2Cm_bWriteCBytes(0x68,txCBuf,1,I2Cm_NoStop); // kirjoitetaan RTC:n osoite I2C:lle lukua varten
status = I2Cm_fReadBytes(0x68,rxBuf,7,I2Cm_RepStart); // haetaan nykyinen aika RTC:ltä
SleepTimer_SyncWait(0x02, SleepTimer_WAIT_RELOAD); // viivästetään hieman napin painalluksen takia
if (check_keypad()!= 0xFF)
{
if (check_keypad() == 0x0E) // lisätään valittua muuttujaa, mikäli painetaan E
{
rxBuf[i]++; // lisätään valittua muuttujaa esim. minuutteja lisää
change = 1; // vaihtolippu 1:een, jotta tajutaan viedä uudet aikatiedot RTC:lle
}
else if (check_keypad() == 0x0D) // vähennetään valittua muuttujaa, mikäli painetaan D
{
rxBuf[i]--; // vähennetään valittua muuttujaa esim. minuutteja vähemmän
change = 1; // vaihtolippu 1:een, jotta tajutaan viedä uudet aikatiedot RTC:lle
}
else if (check_keypad() == 0x0A && i<6) // vaihdetaan valittua muuttujaa, mikäli painetaan A
{
i++;
}
else if (check_keypad() == 0x0B && i>0) // vaihdetaan valittua muuttujaa, mikäli painetaan B
```

```
{
i--;
}
else if (check_keypad() == 0x0F) // poistutaan kelloasetuksesta F:llä
{
clock_loop = 0; // poistutaan kellon asetuksesta
}
}
if(change)
{
for(y=0;y<6;y++)
{
txTime[y+1]=rxBuf[y]; // viedään RTC:lle muutetut tiedot takaisin
}
I2Cm_bWriteBytes(0x68,txTime,9,I2Cm_CompleteXfer); // lähetetään I2C:n yli RTC:lle uudet aikatiedot
change = 0; // asetetaan vaihtolippu takaisin 0:aan
}
I2Cm_bWriteCBytes(0x68,txCBuf,1,I2Cm_NoStop); // asetetaan lukua varten I2C:lle RTC:n osoite
status = I2Cm_fReadBytes(0x68,rxBuf,7,I2Cm_RepStart); // luetaan nykyaika RTC:ltä
LCD_Position(0,0);
LCD_PrCString("                "); // tyhjennetään LCD:n tulosteet riviltä 1.
LCD_Position(1,0);
LCD_PrCString("                "); // tyhjennetään LCD:n tulosteet riviltä 2.
LCD_Position(0,0);
LCD_PrHexByte(rxBuf[6]); // tulostetaan vuosi
LCD_Position(0,2);
LCD_PrHexByte(rxBuf[5]); // tulostetaan kuukausi
LCD_Position(0,4);
LCD_PrHexByte(rxBuf[4]); // tulostetaan monesko päivä
LCD_Position(0,6);
LCD_PrHexByte(rxBuf[3]); // tulostetaan viikonpäivä
LCD_Position(0,8);
LCD_PrHexByte(rxBuf[2]); // tulostetaan tunnit
LCD_Position(0,10);
LCD_PrHexByte(rxBuf[1]); // tulostetaan minuutit
LCD_Position(0,12);
LCD_PrHexByte(rxBuf[0]); // tulostetaan sekunnit
switch (i) // tulostetaan valinnan mukaan LCD:lle tieto siitä, mitä ollaan muuttamassa
{
case 6: // vuoden vaihtovalinta
LCD_Position(1,0);
LCD_PrCString("YY");
break;
case 5: // kuukauden vaihtovalinta
LCD_Position(1,2);
LCD_PrCString("MM");
break;
```

```
case 4: // päivän vaihtovalinta
LCD_Position(1,4);
LCD_PrCString("DD");
break;
case 3: // viikonpäivän vaihtovalinta
LCD_Position(1,6);
LCD_PrCString("WD");
break;
case 2: // tunnin vaihtovalinta
LCD_Position(1,8);
LCD_PrCString("HH");
break;
case 1: // minuutin vaihtovalinta
LCD_Position(1,10);
LCD_PrCString("MM");
break;
case 0: // sekunnin vaihtovalinta
LCD_Position(1,12);
LCD_PrCString("SS");
break;
default:
LCD_Position(1,0);
LCD_PrCString("Errorr!");
break;
}
}
} // ---------- Uuden ajan asetusfunktio loppuu ----------

void add_new_course(void) // uuden kurssin lisäysfunktio
{
i=0; // apumuuttuja
new_course_done = 1; // apumuuttuja kurssi-ID:n lisäysfunktiota varten
LCD_Position(0,0);
LCD_PrCString("Lisataan ID          ");
LCD_Position(1,0);
LCD_PrCString("                     ");
while(new_course_done) // pyöritetään valintaa niin kauan, kunnes käyttäjä hyväksyy kurssi-ID:n
{
SleepTimer_SyncWait(0x04, SleepTimer_WAIT_RELOAD); // viivästetään hieman napin painalluksen takia
if (check_keypad()!= 0xFF && check_keypad()!= 0x0F && check_keypad()!= 0x0E && check_keypad()!= 0x0C &&
check_keypad()!= 0x0D && check_keypad()!= 0x0A && check_keypad()!= 0x0B)
{// tarkistetaan, että käyttäjä antaa numeroita ID:ksi
courseBuf[i] = check_keypad(); // viedään annettu numero puskuriin
LCD_Position(1,i); // tulostetaan annettu ID nähtäväksi
LCD_PrHexByte(courseBuf[i] << 4); // shiftataan numero tulostusta varten kohdilleen, että numero ei näy muodossa
"01" vaan "1"
LCD_Position(1,i+1); //siirretään tulostuksen paikkaa sen mukaan, monennestako ID-numerosta on kyse
```

