Markus Pyykkö

Ohjelmistosuunnittelu e-lukulaitteeseen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Elektroniikan koulutusohjelma

Insinöörityö

15.9.2012

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Markus Pyykkö Ohjelmistosuunnittelu e-lukulaitteeseen xx sivua + x liitettä 15.9.2012
Tutkinto	insinööri (AMK)
Koulutusohjelma	Elektroniikan koulutusohjelma
Suuntautumisvaihtoehto	sulautetut järjestelmät
Ohjaaja(t)	tutkimuspäällikkö Sampo Nurmentaus lehtori Timo Kasurinen

Insinöörityön tavoitteena oli toteuttaa ohjelmisto E-lukulaitteeseen, joka on sähköisten kirjojen ja lehtien lukemiseen tarkoitettu, kannettava elektroninen laite. Laitteen tarkoituksena oli toteuttaa ensimmäinen prototyyppi sähköisestä sanomalehdestä, joka vastaanottaisi sanomalehden päivittäin käyttäen digitaalista televisioverkkoa. Laite oli osa Metropolian Electria –tutkimuslaitoksen Broadcast-projektia, jonka tarkoituksena oli tutkia sähköisen sanomalehden toteuttamismahdollisuuksia, ja ihmisten suhtautumista sellaiseen.

Komponentit oltiin valittu suurimmalta osin etukäteen ja ohjelmistokehityksen osalta työksi jäi pääasiassa niiden saattaminen toimimaan yhdessä mahdollisimman hyvin ja tehokkaasti. Itse laitetta ei edes yritetty kytkeä suoraan televisioverkkoon, vaan alusta lähtien päätettiin että laitteessa olisi itsessään vain bluetooth. Televisioverkkoon kytkeytyminen hoidettiin erillisellä laitteella, jonka kanssa lukulaite keskusteli bluetoothin välityksellä.

Lukulaitteen kehitys onnistui pienistä ongelmista huolimatta hyvin, ja käyttökokemus saatiin myös hiottua miellyttäväksi ja helposti lähestyttäväksi. Laite oli myös näyttävä, sillä siitä saatiin ohuin e-lukulaite maailmassa. Laitetta pilotoitiin ensiksi Sanoma WSOY:n ja Dna Oy:n järjestämässä pilottiprojektissa Vantaalla, ja tätä palautetta käytettiin laitteen toisen version kehitykseen. Laitteen toista versiota pilotoitiin sitten myöhemmin Kiinassa kahdessa eri paikassa, Kiinan valtiollisen sanomalehden People’s Dailyn kanssa.

Avainsanat	

Author(s) Title Number of Pages Date	Markus Pyykkö Software design for an e-reader xx pages + x appendices 15 September 2012
Degree	Bachelor of Engineering
Degree Programme	Electronics
Specialisation option	Embedded systems
Instructor(s)	Sampo Nurmentaus, Project Manager Timo Kasurinen, Principal Lecturer
Keywords	

Sisällys

Lyhenteitä ja käsitteitä

JTAG Joint Test Action Group. JTAG eli IEEE 1149.1 on määritelmä väylästä, joka mahdollistaa laitteen ohjelmoimisen lisäksi myös matalan tason debuggauksen kuten *singlesteppauksen, breakpointit* ja *muistin tarkastelun*.

FSMC Flexible Static Memory Controller . FSMC-väylä on hyvin perinteinen tietokoneen muistiväylä, jollainen on STM32-kontrolleriin lisätty pääasiassa ulkoisten muistien käyttöä varten. FSMC tukee suurta määrää erilaisia signalointikonventioita SRAM- sekä FLASH-muistien kanssa.

DMA Direct Memory Access). DMA-yksiköt ovat useista eri tietokonejärjestelmistä sekä integroiduista kontrollereista löytyviä tiedonsiirtoyksiköitä, joiden tehtävänä on siirtää nopeasti tietoa muistin tai lisälaitteiden välillä. Ajatuksena on se, ettei prosessorin tarvitse osallistua tähän lainkaan.

UART Universal Asyncronous Receiver Transmitter. UART-portit ovat sarjamuotoiseen kommunikaatioon kykeneviä portteja joilla yhdistetään usein lisälaitteita toisiinsa.

FAT File Allocation Table on Microsoftin alunperin 80-luvulla kehittämä yksinkertainen tiedostojärjestelmä, joka on yksi parhaiten tuetuista tiedostojärjestelmistä maailmalla.

Parseri Parseri tarkoittaa ohjelman osaa joka tulkitsee jossain muodossa olevan materiaalin ja jäsentelee sen johonkin toiseen muotoon.

Inband-signalointi Sekä siirrettävä data että kontrolli-informaatio siirretään käyttäen samaa kommunikaatiokanavaa siten, ettei näitä ole enkapsuloitu esimerkiksi paketeiksi erilleen toisistaan.

TFT-matriisi Ohutkalvotransistorimatriisi, jota käytetään E Inkin näytössä vaihtelevan sähkökentän luomiseen.

Haara	Versionhallinnassa haaran luominen tarkoittaa että yhdestä versiosta lähtee enemään toinen kehityspolku, joka ehkä yhdistyy alkuperäiseen polkuun sen jälkeen kuin haaraa ei enää käytetä.
Pull-media	Pull-media tarkoittaa

1 Johdanto

Tämän insinöörityön aiheena on ohjelmistosuunnitteluprosessin kuvaus ja arviointi E-lukulaitteeseen, joka oli osa metropolian Broadcast-projektia. Itse komponentit olivat valittu suurimmaksi osaksi jo etukäteen, ja käsiteltävä työ koostui tutustumisesta komponenttien toimintaan, elektroniikkasuunnittelun konsultoimisesta software-kehittäjän näkökulmasta, sekä itse ohjelmiston suunnittelusta ja kirjoittamisesta.

1.1 Broadcast-projekti

Broadcast-projekti aloitettiin Metropolian tutkimuskeskus Electriassa vuonna 2011, ja sen tarkoituksena oli tutkia sanomalehden kaltaisen median toteuttamista elektronisesti. [1] Projekti toteutettiin yhteistyössä Shanghailaisen NERC-DTV:n kanssa, joka on Kiinan kansallinen televisioverkon tutkimuskeskus. Projektissa tutkittu sähköinen sanomalehti olisi siis vähävirtainen ja halpa tapa jakaa sanomalehti esimerkiksi syrjäseuduille käyttäen jo olemassa olevaa digitaalista televisioverkkoa. Käyttökohteina tälle olisivat esimerkiksi monet Kiinan seuduista joissa ei ole kovin luotettavaa sähköverkkoa.

Laitetta suunniteltiin useiden eri tahojen kanssa samalla kun sähköisen sanomalehden tärkeimpiä vaatimuksia kartoitettiin. Lopullista laitetta pilotoitiin projektin lopuksi Suomessa ja Kiinassa. Näin kerättäisiin palautetta ihmisiltä siitä, että millaisiksi he kokivat laitteen ja mitä ominaisuuksia he haluaisivat nähdä perinteisen sanomalehden korvaajassa.

1.2 Vaatimukset

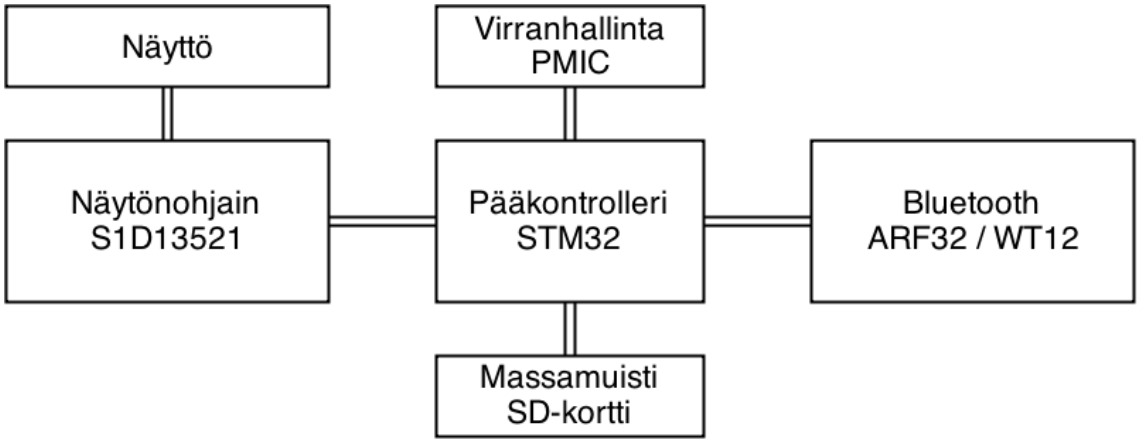
Käyttökokemus piti myös saada mahdollisimman lähelle tavallista sanomalehteä, erityisesti niin että laitetta ei miellettäisi *laitteeksi*. Tämä tarkoitti käytännössä sitä että laite ei olisi havaittavasti *päällä* tai *poissa päältä*, vaan olisi aina käyttövalmis. Samasta syystä myös kuvan piti pysyä näytöllä jatkuvasti. Laite yritettiin saada myös mahdollisimman ohueksi, ja tämä onnistuikin käyttämällä laitteen emolevyä osana mekaanista rakennetta.

