Maarit Tötterman

Ohjelmapohjan toteuttaminen TFT-näytön käyttöönottoa varten

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

14.1.2013

Metropolia

Tekijä Otsikko Sivumäärä Aika	Maarit Tötterman Ohjelmapohjan luominen TFT-näytön käyttöönottoa varten 50 sivua + 2 liitettä 14.1.2013
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	tietoliikennetekniikka
Ohjaajat	projektipäällikkö Kai Tötterman lehtori Kimmo Sauren

Insinöörityön tavoitteena oli toteuttaa ohjelmapohja TFT-näytön ohjaamiseen. Ohjelmapohja tehtiin Amplition Oy:n käyttöön, ja sen tarkoitus on helpottaa TFT-näyttöjen käyttöönottoa tulevissa projekteissa.

Opinnäytetyössä valittiin kehitysympäristö yrityksen ennalta valitsemalle mikro-ohjaimelle. Työssä käytettiin SAM3S-EK2-kokeilukorttia ja siinä olevaa Atmel SAM3SD8 -mikro-ohjainta. Kehitysympäristöksi valittiin IAR Embedded Workbench for ARM -ohjelma. Valintaan vaikuttivat asiantuntijan mielipide, monipuoliset työkalut ja esimerkkiohjelmien sekä tuen saatavuus. Insinöörityössä käytettiin myös Ramtex Graphic Dot Matrix Color LCD Driver -grafiikkakirjastoa sekä Color LCD Icon Editor -ohjelmaa.

Ennen varsinaisen ohjelmapohjan kehittämistä tutustuttiin kokeilukortin ominaisuuksiin ja internetistä saataviin esimerkkiohjelmiin. Piirivalmistajat tarjoavat useasti esimerkkiohjelmia ja valmiita kirjastoja, joita voidaan hyödyntää tuotekehityksessä. Atmel tarjoaa kattavat kirjastot insinöörityössä käytetylle kokeilukortille, ja näitä kirjastoja hyödynnettiin myös ohjelmapohjaa luotaessa.

Ohjelmapohjaan integroitiin myös grafiikkakirjasto, jotta ohjelmalla pystytään tulostamaan näytölle helposti grafiikkaa. Grafiikkakirjasto on kokoelma funktioita, joiden on tarkoitus helpottaa ohjelmoijan työtä. Kirjastossa on valmiit funktiot yksinkertaisien kuvioiden piirtämiseen näytölle sekä funktiot, joiden avulla näytölle saadaan vietyä valmis kuva. Kirjastoon voi myös lisätä itse tehtyjä fontteja.

Lopputuloksena työssä saatiin sovellusohjelma, joka todistaa ohjelmapohjan toimivuuden. Sovellusohjelma tulostaa näytölle mediasoittimen käyttöliittymän, jossa on tekstiä, kuva ja symboleita, sekä juokseva kello. Sovellusohjelmassa käytetään myös yhtä kokeilukortilla olevaa painiketta, joka esittää mediasoittimen play/pause-nappia. Lopputulosta voidaan pitää onnistuneena, koska ohjelmapohja toimii, ja näytölle pystyttiin tulostamaan halutut asiat. Lisäksi ohjelmapohjan ansiosta käyttöliittymän grafiikan päivittäminen on helppoa.

Avainsanat	mikro-ohjain, C-ohjelmointikieli, sulautettu järjestelmä, TFT-näyttö

Author Title Number of Pages Date	Maarit Tötterman Creating a software base for taking TFT-display into use 50 pages + 2 appendices 14 January 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Telecommunications and Data Networks
Instructors	Kai Tötterman, Project Manager Kimmo Sauren, Senior Lecturer

The purpose of this bachelor's thesis was to create a code base in C-programming language that will take a TFT-display into use. The program was created for Amplition Oy. During this project an integrated development environment (IDE) was chosen for the Atmel SAM3SD8 microcontroller. The microcontroller that was used was chosen by Amplition Oy. Amplition also ordered an evaluation kit called SAM3S-EK2 from Atmel.

IAR Embedded Workbench for ARM was chosen as the IDE for this project. The decision was made based on expert opinion, the selection of tools and the demo projects that are available for this software. Amplition also purchased a graphics library from Ramtex for this project. Before actually designing the program base it was necessary to get familiar with the evaluation kit and libraries that are offered by Atmel for the evaluation kit. The graphics library was also included in the program base.

As a result of this project a working program base was created. This was showcased with a simple program that prints a media player user interface on the display. There is text with self-created font, pictures created from bitmap images and also a running clock. Because the software base works it can be concluded that the project was successful.

Keywords	Embedded system, C-programming, TFT-Display, microcontroller

Sisällys

1 Johdanto 3

2 Järjestelmän komponentit 3

2.1 Kokeilukortin ominaisuudet 5

2.2 TFT-näyttö 6

2.3 Mikro-ohjain 9

3 Näytön ohjaaminen 10

3.1 Näytön liittäminen mikro-ohjaimeen 11

3.2 Näytön päivitysnopeus 13

3.3 Värisyvyys 14

3.4 Grafiikan tallentaminen muistiin 17

4 C-ohjelmointikieli 18

4.1 C-ohjelmointi sulautetuissa järjestelmissä 18

4.2 Ohjelmakirjastojen käyttäminen 19

4.3 Virheiden etsintä 20

5 Kehitysympäristö 20

5.1 Kehitysympäristön valinta 20

5.2 JTAG-emulaattori 21

5.3 Grafiikkakirjasto 22

6 Työn valmistelu 23

6.1 Käytössä olevat työkalut ja laitteet 23

6.2 Esiasennettuun kokeiluohjelmaan tutustuminen 24

6.3 Netistä saatavat esimerkkiohjelmat 26

6.4 Projektin käyttöönotto IAR-kehitysympäristössä 27

6.5 Mikro-ohjaimen ohjelmoiminen USB-portin kautta 27

6.6 Mikro-ohjaimen ohjelmoiminen JTAG emulaattorilla 30

6.7 Valittuun esimerkkiohjelmaan tutustuminen 31

7 Työn toteutus 32

7.1 Uuden projektin luominen IARiin 32

7.2 Projektin asetukset IARissa 33

7.3 Ohjelmapohjan luominen 35

7.4 Grafiikkakirjaston käyttö 38

7.5 Sovellusohjelma 40

7.6 Lopputuloksen arviointi 45

8 Yhteenveto 47

Lähteet 49

Liitteet

Liite 1. Sovellusohjelman vuokaavio

Liite 2. Sovellusohjelman ohjelmakoodi

1 Johdanto

Amplition Oy on vuonna 2007 perustettu yritys, joka kehittää elektroniikkaa kuluttajakäyttöön. Yritys on aloittamassa uusia tuotekehitysprojekteja, joissa tullaan käyttämään graafisia TFT-näyttöjä. Insinöörityö tilattiin näitä projekteja silmälläpitäen.

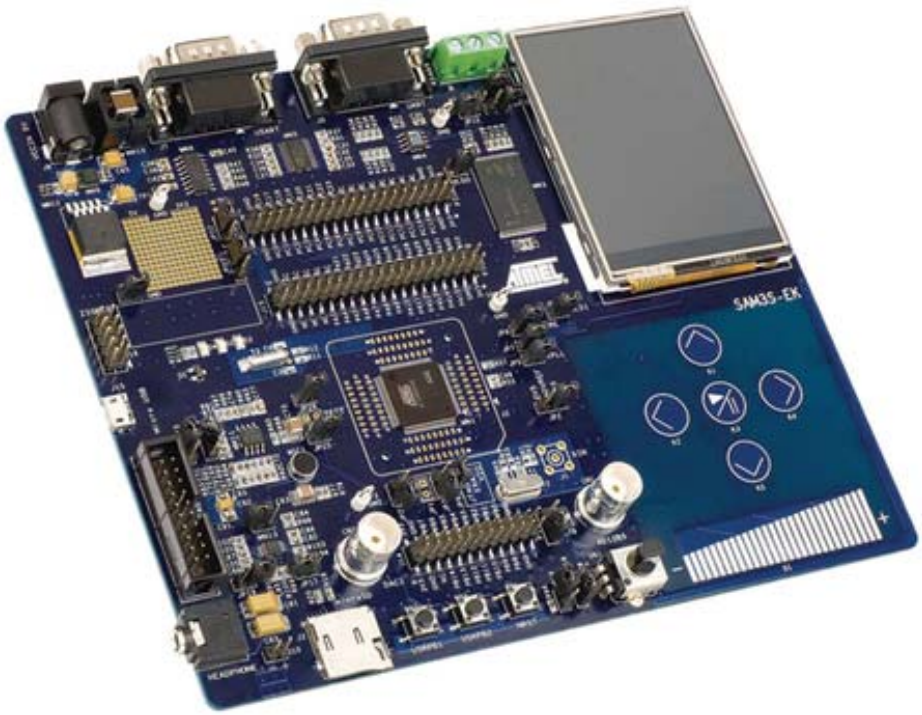
Insinöörityön tavoitteena oli valita sopiva kehitysympäristö Amplition Oy:n ennalta valitsemalle Atmel SAM3SD8 -mikro-ohjaimelle ja luoda ohjelmapohja C-ohjelmointikielellä. Tarkoitus on, että ohjelmapohja mahdollistaa jatkossa TFT-näytön (Thin-Film Transistor) käyttöönoton nopeasti ja helposti yrityksen tulevissa tuotekehitysprojekteissa.

Amplition oli tilannut insinöörityötä varten Atmel SAM3S-EK2 kokeilukortin, joka toimi koko projektin pohjana. Kokeilukortin lisäksi Amplition oli tilannut myös SAM-ICE-JTAG-emulaattorin mikro-ohjaimen ohjelmointia ja ohjelman virheen etsintää varten. Insinöörityön aikana tilattiin lisäksi Ramtex Graphic Dot Matrix Color LCD Driver -grafiikkakirjasto, Color LCD Icon Editor -ohjelma ja FTDI TTL-232R-3V3 sarjaliikennemuunnin.

Insinöörityöraportissa perehdytään ensin TFT-näyttötekniikkaan ja näytön käyttöön vaikuttaviin tekijöihin, kuten värisyvyyteen ja päivitysnopeuteen. Raportissa käydään läpi myös kehitysympäristön valinta ja työssä käytettävät työkalut. Tarvittavien valmistelujen jälkeen kerrotaan ohjelmapohjan toteutuksesta ja sen päälle tehdystä sovellusohjelmasta.

2 Järjestelmän komponentit

Mikropiirien valmistajat suunnittelevat monesti erilaisia kokeilukortteja, joilla voidaan mikropiirin suorituskyvyn arvioinnin lisäksi helpottaa myös tuotekehitystyötä. Piirivalmistajat antavat yleensä ilmaiseksi kokeilukorttien piirikaaviot, osaluettelot, valmistustiedostot sekä ohjelmakoodit, joita käytetään tuotekehitysprosessin pohjana.



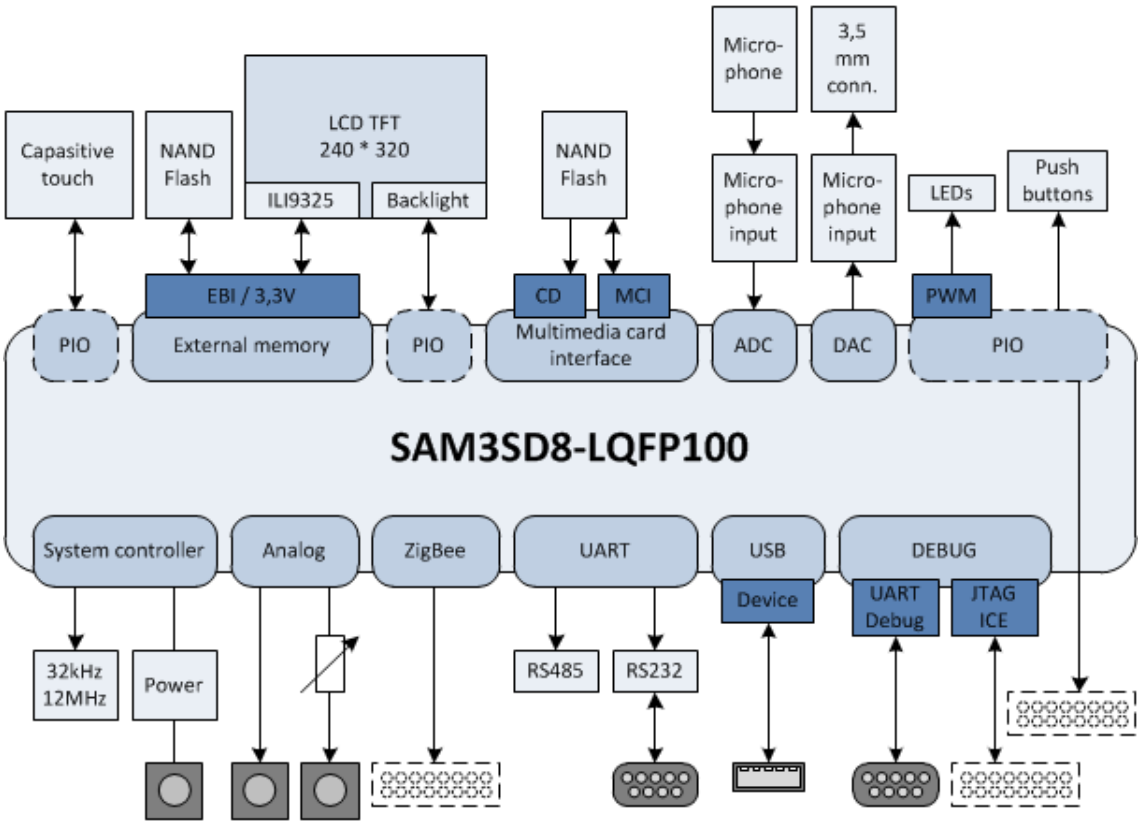
Kuva 1. SAM3S-EK2-kokeilukortti [1].

Atmel on suunnitellut insinöörityössä käytettävälle mikro-ohjaimelle oman kokeilukortin, SAM3S-EK2 (kuva 1). Kortilla on mikro-ohjaimen lisäksi useita erilaisia lisätoimintoja, kuten painonappeja, merkkivaloja, muistikorttipaikka, mikrofoni sekä kuulokeliitin. Kokeilukortilla on myös työssä käytetty 2,8”:n TFT-näyttö. Kortilla olevaa mikro-ohjainta voidaan ohjelmoida sekä USB-portin että JTAG-liitynnän kautta.

Atmelin kotisivuilta on saatavana kokeilukortin datalehden lisäksi kaikki sen valmistustiedostot ja kytkentäkaaviot. Kytkentäkaavioita tarvittiin insinöörityössä mm. määriteltäessä mikro-ohjaimen I/O-nastoja. Kotisivuilta on lisäksi saatavana useita erilaisia C-kielisiä esimerkkiohjelmia sekä piiriin liittyviä ohjelmakirjastoja. Nämä esimerkkiohjelmat ja kirjastot on tarkoitettu tuotekehitysprosessin tueksi. Sulautettuja järjestelmiä ohjelmoitaessa ei ole tarkoitus kirjoittaa kaikkea itse, vaan hyödyntää mahdollisuuksien mukaan valmistajan tarjoamia valmiita ohjelmia ja kirjastoja.

2.1 Kokeilukortin ominaisuudet

SAM3S-EK2-kokeilukortti on tehty Atmel SAM3SD8 -mikro-ohjaimen ympärille. Mikro-ohjain on 100-jalkaisessa LQFP100-kotelossa. Mikro-ohjaimen ympärillä ovat juotosreiät sen jokaiselle jalalle. Kortilla on ulkoinen 12 MHz:n kide, jonka vieressä on paikka SMB-liittimelle ulkoista kellosignaalia varten. Lisäksi kortilla ovat BNC-liittimet mikro-ohjaimen D/A- ja A/D-muuntimille. Kortilla on myös 32,768 MHz:n kide reaaliaikakelloa varten. [2, s. 10.]



Kuva 2. SAM3S-EK2-kokeilukortin lohkokaavio.

Kuvassa 2 on esitetty SAM3S-EK2-kortin lohkokaavio. Edellä mainittujen ominaisuuksien lisäksi kortilla on lukuisia muita mikro-ohjaimeen kytkettyjä osioita, joita voidaan käyttää apuna mikro-ohjaimen arvioinnissa ja ohjelmakehityksessä. Kortilla on mm.:

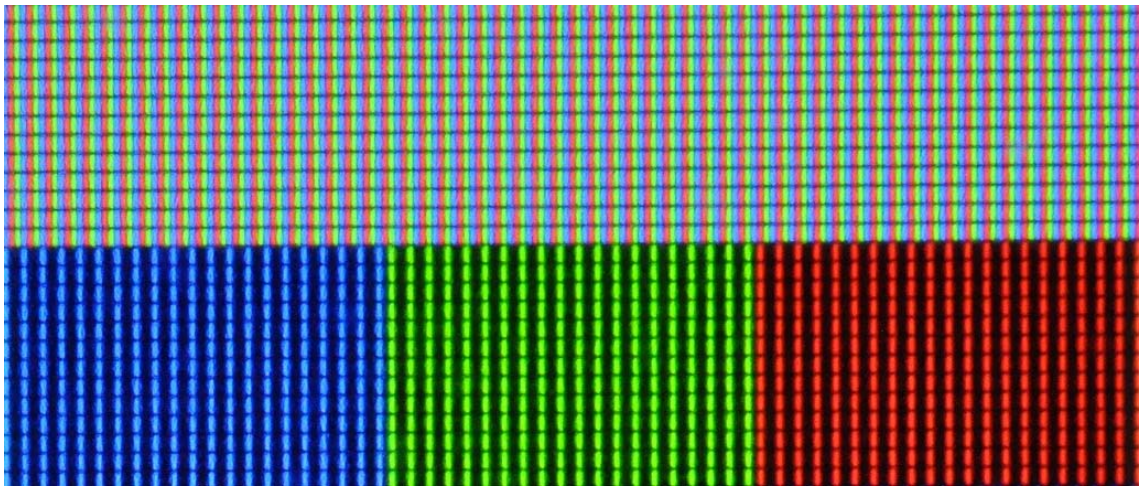
- ulkoinen NAND Flash -muistipiiri
- 2,8”:n resistiivinen TFT-kosketusnäyttö
- UART- ja USART-portit jännitetasonmuuttajilla
- mikrofoni
- 3,5 mm:n kuulokeliitin
- SD/MMC-muistikorttipaikka
- kolme painiketta
- kapasitiivisia hipaisupainikkeita
- USB-liitin
- JTAG/ICE-portti
- merkkivaloja
- mikro-ohjaimen A/D-muuntimeen kytketty potentiometri.