```
LCD_PrCString(" "); // tulostetaan shiftatun 0:n tilalle tyhjä
i++; // mikäli numero annettu, siirretään numeropaikka seuraavaan
}
else if (check_keypad() == 0x0F) // hyväksytään annettu ID painalla F
{
new_course = 1; // lisätään "lippu", että uusi ID on annettu
new_course_done = 0; // poistutaan ID:n antamisesta
}
else if (check_keypad() == 0x0C) // mikäli halutaan peruuttaa ID:n lisäys, painetaan C:tä
{
LCD_Position(1,0);
LCD_PrCString("Ei annettu ID:ta!   ");
SleepTimer_SyncWait(0x10, SleepTimer_WAIT_RELOAD); // viivästys, että käyttäjä kerkeää lukemaan LCD-näytöltä poistumisviestin
SleepTimer_SyncWait(0x10, SleepTimer_WAIT_RELOAD);
new_course_done = 0; // poistutaan ID:n antamisesta
level--; // tiputetaan tasoa alemmas alkuvalikkoon asti
}
else if (i==4) i=0; // mikäli 4 numeroa annettu, aloitetaan alusta, mikäli käyttäjä haluaa korjata valinnan
}
new_course_id = new_course_id + courseBuf[3]; // viedään saatu ID integeriksi
new_course_id = new_course_id + (courseBuf[2] << 4); // shiftataan numerot järjestyksessä oikeille paikoille 16-bit integeriin
new_course_id = new_course_id + (courseBuf[1] << 8);
new_course_id = new_course_id + (courseBuf[0] << 12);
crs_id[0] = courseBuf[0];
crs_id[1] = courseBuf[1];
crs_id[2] = courseBuf[2];
crs_id[3] = courseBuf[3];
} // ------------------- Uuden kurssi-ID:n lisäysfunktio loppuu ----------------------------

void save_information(void) // tietojen tallennusfunktio
{
ReadAddress[0] = 0x00;
I2Cm_bWriteBytes(0x50, ReadAddress, 1, 0); // haetaan tieto ensimmäisestä vapaasta muistipaikasta
I2Cm_fReadBytes(0x50, ReadAddress, 1, 0); //
RAMbuffer[0] = ReadAddress[0]+1; // kirjoitetaan datan aloituskohta
RAMbuffer[1] = crs_id[0]; // viedään datat bufferiin, jolla voidaan viedä tieto I2C:n ylitse eepromille
RAMbuffer[2] = crs_id[1];
RAMbuffer[3] = crs_id[2];
RAMbuffer[4] = crs_id[3];
RAMbuffer[5] = tag_id[0];
RAMbuffer[6] = tag_id[1];
RAMbuffer[7] = tag_id[2];
RAMbuffer[8] = tag_id[3];
RAMbuffer[9] = tag_id[4];
I2Cm_bWriteBytes(0x50, RAMbuffer, 10, 0);
```

```
RAMbuffer2[0] = ReadAddress[0]+2;
RAMbuffer2[1] = timeBufIn[0];
RAMbuffer2[2] = timeBufIn[1];
RAMbuffer2[3] = timeBufIn[2];
RAMbuffer2[4] = timeBufIn[3];
RAMbuffer2[5] = timeBufIn[4];
RAMbuffer2[6] = timeBufIn[5];
RAMbuffer2[7] = timeBufIn[6];
RAMbuffer2[8] = timeBufOut[0];
RAMbuffer2[9] = timeBufOut[1];
RAMbuffer2[10] = timeBufOut[2];
RAMbuffer2[11] = timeBufOut[3];
RAMbuffer2[12] = timeBufOut[4];
RAMbuffer2[13] = timeBufOut[5];
RAMbuffer2[14] = timeBufOut[6];
I2Cm_bWriteBytes(0x50, RAMbuffer2, 15, 0);
WriteAddress[0] = 0x00;
WriteAddress[1] = ReadAddress[0]+2;
I2Cm_bWriteBytes(0x50, WriteAddress, 2, 0);
}
void USB_PrintData(void)
{
for(i=0;i<10;i++)
{
ReadAddress[0] = 0x00+i;
I2Cm_bWriteBytes(0x50, ReadAddress, 1, 0);
I2Cm_fReadBytes(0x50, WriteBuffer, 16, 0);
for(i=0;i<17;i++)
{
USBUART_Write(pData, WriteBuffer[i]);
}
}
}// ------------------- tietojen tallennusfunktio loppuu ----------------------------
```