Laitteen piti myös vastaanottaa uusi sanomalehti televisioverkon välityksellä. Suoraan tätä ei kuitenkaan edes yritetty tehdä, vaan tätä varten kehitettiin erillinen digiboksia muistuttava gapfiller-laite, joka vastaanotti lehden televisioverkosta ja välitti sen bluetoothia käyttäen itse lukulaitteelle.

Myös pitkä akunkesto ja halvat komponentit olivat tärkeitä. Tämän vuoksi pääprosessori oli todella vähävirtainen, ja yksi tavoitteista olikin saada rakennettua e-lukulaite käyttäen tälläistä. Kaikissa olemassa olevissa lukulaitteissa oli käytössä tehokkaat applikaatioprosessorit, jotka olivat tähän tarkoitukseen hieman liian järeitä. Näiden virrankulutus olisi ollut tarpeettoman suuri laitteen vaadittuihin ominaisuuksiin nähden.

Kuva 1. Tutkimuspäällikkö Nurmentaus esittelemässä lopullista lukulaitetta

2 Käytetyt laitteiston komponentit



Kuvio 1. Yksinkertaistettu kuvio relevanteista laitteiston osista

2.1 Mikro-ohjain STM32

Mikro-ohjaimeksi oltiin valittu isoin STMicroelectronicsin *ARM Cortex-M3* pohjaisista piireistä. Syy juuri tämän kontrollerin valintaan oli se ettei kehitysvaiheessa tarvisi ruveta optimoimaan koodia ennenaikaisesti pientä tilaa varten, vaan käytettävissä olisi suhteellisen suuri määrä muistitilaa ja integroituja lisälaitteita kokeiluja varten. Valinta osoittautui oikeaksi, sillä *FSMC-muistiväylää* ei olisi löytynyt lainkaan pienemmistä malleista, ja ilman sitä tiedonsiirto tämän pääkontrollerin ja näytönohjaimen välillä olisi ollut hyvin hidasta ja vaikeaa.

ARM-pohjaisen prosessorin sisältävänä kontrollerina myös STM32:n ohjelmistokehitykseen voitiin käyttää C-ohjelmointikieltä luotettavien ja paljon testattujen työkalujen kanssa. Kääntäjäksi valittiin GCC-pohjainen *Mentor Graphicsin* ylläpitämä *CodeSourcery*.[2] Tämä kääntäjä mahdollisti Unix-järjestelmille ominaisen ohjelmistonkehitysprosessin, josta työryhmällä oli jo ennestään kokemusta.

Mikro-ohjaimen kanssa kanssakäyminen hoidettiin *JTAG-väylää* käyttäen. Tätä varten hankittiin pari kappaletta Flyswatter –nimisiä ohjelmointilaitteita Tin Can Toolsilta.[3] Itse ohjelmistona toimi OpenOCD.[4]

2.1.1 Lisälaitteet

STM32 sisältää hyvin laajan kirjon erilaisia lisälaitteita. Näistä tärkeimmät meille olivat *FSMC-väylä, DMA-yksiköt, UART-portit* sekä *RTC-kello.*

STM32-kontrollerin muistiväylien rakenteesta johtuen pääprosessoria ei tarvitse pysäyttää tiedonsiirron ajaksi, jonka vuoksi DMA-yksiköt mahdollistavat rajoittuneen rinnakkaisprosessoinnin. DMA-yksiköitä käytettiin hyvin laajasti, sillä 8MHz kellotaajuudella kymmenien megatavujen kokoisten tietomäärien liikuttelu olisi ollut todella hidasta.

RTC-kello oli myös tärkeä, sillä se on toteutettu STM32:ssä erillisenä yksikkönä ja on käytännössä ainut lisälaite jonka voi ohjelmoida herättämään kontrollerin unesta ilman ulkoista tekijää.

2.2 Näyttö 9.7” E ink Vizplex/Pearl

Yleisesti sormitietokoneissa käytetyt LCD-näytöt taustavaloineen kuluttivat niin paljon virtaa kuvaa näyttäessään että ne siivilöityivät heti alussa pois vaihtoehdoista. [5] LCD-näyttöjen lukukokemus ei myöskään vastaa sanomalehteä, sillä näytöissä oleva taustavalo aiheuttaa helposti rasitusta silmille, sekä ympäristön voimakas diffuusiovalaistus (kuten auringonvalo) tekisi lehdestä lukukelvottoman. Vaatimukseksi tulikin elektronisen paperin käyttö, jonka takia laite tulisi muistuttamaan todennäköisesti hyvin paljon markkinoilla jo olevia E-lukijoita.

Koska tarkoituksena oli tehdä laite joka tuntuisi aina käyttövalmiilta, oli valinta näytöksi E-paperi, joka tarkoitti käytännössä Sipixin tai E inkin valmistamaa näyttöä. Sipixin näyttöä kokeiltiin aluksi, mutta E Inkin näyttö valittiin käyttöön, sillä se pystyi näyttämään 16 harmaasävyä.

2.3 Näytönohjain Epson S1D13521

13521 on Epsonin valmistama ja E ink Vizplex-näyttöjen ajamiseen tarkoitettu näytönohjain. Piiri sisältää useilla eri tavoilla ohjelmoitavan TFT-matriisien ajamiseen tarkoitetun grafiikkasekvensserin, ja se lataa kuvat erillisestä SDRAM-muistista. E-paperinäyttöjen ajamiseen tarkoitettu aaltomuototiedosto, sekä grafiikkasekvensserin parametrit on tallennettu erilliselle flash-muistipiirille.

Tiedonsiirtoon piirissä oli 16-bittinen Intel 80 -tyyppinen muistiväylä, jonka kautta sille annettiin komentoja sekä lähetettiin kuvat. Tämä oli kytketty STM32:een FSMC:n kautta. Käytännössä piiriä käytettiin niin että sen käynnistämisen jälkeen sille lähetettiin kuva sekä komento jolla kuva piirrettiin näytölle. Isoa SDRAM-piiriä käytettiin myös välimuistina, johon kuvia voitiin lähettää jo etukäteen näyttämistä varten.

2.4 Bluetooth-ohjain ARF32 / Bluegiga WT12

Sekä ARF32 että WT12 ovat yleishyödyllisiä bluetooth-kontrollereja. Projektissa käytettiin aluksi ARF32:sta WT12:n huonon saatavuuden takia puoliväliin asti, mutta sen jälkeen vaihdettiin WT12:een sillä sen valmistaja Bluegiga oli yksi projektin partnereista.

Molemmat ovat peruskäytettävyydeltään hyvin samanlaisia CLASS 2 –luokan bluetooth-piirejä, jotka sisältävät UART-portin kommunikointiin mikro-ohjaimen kanssa. Komentosyntaksi on molemmissa täysin erilainen, mutta koska piirejä käytettiin lähinnä SPP-tilassa, ei niiden välillä ollut juurikaan eroa.

3 Ohjelmistokehityksen käytännöt projektissa

Koko ohjelmistokehitysprosessia kuvaa varmastikin parhaiten sana *debaatti*. Asioista väiteltiin aina kuin vain suinkin aika antoi periksi, ja väittelyjen aiheet vaihtelivat aina muuttujien nimeämisestä suuriin arkkitehtuurisiin päätöksiin. Jatkuva väittely oli kuitenkin hyödyllinen työväline, sillä lopputuloksessa näkyi suoraan että asioita on mietitty ja viilattu. Ohjelmistosta saatiin lopulta erittäin vakaa, ja harvatkin kaatuilut johtuivat pääasiassa satunnaisista laitteistovioista.

Kehitys alkoi käytännössä kahden ohjelmoijan toimesta, jotka toimivat pääosin omilla tahoillaan. Tällöin kaikki ohjelmisto koostui pääasiassa pienistä testiohjelmista, joilla pääasiassa oltiin kokeiltu joko STM32:n sisäisten-, tai ulkoisen lisälaitteiden toimintaa. Tämä ei kuitenkaan ollut kovin pitkäikäinen ratkaisu, sillä kun ohjelmisto alkoi löytää lopullista muotoaan, tarvittiin keskitettyä säilöä lähdekoodeille. Tämän *versionhallinta-ohjelman* piti myös ylläpitää listaa lähdekoodiin tehdyistä muutoksista, sillä kehittäjien lisääntyessä suullinen kommunikaatio ei ollut aina mahdollista tai järkevää.

3.1 Git-versionhallinta

Git-versionhallinta on Linus Torvaldsin kehittämä työkalu alun perin Linux-käyttöjärjestelmän kehitystä varten. Se erottuu muista yleisesti käytetyistä versionhallintajärjestelmistä (kuten CVS ja SVN) esimerkiksi niin, että uusien kehityshaarojen teko (engl. *Branches*) ei ole vain helppoa, vaan myös suositeltua. Git on myös niinsanottu *hajautettu versionhallinta*, mikä tarkoittaa että Git ei juurikaan tee erottelua palvelinten ja työasemien välillä, ja kehityshaaroja voi suoraan hakea muilta käyttäjiltä samaan tapaan kuin palvelimilta.

Kehitysprosessi muovautui hyvin läheiseksi tämän haaroittumisen kanssa. Yleisesti jonkun uuden ominaisuuden kehittäminen tarkoitti uuden kehityshaaran luomista, johon pystyi rauhassa tekemään kaikki muutokset mitä uuden ominaisuuden kehitys vaati. Sen jälkeen näistä muutoksista tehtiin niin sanottu vain muutokset sisältävä *patch*, joka sitten käytiin läpi ja integroitiin päähaaraan. Näin kehitysvaiheessakin olevien ominaisuuksien kanssa voitiin käyttää versionhallintaa, ja kaikki päähaaraan päätyvä koodi oli pääosin kaunista ja toimivaa.