[2, s. 10.]

Kokeilukortin mukana toimitetaan AC/DC-verkkovirtalähde, USB-kaapeli sekä RS232-sarjakaapeli.

2.2 TFT-näyttö

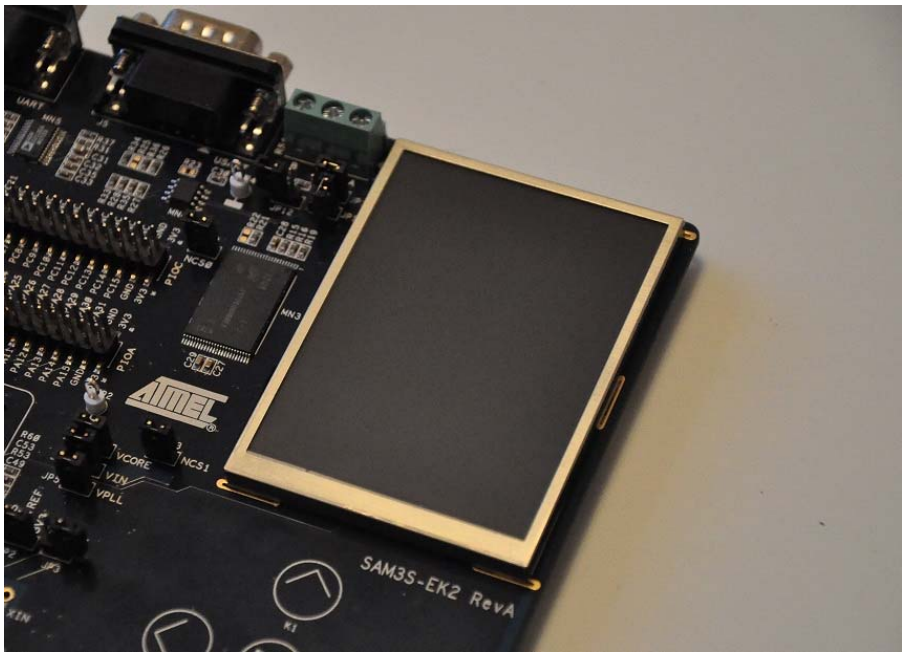
TFT-näyttö on eräänlainen nestekidenäyttö LCD (Liquid Crystal Display), jossa hyödynnetään ohutkalvotransistoritekniikkaa TFT (Thin-Film Transistor). TFT-näyttö on ns. aktiivimatriisinäyttö, jossa yksittäistä näytön pikseliä voidaan ohjata sille omistetun transistorin avulla. [3.]



Kuva 3. Lähikuva TFT-näytöstä.

TFT-värinäytössä jokainen pikseli on jaettu kolmeen osaan: punaiseen, vihreään ja siniseen suodattimeen. Näiden keskinäisillä suhteellisilla kirkkauksilla saadaan pikseli näkymään eri väreissä. [3.] Kuvassa 3 on lähikuva insinöörityössä käytetystä TFT-näytöstä. Kuvan yläosa näyttää kauempaa katsottuna valkoiselta, jonka alla on sininen, vihreä ja punainen alue. Kuvassa pikselien yksittäiset väriosiot ovat selkeästi näkyvissä, minkä ansiosta kuvasta saa melko hyvän käsityksen TFT-näytön toiminnasta.

TFT-näytöt tarvitsevat aina toimiakseen valkoisen taustavalon. Värisuodattimet ovat puikkomaisia kiteitä, joiden asentoa käännellään ohjaussignaalien mukaan. Kun kiteet ovat taustavaloon nähden kohtisuorassa, valo ei läpäise niitä. Kiteiden asento määrää, minkä väristä valoa näytöstä pääsee ulos eli minkä värinen on yksi pikseli. [4, s. 4.]



Kuva 4. SAM3S-EK2-kokeilukortin 2,8”:n TFT-näyttö.

Kuvassa 4 on SAM3S-EK-kokeilukortin näyttö, jonka mallinumero on FTM280C34D. Näytöstä ei ole saatavana datalehteä, mutta kokeilukortin käyttöohjeessa siitä on annettu jokin verran tietoa. Näytön koko on 2,8” ja resoluutio on 240 x 320 pikseliä (QVGA). Kyseessä on pystysuuntainen näyttö, jonka aktiivialueen leveys on 240 pikseliä ja korkeus 320 pikseliä. [2, s.14.] Näytön värisyvyys on enimmillään 18 bittiä, mikä tarkoittaa sitä, että jokaisen pikselin kullekin värille on käytössä 6 bittiä. 18 bitillä voidaan toistaa 262 144 eri värisävyä (2^{18}=262 144).

TFT-näytöissä on käytännössä aina näyttöön integroitu ohjainpiiri. Työssä käytetyssä näytössä ohjainpiirin malli on ILI9325. ILI9325 on Ilitekin integroitu piiri, joka on suunniteltu ohjaamaan 240 x 320 pikselin TFT-näyttöä. Piirillä on 172 800 tavua sisäänrakennettua RAM-muistia (Ramdom-Access Memory), johon näytön pikselien tila tallennetaan. Piiri ohjaa suoraan näytön ohutkalvotransistoreja ja tarjoaa helppokäyttöisiä liityntöjä näytön käyttäjälle. ILI9325:ssa on neljä erilaista liityntää (system interface):

- 8-, 9-, 16- tai 18-bittinen i80
- SPI (Serial Peripheral Interface)
- 6-, 16-, tai 18-bittinen RGB
- VSYNC.

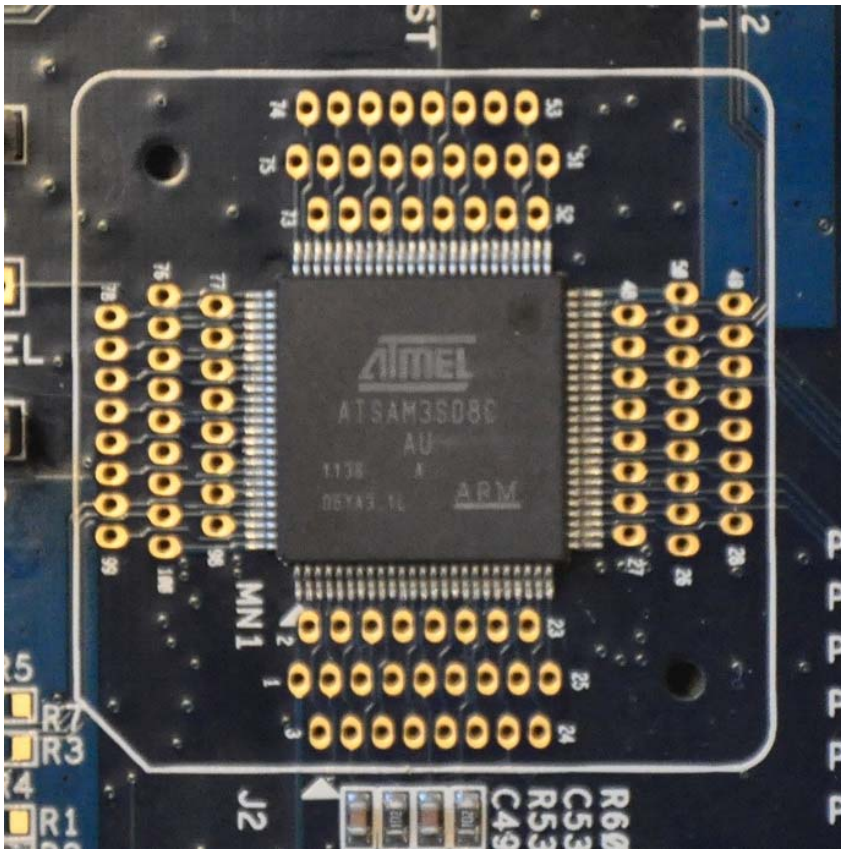
[5, s. 7.]

TFT-värinäytöt ovat nykyään erittäin yleisiä helpon saatavuuden ja edullisen hinnan vuoksi. Näyttöjen ohjaaminen on kuitenkin melko monimutkaista, ja ohjaustapa vaihtelee suuresti näytön ohjainpiiristä riippuen.

2.3 Mikro-ohjain

TFT-näytön ohjaamiseen tarvitaan joko mikroprosessori tai mikro-ohjain. Mikroprosessorit ovat yleensä nopeampia ja tehokkaampia kuin mikro-ohjaimet, mutta samalla ne ovat myös kalliimpia ja paljon monimutkaisempia käyttää.

Pieniä ja yksinkertaisia näyttöjä, kuten työssä käytettyä FTM280C34D:tä, voidaan ohjata myös mikro-ohjaimella. Mikro-ohjaimessa on samaan piiriin integroitu suorittimen lisäksi mm. ohjelma- ja datamuistit, ajastimia, oskillaattoreita sekä erilaisia oheispiirejä, kuten A/D- ja D/A-muuntimia. [6.] Lisäksi mikro-ohjaimissa on yleiskäyttöisiä ohjelmoitavia I/O-puskureita, jotka kykenevät antamaan ja ottamaan useita milliampeereita virtaa. Useimmissa mikro-ohjaimissa on sisäänrakennettu oskillaattoripiiri, jota voidaan käyttää prosessorin kellona. Tällöin mikro-ohjain ei periaatteessa tarvitse lainkaan ulkopuolisia komponentteja.



Kuva 5. ATSAM3SD8C-mikro-ohjain SAM3S-EK2-kokeilukortilla.

Kuvassa 5 on insinöörityössä käytetty Atmel ATSAM3SD8CA-CU -mikro-ohjain, jonka Amplition oli etukäteen valinnut. Valinta oli tehty komponentin ominaisuuksien, hinnan ja valmistajan tuen perusteella. Kyseessä on 32-bittinen ARM Cortex M3 -ytimeen perustuva mikro-ohjain, joka toimii nopeimmillaan 64 MHz:n taajuudella. Piirissä on 512 kB flash-ohjelmamuistia, 64 kB RAM-muistia sekä 16 kB ROM-muistia (Read-Only Memory), johon on jo tehtaalla tallennettu erilaisia Boot Loader -rutiineja. Mikro-ohjain on 100-jalkaisessa LQFP-kotelossa, ja siinä on 79 yleiskäyttöistä I/O-nastaa. [7, s. 1.]

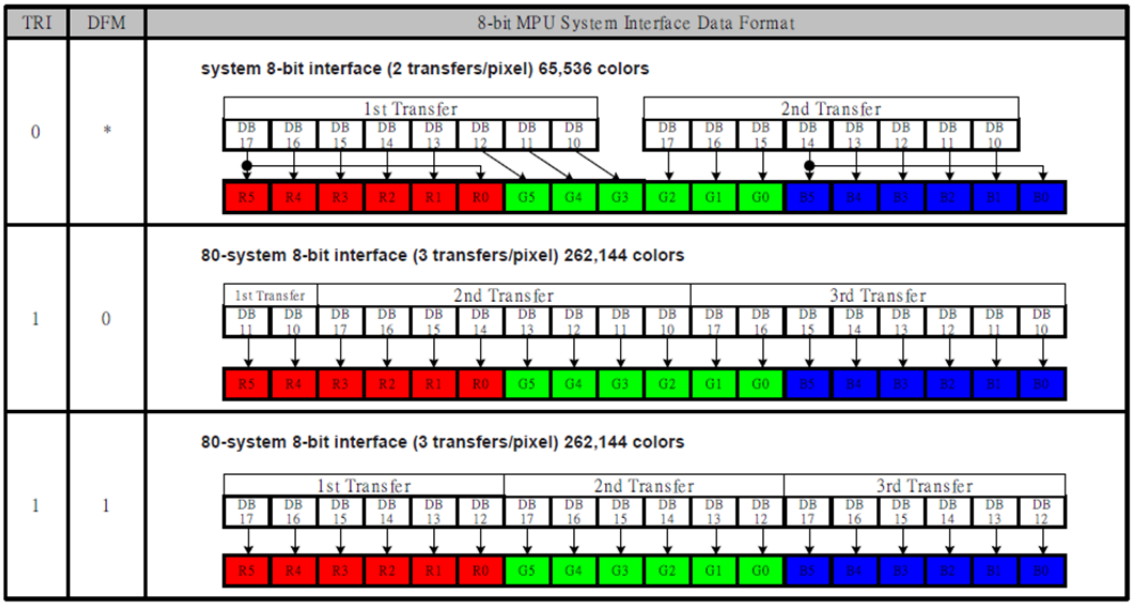
3 Näytön ohjaaminen

Väri-TFT-näytön ohjaaminen on monimutkainen prosessi. Näytössä ohjataan erikseen jokaista pikseliä, joita insinöörityössä käytetyssä, melko yksinkertaisessa näytössä, on 76 800 kappaletta (240 x 320 pikseliä). Jokaiselle pikselille voidaan kirjoittaa pikselin väri jopa 18-bittisenä, minkä vuoksi näytölle pitää siirtää todella suuria tietomääriä. Vaikeutta lisää se, että kuva pitää saada tulostettua mahdollisimman nopeasti näytölle, mikä vaatii suurta tiedonsiirtonopeutta.

Monimutkaisen grafiikan nopea päivittäminen näytölle vaatii sitä ohjaavalta prosessorilta suurta laskentatehoa, nopeaa tiedonsiirtoväylää ja riittävästi muistia. Näyttöä ohjaava ohjelma tulee vääjäämättä monimutkaiseksi, jotta näytölle voidaan tulostaa tekstiä ja grafiikkaa helposti ymmärrettävällä tavalla. Tarkoitus on, että ohjelman avulla näytölle voidaan tulostaa haluttua tekstiä tai grafiikkaa käytöltään yksinkertaisilla funktioilla, sen sijaan että pikseleitä ohjattaisiin yksi kerrallaan.

3.1 Näytön liittäminen mikro-ohjaimeen

TFT-näyttö pitää kytkeä sähköisesti mikro-ohjaimeen, jotta näyttöä voidaan käyttää. Tätä varten on olemassa useita erilaisia liityntätapoja. Insinöörityössä käytetyn näytön ohjainpiirissä on neljä erilaista liityntää: i80, SPI, RGB ja VSYNC [5, s. 7]. Näistä kokeilukortilla käytössä on 8-bittinen i80.



Kuva 6. 8- bittinen tiedonsiirto [5, s. 28].

i80 on rinnakkainen tiedonsiirtotapa, jossa tieto siirretään näytölle 8-, 9-, 16- tai 18-bittisenä. Tätä varten näytössä on enimmillään 18 bittiä leveä dataväylä, johon tieto saadaan syötettyä hyvinkin nopeasti. SAM3S-EK2-kokeilukortissa käytetään kuitenkin 8-bittistä i80-liityntää. Tällöin tieto syötetään näytölle kahdeksan bitin eli yhden tavun osissa. Tähän käytetään kahdeksan bitin dataväylää, joka käytännössä tarkoittaa kahdeksaa rinnakkaista vetoa piirikortilla mikro-ohjaimen lähtönastoista näytön

ottonastoihin. Tiedonsiirron leveys ei vaikuta kuitenkaan näytön värisyvyyteen, joka on enimmillään 18 bittiä. 8-bittiä leveällä väylällä tämä tarkoittaa sitä, että kutakin pikseliä varten joudutaan siirtämään kolme tavua tietoa. Näytöllä voidaan käyttää 18 bitin sijaan myös 16 bitin värisyvyyttä, jolloin värisävyjen määrä putoaa 262 144 kappaleesta 65 536 kappaleeseen. Tällöin riittää, että kutakin pikseliä kohden siirretään kaksi tavua tietoa. Kuvassa 6 esitetään, kuinka ILI9325-näytönohjainpiiri täyttää puuttuvat kaksi bittiä automaattisesti, jotta sen sisäinen 18 bitin rekisteri tulee täyteen. [5, s. 28.]

Dataväylän lisäksi näyttö tarvitsee toimiakseen CS-, RS-, WR- ja RD-linjat. CS on Chip Select -signaali, jolla kyseinen näyttöpiiri otetaan käyttöön. CS on 0-aktiivinen tarkoittaen sitä, että kun se on loogisessa nollassa eli kytketty maahan, piiri on aktiivinen ja käytettävissä. Kun linjalle kirjoitetaan looginen 1 eli mikro-ohjaimen käyttöjännite, piiriä ei ole valittu, eikä sitä siten voida käyttää. [5, s. 10.]

RS-signaalilla (Register Select) valitaan, kirjoitetaanko tietoa näytönohjaimen index-rekisteriin (IR) vai data-rekisteriin (WDR). Kirjoitus tapahtuu index-rekisteriin, kun RS-signaali on 0 ja data-rekisteriin, kun se on 1. WR-signaalilla (Write) kirjoitetaan tietoa näytölle, kun taas RD-signaalilla (Read) tietoa luetaan näytöltä. [5, s. 10.]

Taulukko 1. Näytön kytkentä mikro-ohjaimeen.

Signaalin nimi	Mikro-ohjain		Näyttö	
	Jalkanro	I/O-nasta	Liittimen kontakti	Näytön signaali
Data 0	25	PC0	9	DB10
Data 1	47	PC1	8	DB11
Data 2	43	PC2	7	DB12
Data 3	40	PC3	6	DB13
Data 4	37	PC4	5	DB14
Data 5	35	PC5	4	DB15
Data 6	32	PC6	3	DB16
Data 7	29	PC7	2	DB17
Chip Select	19	PC15	24	CS
Register Select	80	PC19	23	RS
Write	58	PC8	22	WR
Read	68	PC11	21	RD

Taulukossa 1 on esitetty SAM3S-EK2-kokeilukortilla näytölle menevät signaalit ja niiden kytkennät mikro-ohjaimessa ja näytössä.

Ohjaussignaalien lisäksi myös näytön taustavaloa pitää ohjata. SAM3S-EK2-kokeilukortilla tätä varten on erityinen ohjainpiiri AAT3155ITP-T1. Ohjainpiiri on kytketty mikro-ohjaimen PC13-nastaan.