Kaikkia muutoksia ei kuitenkaan käytetty tämän tarkistusprosessin kautta. Pienemmät muutokset (joita olivat esimerkiksi pienet bugikorjaukset) laitettiin suoraan versionhallintaan. Myöskään ohjelmakoodin ulkonäköön liittyviä muutoksia ei tarkistettu erikseen, sillä koodin ulkonäön parantuminen lisää aina myös sen hallittavuutta, ja täten on myös aina hyvä muutos.

3.2 Laaduntarkkailun menetelmät

Tärkeimmät laaduntarkkailuun käytetyt menetelmät olivat *staattinen tarkistus*, sekä pariohjelmointi.

Staattisella tarkistuksella tarkoitetaan yleisesti kaikkea kääntäjän varoituksista aina ”älykkäämpiin” ohjelmiin, jotka yrittävät tunnistaa ohjelmakoodista epäilyttäviä rakenteita. Tämä projekti oli kuitenkin niin suhteellisen yksinkertainen, että tyydyimme kääntäjän tarjoamiin lisävaroituksiin sekä lippuun, joka aiheutti kääntämisprosessin epäonnistumisen yhdenkin varoituksen takia. Nämä estivät jo hyvin monien epäturvallisten ohjelmointirakenteiden käytön.

Toinen laaduntarkastusmenetelmä oli pariohjelmointi. Käytännössä tämä tarkoitti sitä, että joku ohjelmoija otti työkseen jonkin ominaisuuden toteuttamisen, tai ison bugin korjauksen. Kun tämä työ alkoi lähestyä valmista, korjausehdotus käytännössä vain kirjoitettiin uudelleen kahden ohjelmoijan toimesta. Tämä näennäisesti aikaa vievä prosessi oli kuitenkin hyvin kannattava, sillä kaksi silmäparia auttoi huomaamaan paremmin logiikassa olleet aukot. (kuten sellaiset rajatapaukset, joita ei käsitelty oikein) Myöskin yleistä koodin laatua saatiin viilattua paremmaksi jo heti integraatiovaiheessa, sillä esimerkiksi *globaaleja* muuttujia saatettiin pystyä poistamaan.

4 Ohjelmistokehityksen osat

4.1 Laitteiston käyttöönotto

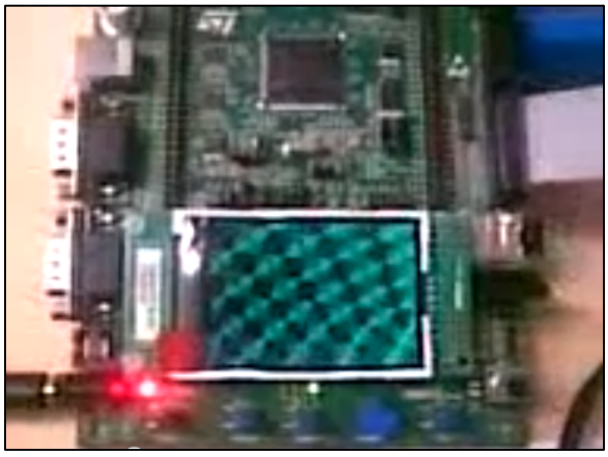
4.1.1 Tutustuminen mikro-ohjaimeen

Laitteistoon tutustuminen aloitettiin käyttämällä STM32 Evaluation Board –kittiä [6] (johon tästä lähtien viitattu nimellä Evalboard), joka sisälsi kaikki tarvittavat lisälaitteet kuten SD-korttilukijan, LCD-näytön, sekä useita liitäntöjä ulkomaailmaan. Mukana tuli myös paljon esimerkkikoodeja, joiden avulla uuden alustan käyttöönotto oli helppoa.

Evalboardin käyttö mahdollisti yhtäaikaisen ohjelmiston ja laitteiston kehityksen. Se tarjosi myös vakaan pohjan jonka päälle ohjelmistokehitys voitiin aloittaa.

Evalboardille kirjoitettiin useita erilaisia testiohjelmia ensimmäisten viikkojen aikana, jotta mikro-ohjaimen mahdollisuudet sekä yleinen työskentelyn prosessi tulisivat tutuiksi. Yksi näistä testiohjelmista on demoscenestä tuttu ns. *rotozoomer*, joka utilisoi DMA-yksiköitä kuvan siirtämiseen näyttömuistiin samalla kun prosessori laskee seuraavaa animaatioruutua. [7] [Kuva 2.]

Testiohjelmien joukossa olivat myös *raycasteri joka latasi valmiiksi* laskettua seinäsegmenttigrafiikkaa SD-muistikortilta suoraan näyttömuistiin, pyörivä kuutio, sekä useita satunnaisia testejä jotka liittyivät tiedonsiirtoon, vahtikoiran käyttöön ja muiden lisälaitteiden hyödyntämiseen.



Kuva 2. Evalboard ja rotozoomer

4.1.2 FAT-tiedostojärjestelmän ajuri

Melko varhaisessa vaiheessa kehitystä päätettiin että laitteessa olisi hyvä käyttää FAT-tiedostojärjestelmää. Vaikka se ei sovellukkaan Flash-muisteille erityisen hyvin, sen yleinen yksinkertaisuus ja hyvä tuki eri käyttöjärjestelmissä hyödyttäisi ja nopeuttaisi laitteen kehittämistä. SD-kortin käyttö laitteen massamuistina tuki myös valintaa, sillä oli hyödyllistä että muistikortin sisältöä pystyi käsittelemään myös tavallisella työasemalla

FAT-tiedostojärjestelmä perustuu siihen että tallennusmedian sektorit kasataan klustereiksi, ja kaikki tiedostot ovat tallennettu osissa näihin klustereihin. Median alussa on myös juurihakemiston klusterin numero, josta kulkemalla eteenpäin ajuri voi löytää halutun tiedoston levyltä.

Tallennusmedian alussa on taulukko käytetyistä klustereista, jota tiedostojärjestelmän ajuri käyttää löytämään tyhjän tilan levyltä. Tämä taulukko sisältää myös tiedon siitä missä klusterissa tiedosto jatkuu. Nimenomaan tämän tilanvaraustaulukon takia FAT on huono valinta Flash-muisteille koska sektorien käyttö ei jakaudu tasaisesti koko muistin alueelle.

Käyttöön valittiin ajuri nimeltään *FatFS*, joka on nimenomaan pienille mikro-ohjaimille suunnattu implementaatio FAT -ajurista. [8] FatFS on ohjelmoitu käyttäen standardia Ansi C:tä, ja sisältää paljon viritysmahdollisuuksia erilaisiin käyttötarkoituksiin. Hyvä esimerkki FatFS:n taipuisuudesta on se, että eräässä applikaatiossa sitä ajetaan vuonna 1975 markkinoille tulleella 6502-prosessorilla. [9]

Ajurin integroimiseksi sille piti toteuttaa laitteistorajapinta, eli käytännössä funktiot joiden avulla se pystyi se pystyi olemaan vuorovaikutuksessa tallennusmedian kanssa. Rajapinnan toteutuksen pohjana käytettiin erästä esimerkkikoodia joka esitteli STM32:n SDIO-väylän käyttöä.

Ajuri osoittautui erittäin vakaaksi koko projektin ajan, ja se pysyi usein toimivana jopa tiedostojärjestelmän korruptoituessa hieman esimerkiksi odottamattoman virtakatkoksen takia.

4.1.3 Konsoli

Myös *konsoli* toteutettiin kehityksen alkuvaiheessa. Tässä tapauksessa konsolilla tarkoitetaan laitteen ylläpitoon tarkoitettua komentorivikäyttöliittymää, jota pystyi käyttämään lukulaitteen sarjaportin kautta kun sen käynnisti uudelleen pitäen samalla pohjassa koti-näppäintä.

Konsolia käytettiin alunperin lähinnä bluetooth-moduulin sekä näytönohjaimen debuggaukseen. Myös tiedostojärjestelmän ajurin stressitestausta tehtiin konsolista käsin. Myöhemmin konsolin avulla suoritettiin sitten protosarjojen massatestausta, gapfillerin osoitteiden ohjelmointia, sekä korruptoituneiden tiedostojärjestelmien korjausta. Kuva 3.