3.2 Näytön päivitysnopeus

Se, kuinka nopeasti haluttu tieto saadaan siirtymään näytölle, vaikuttaa suoraan siihen, kuinka nopeasti näytölle piirtyy kuva. Jos tiedonsiirto on hidasta, ihmissilmä ehtii nähdä kuvan piirtymisen näytölle. Huonossa tapauksessa kuvan tulostuminen näytölle voidaan mitata jopa sekunneissa. Kuva piirretään näytölle pikseli kerrallaan aloittaen vasemmasta yläkulmasta. Hitaasti päivittyvällä näytöllä kuva vaikuttaa tulostuvan rivi kerrallaan ylhäältä alas.

Näytön päivitysnopeuteen vaikuttaa moni asia. Mitä suurempia näytön resoluutio ja värisyvyys ovat, sitä enemmän tietoa näytölle pitää siirtää. Insinöörityössä käytetyn näytön resoluutio on 240 x 320 pikseliä ja värisyvyys 18 bittiä. Tällöin yhden täyden ruudun vaatima tietomäärä voidaan laskea kaavalla:

$$240 \times 320 \times 18\ bit = 1\ 382\ 400\ bit$$
$$1\ 382\ 400\ bit = 172\ 800\ tavua$$

172 800 tavua on sama määrä kuin ILI9325-ohjaimessa on sisäistä RAM-muistia, ks. luku 2.2. Näytönohjain pystyy siis muistamaan yhden täyden ruudun jokaisen pikselin värin. Tämä on olennainen tieto näytönohjauksessa.

Näyttö olisi hyvä pystyä päivittämään alle sadassa millisekunnissa. Tämäkin on kokeellisesti todettu olevan vielä varsin hyvin havaittavissa ihmissilmällä. Sadan millisekunnin päivitysnopeuden vaatima tiedonsiirtonopeus voidaan laskea kaavalla:

$$1\ 382\ 400\ bit/0{,}1\ sek = 13\ 824\ 000\ bit/sek$$
$$13\ 824\ 000\ bit/sek = 13{,}824\ Mbit/sek$$

Tämä on jo melko nopea tiedonsiirtonopeus, vaikka näytön päivitysnopeus on kuitenkin vielä melko vaatimaton. Esimerkiksi liikkuva kuva vaatii päivitysnopeuden, joka on 25

kuvaa sekunnissa. 25 kuvaa sekunnissa tarkoittaa sitä, että yhden kuvan tulostus kestää vain 40 millisekuntia:

$$1\ sek/25 = 0{,}04\ sek$$
$$0{,}04\ sek = 40\ msek$$

Tällöin tiedonsiirtonopeuden pitäisi olla:

$$1\ 382\ 400\ bit\ /\ 0{,}04\ sek = 34\ 560\ 000\ bit/sek$$
$$34\ 560\ 000\ bit/sek = 34{,}56\ Mbit/sek$$

Ennen kuin tietoa voidaan siirtää, se pitää valmistella siirrettäväksi. Kuva pitää lukea muistista, teksti tulostaa tai haluttu kuvio laskea, jonka jälkeen se voidaan kirjoittaa. Tämä saattaa olla laitteistosta riippuen erittäin hidasta, mikä asettaa entistä tiukempia vaatimuksia tiedonsiirtonopeudelle.

Kun näyttöä ohjataan melko yksinkertaisella mikro-ohjaimella, sen päivitysnopeus jää usein kohtalaisen vaatimattomaksi. Käytännössä tämä tarkoittaa sitä, että näytöllä ei voida esittää liikkuvaa kuvaa. Näytön oheiselektroniikkaa ja ohjelmaa suunniteltaessa on kuitenkin mahdollista parantaa näytön päivitysnopeutta. Koska kokeilukortilla elektroniikka on jo valmiiksi suunniteltu, eikä se varsinaisesti liity tämän insinöörityön aihealueeseen, siihen ei paneuduta tämän syvällisemmin.

3.3 Värisyvyys

Kuten aikaisemmin jo todettiin luvussa 2.2, insinöörityössä käytetyn näytön värisyvyys on enimmillään 18 bittiä. Koska työssä käytetty mikro-ohjain on 32-bittinen, näytön yhden pikselin kaikki bitit voidaan käsitellä samaan aikaan. On myös olemassa 16- ja 8-bittisiä mikro-ohjaimia. Näillä kunkin pikselin bitit jouduttaisiin jakamaan kahteen tai kolmeen erään. Tämä hidastaisi mikro-ohjaimen tiedonkäsittelyä huomattavasti. Lisäksi työssä käytetty mikro-ohjaimen suurin kellotaajuus on 64 MHz, kun se pienemmillä 16- ja 8-bittisillä mikro-ohjaimilla on huomattavasti pienempi. Näytön ohjaamisessa lähtökohtana voidaankin pitää riittävän tehokkaan mikro-ohjaimen valitsemista.

Myös näytönohjausväylän leveys vaikuttaa olennaisesti tiedonsiirtonopeuteen. Insinöörityössä käytetyssä näytössä voidaan käyttää enimmillään 18 rinnakkaista datalinjaa. Kun nämä kaikki ovat käytössä, voidaan kaikki 18 bittiä siirtää yhdellä kertaa näytölle. Tämä on mahdollista vain, jos mikro-ohjain on 32-bittinen. 16- tai 8-bittisillä mikro-ohjaimilla ei voida käsitellä 18 bittiä rinnakkain. Vaikka SAM3S-EK2-kokeilukortilla on 32-bittinen mikro-ohjain, näytön ohjausväylän leveys on vain 8 bittiä. Toisin sanoen mikro-ohjaimen ja näytön välillä on vain kahdeksan rinnakkaista datalinjaa. Tässä tapauksessa jokaisen pikselin 18 bittiä joudutaan jakamaan kolmeen eri tavuun, jotka kirjoitetaan peräkkäin näytölle. Tiedonsiirtonopeus putoaa tämän vuoksi kolmasosaan suurimmasta mahdollisesta.

Myös mikro-ohjaimen I/O-nastoilla on merkitystä. Tavallinen yleiskäyttöinen GPIO-nasta (General Purpose I/O) on liian hidas näytön ohjaamiseen. Kun halutaan asettaa useita rinnakkaisia GPIO-nastoja samaan aikaan haluttuun tilaan, nämä todellisuudessa asettuvat vuoronperään yksi kerrallaan kyseiseen tilaan. Tähän menee vähintään yksi kellojakso jokaista nastaa kohti. Ohjelmasta ja mikro-ohjaimesta riippuen tähän voi mennä myös useita kellojaksoja nastaa kohden. 240 x 320 pikselin näytölle pitää siirtää enimmillään 1 382 400 bittiä kerralla. Jos jokaisen bitin siirtoon käytettäisiin useita kellojaksoja pelkästään I/O-nastojen asetteluun, päivitysnopeus olisi erittäin huono. [8.]

Siksi mikro-ohjaimessa pitää olla käytettävissä DMA (Direct Memory Access) I/O-nastoja. Nämä nastat ovat suoraan yhteydessä mikro-ohjaimen sisäiseen dataväylään, jolloin halutun tiedon siirtäminen I/O-nastoille käy erittäin nopeasti. [9.] SAM3S-EK2:ssa ovat juuri tällaiset I/O-nastat käytössä näytön dataväylällä.

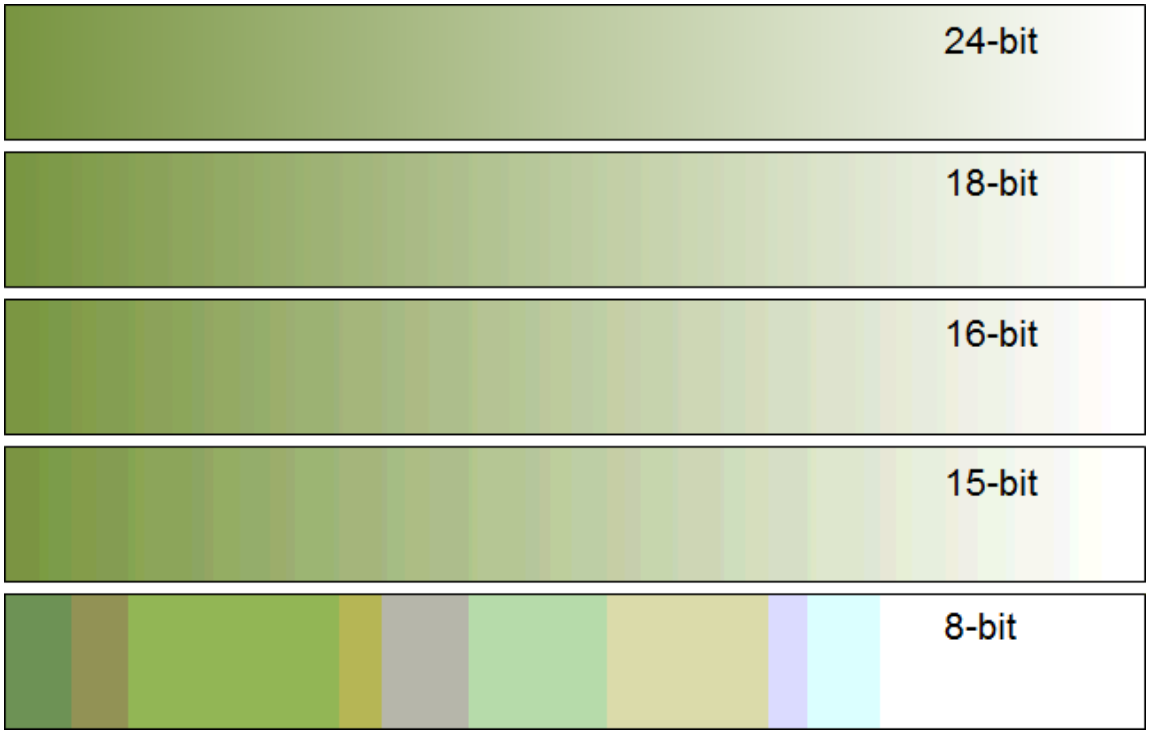
Toteutusta suunniteltaessa pitää päättää, kuinka monta väriä tarvitaan halutunlaisen grafiikan esittämiseen. Työssä käytetyn näytön enimmäisvärisyvyys on 18 bittiä. Käytetty värisyvyys voidaan valita myös 16- tai 8-bittiseksi. Käytettävissä olevien värien määrä voidaan laskea kaavalla:

$$lkm = 2^x$$

lkm on värien lukumäärä
x on värisyvyys bitteinä

Tällöin 18 bitillä voidaan toistaa 2^{18} = 262 144 eri väriä. 16-bittinen värisyvyys mahdollistaa 2^{16} = 65 536 eri väriä, ja 8-bittinen värisyvyys antaa vain 2^{8} = 256 eri väriä. Jos tarkoitus on näyttää vain yksinkertaisia kirjaimia ja numeroita sekä mahdollisesti joitain selkeitä symboleja, jopa kahdeksan bitin värisyvyys voi riittää. 16 bitillä voidaan piirtää jo jonkin verran pehmeitä muotoja, mutta esimerkiksi laajemmat liukuvärit eivät toistu niin hyvin kuin 18 bitillä.

Jos sekä mikro-ohjain että väylä kykenevät käsittelemään 18 bittiä rinnakkain, ei ole syytä käyttää pienempää värisyvyyttä. Kokeilukortilla on käytössä vain 8-bittinen dataväylä, joten värien määrästä tinkimällä voidaan nostaa näytön päivitysnopeutta. Kahdeksan bitin väylällä joudutaan 18 bittiä siirtämään kolmella tavulla. Jos värisyvyys rajoitetaan 16 bittiin, riittää kaksi tavua. Tällöin yhden pikselin tiedonsiirtoon menee kolmasosa vähemmän aikaa.



Kuva 7. Liukuvärejä eri värisyvyyksillä.

Kuvassa 7 esitetään eri värisyvyyksien vaikutus liukuväriin. Alkuperäinen, tietokoneella tehty 24-bittinen bittikarttakuva (bitmap-kuva) on muutettu 18-, 16-, 15- ja 8-bittisiksi. 15-bittinen värisyvyys on otettu mukaan siksi, että insinöörityössä käytetty 16 bitin värisyvyys ei ole kolmella jaollinen. TFT-näytöissä jokaisessa pikselissä on kolmen värisiä kiteitä, joita jokaista säädetään erikseen. Pikseliä ohjaava 16-bittinen sana ei

jakaudu tasan kaikkien värien kesken, vaan se saattaa jakautua esimerkiksi seuraavalla tavalla: punaiselle viisi bittiä, vihreälle kuusi bittiä ja siniselle viisi bittiä. 15 bittiä jakaantuu kolmen värin kesken tasaisesti, jolloin jokaista väriä säädetään viiden bitin tarkkuudella. Kuvasta 7 huomataan, että tällä yhden bitin erolla saattaa olla merkittävä vaikutus värintoistokykyyn.

3.4 Grafiikan tallentaminen muistiin

Suunnittelussa on kiinnitettävä huomiota siihen, mille näytölle piirrettävä grafiikka tallennetaan.. Yksi täysi näytöllinen grafiikkaa 18 bitin värisyvyydellä tarvitsee 1 382 400 bittiä eli 172,8 kB muistia. Täysi näytöllinen tarkoittaa käytännössä esimerkiksi yhtä taustakuvaa. Tyypillisesti näytöllä on taustakuvan lisäksi myös muita grafiikkaelementtejä, kuten ikoneita ja tekstiä. Tiedon noutaminen muistista on hidasta. Muistista riippuen tiedon lukeminen voi viedä huomattavasti enemmän aikaa kuin sen kirjoittaminen näytölle.

Nopein ja yksinkertaisin muisti on mikro-ohjaimen ohjelmamuisti. Kokeilukortilla käytetty Atmel ATSAM3SD8CA-CU -mikro-ohjain sisältää 512 kB ohjelmamuistia. Tähän muistiin ei siis mahdu edes kolmea näytön taustakuvaa. Lisäksi itse ohjelma ja grafiikkakirjasto tarvitsevat oman tilansa muistista. Täytyy myös huomioida, että 512 kB on melko paljon mikro-ohjaimen ohjelmamuistiksi. Hyvin tyypillinen muistimäärä voi olla esimerkiksi 32 kB, ja pienemmissä 8-bittisissä mikro-ohjaimissa on vain muutama kilotavu ohjelmamuistia.

Jos käytössä on isomuistinen mikro-ohjain, kuten tässä insinöörityössä oli, tarvittava grafiikka voidaan saada mahtumaan ohjelmamuistiin. Tällöin haluttu grafiikka pitää vain suunnitella etukäteen, jotta voidaan laskemalla arvioida muistin riittävyyttä.

Lähes yhtä nopea kuin mikro-ohjaimen sisäinen flash-ohjelmamuisti on ulkoinen DMA-kytketty muisti. Tämän nopeus tosin riippuu muistiväylän leveydestä. Leveä muistiväylä vie puolestaan paljon I/O-nastoja, joita mikro-ohjaimissa on aina rajallinen määrä. 32-bittiselle mikro-ohjaimelle optimaalinen väylän leveys olisi 32 bittiä, mutta se veisi 32 I/O-nastaa, mikä on useimmiten liikaa. Lisäksi näiden I/O-nastojen pitäisi olla DMA I/O -nastoja, joita on vielä rajoitetummin saatavilla. Esimerkiksi tässä työssä käytetyssä mikro-ohjaimessa on vain kahdeksan data-DMA-nastaa, jotka on jaettu piirilevyllä

olevan ulkoisen muistipiirin ja näytön välillä. Tämän takia näytölle siirrettävä 18-bittinen tieto pitää pilkkoa kolmeen osaan sekä muistista luettaessa että näytölle kirjoitettaessa.

On olemassa muitakin ulkoisia muistipiirejä. Erittäin yleinen ulkoinen muisti on SPI-liityntäinen sarjamuisti. Tämän kytkeminen mikro-ohjaimeen vie vain neljä I/O-nastaa. Tieto siirretään muistille sarjamuotoisena, jolloin tämän liitynnän pitää olla erittäin nopea. Mikro-ohjaimissa on usein SPI:tä varten omistettuja nastoja, jotka kykenevät erittäin nopeaan tiedonsiirtoon.

4 C-ohjelmointikieli

C-kieltä käytetään yleisesti sulautettujen järjestelmien ohjelmointiin. Aikaisemmin sulautettuja järjestelmiä ohjelmoitiin laiteläheisellä assembly-kielellä, mutta nykyisin C-kääntäjiä on saatavilla useisiin erilaisiin mikro-ohjaimiin. C-kielestä onkin tullut sulautettujen järjestelmien yleiskäyttökieli. [10, esipuhe, s.2.] Insinöörityön vaativuuden vuoksi C-kieli oli järkevin ja käytännössä ainoa mahdollinen valinta ohjelmointikieleksi.

C-ohjelmointikieli on hyvin siirrettävää. Koska C-kieli on standardoitua, se toimii eri prosessoriympäristöissä. Lisäksi sen käyttöön löytyy tarvittaessa paljon apua, kuten kirjallisuutta, työkaluja ja internet-keskusteluja. C-kääntäjiä löytyy sekä ilmaisia että maksullisia versioita useille eri mikro-ohjaimille. C-kieli on melko laiteläheistä, mutta lausemuotoisuutensa takia helposti opittavaa. [10, luku 3, s.3–4.]

Alun perin C-kielen epävirallinen standardi tunnettiin nimellä K&R. Nimi tulee alkuperäisen kuvauksen kirjoittaneiden Kerninghamin ja Ritchien mukaan. Vuonna 1989 valmistui ANSI C -standardi, jota täydennettiin 1999. [11.] Jos ohjelmassa käytetään vain standardin mukaista koodia, se on siirrettävissä eri ympäristöihin ja sen kääntäminen onnistuu kaikilla standardia noudattavilla C-kääntäjillä.

4.1 C-ohjelmointi sulautetuissa järjestelmissä

Sulautettujen järjestelmien ohjelmoinnissa C-kielestä käytetään vain pientä osaa. Yleensä ei ole käyttöjärjestelmää, jonka päälle ohjelma kirjoitetaan, joten kaikkia ominaisuuksia ei voida käyttää. Myös muistin koko ja muu laitteisto luovat omat rajansa

C-koodille. Sulautettujen järjestelmien C-kääntäjillä on myös mikro-ohjainkohtaisia erityisominaisuuksia, joita voidaan käyttää hyväksi ohjelmoinnissa. [10, luku 3, s.4–5.]