Konsoli oli toteutettu yksinkertaisena rivieditorina sekä parserina, joka kutsui tarpeen mukaan erilaisia funktioita. bluetooth-moduulin testausta ja ohjelmointia varten konsolissa oli myös passtrough-ominaisuus, jolla kommunikaatio ohitti parserin, ja laitteen sarjaportin avulla pystyi kommunikoimaan suoraan bluetooth-moduulin kanssa.

```
BROADCAST PROJECT
E-PAPER PLATFORM

> PMIC_EP ON
PMIC: 13521 digital power on

> PMIC_EP_VBUS ON
PMIC: 13521 analog power on

> EP_RESET
13521: Restart successful

> EP_INIT_PLL
13521: Initializing system clock PLL...
13521: OK

> PMIC_SD ON
PMIC: SD-card power on

> SDIO_INIT
SDIO: SD-Card initialized: 4GB SDHC found
```

Kuva 3. Konsolinäkymä terminaaliohjelmassa

4.2 Tiedonsiirto

Itse televisioverkkoon liittyminen hoidettiin erillisen gapfiller-laitteen välityksellä, ja tämä *viimeinen maili* gapfilleristä lukulaitteelle hoidettiin bluetoothilla. Alunperin oli tarkoitus käyttää projektissa partnerinakin olevan Bluegigan WT12-moduulia. Sen toimitusaikataulut kuitenkin venyivät mahdottomiksi, ja hätäratkaisuna päätettiin ottaa käyttöön ARF32 –niminen moduuli. Molemmat moduulit olivat kytketty STM32:een UART-portin kautta ja molemmat käyttivät inband-signalointia. Kun Bluetooth-yhteys oli muodostettu, moduuli käyttäytyi niinkuin olisi ollut tavallinen sarjakaapeli. (Bluetoothin SPP-profiili) Näiden puitteiden sisällä käytettiin samaa protokollaa.

Käytännössä ohjelmistossa näkyneet erot näiden moduulien välillä liittyivät komentojen muotoon, sekä yhteydenmuodostuksen yksityiskohtiin. WT12 tarjosi helpon ASCII-merkeillä ohjattavan tekstikäyttöliittymän, kun taas ARF32:ta ohjattiin vaihtelevankokoisilla paketeilla, joiden muodostus ja tulkinta oli huomattavasti monimutkaisempaa. Tämän vaikutus näkyi kuitenkin pääasiassa vain kun uusi laite otettiin käyttöön, ja silloin kun bluetooth-pareja laitteiden ja gapfillerien välillä jouduttiin poistamaan tai muokkaamaan.

4.2.1 Datapurskeet ja inband-signaloinnin tuomat haasteet

Protokolla suunniteltiin siten että se hyödyntäisi mahdollisimman paljon STM32:n sisäisiä oheislaitteita. Käytännössä tämä tarkoitti DMA-yksikön käyttämistä datan vastanottoon bluetooth-moduulilta, samaan aikaan kun toista muistipuskuria kirjoitettiin SD-kortille. Nämä purskeet voitaisiin sitten mitoittaa niin, että molempiin kuluisi suurinpiirtein yhtä kauan.

Alunperin tätä kokeiltiin käyttäen tavallista FTDI:n USB-UART –adapteria, sekä logiikka-analysaattoria jolla nähtiin tarkasti purskeet ja niiden ajoitus. Koska täysin determinististä ajoitusta ei ollut mahdollista toteuttaa lähettäjä- eikä vastaanottopäässä, protokollaan määriteltiin että vastaanottaja lähettää tietyn kontrollimerkin ilmoittamaan lähettäjälle että on valmis vastaanottamaan dataa. Siirron nopeudeksi määriteltiin 460.8kBaud/s, ja ilman DMA:ta olisi prosessorille jäänyt vastaanottoon noin 17 kellosykliä per bitti.

Alunperin noin 24kB kokoisilla purskeilla saatiin lähes 99% käyttöaste siirtolinjalle. Myöhemmin laitteen ominaisuuksien lisääntyessä ja muistin vähentyessä jouduttiin siirtymään 16kB siirtopuskureihin, mutta siirtolinjan käyttöaste saatiin silti riittävän hyväksi. Hyvä linjan käyttöaste oli tärkeää, sillä se tarkoitti siirtoajan lyhenemistä ja täten pienempää virrankulutusta.

DMA-vetoisessa purskeiden vastaanottamisessa on kuitenkin se ongelma, ettei se toimi kovinkaan hyvin inband-signaloinnin kanssa. Käytännössä tämä siis tarkoittaa että jos vaikkapa linkki katkeaa, moduuli lähettää datan seassa tiedon sen katkeamisesta eikä tätä voi erottaa oikeasta datasta mitenkään. Pahimmillaan kontrolli-informaatio saatetaan tulkita esimerkiksi purskeen loppumisen kohdalla takia oikeaksi dataksi, ja tämä voi aiheuttaa datan korruptoitumista. Perinteinen tapa ratkaista tämä olisi käyttää keskeytyspohjaista vastaanottoa *escape-sekvenssien* kanssa, mutta tällöin DMA-yksiköiden käytöstä saatu nopeus/luotettavuushyöty olisi menetty. Tämän sijaan moduuleista kytkettiin kaikki kontrolli-informaation lähetys pois, ja DMA-siirron pysähtyminen havaittiin käyttämällä ajastinkeskeytystä. Näin data pysyi aina kelvollisena (kontrolli-informaatiota ei päässyt vahingossa sen joukkoon) ja tiedonsiirto voitiin keskeyttää hallitusti.

4.2.2 Protokolla

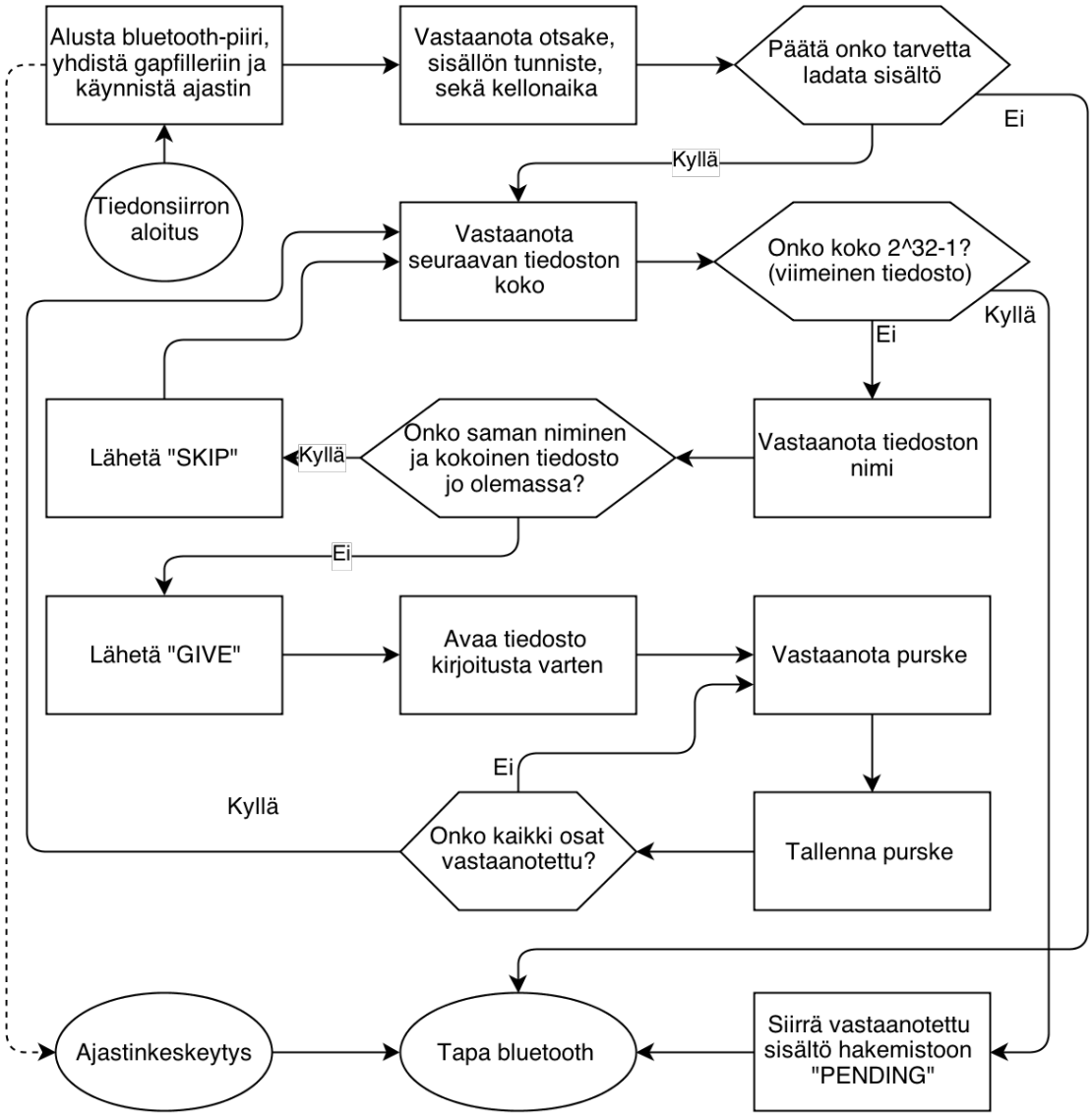
Itse tiedonsiirtoprotokolla sisältää vain muutamia ohjauskomentoja. Käytännössä lähettävä pää kertoo sisällön tunnisteen, ja vastaanottopää voi heti tässä kohtaa katkaista yhteyden jos se ei halua tarjottua sisältöä. Tämä tapahtuu esimerkiksi silloin jos tarjottu sisältö on jo ladattu. Muussa tapauksessa vastaanotetaan tiedoston nimi ja koko sekä tiedostokoon mukainen määrä purskeita. Protokollan toiminta on kuvattu kuviossa 1.

Ennen itse tiedoston vastaanottoa vastaanottava pää voi vielä tarkistaa onko kyseisen kokoinen ja niminen tiedosto jo olemassa. Tässä vaiheessa vastaanottaja voi ilmoittaa lähettäjälle että haluaa ohittaa kyseisen tiedoston. Tämä mahdollistaa keskeytyneen tiedonsiirron jatkamisen useissa tapauksissa.

Protokollassa on määritelty myös tapa kertoa nykyinen laitteelle, sillä on mahdollista että sisäinen *RTC-kello* on käynnistynyt uudelleen esimerkiksi akun loppumisen takia, ja kellonaika on tärkeä tietää esimerkiksi silloin kun kirjoitetaan lokia käyttäjän tekemisistä.

Ikään kuin protokollan osana on myös vielä pieni sisällönhallinta, mikä koostuu kahdesta hakemistosta, nimeltään *partial* ja *pending*.

Vastaanotettaessa uutta sisältöä (jonka tunniste on ennestään tuntematon) hakemisto *partial* tyhjennetään ja siirto aloitetaan. Jos sen sijaan vastaanotetaan jo *partial* hakemistossa olevalla tunnisteella olevaa tietoa, hakemistoa ei tyhjennetä vaan siirtoa jatketaan. Kun siirto on saatu valmiiksi (eli kun kaikki tiedostot on vastaanotettu), sisältö siirretään hakemistoon *pending.* Tämän jälkeen tiedonsiirtoyksikkö ilmoittaa pääohjelmalle että uutta sisältöä on ladattu, ja pääohjelma päättää omalla logiikallaan milloin *pending*-hakemiston sisältö otetaan käyttöön.



Kuvio 2. Kaavio tiedonsiirron toiminnasta

4.3 Näyttö, näytönohjain ja niiden ohjaus

E Inkin näyttöteknologia perustuu musteeseen, jonka väriä voidaan vaihtaa sähkökentän avulla. Koska kyseessä on muste, näyttö ei rasita silmiä samalla tavalla kuin esimerkiksi LCD-paneeli, joka perustuu taustavalon spektrin suodatukseen nesteessä kelluvien kiteiden avulla. Toisin kuin LCD-näyttöjen nesteessä kelluvat kiteet, E Inkin mustepartikkelit ovat öljyllä täytettyjen pallojen sisällä, ja niiden liikuttelu vaatii huomattavasti enemmän energiaa. Tämä liikuttelun vaikeus aiheuttaa E Inkin näyttöjen tärkeimmän ominaisuuden eli kuvan *bistabiiliuden*, mikä tarkoittaa että kuva pysyy näytössä ilman että sitä tarvitsee aktiivisesti ylläpitää.

Bistabiilius vaikeuttaa kuitenkin kuvan muodostamista, sillä näytön TFT-matriisi tarvitsee useita suhteellisen korkeita jännitetasoja. Näytönohjaimen pitää myös pitää kirjaa näytön nykyisestä tilasta ja tietää tarkka järjestys jännitteistä joilla pikseleitä ajetaan, eli niin sanottu *waveform*. Bistabiilien mustepartikkelien ominaisuudet vaihtelevat myös lämpötilan muuttuessa, ja useiden päivityksien jälkeen näytönohjain ei voi enää tietää partikkeleiden tarkkaa asemaa. Tämän vuoksi näyttö pitää nollata täysin silloin tällöin. Myös näyttöjen valmistusprosessin vaihtelun vuoksi eri valmistuserillä tarvitsee käyttää aina sille tarkoitettua aaltomuotoa parhaan lopputuloksen saamiseksi.

Virallisesti Epsonin S1D13521 –näytönohjaimen ohjelmointi tapahtuu käyttäen piiriin integroitua komentosekvensseriä Epsonin tarjoaman komentokoodin kanssa. Epsonin tarjoama komentokoodi helpottaa piirin käyttöönottoa, sillä se tarjoaa paljon valmiita komentoja useiden eri toimintojen suoritukseen. Komennot ovat käytännössä makroja jotka suorittavat erilaisia rekisterikirjoituksia perustuen annettuihin parametreihin. Esimerkiksi ruudun päivityksen käynnistävä komento kirjoittaa grafiikkasekvensserin rekistereihin muistiosoitteen jossa kuva sijaitsee, päivityksen koon, päivitysmenetelmän, sekä asettaa bitin, jolla päivityssykli aloitetaan.

Näytönohjaimen käyttöönotto ei ollut täysin ongelmatonta, sillä johtuen projektin kiireellisestä luonteesta laitteistosuunnitteluun ei ollut tarpeeksi aikaa. Ironisesti tämän vuoksi jouduimme käyttämään useita viikkoja ongelmien selvittämiseen, ja näytönohjaimen ajuria ei saatu viilattua paljoakaan sen jälkeen kun se oli todettu toimivaksi.

Ajurin rajapinta muodostui lopulta melko ohueksi kerrokseksi, jonka funktiot olivat lähinnä wrappereita Epsonin komentokoodille. Ajuri tarjosi seuraavat ominaisuudet

- näytönohjaimen käynnistämisen, sekä *workaroundin* häiriöongelmaa varten, jonka vuoksi näytönohjaimen kellogeneraattori ei vakautunut luotettavasti
- näytönohjaimen virtatilan vaihtamisen
- komentokoodin ja waveformin flash-muistin ohjelmoimisen
- kuvan lataamisen näytönohjaimen muistiin
- kuvan päivittämisen näytölle

Kuvan lataamiseksi muistiin käytettiin jälleen DMA-yksiköitä, jolla pystyttiin lähettämään kuvadataa näyttömuistiin samaan aikaan kun sitä ladattiin SD-kortilta. Nopea kuvan lähetys muistiin oli tärkeää, sillä monissa tapauksissa se piti suorittaa näppäimen painalluksen ja näytönpäivityksen välissä. Kuvan lähettämiseen kuluva aika on käänteisesti verrannollinen käyttökokemuksen sujuvuuteen, ja tämän takia sen optimointi oli ensiarvoisen tärkeää.

Näytönohjaimen käynnistyksen kanssa ilmenneet ongelmat johtuivat ilmeisesti hieman epävakaasta kellosignaalista, ja jostain syystä näytönohjain jäi välillä odottelemaan ikuisesti sen vakautumista. Ratkaisuna tähän kirjoitettiin rutiini, joka käynnisti näytönohjaimen aina uudelleen jos kellosignaali ei vakautunut tarpeeksi lyhyessä ajassa.

Ongelmana oli myös näyttömuistin tyhjeneminen aina kun näytönohjaimelta katkaistiin virta. Luotettavaa ruudunpäivitystä varten näytönohjaimen tarvitsi tietää näytöllä oleva kuva, ja virrankatkaisun jälkeen tämä ei ollut mahdollista. Yleisesti E Inkin näyttöjä käyttävät lukulaitteet tyhjentävät ruudun kokonaan tälläisissä tilanteissa, mutta tässä laitteessa melkein 700 millisekuntia kestävä välähdys olisi ollut sietämätön, sillä laitteessa ei saisi olla erillistä käyttäjälle näkyvää on/off –tilaa. Kompromissina riitti, että käynnistymisen jälkeen ensimmäisten sivunvaihtojen yhteydessä suoritettiin kaksi peräkkäistä ruudunpäivitystä, joka riitti putsaamaan näyttöä tarpeeksi ilman käyttökokemuksen muuttumista.

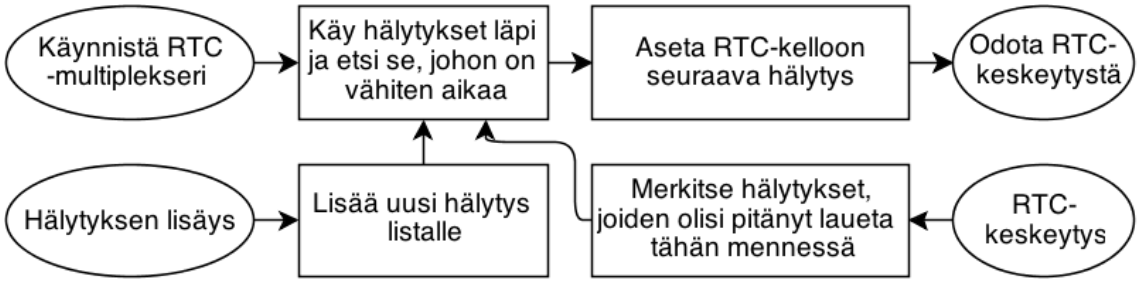
4.4 Muut ohjelman osat

4.4.1 RTC-monivalitsin

STM32:n sisältämä RTC-kello sijaitsee samalla piillä erillsenä yksikkönä, ja se pysyy käynnissä myös tilanteissa joissa esimerkiksi itse Cortex-M3 –ydin on pysäytetty. Tämä RTC-kello on myös ainut sisäinen lisälaite, joka pystyy herättämään virransäästötilassa olevan mikro-ohjamen keskeytystä käyttäen. RTC-kello käyttää sisäistä matalan kellotaajuuden oskillaattoria (LSI) ja korottaa laskuria. Sille voidaan asettaa myös hälytyskeskeytys, mikä laukaistaan kun laskuri saavuttaa tietyn arvon.

Lukulaitteessa oli kuitenkin tarvetta asettaa useita tapahtumia, joiden piti herättää nukkuva mikro-ohjain.Tätä varten hälytyskeskeytykseen implementoitiin monivalitsin, joka hallinoi useita virtuaalisia hälytyskeskeytyksiä yhdellä fyysisellä keskeytyksellä. Monivalitsimen toiminta on kuvattu kuviossa 3.