Ohjelman laiteläheisyyden vuoksi sulautettuja järjestelmiä on käytännössä mahdotonta ohjelmoida täysin standardin mukaisella C-kielellä. Sulautetussa järjestelmässä on pystyttävä ohjelmallisesti ohjaamaan sekä mikro-ohjainta että useita oheispiirejä. Oheispiirien ohjaukseen ei ole standardin mukaisia komentoja, vaan ne on tehtävä mikro-ohjainkohtaisesti usein jopa bittitasolla.

Insinöörityössä pyrittiin käyttämään mahdollisimman paljon standardin mukaista C-kieltä, jotta ohjelma olisi helposti siirrettävissä toiselle mikro-ohjaimelle. Työssä tehtiin ajurit alemmalle tasolle, joita ylemmän tason ohjelma käyttää. Näin ollen ajureita muokkaamalla on mahdollista saada ohjelma toimimaan myös toisella mikro-ohjaimella. Täytyy kuitenkin ottaa huomioon, että ohjelman siirto toiselle mikro-ohjaimelle vaatii ohjelmaan aina joitakin muutoksia sekä kohtalaisen selvitystyön toisen mikro-ohjaimen toiminnasta.

4.2 Ohjelmakirjastojen käyttäminen

C-kielen suppeuden vuoksi ohjelmaan liitetään aina ohjelmakirjastoja. Kirjastot koostuvat erilaisista määritelmistä ja funktioista, joilla laajennetaan C-kielen ominaisuuksia. Yleisesti käytettyjen kirjastojen, kuten stdio.h:n, lisäksi sulautettujen järjestelmien ohjelmoinnissa käytetään useita laiteläheisiä kirjastoja. Kirjastot myös helpottavat ohjelman siirrettävyyttä. [12.]

Sulautettuja järjestelmiä ohjelmoitaessa kannattaa käyttää mahdollisimman paljon valmiita kirjastoja. Usein mikropiirivalmistajat tarjoavat kirjastoja, joita voi käyttää ohjelman kirjoittamisen apuna. Kirjastojen tarkoitus on antaa ohjelmoijan käyttöön valmiita ohjelmallisia työkaluja ja ajureita, jotka helpottavat ja nopeuttavat komponentin käyttöönottoa. Mikro-ohjaimiin on saatavilla usein valmiita kirjastoja esimerkiksi ajastimien, A/D-muuntimien ja muiden oheispiirien käyttöön. Kirjastoja on saatavana myös mikro-ohjaimen ulkopuolisille mikropiireille, kuten erilaisille antureille ja ohjainpiireille.

4.3 Virheiden etsintä

Virheiden etsinnällä tarkoitetaan ohjelman vaiheittaista läpikäymistä ja virheiden paikallistamista. Ohjelman kirjoitus sisältää yleensä aina joitakin virheitä eli jokin osa ohjelmasta ei toimi oletetulla tavalla. Ilman virheiden etsintätyökalua tällaisen virheen paikallistaminen voisi olla suuri työ. Usein ohjelmoija ei huomaa omia virheitään helposti tai ohjelma saattaa olla niin monimutkainen, että virheen etsimiseen menee kauan aikaa.

Virheiden etsintätyökaluilla ohjelmaa pystytään ajamaan esim. funktio kerrallaan, jolloin suorituksessa tapahtuvat virheet on helppo paikallistaa. Ohjelmaan voidaan asettaa myös tiettyjä pysäytyskohtia, joihin ohjelma pysähtyy. Näin voidaan esimerkiksi suorittaa jokin pidempi for-silmukka loppuun ja jatkaa siitä eteenpäin funktio kerrallaan.

5 Kehitysympäristö

Kehitysympäristö IDE (Integrated Development Environment) on kokoelma työkaluja, jotka helpottavat ohjelmointia. Kehitysympäristöön voi kuulua esimerkiksi kääntäjä, koostaja (assembler), linkitin ja virheiden etsintätyökalu. Työkalut on koottu yleensä yhdeksi ohjelmaksi, jossa on graafinen käyttöliittymä. Kehitysympäristön tarkoitus on helpottaa sulautetun järjestelmän ohjelmointityötä tuomalla monipuolisesti työkaluja ohjelmoijan käyttöön samaan käyttöliittymään. Myös tekstieditorilla olisi mahdollista muokata C-ohjelmaa, mutta hyvä kehitysympäristö tekee koodin muokkauksesta paljon helpompaa. [13, s. 9.]

5.1 Kehitysympäristön valinta

Insinöörityössä oli alun perin tarkoitus käyttää Atmel Studio 6 -ohjelmaa kehitysympäristönä. Kyseinen ohjelma on ilmeinen vaihtoehto, koska projektissa käytetään Atmelin piiriä. Atmel Studio 6 -ohjelmaan on saatavilla paljon esimerkkiohjelmia ja valmiita kirjastoja. Internetissä löytyy myös paljon keskusteluja, joista olisi ollut apua kyseisen ohjelman käytössä. Ohjelma olisi ollut myös varmasti yhteensopiva Atmelin mikro-ohjaimien kanssa.

Ennen kuin kehitysympäristö valittiin lopullisesti, aiheesta käytiin keskusteluja projektipäällikkö Zhongliang Hun kanssa. Zhongliang Hulla on tältä alalta erittäin vahva kokemus. Hän on työskennellyt vuosia sulautettujen järjestelmien ohjelmoijana Espotel Oy:ssä, joka on Suomen suurin sulautettujen järjestelmien suunnittelutoimisto. Keskusteluissa kävi ilmi, että Studio 6 on erittäin hankalakäyttöinen ja epäintuitiivinen. Zhongliang on itse joskus kokeillut kyseistä ohjelmaa, eikä hän pitänyt sen käyttöliittymästä. Hän suositteli kehitysympäristöksi IAR -ohjelmaa. [8.]

IAR on ammattipiireissä erittäin paljon käytetty kehitysympäristö. Ohjelmasta löytyvät omat versiot usealle eri prosessoriperheelle. Koska tässä työssä käytettiin ARM Cortex M3 -ytimellä varustettua mikro-ohjainta, sopiva IAR-versio oli IAR Embedded Workbench for ARM. Varsinaisen lisenssin hinta on 3800 €, mutta valmistajan kotisivuilta saa ladata ilmaiseksi kokeiluversion. Kokeiluversioita on kaksi erilaista. Toinen on 30 päivän aikarajoitettu, mutta kokorajoittamaton versio, ja toinen on 30 kB:n kokorajoitettu, mutta aikarajoittamaton versio. [14.]

Valittavan kehitysympäristön tuli olla laajalti alalla käytetty ja monipuolinen. Lisäksi sen tuli tukea useita eri mikro-ohjaimia, jotta sitä voidaan käyttää myös tulevaisuuden projekteissa. Koska insinöörityöhön oli valittu Atmelin piiri, vaatimuksena kehitysympäristölle oli myös piirivalmistajan tuki. IAR Embedded Workbench for ARM tarjoaa kaikki tarvittavat työkalut sulautetun järjestelmän ohjelman kirjoittamiseen, testaamiseen ja virheiden etsintään. Ohjelmaan kuuluu kääntäjä, linkitin, koostaja ja monipuolinen virheiden etsintätyökalu. Tämän lisäksi Atmelilta löytyy valmiita kirjastoja ja esimerkkiohjelmia, jotka on tehty erityisesti IARille. [15.]

Kaikki seikat huomioon ottaen insinöörityön kehitysympäristöksi valikoitui IAR Embedded Workbench for ARM:n kokorajoitettu kokeiluversio. 30 päivän aikarajoitettu versio ei tullut kysymykseen, koska työn tekemiseen kuluisi todennäköisesti pidempi aika. Toisaalta oli oletettavaa, että ohjelman koko ei ylittäisi tuota 30 kB rajaa.

5.2 JTAG-emulaattori

Kehitysympäristöön valittiin myös JTAG-emulaattori (Joint Test Action Group). JTAG on yleisnimitys IEEE 1149.1 -standardille, joka määrittelee ohjelmiston testaukseen käytettävän JTAG-portin [16]. Alun perin se kehitettiin piirikorttien testaukseen, johon

sitä käytetään vieläkin. Tänä päivänä JTAG on eniten käytetty mikro-ohjaimien ja mikroprosessorien virheiden etsintään tarkoitettu liityntä, ja lähes kaikki nykyaikaiset mikroprosessorit tukevat JTAGia. Sulautettujen järjestelmien ohjelmoinnissa JTAGista on paljon hyötyä. Se mahdollistaa muun muassa ohjelman ajon askel kerrallaan tai pysäytyspisteiden asettamisen haluttuihin kohtiin ohjelmaa. [17.]

Kuva 8. SAM-ICE JTAG emulaattori [18].

Atmel myy SAM-ICE-nimistä JTAG-emulaattoria (kuva 8), joka on suunniteltu Atmelin SAM3-, SAM7- ja SAM9-mikro-ohjaimille [19]. Koska työssä käytetään SAM3-mikro-ohjainta, SAM-ICE oli luonteva valinta JTAG-emulaattoriksi.

5.3 Grafiikkakirjasto

Grafiikkakirjasto on kokoelma ohjelmia, jotka on suunniteltu avustamaan grafiikan tulostamista näytölle sulautetuissa järjestelmissä. Yleensä kirjasto sisältää optimoituja funktioita, joiden avulla saadaan helposti piirrettyä grafiikkaa näytölle. Grafiikkakirjaston funktiot on mahdollista suorittaa ohjelmallisesti ajettuna suoraan CPU:lla (Central Processing Unit), ja näin yleensä toimitaankin sulautettujen järjestelmien kohdalla. Tietokoneissa voidaan käyttää GPU:ta (Graphics Processing Unit) nopeuttamassa funktioiden suorittamista. Grafiikkakirjaston funktioita käyttämällä näytölle saadaan tulostetuksi kuvia, kuvioita ja tekstiä. Ohjelmoijan aikaa säästyy, kun ei tarvitse itse luoda ja optimoida grafiikkafunktioita. [20.]

Sulautetuille järjestelmille tarkoitettuja grafiikkakirjastoja on rajallinen määrä. Tähän projektiin tarvittiin kirjasto, joka tukee projektissa käytetyn näytön ILI9325-ohjainpiiriä.

Ramtex tarjoaa grafiikkakirjastoja ja ajureita pienille ja keskisuurille TFT-LCD-näytöille. Yritykseltä löytyy kirjastoja usealle eri näytönohjaimelle ja tässä projektissa käytetty ILI9325-näytönohjain oli myös tuettujen listalla. Projektissa päädyttiin käyttämään Ramtex Graphic Dot Matrix Color LCD Driver -grafiikkakirjastoa.

6 Työn valmistelu

6.1 Käytössä olevat työkalut ja laitteet

Insinöörityötä varten käytössä oli tietokone Windows 7 -käyttöjärjestelmällä, SAM3S-EK2-kokeilukortti ja JTAG-emulaattori (Atmel SAM-ICE). Kehitysympäristönä käytettiin IAR Embedded Workbench -ohjelmaa ja grafiikan tulostamiseen näytölle Ramtex Graphic Dot Matrix Color LCD Driver -grafiikkakirjastoa. Näytölle tulostettavat kuvat luotiin Microsoft Paint -ohjelmalla ja käännettiin oikeaan muotoon Color LCD Icon Editor -ohjelmalla.

Kuva 9. FTDI TTL-232R-3V3 -sarjaliikennemuunnin [21].

Lisäksi työssä käytettiin FTDI TTL-232R-3V3 -sarjaliikennemuunninta (kuva 9), jolla kokeilukortti liitettiin tietokoneen USB-porttiin. Tämän avulla pystyttiin luomaan terminaaliyhteys tietokoneen ja mikro-ohjaimen välille. Yhteyden luomiseen käytettiin PuTTY-terminaaliohjelmaa.

6.2 Esiasennettuun kokeiluohjelmaan tutustuminen

Kokeilukortille on esiasennettu ohjelma, joka käynnistyy automaattisesti, kun kokeilukorttiin kytketään virta. Kokeiluohjelmassa esitellään kortin eri ominaisuuksia, joita pääsee testaamaan painamalla kosketusnäytön eri ikoneita.

Kuva 10. Kokeiluohjelman päävalikko.

Kuvassa 10 on kokeiluohjelman näytön päänäkymä. Näytöllä vasemmassa yläkulmassa oleva ääni-ikoni ei ole tällä kokeilukortilla käytössä, mikä kerrotaan myös Atmelin julkaisutiedotteesta [22]. Seuraavasta ikonista pääsee katsomaan näytön ominaisuuksia, värintoistoa ja kuvioiden piirtonopeutta. Mikropiiri-kuvakkeesta aukeaa graafinen esitys mikro-ohjaimen ominaisuuksista. Esityksessä on 14 sivua, ja ne vaihtuvat automaattisesti. Työkalu-ikonista pääsee määrittelemään kellon ajan, päivämäärän sekä näytön kirkkauden. Täällä voi tehdä myös kosketusnäytön kalibroinnin. Qtouch-kuvakkeesta pääsee testaamaan kokeilukortilla olevia

hipaisukytkimiä. Hipaisukytkimillä voi myös ohjata valintaa päävalikossa. Akku-kuvakkeesta pääsee kokeilemaan SAM3S-EK2-kokeilukortin virransäästö-ominaisuuksia. Vaihtoehtoina ovat backup-, wait- ja sleep-toiminnot. Viimeisestä info-kuvakkeesta näkee kokeiluohjelman version.

Kokeiluohjelma käsittelee monipuolisesti kokeilukortin eri ominaisuuksia. TFT-näytön pienen resoluution vuoksi joistakin teksteistä on vaikea saada selvää. On myös mahdollista, että teksti on liian pientä ja huonolaatuista ja sen takia epäselvää. Kokeiluohjelmaa tutkimalla on havaittavissa myös kosketusnäytön epätarkkuutta ja se, ettei näytön kalibrointi aina onnistu, vaikka yrittäisi tehdä sen mahdollisimman tarkasti. Huomattavaa on myös se, että näyttöä ei voi katsoa kovin sivusta, koska silloin osa väreistä häviää kokonaan. Ohjelma on kuitenkin hyvin havainnollinen, ja siinä tulevat esille kokeilukortin ominaisuudet ja mahdollisuudet.

Kokeilukortin voi myös kytkeä mukana tulleella USB-kaapelilla tietokoneen USB-porttiin. Windows tunnistaa kokeilukortin automaattisesti uudeksi asemaksi, ja kokeilukortilla olevia tiedostoja pääsee selaamaan. Tiedostot voi myös varmuuskopioida tietokoneelle. Kokeilukortilla ovat lisäksi DevStart-nimiset html-muotoiset esittelysivut, jotka aukeavat tietokoneen selaimeen.

Kokeilukortin esittelyohjelma

Tietokoneen selaimeen avautuvissa DevStart-sivuissa käydään läpi, miten kokeilukortin muisti tyhjennetään ja kuinka mikro-ohjaimelle saa ohjelmoitua oman ohjelman. Myös kokeiluohjelman palauttaminen kokeilukortille on käsitelty samaisessa esittelyohjelmassa. Atmelin sivuilta löytyy vielä laajempi esittelyohjelmapaketti, jossa on mm. toinen kokeiluohjelma kortin ominaisuuksien testaamiseen. Tämän KitsFiles-nimisen esittelyohjelmapaketin läpikäyminen helpottaa kokeilukortin käyttöön tutustumisessa.

Muistin tyhjentäminen

Jotta kokeilukortille voidaan ohjelmoida oma ohjelma, täytyy mikro-ohjaimen muisti ensin tyhjentää. Kortilla olevan oikosulkupalan numero 3 (JP3) sulkeminen tyhjentää ja alustaa (initialisoi) mikro-ohjaimen sisäiset muistit, kun kokeilukorttiin kytketään virta. DevStart-esittelyohjelmassa on lueteltu kaikki kokeilukortilla olevat oikosulkupalat ja

niiden oletusasennot. Oikosulkupalan voi avata virran ollessa päällä, jolloin mikro-ohjaimen ohjelmointi on mahdollista.

Ohjelmointitavat

Kokeilukortilla olevan mikro-ohjaimen voi ohjelmoida kahdella tavalla. Ensimmäinen tapa on USB-liittimen kautta tapahtuva ohjelmointi. Tämä vaatii, että tietokoneelle asennetaan SAM-BA-niminen ohjelma ja oikeanlaiset USB-ajurit.

SAM-BA on Atmelin ohjelma, joka tarjoaa työkalut SAM3-, SAM7- ja SAM9-mikro-ohjainten ohjelmointiin [23]. Mikro-ohjaimessa on sisäänrakennettuna SAM-BA Boot Loader, joka mahdollistaa flash-muistin ohjelmoimisen SAM-BA-ohjelman käyttöliittymän avulla.

Toinen ohjelmointitapa on käyttää JTAG-emulaattoria, joka kytketään tietokoneen USB-porttiin ja kokeilukortin JTAG/ICE-porttiin. Insinöörityössä käytettävää SAM-ICE-JTAG-emulaattorin käyttöä varten tarvitaan Windowsille ajurit, jotka voidaan ladata internetistä SEGGERin sivuilta [24].

6.3 Netistä saatavat esimerkkiohjelmat

Atmelin kotisivuilta on saatavilla suuri määrä erilaisia esimerkkiohjelmia SAM3S-EK2-kokeilukortille. Esimerkkiohjelmat ovat ladattavissa eri C-kääntäjille, myös IARille. Insinöörityössä käytettiin IARin versiota 6.4. Tälle versiolle ei suoraan löydy esimerkkiohjelmapakettia Atmelin sivuilta, mutta version 6.3 ohjelmapaketti on yhteensopiva myös tämän uudemman version kanssa. Insinöörityötä varten ladattiin Atmelin sivuilta SAM3S8/SD8 IAR EWARM 6.30 -esimerkkiohjelmapaketti.

Ohjelmapaketti sisältää esimerkkiohjelmien lisäksi laajat kirjastot. Kirjastoja voi käyttää myös tukena oman ohjelman luomisessa. Ohjelmapaketissa olivat perustoimintoesimerkkiohjelmien lisäksi esimerkkiohjelmat mm. tiedostojärjestelmästä (filesystem example), grafiikasta (graphics example) ja hipaisunäppäimistä (Qtouch example). Näytön ohjaamista parhaiten havainnollistaa grafiikka-esimerkkiohjelma.