Virtuaalisia keskeytyksiä varten määriteltiin käyttöliittymä, jolla tapahtuman saattoi ajastaa tapahtumaan tietyn ajan kuluttua. Jokaisen ajastuksen yhteydessä monivalitsin kävi läpi kaikki virtuaaliajastimet ja asetti fyysisen ajastimen sen virtuaaliajastimen mukaisesti, jonka pitäisi hälyttää ensimmäisenä. Ajastinkeskeytyksessä kaikkia virtuaaliajastimia edistettiin oikeasti kuluneen ajan verran ja fyysinen ajastin asetettiin jälleen. Näitä virtuaaliajastimia käytettiin sekä tiedonsiirron aloittamiseen, SD-kortin virransäästötilan hallintaan että naisen ovulaatiosyklin käynnistymiseen.

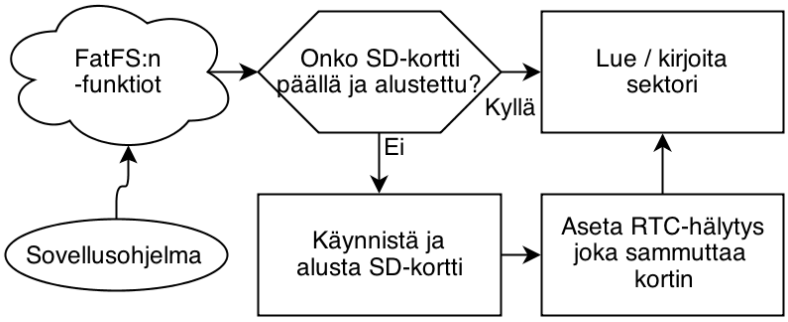


Kuvio 3. RTC-monivalitsimen toiminta

4.4.2 SD-kortin virransäästö

SD-kortin virransäästö toteutettiin FAT-ajurin tarvitseman laitteistorajapinnan sisällä. Sektorien lukemiseen ja kirjoittamiseen tarkoitetuissa funktioissa kytkettiin aina tarvittaessa SD-korttiin virta päälle, sekä asetettiin ajastin jolla se kytkettiin pois päältä. Aluksi suunniteltiin, että SD-kortin virranhallinta hoidettaisiin samalla tavoin kuin näytönohjaimenkin, jossa ohjelma kytki eksplisiittisesti virran päälle korttiin kun sitä tarvittiin. Pian kuitenkin todettiin, että implisiittinen ratkaisu olisi järkevämpi.

Tiedostojärjestelmän käsittely on kutakuinkin järjestelmän perusominaisuus ja luonteeltaan niin tiheää, että eksplisiittinen virranhallinta olisi tuonut ohjelmaan tarpeetonta monimutkaisuutta, ja mahdollisesti jopa aiheuttanut tilanteita jossa korttia oltaisiin kytketty päälle ja pois vähän väliä. Virran kytkemistä edestakaisin haluttiin välttää, sillä kortin käynnistämiseen liittyi aina myös sen alustuprosessi, joka olisi useasti tehtynä aiheuttanut paljon turhaa työtä. SD-kortin virranhallinta on kuvattu kuviossa 4.



Kuvio 4. SD-kortin virranhallinta

4.4.3 LED-valon ohjaus

Omassa laitteistossamme oli LED-valo, jota käytettiin lähinnä sisäisen tilan visualisointiin. Lediä välkytettiin esimerkiksi tiedonsiirron yhteydessä aina uuden paketin saavuttua, sekä kun kuvaa ladattiin näytönohjaimen muistiin. Näin prototyyppilaitteiden sisäisestä toiminnasta sai suurpiirteisen kuvan, joka helpotti ongelmatilanteiden ratkaisua.

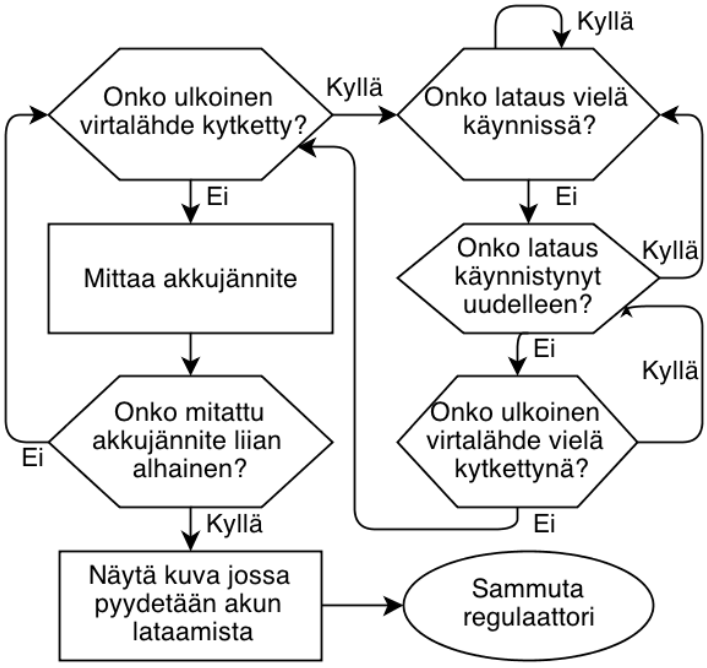
Ulkoisen virtalähteen ollessa kiinni LED-valo saatiin sykkimään hitaasti PWM-kontrolleria käyttäen. Ledin eri sykkimiskuvioiden hallintaan kirjoitettiin rutiini joka toteutti ikään kuin pinon, ja näin ledillä pystyi signaloimaan relevanttia informaatiota, vaikka tapahtumat vaihtuivatkin nopealla tahdilla.

4.4.4 Latauksen logiikka ja akun tilan tunnistus

Akkulaturin indikaattorilogiikka jouduttiin toteuttamaan melko monimutkaisella tavalla, sillä alkupään laitteistoversioissa latauspiiri ei ilmoittanut häiriöiden takia aina luotettavasti että oltaisiin kiinni laturissa. Ongelmana oli myös se, että laturipiiri statuspinnin lukemiseen käytetty STM32:n GPIO-pinni pystyi tunnistamaan vain kaksi tilaa (0V ja 3.3V), kun taas latauspiirin statuspinni oli kolmitilainen. (0V, 3.3V ja Hi-Z). Itse lataus hoitui piirin toimesta täysin autonomisesti. Latauksen logiikka on kuvattu kuviossa 5.

Myös tilanteessa, jossa ulkoista virtalähdettä ei ollut kytketty, latauslogiikan yhteydessä mitattiin akkujännitettä. Akkujänniteen mittaamiseen käytettiin STM32:n sisäistä ADC-muunninta, jonka digitoimasta arvosta poistettiin kohinan ja näytönpäivityksen aiheuttamat varianssit käyttäen juoksevaa keskiarvoa. Keskiarvon ollessa liian alhainen, näytölle ajettiin kuva joka pyysi akun lataamista ja laitteen pääregulaattori ajettiin alas.

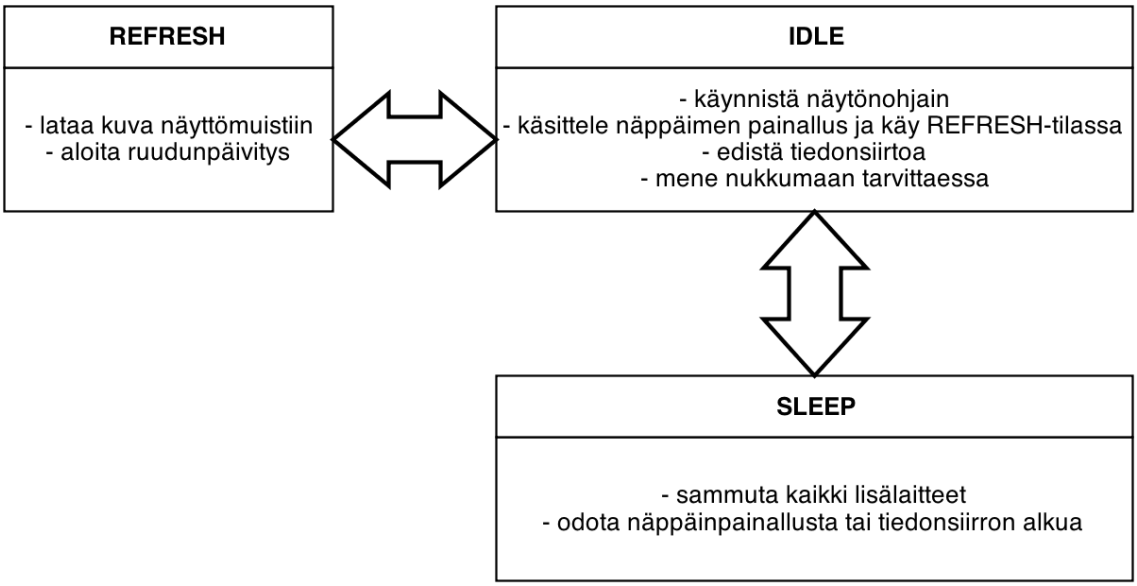
Tämän kolmitilaisuuden emuloimiseksi GPIO-pinnin sisäisiä ylös- ja alasvetovastuksia käytettiin vuorotellen siten, että pystyttiin arvioimaan todennäköisyys latauksen tilasta. Koska ylös- ja alasvetovastukset ovat melko suuriresistanssisia (500kohm), jouduttiin pinnin tilan lukemista myös viivyttämään jonkun aikaa että sen sisäinen kapasitanssi ehti purkautua.