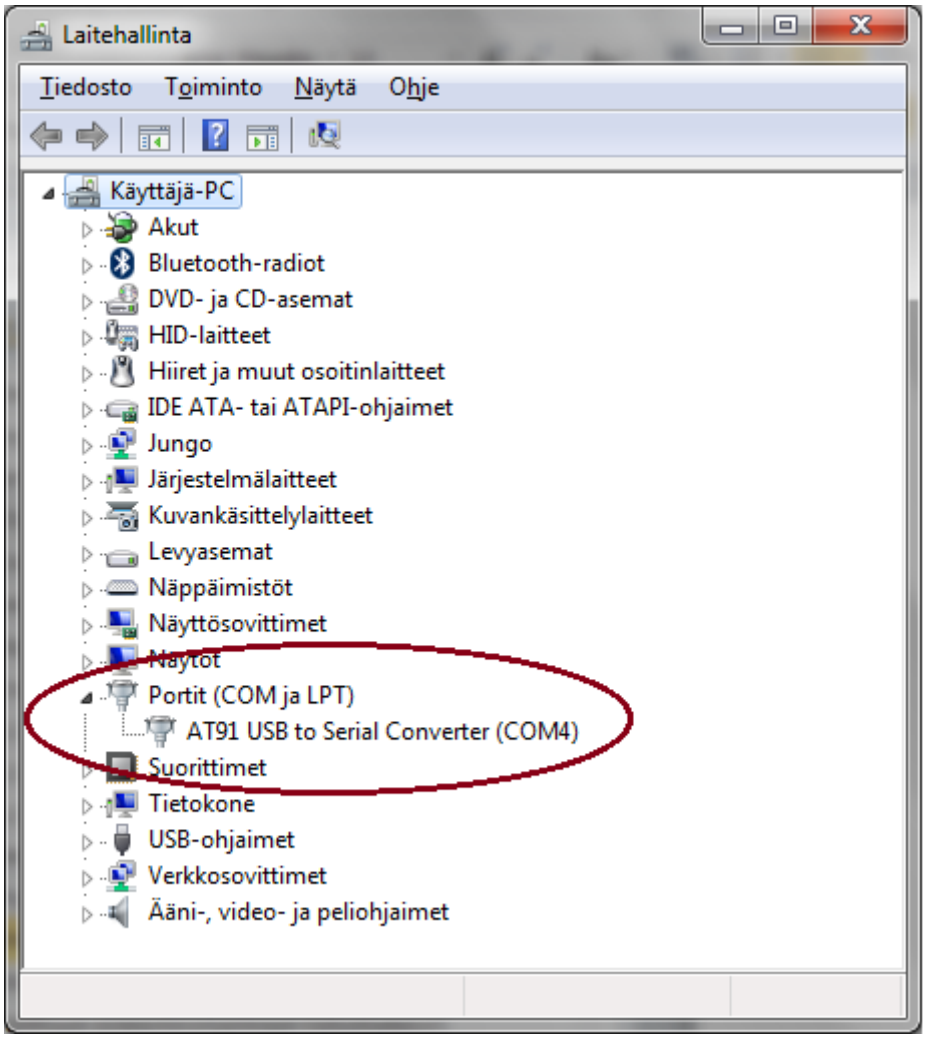
6.4 Projektin käyttöönotto IAR-kehitysympäristössä

Grafiikka-esimerkkiohjelman avaaminen IAR-kehitysympäristössä on hyvin suoraviivaista. Esimerkkiohjelman kansiosta löytyy .eww-päätteinen tiedosto, joka aukeaa suoraan IARiin. Kun tiedosto avataan IARiin, aukeaa samaan työtilaan kolme eri projektia. Kaksi näistä sisältää tarvittavat kirjastot ja kolmas itse suoritettavan ohjelman. Grafiikka-esimerkkiohjelma kääntyi IARissa ilman virheitä, ja tuloksena saatiin kaksi binääritiedostoa, sram.bin ja flash.bin.

6.5 Mikro-ohjaimen ohjelmoiminen USB-portin kautta

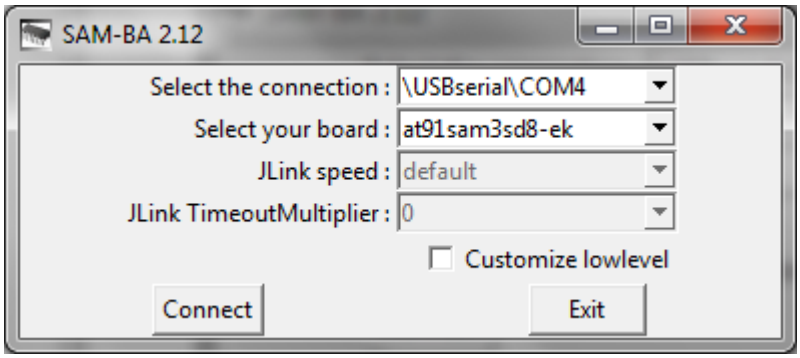
Kuten luvussa 6.2 todettiin, mikro-ohjaimen voi ohjelmoida kahdella tavalla. USB-portin kautta ohjelmoitaessa käytetään SAM-BA-ohjelmaa. SAM-BA voidaan ladata Atmelin nettisivuilta ja sen asentaminen on yksinkertaista.

SAM-BA tarvitsee oikeat USB-ajurit toimiakseen. Ohjelma käyttää USB-porttia sarjamuodossa. Windows ei osaa automaattisesti asentaa oikeita ajureita, kun USB-johto kytketään kokeilukortin ja tietokoneen välille. Oikean ajurin asentaminen ja asetusten saaminen kohdalleen oli työlästä. Atmel tarjoaa ohjeita, miten Windows 7 -käyttöjärjestelmään asennetaan oikeat ajurit. Ohjeet ovat tosin puutteelliset eikä USB-ajureiden asennus onnistu niiden avulla. Atmelin keskustelufoorumilta löytyy nimimerkin Haidosu kirjoittama ohje, jota seuraamalla USB-ajurit saadaan asennettua oikein.



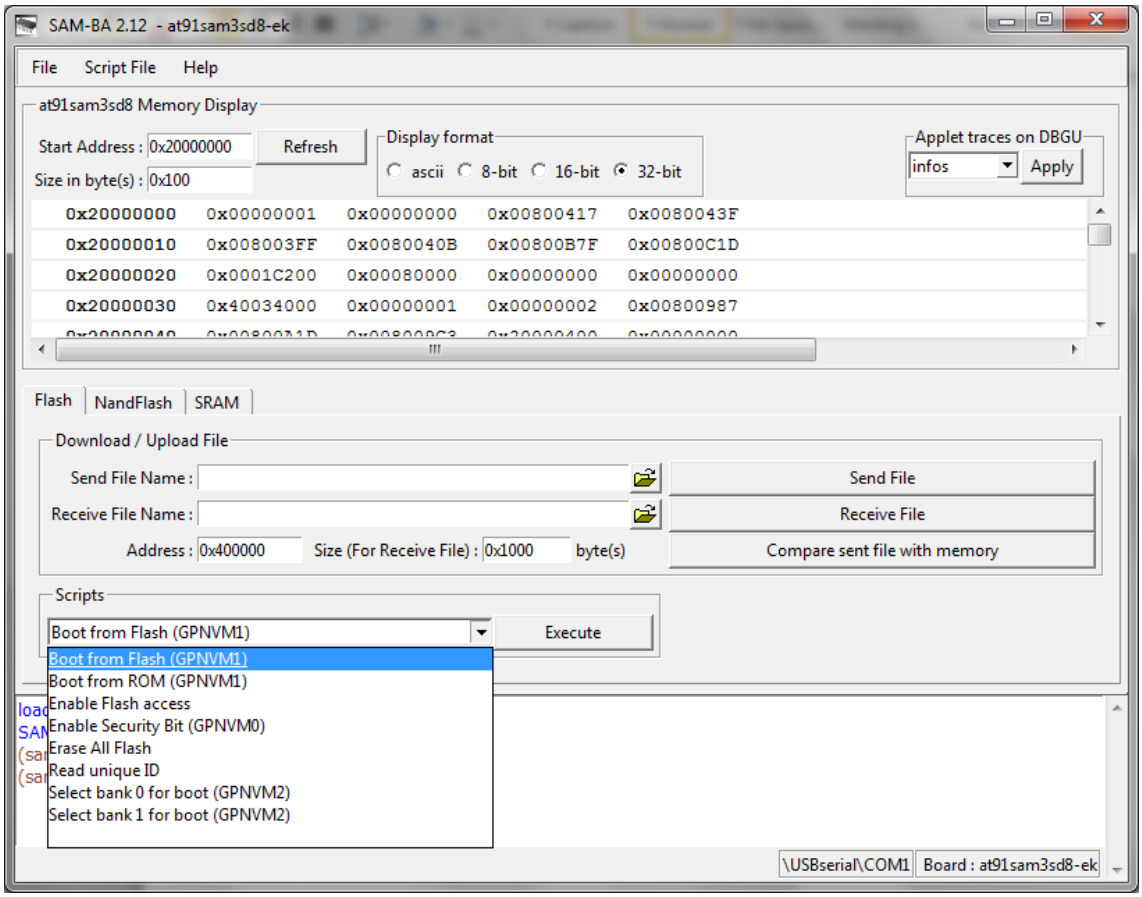
Kuva 11. Oikein asentuneet ajurit Windowsin laitehallinnassa.

Kuvassa 11 on laitehallinnan näkymä USB-johto kytkettynä ja laiteajurit oikein asennettuina. Jos käytössä ovat väärät ajurit, näkyy USB-yhteys USB-ohjaimien alla eikä Porttien (COM ja LPT) alla. Tässä tapauksessa yhteys käyttää COM4-porttia, mutta numero voi olla myös jokin toinen.



Kuva 12. SAM-BA-valikko.

SAM-BAn ohjelmointiympäristö käynnistetään SAM-BAn valikkoikkunasta (kuva 12) valitsemalla, mitä yhteyttä halutaan käyttää ja mihin kokeilualustalle yhteys muodostetaan. Jos USB-ajurit ovat asentuneet oikein, löytyy oikea yhteysvaihtoehto alasvetovalikosta. Samoin kokeilukortin malli löytyy alasvetovalikosta. Yhteyden muodostamisen jälkeen aukeaa ohjelman varsinainen käyttöliittymä.



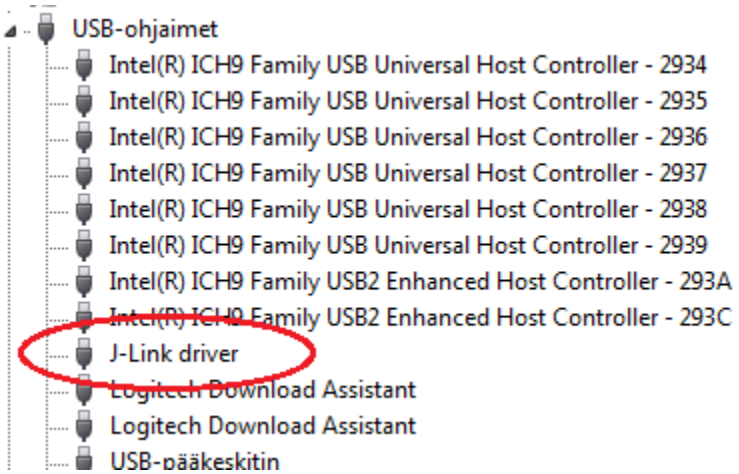
Kuva 13. SAM-BA-käyttöliittymä.

Mikro-ohjain ohjelmoidaan noudattamalla Atmelin DevStart-käyttöohjeita.. Ensin flash-muisti asetetaan kirjoitettavaan tilaan. Tämän jälkeen haetaan luvussa 6.4 mainittu flash.bin-tiedosto ja tallennetaan se mikro-ohjaimen muistiin. Tallennuksen jälkeen flash-muisti asetetaan takaisin boot from flash -tilaan (kuva 13), jolloin uudelleen käynnistettäessä ohjelma suoritetaan suoraan muistista.

Kokeilukortilla olevan mikro-ohjain on ohjelmoitavissa USB-portin kautta, mutta tätä ohjelmointitapaa käytettäessä virheiden etsintätyökalut eivät ole käytettävissä. Sulautettujen järjestelmien ohjelmoinnissa on erittäin tärkeää, että ohjelmasta päästään etsimään virheitä, joten yksinään USB-ohjelmointi ei ole toimiva ratkaisu.

6.6 Mikro-ohjaimen ohjelmoiminen JTAG emulaattorilla

Toinen tapa ohjelmoida kokeilukortilla oleva mikro-ohjain on käyttää JTAG-emulaattoria. JTAG-emulaattorin käyttö mahdollistaa myös ohjelman virheiden etsinnän. Insinöörityössä käytettiin Atmelin SAM-ICE-emulaattoria, joka on luonteva valinta käytettäväksi Atmelin mikro-ohjaimien kanssa. SAM-ICE on SEGGERin Atmelille valmistama JTAG-emulaattori. SEGGER valmistaa myös omalla nimellään JTAG-emulaattoreita, joita olisi yhtä hyvin voinut käyttää tässä projektissa.



Kuva 14. Windowsin laitehallintaikkuna SAM-ICE kytkettynä.

SAM-ICE-emulaattorin käyttöönotto vaatii oikeiden ajurien asentamista. Ajurit on ladattavissa SEGGERin internet-sivuilta. Ajureiden asentamisen jälkeen emulaattoria voidaan käyttää suoraan IAR-kehitysympäristöstä. Ajureiden asentuminen kannattaa

varmistaa Windowsin laitehallinnasta (kuva 14). Emulaattori kytketään tietokoneen USB-väylään, josta se saa myös tarvitsemansa virran.

SAM-ICE-emulaattorin avulla ohjelman saa tallennettua mikro-ohjaimen flash-muistiin ja ohjelmasta pystyy myös paikallistamaan virheitä virheiden etsintätyökalujen avulla. Työkalut ovat verrattain helppokäyttöisiä ja intuitiivisia. Koska ohjelmaa kehitettäessä syntyy aina virheitä, JTAG-emulaattorin käyttö on oikea ratkaisu sulautettujen järjestelmien ohjelmointiin ja ohjelman virheiden etsintään.

6.7 Valittuun esimerkkiohjelmaan tutustuminen

Tutkittavaksi valittu esimerkkiohjelma (graphics example) on nimeltään smc_lcd. Esimerkkiohjelma on kaksiosainen. Ensimmäisessä osassa konfiguroidaan staattinen muistinohjain SMC (Static Memory Controller) ohjaamaan kokeilukortilla olevaa TFT-näyttöä. Ohjelman toisessa osassa demonstroidaan SPI-liitynnän käyttöä kosketusnäytön ajureiden ohjaamiseen. Insinöörityön tavoitteen kannalta ensimmäinen osa on mielenkiintoisempi.

Kuva 15. TFT-näytön kuva esimerkkiohjelmasta.

Ohjelma tulostaa TFT-näytölle värikuvion, tekstiä, ympyröitä, neliöitä ja viivoja (kuva 15). Värikuvio on laskennallisesti piirretty, eli se ei ole laitteen muistiin tallennettu

graafinen kuva. Esimerkkiohjelmaa muokkaamalla näytölle saadaan erilaisia ja erikokoisia kuvioita, tekstiä ja värejä. Myös taustaväriä voidaan muuttaa.

Ohjelma on kirjoitettu hyvän ohjelmointitavan mukaisesti, ja sitä on kommentoitu selkeästi toiminnallisuuden ymmärtämisen helpottamiseksi. Ohjelma on myös kerroksellinen eli ylimmällä tasolla kutsutaan funktioita, jotka on määritelty alemmilla tasoilla. Tämä tekee ohjelman tulkitsemisesta verrattain helppoa. Aivan alimman tason määrittelyt, kuten esimerkiksi mikro-ohjaimen I/O-nastojen määrittelyt, tehdään Atmelin luomissa kirjastoissa.

7 Työn toteutus

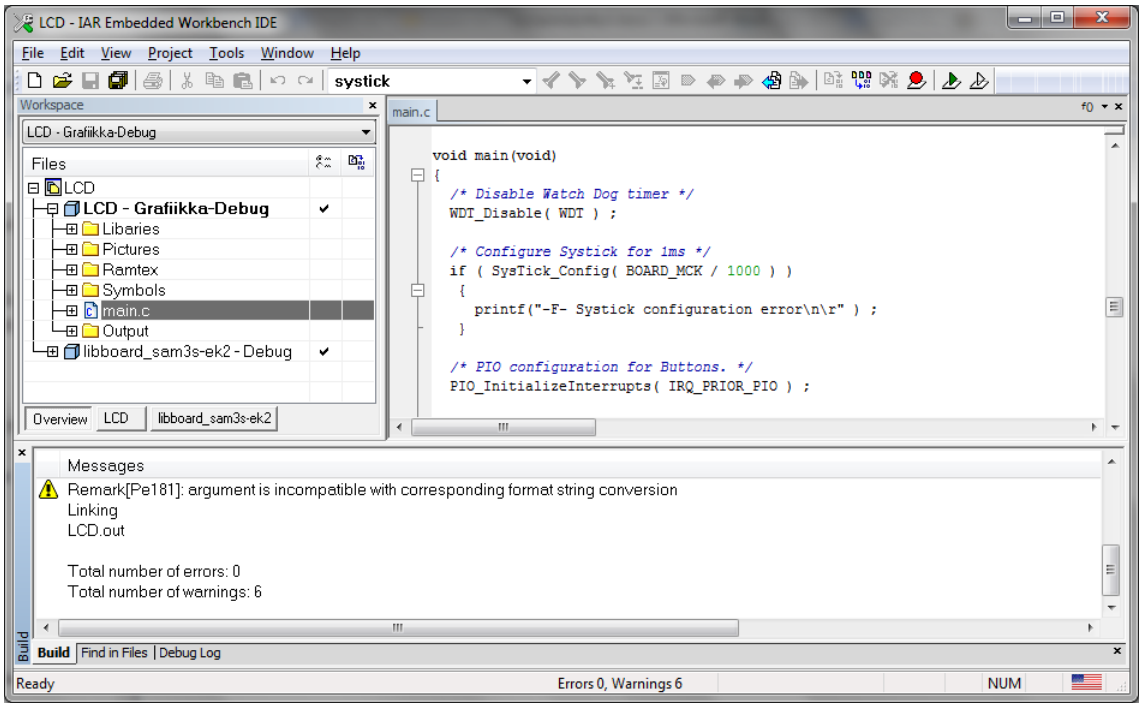
Kuten luvussa 5.1 todetaan, insinöörityöhön valittiin kehitysympäristöksi IAR Embedded Workbench for ARM -ohjelma. Ohjelmasta ladattiin internetistä ilmainen 30 kB:n kokorajoitettu kokeiluversio. Tulevia projekteja varten Amplition Oy voi ostaa kehitysympäristöstä virallisen ja rajoittamattoman version.