Kuvio 5. Latauksen logiikka

4.5 Lukulaitteen pääohjelma

4.5.1 Pääohjelman ensimmäinen prototyyppi



Kuvio 6. Kuvio pääohjelman ensimmäisestä versiosta

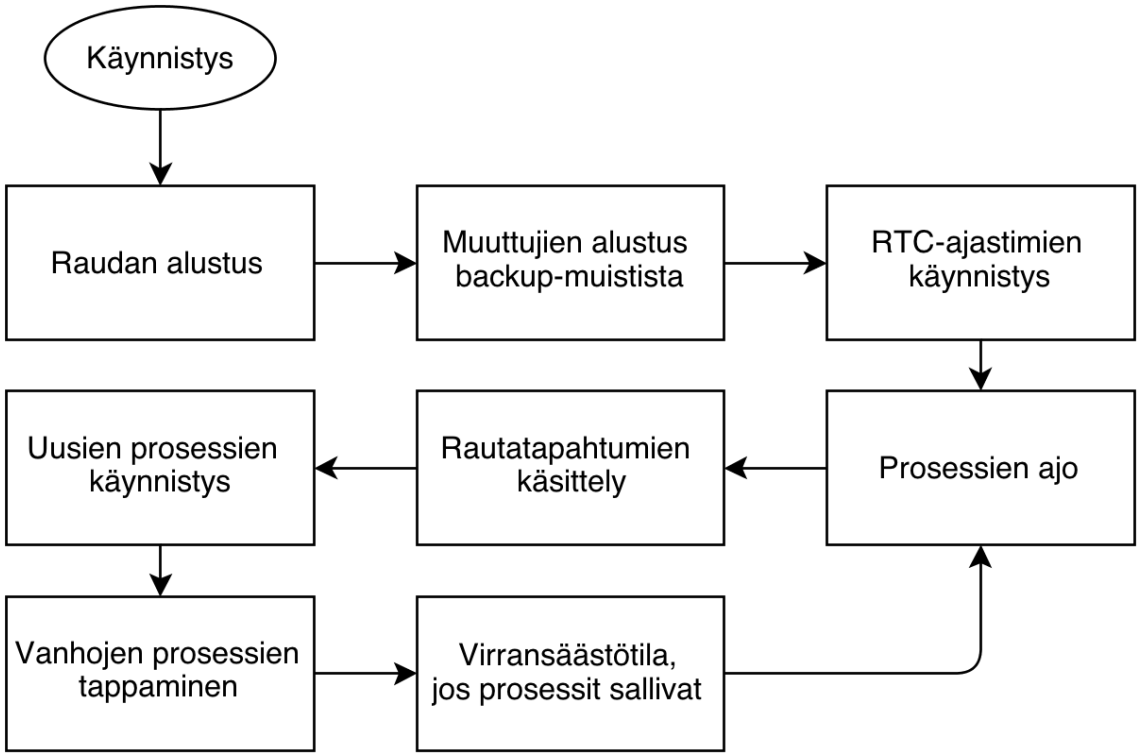
Tämä primitiivinen pääohjelma koostui kolmesta tilasta nimeltään *SLEEP, IDLE* ja *REFRESH.* Tilassa *SLEEP* laite sammutti kaikki lisälaitteet ja jäi odottelemaan näppäinpainalluksen aiheuttamaa keskeytystä. *IDLE*-tilassa taas laite käsitteli näppäinpainalluksen aiheuttaman keskeytyksen ja käytti laitteen *REFRESH*-tilan kautta, jossa suoritettiin ruudunpäivitys. Nämä tilat voi nähdä visuaalisesti kuviossa 5.

Kun laitteen kaikki yksittäiset ohjelman osat alkoivat konkretisoitua, kävi selväksi että tällainen tilakone ei todellakaan tulisi riittämään pitkälle. Tiedonsiirto esimerkiksi tapahtui käyttäen corutiinia ja tätä kutsuttiin *IDLE*-tilan silmukasta. Vaikka tämä oli sinäänsä toimiva lähestymistapa, kaikki tilasiirtymät vaativat usean poikkeuksen tarkistamista erikseen ja rupesi vaikuttamaan että ohjelma kasvaisi pian liian monimutkaiseksi.

4.5.2 Prosesseihin perustuva ”käyttöjärjestelmä”

Yhteistyömoniajoon perustuva pääohjelma perustuu ajatukseen, että kaikki järjestelmän osat ovat *prosesseja*, joiden syntymä ja kuolema hallitsevat järjestelmän lisälaitteita. Prosessit pystyvät käynnistämään sekä tappamaan muita prosesseja perustuen erilaisiin laitteistotapahtumiin. Esimerkiksi nukkumisprosessissa näppäinpainallus käynnistää lukulaiteprosessin, joka taas puolestaan tappaa itsensä ja käynnistää nukkumisprosessin, jos laite on ollut käyttämättä tarpeeksi pitkään. Samoin tiedonsiirron RTC-hälytys käynnistää tiedonsiirtoprosessin.

Virransäästön hallinta perustuu ajatukseen, että prosessit voivat dynaamisesti määrittää ne lisälaitteet joita tarvitsevat eri tilanteissa ja käyttöjärjestelmä menee syvimpään unitilaan, missä nämä laitteet ovat toiminnassa. Esimerkiksi tiedonsiirto tarvitsee DMA-yksiköitä jatkuvasti, jonka takia tiedonsiirron aikana itse Cortex-M3 –ydin pystyy nukkumaan, mutta syvempiin unitiloihin ei mennä. Käyttöjärjestelmän prosessisilmukan toiminta näkyy kuviossa 6. Keskeytys (esim. näppäinpainallus) herättää laitteen aina.



Kuvio 7. Prosessipohjaisen ”käyttöjärjestelmän” toiminta

4.5.3 Pääohjelman lopullinen versio

Prosessijärjestelmän päälle rakennettiin sitten lukulaitteen käyttöliittymän toimintalogiikka. Käyttöliittymä koostui kolmesta erillisestä prosessista nimeltään *KEYWAKE, READER* sekä *PAGECACHER.* Toiminta näkyy kuviossa 8.

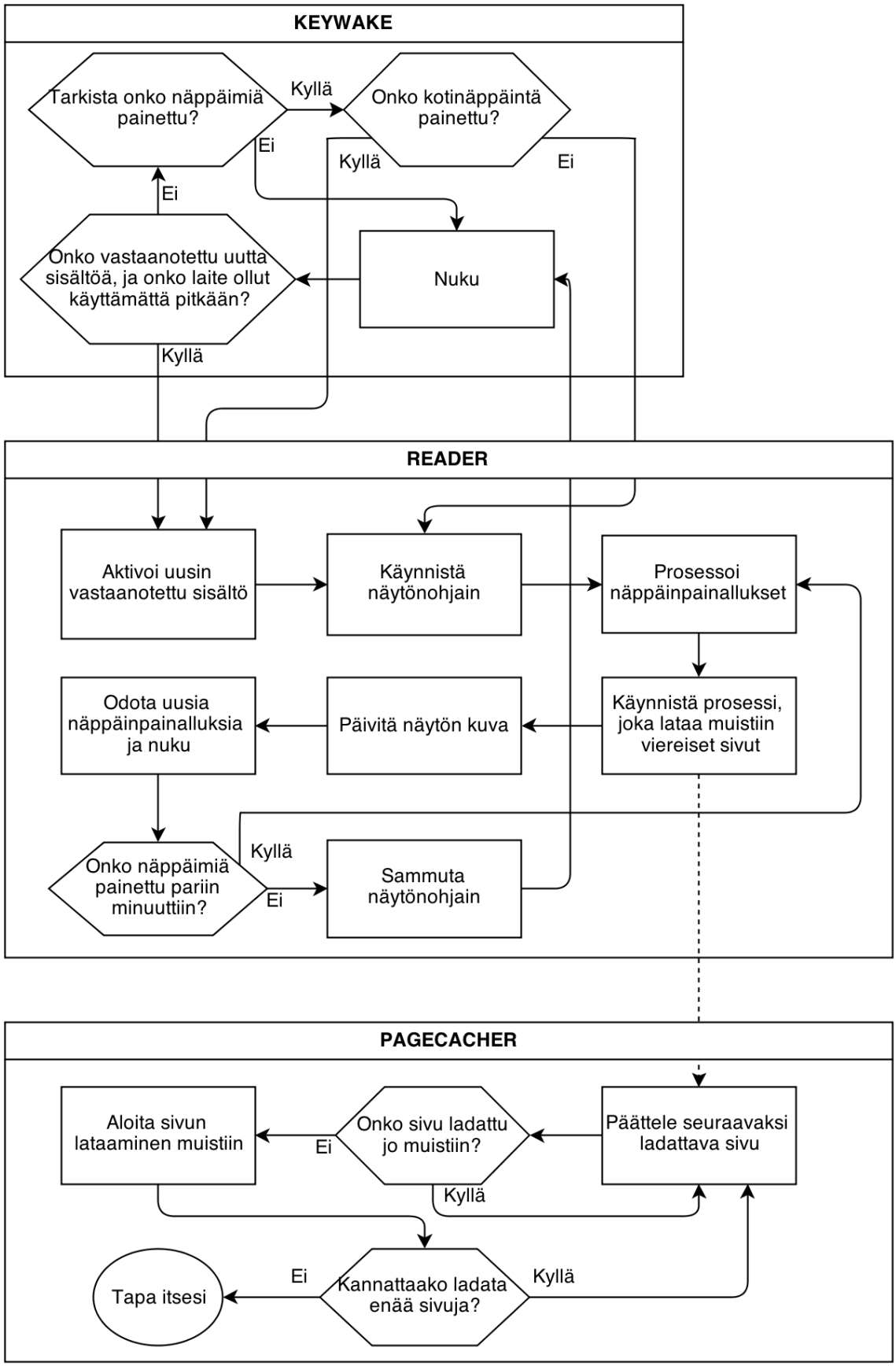
Suurin hyöty uudesta pääohjelmasta ja prosessipohjaisesta toteutuksesta oli se, että ohjelmakoodin laatu parani huomattavasti. Eri osat saatiin erottumaan hyvin toisistaan, mikä taas puolestaan nosti kehityksen tehokkuutta, kun lisälaitteiden hallinta ei tarkoittanut enää alati monimutkaistuvaa poikkeuksien ja erikoisehtojen verkkoa.