Työn toteutuksessa luotiin ensin uusi projekti, jonka asetukset määriteltiin oikeiksi. Tämän jälkeen päästiin tekemään varsinaista työtä eli luomaan ohjelmapohjaa. Ohjelmapohja sisältää Atmelin valmiita kirjastoja, erilaisia ajureita sekä grafiikkakirjaston.

Ohjelmapohjan päälle suunniteltiin ja toteutettiin sovellusohjelma. Sovellusohjelma esittää mediasoittimen graafista käyttöliittymää, jota ohjataan kokeilukortin painikkeella. Kehitysympäristön kokorajoituksen vuoksi sovellusohjelmassa ei voitu käyttää alun perin suunniteltua taustakuvaa. Lopuksi arvioitiin ohjelman ja laitteiston suorituskykyä silmämääräisesti.

7.1 Uuden projektin luominen IARiin

IAR-kehitysympäristö tukee laajalti eri mikro-ohjaimia, joten ennen kuin ohjelmaa päästään varsinaisesti luomaan, täytyy kehitysympäristön asetukset määritellä oikeiksi. Oikeilla asetuksilla kääntäjä osaa tuottaa käytössä olevalle mikro-ohjaimelle sopivaa binäärikoodia.



Kuva 16. IAR Embedded Workbench käyttöliittymä.

IAR-kehitysympäristön työtilan (workspace) pystyy muokkaamaan itselle sopivaksi. Kuvassa 16 on insinöörityössä käytetty työtila. Oletuksena ikkunassa on projektin rakennepuu, ohjelman kirjoitukseen tarkoitettu tila ja tietoikkuna, johon tulevat käännöksen aikana mm. virheilmoitukset. Kun IARiin luodaan uutta projektia, kannattaa aina luoda kokonaan uusi työtila, jotta samannimiset tiedostot eivät mene projektien kesken sekaisin. Tästä esimerkkinä on main.c, joka on yleensä käytössä kaikissa projekteissa.

Ennen kuin IARilla aloitetaan uusi projekti, tietokoneelle kannattaa luoda valmiiksi projektikansio. Projektikansioon tallennetaan kaikki kyseiseen projektiin tarvittavat tiedostot, mm. käytettävät kirjastot, grafiikat ja ohjelmatiedostot.

7.2 Projektin asetukset IARissa

IARissa asetukset ovat aina projekti- ja konfiguraatiokohtaisia. Jos työtilassa on auki esimerkiksi kaksi projektia, joissa molemmissa on sekä debug- että release-konfiguraatiot, on mahdollista määrittää neljät eri asetukset. Insinöörityössä käytettiin vain debug-konfiguraatiota, koska työn tarkoituksena oli luoda ohjelmapohja tuleville projekteille eikä asiakkaalle luovutettavaa julkaistavaa ohjelmaa.

Kääntäjän asetukset

Kääntäjän yleisissä asetuksissa valitaan ohjelmoitava piiri, joka insinöörityössä on Atmelin SAM3SD8C. Asetuksissa täytyy myös määritellä kääntäjän tuottaman tiedoston muoto, joka oletuksena on suoritettava tiedosto (.exe). Ohjelmoitaessa mikro-ohjainta, jossa ei ole käyttöjärjestelmää, tiedoston muoto täytyy muuttaa binääritiedostoksi (.bin).

IARissa on mahdollista määrittää, miten kääntäjä optimoi ohjelman käännösvaiheessa. Mahdollisuudet ovat nopeuden tai tiedoston koon mukaan optimoitu ohjelmakoodi. Sulautetuissa järjestelmissä mikro-ohjaimella ei yleensä ole suoritusmuistia käytettävissä kovin paljon, joten ohjelmakoodin optimointi on järkevää tehdä koon mukaan. Tämä asetus tuottaa mahdollisimman pienen koodin, mutta sen suorittaminen saattaa olla hitaampaa.

Yleisistä asetuksista voidaan myös määritellä esikääntäjän asetukset. Asetuksissa voidaan määritellä esimerkiksi useita kansioita, joista etsitään ohjelman kääntämisen aikana tarvittavia tiedostoja.

Muut asetukset

Koostajan asetuksissa voidaan määritellä, että se tuottaa virheiden etsintäinformaatiota. Lisäksi asetuksista voidaan valita, että koostaja tulostaa ns. list-tiedoston, johon tulostuu valinnan mukaisesti tietoa esikääntäjän tekemistä muutoksista. Nämä voidaan määritellä jokaiselle projektille tarkoituksen mukaisesti. Insinöörityössä määriteltiin koostaja tuottamaan virheiden etsintäinformaatiota sekä list-tiedoston mahdollisten ongelmien ratkaisemista helpottamaan.

Tuotettavan tiedoston nimi ja muoto voidaan määrittää output converter -asetuksissa. Tiedoston nimi kannattaa valita niin, että se on mahdollisimman havainnollinen, ja muoto määritellään käyttötarkoituksen mukaan. Insinöörityössä haluttiin, että kääntäjä tuottaa binääritiedoston eli tiedoston muodoksi valittiin .bin. Virheiden etsintäasetuksissa määriteltiin, että käytössä on virheiden etsintätyökalu J-Link/J-Trace. JTAG-emulaattorille voidaan myös määritellä tiedonsiirtonopeus. Insinöörityössä käytettävän emulaattorin nopeus määriteltiin mukautuvaksi (adaptive).

7.3 Ohjelmapohjan luominen

Insinöörityön tavoitteena oli luoda ohjelma, joka toimii pohjana yrityksen tulevissa projekteissa. Ohjelman tarkoitus on helpottaa TFT-näytön käyttöönottoa ja sen ohjaamiseen tarvittavan ohjelman kehitystyötä. Ohjelma pyrittiin luomaan niin, että se olisi mahdollisimman helposti käytettävissä erilaisissa ympäristöissä.

Ohjelman kerroksellisuus

Sulautettujen järjestelmien ohjelmakoodi on hyvin laiteläheistä, minkä vuoksi ei ole mahdollista luoda vapaasti laiteympäristöstä toiseen siirrettävää ohjelmaa. Ohjelmassa muutetaan mm. mikro-ohjaimen yksittäisten nastojen tilaa, mikä on luonnollisesti komponenttikohtaista. Jotta ohjelma voidaan siirtää mahdollisimman helposti laiteympäristöstä toiseen, siitä on hyvä tehdä kerroksellinen. Tämä tarkoittaa sitä, että ensin määritellään kyseisen ympäristön sähköinen toteutus, kuten käytettävät mikro-ohjaimen nastat ja kellotaajuus, jonka päälle luodaan erilaisia ajureita. Tällöin ylemmän tason ohjelma on siirrettävissä toiselle alustalle, ja vain alimman tason nastamäärittelyt täytyy tehdä uudelleen.

Kuva 17. Ohjelmapohjan kerroksellisuus.

Kuvassa 17 on esitetty insinöörityön ohjelmapino. Alimmalla tasolla on tehty laitekohtaiset asetukset, joiden päälle on luotu laiteajurit. Bussim.c on grafiikkakirjaston ajuritiedosto, jolla grafiikkakirjasto integroidaan osaksi ohjelmaa. Ylimmällä tasolla on

grafiikkakirjasto ja itse sovellusohjelma. Seuraavissa kappaleissa käydään tarkemmin läpi ohjelmapinon kerroksia.

Mikro-ohjaimen nastojen määrittely

Alimmalla tasolla määritellään laiteympäristön ominaisuudet, kuten käytettävät nastat ja kellotaajuus. Insinöörityössä käytettiin valmista Atmelin kokeilukorttia, johon on saatavissa valmistajan luomia kirjastoja. Kirjastoista löytyy board.h-tiedosto, jossa kyseisen kokeilukortin alimman tason määrittelyt on tehty valmiiksi. Tiedostossa määritellään mm. mihin nastoihin näyttö, painikkeet ja muut oheiskomponentit on kytketty.

Kun tulevaisuudessa suunnitellaan oma kytkentä, pitää luoda oma vastaava tiedosto, jossa määritellään kyseisen kytkennän ominaisuudet. Parhaassa tapauksessa ohjelman siirtämiseksi riittää pelkästään uuden board.h-tiedoston luominen, jos ylemmän tason ohjelmakoodi on tehty riittävän hyvin.

Näytön ajurit

Kokeilukortin näytölle löytyvät myös valmiit ajurit Atmelin kirjastoista. ILI9325.h ja ILI9325.c ovat Atmelin tekemiä ajuritiedostoja, joita voidaan käyttää kaikkien ILI9325-ohjainpiiriä käyttävien näyttöjen kanssa. Mikäli käytetään näyttöä, jossa on jokin muu ohjainpiiri, joudutaan nämä tiedostot korvaamaan kyseisen ohjainpiirin ajuritiedostoilla.

ILI9325.h-tiedostossa määritellään näytönohjaimen rekisterit ja niiden käyttöön tarkoitetut makrot. ILI9325.c-tiedostosta löytyy mm. erilaisia funktioita, joilla valmistellaan, kirjoitetaan ja luetaan näytön rekistereitä.

Näytön initialisointi

Ennen kuin näytölle voidaan kirjoittaa tekstiä tai piirtää grafiikkaa, se täytyy initialisoida. Insinöörityössä kirjoitetussa ohjelmassa hyödynnetään Atmelin kirjastoista löytyviä ajureita näytön käyttöönotossa. Esimerkkiohjelmaa tutkimalla käy ilmi, että tätä varten on tehty valmiita funktioita, joita käytettiin ohjelmassa. Kun IARin työtilaan lisätään Atmelin tarjoamat kirjasto-projektit, näytön initialisointi onnistuu kutsumalla funktiota LCDD_Initialize. Funktio määrittelee tarvittavat nastat, initialisoi staattisen

muistinohjaimen (SMC) ja suorittaa initialisointisekvenssin. Tämän jälkeen näytön taustavalo syttyy, mutta itse näyttö on vielä sammutettuna.

Näytön saadaan päälle kutsumalla funktiota LCDD_On. Tämä on myös Atmelin kirjastoissa määritelty funktio, joka kirjoittaa näytönohjaimen rekistereihin oikeat arvot, joilla näyttö saadaan päälle. LCDD_Initialize- ja LCDD_On-funktioiden jälkeen näyttö on valmis käytettäväksi. Insinöörityön tavoitteena oli käyttää grafiikkakirjastoa tekstin ja grafiikan luomiseen, joten seuraava tehtävä oli grafiikkakirjaston käyttöönotto.

Grafiikkakirjaston integroiminen

Ramtex lähettää grafiikkakirjaston mukana painetun oppaan, jossa kerrotaan grafiikkakirjaston käytöstä ja oikeanlaisten ajureiden luomisesta. Ramtexin grafiikkakirjastoa voidaan käyttää mikro-ohjaimissa ja myös erillisissä mikroprosessoreissa. Käytettäessä mikroprosessoria näyttö kytketään sen dataväylään, mikä on nopein mahdollinen tapa käyttää näyttöä. Mikro-ohjaimissa dataväylä on mikropiirin sisällä eikä siihen yleensä päästä käsiksi. Tämän vuoksi mikro-ohjaimia käytettäessä dataväylää pitää simuloida piirin I/O-nastoilla. Tätä varten grafiikkakirjastossa on määritelty GHW_SINGLE_CHIP-makro, joka pitää ottaa käyttöön IARin esikääntäjän asetuksissa. [25.]

Jotta grafiikkakirjaston funktioita voidaan käyttää, täytyy ensin luoda ajurit, joilla kirjasto integroidaan osaksi ohjelmaa. Ajureiden avulla grafiikkakirjasto osaa ohjata käytettävää näytönohjainta oikein. Grafiikkakirjaston ajuritiedosto on nimeltään bussim.c, joita Ramtexissa on useita erilaisia. Oikea tiedosto valitaan käytettävän näytönohjainpiirin mukaan. Tässä projektissa käytettiin ILI9325-piiriä, jolloin oikea bussim.c löytyy ILI9320-kansiossa olevan readme.txt-tiedoston ohjeiden avulla. Bussim.c on tiedosto, jolla Ramtexin grafiikkakirjasto liitetään käytössä olevan ympäristön laiteajureihin. Se toimii sovittimena, joka ohjaa kaiken grafiikkakirjastolta tulevan tiedon oikeisiin rekistereihin.

ILI9325:ssä on kahdenlaisia rekistereitä, index- ja status-rekistereitä. Status-rekistereistä voidaan lukea erilaista tietoa näytöltä, mutta tätä ominaisuutta ei käytetä tässä insinöörityössä. Index-rekistereihin voidaan vain kirjoittaa tietoa. Niillä tehdään erilaisia asetuksia ja säätöjä, kuten initialisointi sekä virta- ja gamma-asetuksia. Näytölle kirjoitettavan grafiikan kannalta tärkeimmät index-rekisterit ovat osoitteissa

0x20-0x22. 0x20 ja 0x21-osoitteisiin kirjoitetaan sen pikselin koordinaatit, joista alkaen näytölle halutaan kirjoittaa grafiikkaa. 0x22-osoitteeseen kirjoitetaan pikseli kerrallaan kunkin pikselin haluttu väri.

Jokaisella rekisterillä on osoite ja jokaiseen rekisteriin voidaan kirjoittaa tietoa. Näytön ja mikro-ohjaimen välillä ei ole erillisiä osoite- ja dataväyliä, vaan rekisterin osoite ja sinne kirjoitettava tieto siirretään samaa väylää pitkin. Ensin kirjoitetaan rekisterin osoite, minkä jälkeen kyseiseen rekisteriin kirjoitetaan haluttu tieto. Se, onko kyseessä rekisterin osoite vai sinne kirjoitettava tieto, valitaan sähköisesti näytön RS-signaalilla. Ohjelmassa tämä valinta tehdään bussim.c-tiedostossa.

Tutkimalla ILI9325-näytönohjaimen datalehteä ja kirjastotiedostoja ili9325.c ja ili9325.h selvisi, miten rekisterin osoite ja sinne kirjoitettava tieto erotetaan toisistaan. Valinta toteutettiin if-else-lauserakenteella. Atmelin kirjastoista löytyivät valmiit makrot rekisterin osoitteen ja sinne kirjoitettavan tiedon lähettämiseen mikro-ohjaimelle. Näitä makroja hyödyntäen luotiin ajurit, joilla saatiin grafiikkakirjaston funktiot käyttöön. Jos tulevaisuudessa on tarvetta vaihtaa näytönohjainta, ajurit on muutettava uuden näytönohjaimen mukaisiksi. Tämä vaatii jonkin verran perehtymistä käytettävään näytönohjaimeen. Tällaisten ajureiden avulla ohjelmasta saadaan yleiskäyttöisempi, ja se on helpommin siirrettävissä muihin laiteympäristöihin.

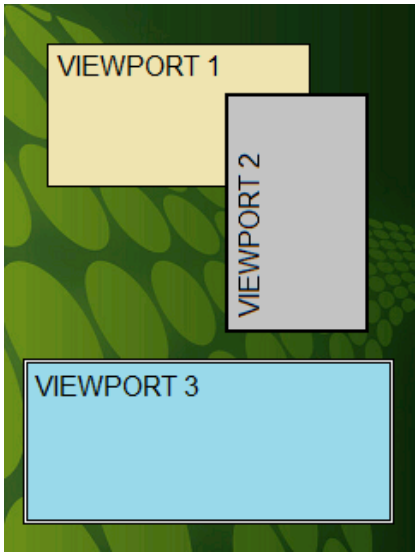
7.4 Grafiikkakirjaston käyttö

Kun grafiikkakirjasto on integroitu osaksi ohjelmaa, sen käyttö on melko yksinkertaista. Grafiikkakirjaston mukana tulee joitakin esimerkkiohjelmia, joita voi käyttää apuna oman ohjelman luomisessa. Kirjaston mukana tulee myös painettu ohjekirja, jossa on esimerkkejä yleisimpien funktioiden käytöstä sekä lueteltu grafiikkakirjaston kaikki funktiot ja niiden käyttö hyvin yksityiskohtaisesti.

Vaikein asia grafiikkakirjastoa käytettäessä on se, että kaikki funktiot ovat grafiikkakirjaston omia. Vaikka ohjelmoija osaisi C-kieltä, hänen täytyy etsiä oikeat funktiot ohjekirjasta. Tietysti normaalit C-kielen funktiot ja rakenteet toimivat myös grafiikkakirjastoa käytettäessä, mutta kirjastossa ovat omat funktiot esimerkiksi tekstin ja symbolien tulostamiseen näytölle.

Viewport

Ramtexin grafiikkakirjastossa voidaan käyttää ns. viewporteja. Viewportit ovat määriteltäviä alueita, joihin voidaan kirjoittaa tekstiä ja piirtää grafiikkaa, aivan kuin kyseessä olisi itsenäinen näyttö. Teksti tai kuva leikataan automaattisesti, mikäli se on isompi kuin viewportin määritelty koko. Viewport voidaan määritellä esimerkiksi taustakuvan päälle, mikä helpottaa grafiikan ryhmittelyä näytöllä. Viewportiin voidaan myös määritellä oma taustakuva tai -väri.



Kuva 18. Esimerkki viewporteista.

Kuvassa 18 on 240 x 320 pikselin kokoisen taustakuvan päälle luotu kolme viewportia. Viewportit voivat sijaita missä tahansa kohtaa näyttöä, ja ne voivat olla myös päällekkäin. Kuvan 18 viewport 1:een voidaan kirjoittaa tekstiä myös alueelle, joka on viewport 2:n alla.

Grafiikan piirtäminen

Näytölle voidaan tulostaa grafiikkaa kahdella tavalla. Ennalta piirretty kuva voidaan lukea muistista ja siirtää näytölle, tai näytölle voidaan laskemalla piirtää yksinkertaisia kuvioita, kuten viivoja ja ympyröitä. Tätä varten grafiikkakirjastossa on erilaisia valmiita funktioita, kuten *grectangle*, joka piirtää halutun kokoisen neliön haluttuun kohtaan, ja *glineto*, jolla voidaan piirtää halutun pituinen viiva mihin tahansa suuntaan [26]. Kuvioiden piirtäminen laskemalla on huomattavasti nopeampaa kuin grafiikan

lukeminen muistista. Yleensä yksinkertaiset kuviot kannattaa piirtää laskemalla ja monimutkaisemmat tallentaa valmiina kuvana järjestelmän muistiin.

Valmiiden kuvien tulostaminen näytölle

Graafista käyttöliittymää GUI (Graphical User Interface) tehtäessä käytännössä aina joudutaan käyttämään grafiikkaa, joka on luotu sitä varten tehdyllä ohjelmalla. Tässä insinöörityössä tällaisia graafisia elementtejä olivat play- ja pause-symbolit sekä levyn kansikuva. Alkuperäisessä formaatissa nämä kuvat olivat bitmap-muotoisia (.bmp), jotka muutettiin symbolimuotoon (.sym) erityisellä Color LCD Icon Editor -ohjelmalla.

Symbolitiedostojen tulostusfunktiot sijaitsevat symbols.c-tiedostossa. Tässä tiedostossa määritellään käytettävät symbolit sekä niiden sijainnit. Varsinainen tulostus tapahtuu *gputsym*-funktiolla.

Fonttien käyttö

Ramtexin grafiikkakirjastoon kuuluu 18 eri fonttia. Grafiikkakirjaston fontit poikkeavat tietokoneiden fonteista. Yleensä mielletään, että kyseessä on sama fontti, vaikka sen kokoa, lihavointia ja kursivointia muutetaan. Ramtexin grafiikkakirjastossa jokainen fontti on kokoelma tietynlaisia ja kokoisia kirjaimia. Jos halutaan esim. Arial-fontti 10 pikselin ja 12 pikselin korkuisena, tarvitaan kaksi eri fonttia. Sama pätee myös lihavointiin ja kursivointiin. Tästä syystä yleensä joudutaan luomaan omia tarkoitukseen sopivia fontteja. Myös tämä tehdään Color LCD Icon Editor -ohjelmalla.

Kun fontti luodaan, syntyy kolme tiedostoa, esimerkiksi Arial.c, Arial.cp ja Arial.sym, joista .c-tiedosto sisällytetään projektiin. Lisäksi fontti pitää esitellä gi_fonts.h-tiedostossa, jotta sitä voidaan käyttää. Fontti valitaan ohjelmassa *gselfont*-funktiolla, minkä jälkeen haluttu teksti voidaan tulostaa näytölle *gputs*-funktiolla.

7.5 Sovellusohjelma

Insinöörityössä luotiin lopuksi yksinkertainen sovellusohjelma, joka käyttää luotua ohjelmapohjaa ja grafiikkakirjastoa. Sovellusohjelmalla todennetaan ohjelmapohjan toimivuus ja esimerkinomaisesti näytetään, miten sitä käytetään.

Kuva 19. Graafinen käyttöliittymä.

Sovellusohjelma esittää mediasoittimen graafista käyttöliittymää (kuva 19). Näytöllä näytetään soivan levyn kansikuva, valittu kappalenumero, kappaleen ja levyn nimi sekä kappaleen kulunut aika. Ohjelmassa käytetään myös kokeilukortin yhtä painiketta, joka esittää laitteen play/pause-nappia. Painiketta ensimmäistä kertaa painettaessa näytöllä oleva pause-symboli muutetaan play-symboliksi ja kappaleen kuluvaa aikaa kuvaava kello käynnistyy. Painiketta uudelleen painettaessa laite siirtyy pause-tilaan, jolloin kello pysähtyy. Myös näytön play-symboli päivitetään takaisin pause-symboliksi.

Ohjelmassa käytetään paljon grafiikkakirjaston omia funktioita ja näytön initialisointiin Atmelin kirjastoja. Ohjelma initialisoi näytön ja kytkee sen päälle, minkä jälkeen näytölle määritellään yksi koko näytön kokoinen viewport. Viewportiin sijoitetaan play-/pause-symbolit, kansikuva, tekstit ja kello. Tekstit on toteutettu itse tehdyllä fontilla, ja symbolit on käännetty bitmap-kuvista.

Grafiikka ja kuvien luominen

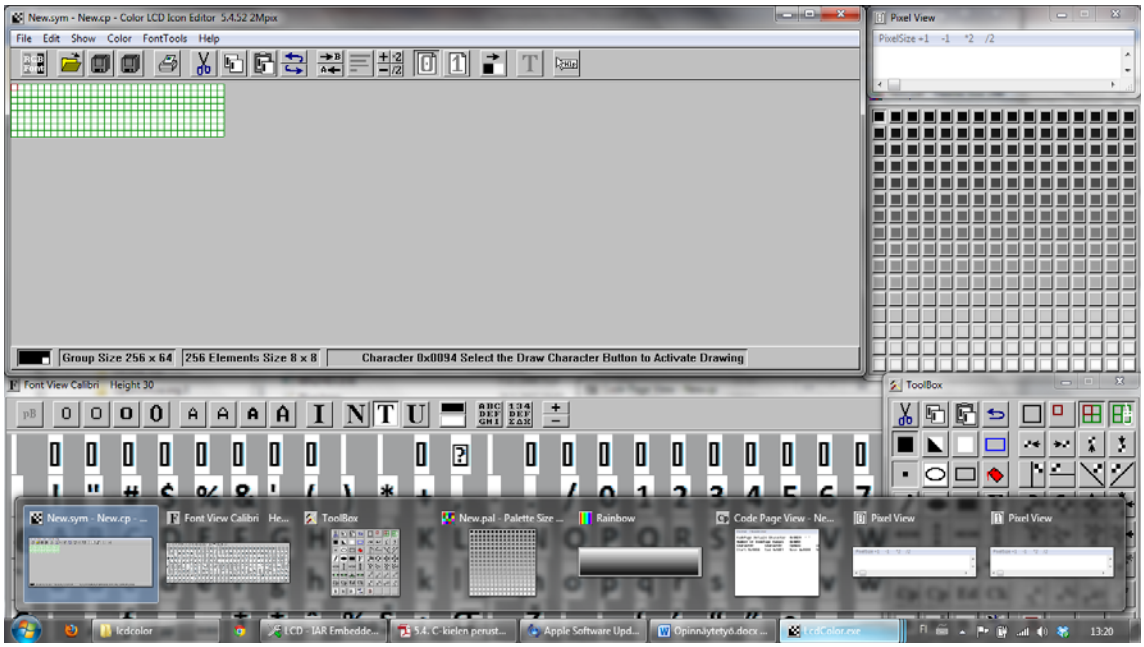
Ohjelmaa varten luotiin kolme graafista elementtiä. Play- ja pause-symbolit piirrettiin Microsoft Paint -ohjelmalla ja kansikuvagrafiikka haettiin internetistä. Kansikuva muutettiin MS Paintilla sopivan kokoiseksi, minkä jälkeen sen ympärille piirrettiin valkoiset kehykset.

Kuva 20. Näytön symbolit mustalla taustalla.

Ohjelmassa käytetyt symbolit ovat kuvassa 20. Kaikki kuvat luotiin aluksi bitmap-muotoisiksi, josta ne käännettiin Color LCD Icon Editor -ohjelmalla symboleiksi.

Kuva 21. Alkuperäisen suunnitelman mukainen graafinen käyttöliittymä.

Alun perin tarkoitus oli tehdä näytölle myös taustakuva ja käyttää erikokoisia fontteja. Kuvassa 21 esitetään alkuperäinen suunnitelma laitteen graafisesta käyttöliittymästä. Koko näytön täyttävää grafiikkaa käyttävä ohjelma tuli kuitenkin liian suureksi. Esimerkiksi pelkkä näytön taustakuva oli kooltaan 225 kB, joka olisi kyllä mahtunut mikro-ohjaimen muistiin, mutta kehitysympäristöstä oli käytössä vain ilmainen 30 kB:n kokorajoitettu versio. Tämän vuoksi käyttöliittymää yksinkertaistettiin. Taustakuvan sijaan näytön tausta väritettiin mustaksi laskemalla ja fonttien lukumäärä supistettiin yhteen. Näillä muutoksilla ohjelman koko jäi alle 30 kB:n.



Kuva 22. Color LCD Icon Editorin käyttöliittymä.

Color LCD Icon Editorilla voidaan avata bitmap-muotoinen kuva ja muokkaamaan sitä. Ohjelman käyttöliittymä ei ole kovin käyttäjäystävällinen. Siihen avautuu useita eri ikkunoita, jotka myös näkyvät Windowsin alapalkissa omina ikkunoinaan (kuva 22). Tämä hankaloittaa esimerkiksi ohjelmasta toiseen siirtymistä. Color LCD Icon Editorin mukana tulee pdf-muotoinen ohje, jossa on selostettu kaikki tärkeimmät työkalut ja esimerkkien avulla kaikki yksinkertaisimmat muunnokset.

Ensimmäiseksi valitaan värisyvyys. Vaihtoehtona on 24, 16, 8 ja 2 bittiä. Valittu värisyvyys näkyy tietokoneen ruudulla suoraan, joten kokeilemalla eri vaihtoehtoja voi reaaliaikaisesti arvioida, mikä on riittävä värisyvyys. Insinöörityössä käytettyihin kuviin valittiin 16 bitin värisyvyys, joka on hyvä kompromissi päivitysnopeuden ja kuvanlaadun välillä (ks. luku 3.3).

Seuraavaksi muutetaan kuva symboliksi. Tällöin Color LCD Icon Editor luo .sym- ja .c-tiedostot, joita voidaan käyttää sovellusohjelmassa. Molemmat tiedostomuodot tallennetaan yhtä aikaa valitsemalla tallenna kaikki (Save All As...).

Fontin luominen

Myös sovellusohjelmassa käytetty fontti luotiin Color LCD Icon Editorilla. Color LCD Icon Editorilla pystytään luomaan helposti uusi fontti mistä tahansa Windowsin fontista.

Tarvittaessa ohjelmalla pystytään muokkaamaan fontteja ja jopa yksittäisiä kirjaimia, mutta hyvä lopputulos saadaan myös käyttämällä oletusasetuksia.

Insinöörityötä varten luotiin fontti käyttäen pohjana Windowsin 24 pikselin Calibri-fonttia. Tämän kokoinen teksti on selvästi luettavissa insinöörityössä käytetyllä TFT-näytöllä. Työssä kokeiltiin myös pienempiä fonttikokoja, mutta ne olivat epäselviä näytölle tulostettuina.

Ohjelman kulku

Sovellusohjelman main-funktio (void main (void)) sijaitsee main.c-tiedostossa, jossa aluksi suoritetaan initialisointi. Initialisoinnissa mm. priorisoidaan keskeytyslähteet ja määritellään paikalliset muuttujat.

Aluksi main-funktiossa otetaan watchdog-ajastin pois päältä, jonka jälkeen konfiguroidaan SysTick-ajastin ja alustetaan painike (nastamäärittely). Näytön initialisoinnin jälkeen siihen tulostetaan tekstit ja kansikuva. Näytön taustaväriksi on grafiikkakirjastossa määritelty musta.

Seuraavaksi ohjelma siirtyy ikuiseen while-silmukkaan, jossa ensin tarkistetaan status-muuttujasta, ollaanko tällä hetkellä play- vai pause-tilassa. Ohjelman alussa status-muuttuja alustetaan siten, että ohjelma alkaa pause-tilasta, jolloin näytölle tulostetaan pause-symboli, ja kello on pysäytettynä. Ohjelma pyörii tässä while-silmukassa jatkuvasti tarkistaen status-muuttujan tilaa.

Kun painiketta painetaan, ohjelma keskeytyy ja siirtyy keskeytysrutiiniin. Keskeytyksessä status-muuttujan tila muutetaan toiseksi, jonka jälkeen palataan takaisin while-silmukkaan.

Kun while-silmukassa huomataan, että status-muuttuja on play-tilassa, siirrytään for-silmukkaan. Tässä silmukassa käynnistetään kello ja päivitetään näytölle play-symboli. Kellossa inkrementoidaan sekunteja. Kun 60 sekuntia tulee täyteen, ne nollataan ja inkrementoidaan minuutteja. 60 minuutin jälkeen kello nollataan. For-silmukkaa suoritetaan niin kauan, kun ohjelma on play-tilassa. Myös tässä silmukassa status-muuttujan tila muutetaan painikkeen keskeytysrutiinissa. Silmukassa tarkistetaan kaksi

kertaa sekunnissa status-muuttujan tila, ja mikäli se on muuttunut, siirrytään takaisin while-silmukan alkuun.

Liitteessä 1 esitetään sovellusohjelman vuokaavio. Vuokaavio on selkeyden vuoksi jaettu neljälle sivulle. Liitteen 1 neljännellä sivulla on sovellusohjelman keskeytysrutiinin vuokaavio. Liitteessä 2 on sovellusohjelman ohjelmakoodi.

7.6 Lopputuloksen arviointi

Lopputuloksena saatiin toimiva ohjelma, joka tulostaa näytölle itse tuotettua grafiikka ja tekstiä sekä juoksevaa aikaa näyttävän kellon. Painonapilla kellon voi pysäyttää, jolloin näytölle vaihtuu pause-kuvake play-kuvakkeen tilalle. Ylimmän tason ohjelma on todennäköisesti melko helposti siirrettävissä toiselle alustalle, kunhan grafiikkakirjaston ajurit tehdään uudelle alustalle sopiviksi.

Siirrettävyyttä heikentää se, että sulautettuja järjestelmiä ohjelmoitaessa täytyy tehdä paljon tutkimustyötä ohjelmoitavasta alustasta ja sen toiminnasta. Alimman tason ohjelmat ja funktiot ovat mikro-ohjainkohtaisia ja myös kehitysympäristön asetukset on määriteltävä aina käytettävän mikropiirin mukaan. Täysin siirrettävää ohjelmaa ei siis voida saavuttaa, mutta kerroksellisen ohjelman ansiosta ylemmän tason koodi on siirrettävissä.

Päivitysnopeus

Näytön päivitysnopeus on kohtalaisen hyvä. Ohjelman alussa näkyy selvästi havaittava valkoinen välähdys ruudulla, kun näyttö initialisoidaan. Initialisoinnin jälkeen näkyy, kun tausta värjätään mustaksi ylhäältä alas. Näytön initialisointi ja taustan tulostaminen näytölle tapahtuu kuitenkin erittäin nopeasti. Ohjelman alun jälkeen näytölle päivitetään tarvittaessa play- tai pause-symboli ja kulunut musiikkikappaleen kestoaika. Näiden päivitys tapahtuu nopeasti, eikä se ole silmin havaittavissa.

Tästä havaitaan, miten päivitettävän alueen koko vaikuttaa päivitysnopeuteen. Kun koko näytön kokoinen tausta tulostetaan mustaksi, mikro-ohjaimelta joudutaan siirtämään paljon tietoa näytölle. Vaikka tämä tehdään laskemalla eli niin nopeasti kuin on mahdollista, vie niin suuren alan päivittäminen kuitenkin sen verran aikaa, että se on

havaittavissa välähdyksenä. Pienten play- ja pause-symbolien tulostumista ei pysty silmin havaitsemaan, vaikka ne luetaan mikro-ohjaimen muistista.

Näyttö päivittyy kuitenkin niin nopeasti, että lopputulosta voidaan pitää onnistuneena. Suurempaan päivitysnopeuteen voidaan käytännössä päästä vain käyttämällä erillistä mikroprosessoria, jossa näyttö voidaan kytkeä suoraan sen dataväylään. Tällöin nykyisen kahdeksan rinnakkaisen bitin sijaan voitaisiin siirtää kaikki käytettävät 16 bittiä rinnakkain, joka jo sellaisenaan kaksinkertaistaisi päivitysnopeuden. 32-bittisellä prosessorilla voitaisiin siirtää yhtä nopeasti myös 18 bittiä rinnakkain, joka on näytön maksimivärisyvyys. Tällöin näytöllä näytettävien värien määrä saataisiin 4...8-kertaiseksi.

Värintoistokyky ja näkyvyys eri kulmista

Insinöörityössä käytetyn näytön värintoistokyky ei ole paras mahdollinen. Suoraan ylhäältä katsottaessa se toistaa värit kohtalaisesti, mutta jos näyttöä katsotaan sivusta, värit haalistuvat eri tahtiin. Musta on hieman harmaan sävyinen myös suoraan ylhäältä katsottaessa, mutta sivusta musta väri näyttää melkein vaalean harmaalta. Tämä johtuu luultavasti ainakin osittain ns. valovuodosta, jossa taustavalo kuultaa kiteiden läpi. Tämä on tuttu ominaisuus myös mm. LCD-televisioista.

Selkeimmin väreistä näkyy punainen, joka säilyttää sävynsä melko hyvin myös sivusta katsottaessa. Vihreä ja sinisen eri murretut sävyt häviävät melkein kokonaan näkyvistä tai muuttuvat harmaiksi sivusta katsottaessa. Puhdas sininen väri säilyttää kuitenkin sävynsä sivusta katsottaessa yhtä hyvin kuin punainen.

Näytöstä ei löytynyt datalehteä, josta olisi voinut katsoa vertailevia arvoja. TFT-näyttöjen datalehdissä yleensä aina kerrotaan näytön katselukulmat eri suunnista. Olisi ollut mielenkiintoista verrata tämän näytön ilmoitettuja katselukulmia omiin havaintoihin.

Painikkeen toiminta

Ohjelmassa siirrytään pause- ja play-tilasta toiseen painamalla kokeilukortilla olevaa painiketta. Painike esittää ohjelmassa soittimen play/pause-nappia, ja sen painaminen saa ohjelmassa aikaan keskeytyksen, jossa tilaa vaihdetaan. Keskeytys tapahtuu painikkeen nousevalla reunalla, eli silloin kun painike vapautetaan. Ohjelma vastaa

painikkeen painallukseen nopeasti. Näytölle päivittyy uusi tila käytännössä heti, kun painike on vapautettu.

Ensimmäisessä ohjelmaversiossa play-tilasta pause-tilaan siirryttäessä pause-symboli päivitettiin näytölle vasta siirryttäessä while-silmukkaan keskeytyksen jälkeen. Tämä aiheutti sen, että pause-symboli päivittyi näytölle huomattavan hitaasti painikkeen painamisen jälkeen. Hidas päivittyminen johtui siitä, että keskeytyksen jälkeen ohjelma vielä odotti Wait-funktion loppuun ennen kuin siirtyi ohjelmassa seuraavaan kohtaan. Pause-symbolin päivittymisen nopeuttamiseksi ohjelmaan tehtiin muutos, niin että symboli päivitetään jo keskeytysrutiinissa, jolloin se päivittyy näytölle käytännössä heti painikkeen painamisen jälkeen.