4.6 Virhetilojen hallinta

Virhetilojen hallintaan laitteessa käytettiin STM32:n sisäistä vahtikoira-lisälaitetta, joka käynnisti laitteen uudelleen, jos tiettyä funktiota ei kutsuttu tasaisin väliajoin. RTC-hälytystä käytettiin kutsumaan funktiota laitteen ollessa käyttämättömänä. Lukulaite myös kirjoitti aina erilliseen backup-muistiin nykyisen sivun numeron, ja vahtikoira-uudelleenkäynnistyksen tapahtuessa laite avasi suoraan sen sivun joka oli tallennettu.

Vahtikoira toimi loppujenlopuksi erittäin hyvin, ja monet virheet pystyttiin hoitamaan kokonaan täysin käyttäjän huomaamatta. Useissa tapauksissa laitteen jumiutuminen johtui kuitenkin näytönohjaimen käynnistysvaikeuksista, ja tämän vuoksi kaatuminen näkyi käyttäjälle hieman hitaana reagoimisena näppäinpainallukseen.

STM32:n vahtikoira lisäsi valitettavasti myös virrankulutusta huomattavasti. Ajatuksena oli, että vahtikoira voitaisiin laittaa virransäästötiloissa pois päältä. Tämä osoittautui kuitenkin mahdottomaksi, sillä STM32 ei salli vahtikoiran sammutusta enää sen jälkeen kun se on kerran käynnistetty. Ilman vahtikoiraa laitteen virrankulutus olisi ollut vain kymmeniä mikroampeereja, mutta vahtikoiran kanssa kulutus nousi lähes milliampeeriin. Tämä ei kuitenkaan ollut varsinaisesti ongelma, sillä akunkesto pysyi silti useassa viikossa ja laitteen akku latautui ulkoisella virtalähteellä parissa tunnissa.



Kuvio 8. Pääohjelman toimintalogiikka ja prosessit

5 Lopputulos ja prosessin arviointi

Laiteet olivat lopulta niin luotettavia, että ne kestivät kuljetuksen Suomesta Kiinaan, kuukauden ajan käytön pilottiperheissä, sekä kuljetuksen takaisin. Vahtikoiran käyttämisestä oli varmastikin paljon hyötyä, sillä pilottien loppua lähestyttäessä useat prototyyppiyksilöt alkoivat tuntua hieman epävakailta. Erityisesti näytönohjaimen käynnistyminen muuttui vaikeaksi useassa laitteessa, mutta näyttöohjaimen käynnistysrutiini piti huolen että laitteet jatkoivat toimintaansa.

Projekti onnistui siinä mielessä, että kaikki sen tärkeimmät vaatimukset saatiin toteutettua laitteeseen. Käyttöjärjestelmättömyys mahdollistaa hyvin hienovaraisen virrankulutuksen viilaamisen, mutta ominaisuuksien ja monimutkaisuuden lisääntyessä saattaa olla parempi idea rakentaa ohjelma valmiin RTOS:n päälle.

Ihmiset arvostivat laitteen yksinkertaista käyttöliittymää. Vaikka erityisesti Kiinassa laitetta vertailtiin tabletteihin useasti, monet kertoivat että *appsittomuus* itse asiassa auttoi lukukokemusta, sillä häiriötekijöiden määrä luettaessa oli minimoitu. Todellisessa tuotteessa pitäisi kuitenkin olla hieman enemmän funktionaalisuutta, kuten mahdollisuus saada useita lehtiä laitteeseen ja vaihdella niiden välillä. Myös kosketusnäyttöä kaivattiin.

Itse ohjelmointitekniikat olivat enemmän tai vähemmän tuttuja jo harrastepohjalta, mutta todellisen elämän ohjelmistokehityksestä ei ollut paljoakaan kokemusta. Erityisesti ryhmätyöskentelyn tärkeys nosti päätään kokonaan uudella tavalla, sillä aiempi kokemus oli lähes yksinomaan projekteista, joissa olin ainut ohjelmoija. Tämä johti helposti alussa tilanteisiin joissa en ottanut ryhmää huomioon sen vaatimalla tavalla, ja yritin tehdä liian montaa ominaisuutta yksin. Hyvän projektipäällikön merkitystä ei myöskään sovi vähätellä tilanteissa jossa etenemissuunta ei ole täysin selvä.

Myös arkkitehtuurisuunnitteluun pitää kiinnittää aivan eri tavalla huomiota, kun kehittäjiä on useita. Yhden ihmisen kotiprojekteissa on aina mahdollista tehdä isojakin muutoksia minne tahansa päin järjestelmää, mutta moniosaisen projektin suunnitteluun etukäteen tarvitsee kiinnittää aivan eri tavalla huomiota. Myös keskustelu on erittäin tärkeää, sillä yhden ihmisen mieli näkee ainoastaan hyvin rajallisesti heikkoudet omissa suunnitelmissaan.

Lähteet

Lähdetieto Lähdetieto Lähdetieto Lähdetieto Lähdetieto Lähdetieto Lähdetieto Lähdetieto Lähdetieto. (Käytetään tyyliä Lähde).

Lähdetieto Lähdetieto Lähdetieto Lähdetieto Lähdetieto Lähdetieto Lähdetieto Lähdetieto Lähdetieto

1 Electria: Broadcast information distribution platform project <http://electria.metropolia.fi/index.php?option=com_content&view=article&id=64&Itemid=68> Luettu 20.3.2013.

2 Mentor Graphics CodeSourcery <http://www.mentor.com/embedded-software/codesourcery> Luettu 20.3.2013.

3 Tin Can Tools: Flyswatter <http://www.tincantools.com/product.php?productid=16134> Luettu 20.3.2013.

4 Open On-Chip Debugger <http://openocd.sourceforge.net/> Luettu 20.3.2013

5 E Ink: Green <http://www.eink.com/green.html> Luettu 20.3.2013.

6 ST Microelectronics: STM3210E-EVAL <http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF204176> Luettu 21.3.2013.

7 YouTube: STM32 CRAZY DEMO EFFECT <http://www.youtube.com/watch?v=UuxbUVqV2lA> Luettu 21.3.2013.

8 FatFS – Generic FAT File System Module <http://elm-chan.org/fsw/ff/00index_e.html> Luettu 21.3.2013.

9 1541 Ultimate <http://www.1541ultimate.net/content/index.php> Luettu 21.3.2013.

10

11

12

13

14

15

16

17

18

19

20 terve

21 Lähdetieto Lähdetieto Lähdetieto Lähdetieto Lähdetieto Lähdetieto Lähdetieto Lähdetieto Lähdetieto.

Liitteen otsikko

Tähän kirjoitetaan liitteen sisältö. Alla on ohje liitteiden poistamiseksi ja lisäämiseksi siten, että ylätunnisteet säilyvät oikeanlaisina

Ohje tarpeettoman liitteen poistamiseksi:

1. Valitse ensin kokonaisuudessaan liitteenä oleva sivu ja poista sen sisältö Delete-näppäimellä.

2. Kun olet tyhjentämäsi liitesivun alussa, paina kerran askelpalautinta (Backspace), jolloin liitettä edeltävä osan vaihto poistuu.

Ohje uuden liitteen lisäämiseksi:

1. Siirrä kohdistin viimeisen olemassa olevan liitesivun loppuun.

2. Valitse Sivun asettelu ja valintanauhasta Vaihdot / Osanvaihdot - Seuraava sivu. Näin loppuun tulostuu uusi liite, mutta sen ylätunnisteessa oleva numero ei ole oikea.

3. Kaksoisnapauta uuden liitesivun ylätunnistetta, jossa on väärä liitteen numero. Jos valintanauhassa näkyy nyt valittuna vaihtoehto ”Linkitä edelliseen”, paina kyseistä painiketta siten, että vaihtoehto ei enää ole valittuna.

4. Korjaa liitteen numero oikeaksi.

Huomaa, että liitteet on päivitettävä sisällysluetteloon manuaalisesti.

Liitteen otsikko

Tähän kirjoitetaan liitteen sisältö.