8 Yhteenveto

Insinöörityössä oli tavoitteena tehdä Amplition Oy:lle ohjelmapohja TFT-näytön ohjaamiseen mikro-ohjaimella. Mikro-ohjaimeksi oli etukäteen valittu Atmel SAM3SD8, joka on melko edullisesta hinnastaan huolimatta varsin monipuolinen. Työssä käytettiin kyseiseen mikro-ohjaimeen perustuvaa SAM3S-EK2-kokeilukorttia, jossa on mukana mm. 2,8”:n 240 x 320 pikselin TFT-näyttö.

Työkaluina käytettiin IAR Embedded Workbench for ARM -ohjelmaa kehitysympäristönä, SAM-ICE-JTAG-emulaattoria, Ramtex Graphic Dot Matrix Color LCD Driver -grafiikkakirjastoa, Color LCD Icon Editor -kuvankäsittelyohjelmaa sekä FTDI TTL-232R-3V3 sarjaliikennemuunninta.

Ohjelmapohjassa käytettiin hyväksi Atmelin valmiita kirjastoja ja ohjelmaan integroitiin grafiikkakirjasto. Grafiikkakirjaston käyttö helpottaa tekstin ja kuvien tulostamista näytölle sekä tekee ohjelmasta paremmin siirrettävän ympäristöstä toiseen. Ohjelmapohjan päälle tehtiin vielä sovellusohjelma, joka tulostaa näytölle mediasoittimen graafisen käyttöliittymän. Sovellusohjelmassa käytetään itse luotua fonttia ja grafiikkaa. Alun perin oli tarkoitus tulostaa näytölle myös taustakuva, mutta ohjelman koko olisi kasvanut liian suureksi ilmaiselle IARin versiolle.

Sovellusohjelma todistaa, että luotu ohjelmapohja toimii, joten insinöörityötä voidaan pitää onnistuneena. Tulevaisuudessa sovellusohjelman siirtäminen toiselle alustalle

vaatii kuitenkin perusteellista tutustumista uuteen ympäristöön. Jos tulevassa projektissa käytetään samaa mikro-ohjainta ja näytönohjainpiiriä kuin insinöörityössä, ohjelmapohjaa voidaan hyödyntää sellaisenaan. Tällöin päästään suoraan kehittämään sovellusohjelmaa ja säästetään aikaa tuotekehityksessä.

Insinöörityössä olisi voitu vielä jatkaa sovellusohjelman kehittämistä, esimerkiksi ottamalla toinen painike käyttöön stop-nappiksi. Jos IARista olisi ostettu maksullinen kokorajoittamaton versio, olisi voitu myös käyttää taustakuvaa ohjelmassa. Jatkossa Amplitionin kannattaakin hankkia IARista maksullinen ja rajoittamaton versio, joka mahdollistaa tulevan ohjelmakehityksen täysipainoisesti.

Lähteet

1 SAM3S-EK2 kuva. 2012. Distrelec. Verkkokuva. <https://www.distrelec.ch/ishop/ImagesProduct/stibo/i_7328721-01.jpg> Luettu 2.1.2013.

2 SAM3S-EK2 User Guide. 2012. Atmel Co.

3 TFT LCD. 2012. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/TFT_LCD>. Päivitetty 29.11.2012. Luettu 2.1.2013.

4 Introduction to graphics and LCD technologies. 2009. Verkkodokumentti. NXP Semiconductors N.V. <http://ics.nxp.com/literature/presentations/microcontrollers/pdf/graphics.lcd.technologies.pdf> Helmikuu 2009. Luettu 2.1.2013.

5 ILI9325 datasheet. 2012. ILI TECHNOLOGY CORP.

6 Sami Saalasti. 1997. Sulautetut järjestelmät, mikro-ohjaimet ja niiden ohjelmointi. Verkkodokumentti. Ohjelmistotekniikan seminaari syksyllä 1997. <http://www.mit.jyu.fi/opiskelu/seminaarit/bak/sulautetut/#E19E3> 8.5.1997. Luettu 2.1.2013.

7 ATSAM3SD8CA-CU Datasheet. 2012. Atmel Co.

8 Zhongliang Hu. 2012. Projektipäällikkö, Espotel Oy, Espoo. Keskustelut 7.11.2012 ja 22.11.2012.

9 Jon Gabay 2012 Designing with TFT Displays. Verkkodokumentti. Digikey. <http://www.digikey.com/us/en/techzone/microcontroller/resources/articles/designing-with-tft-displays.html> 18.4.2012. Luettu 2.1.2013.

10 Pentti Vahtera. 2008. Mikro-ohjaimen ohjelmointi C-kielellä 2. Microsalo oy.

11 C (ohjelmointikieli). Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/C-ohjelmointikieli> Päivitetty 27.11.2012. Luettu 2.1.2013.

12 Lasse Lensu. 2003. Tietorakenteet ja C-kieli. Verkkodokumentti. Lappeenrannan teknillinen yliopisto. <http://www2.it.lut.fi/kurssit/02-03/010533000/lecture_notes/node12.html> 23.1.2003. Luettu 2.1.2013.

13 C-programming. 2012. Verkkodokumentti. Wikibooks.org. <http://en.wikibooks.org/wiki/C_Programming/What_you_need_before_you_can_learn> 22.6.2012. Luettu 2.1.2013.

14 IAR Downloads. Verkkodokumentti. IAR Systems. <http://www.iar.com/en/Service-Center/Downloads/> Luettu 2.1.2013.

15 ATSAM3SD8C. 2012. Verkkodokumentti. Atmel Co. <http://www.atmel.com/devices/SAM3SD8C.aspx?tab=tools> Luettu 2.1.2013.

16 Joint Test Action Group. Verkkodokumentti. Wikipedia. <http://fi.wikipedia.org/wiki/Joint_Test_Action_Group> 14.1.2010. Luettu 2.1.2013.

17 Joint Test Action Group. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/JTAG> 24.12.2012. Luettu 2.1.2013.

18 SAM-ICE. Verkkokuva. EFO LTd. <http://www.mymcu.ru/support/vnutrishemnyy_emulyator_at91samice/> Luettu 2.1.2013.

19 Atmel SAM-ICE. 2012. Verkkodokumentti. Atmel Co. <http://www.atmel.com/tools/ATMELSAM-ICE.aspx> Luettu 2.1.2013.

20 Graphics Libary. 2011. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Graphics_library> 8.9.2011. Luettu 2.1.2013.

21 FTDI TTL-232R-3V3. Verkkokuva. Partco Oy. <http://www.partco.biz/verkkokauppa/popup_image.php?pID=10894&image=0> Luettu 2.1.2013.

22 SAM3S-EK2 release notes. 2011. Verkkodokumentti. Atmel Co. <http://www.atmel.com/Images/SAM3S-EK2_demo_release_note.txt> 15.12.2011. Luettu 2.1.2013.

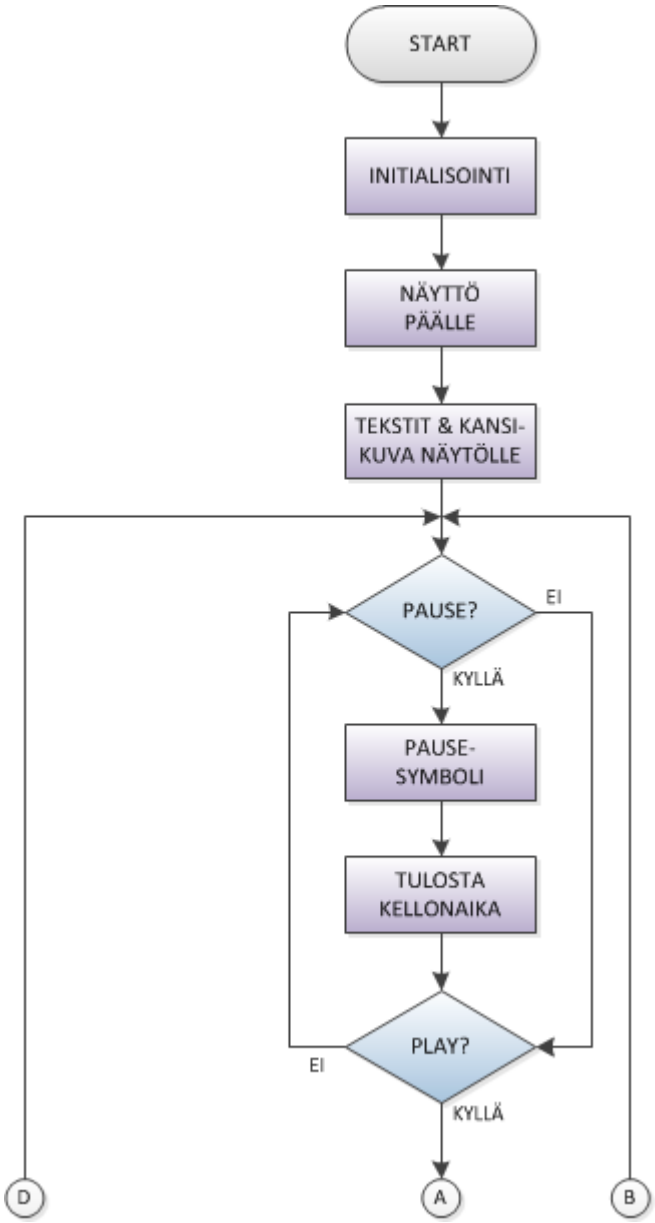
23 Atmel SAM-BA In-system Programmer. 2012. Verkkodokumentti. Atmel Co. <http://www.atmel.com/tools/ATMELSAM-BAIN-SYSTEMPROGRAMMER.aspx> Luettu 2.1.2013.

24 J-Link downloads. Verkkodokumentti. Segger. < http://www.segger.com/jlink-software.html> Luettu 2.1.2013.

25 Graphic Dot Matrix Color LCD Driver Library Manual. 2012. RAMTEX International ApS. Revision 6.43. Syyskuu 2012.

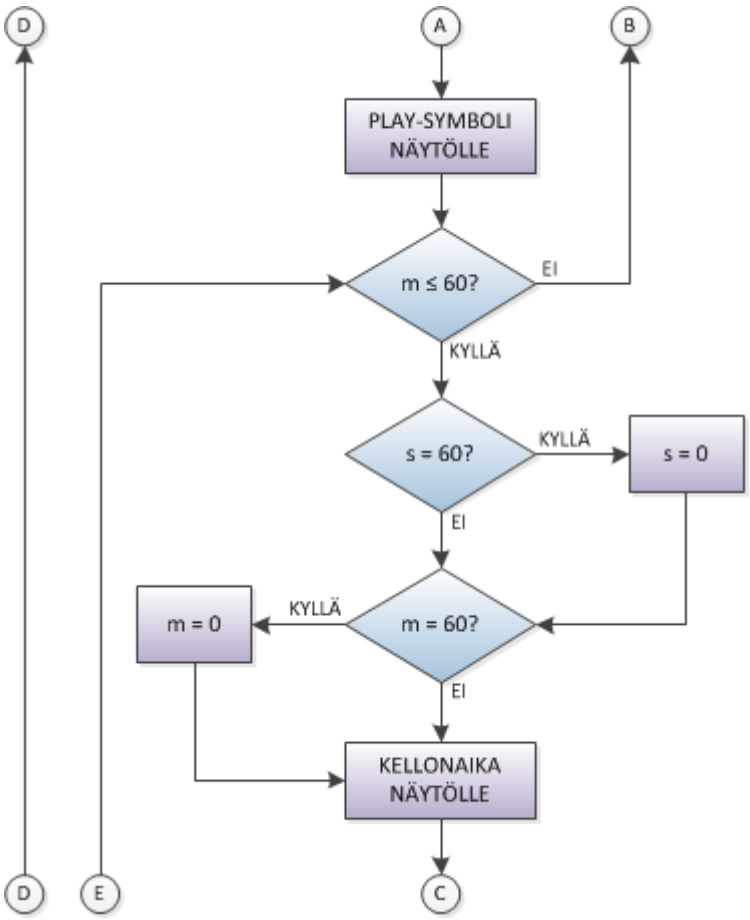
Sovellusohjelman vuokaavio

Sovellusohjelman vuokaavion ensimmäinen osa:



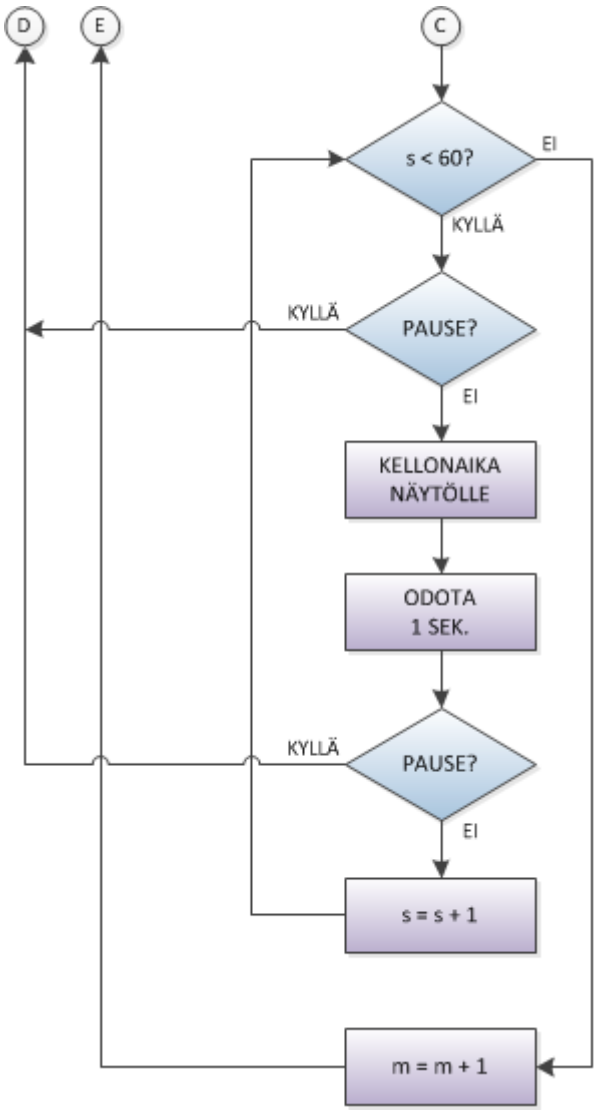
Sovellusohjelman vuokaavio

Sovellusohjelman vuokaavion toinen osa:



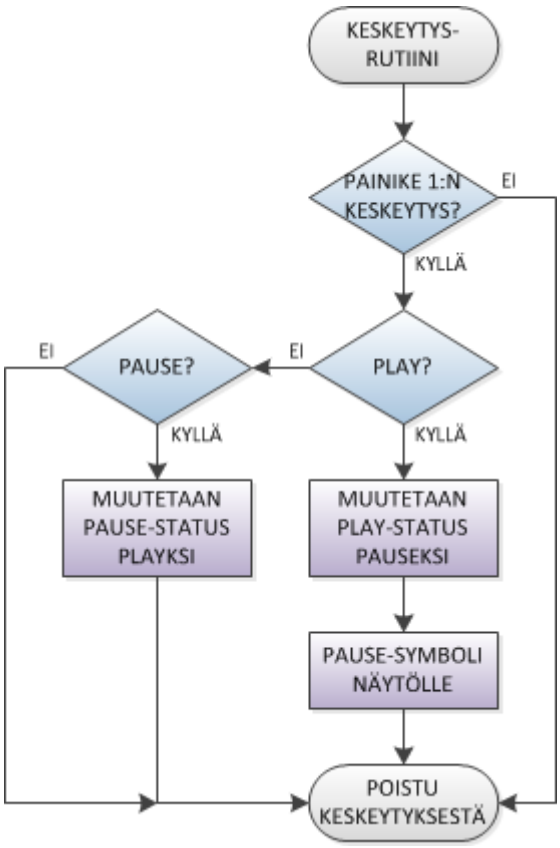
Sovellusohjelman vuokaavio

Sovellusohjelman vuokaavion kolmas osa:



Sovellusohjelman vuokaavio

Sovellusohjelman keskeytysrutiinin vuokaavio:



Sovellusohjelman ohjelmakoodi

```
#include <gdisp.h>
#include "board.h"
#include "symbols.h"
#include <stdint.h>
#include <stdio.h>

/** IRQ priority for PIO (The lower the value, the greater the priority) */
#define IRQ_PRIOR_PIO    0

/** Pushbutton1 pin instance. */
const Pin pinPB1 = PIN_PUSHBUTTON_1 ;

/** Global timestamp in milliseconds since start of application */
volatile uint32_t dwTimeStamp = 0;

/* Local Variables */
int status=0;
int m=0;
int s=0;
char kello[5];

/* Systick handler */
void SysTick_Handler( void )
{
  TimeTick_Increment() ;
}

/* Interrupt routine. Change the status-variable
 * from play to pause and pause to play.*/

static void ProcessButtonEvt( uint8_t ucButton )
{
    if ( ucButton == 0 )
    {
      printf("nappi1\n\r");
      if ( status == 1 )
      {
        status=0;
        pause();
        printf("status on %d\n\r", status);
      }
      else if ( status == 0 )
      {
        status=1;
        printf("status on %d\n\r", status);
```

```
      }
    }
}

/* Handler for button1 rising edge interrupt. */
static void _Button1_Handler( const Pin* pPin )
{
    if ( pPin == &pinPB1 )
    {
        ProcessButtonEvt( 0 ) ;
    }
}

/* Configure the Pushbutton. Configure the PIO as input and
 * generate corresponding interrupt when
 * pressed or released. */
static void _ConfigureButtons( void )
{
    /* Configure pios as inputs. */
    PIO_Configure( &pinPB1, 1 ) ;

    /* Adjust pio debounce filter patameters, uses 10 Hz filter. */
    PIO_SetDebounceFilter( &pinPB1, 10 ) ;

    /* Initialize pios interrupt handlers, see PIO definition in board.h. */
    PIO_ConfigureIt( &pinPB1, _Button1_Handler ) ; /* Interrupt on rising edge
*/

    /* Enable PIO controller IRQs. */
    NVIC_EnableIRQ( (IRQn_Type)pinPB1.id ) ;

    /* Enable PIO line interrupts. */
    PIO_EnableIt( &pinPB1 ) ;
}

void main(void)
{
  /* Disable Watch Dog timer */
  WDT_Disable( WDT ) ;

  /* Configure Systick for 1ms */
  if ( SysTick_Config( BOARD_MCK / 1000 ) )
   {
     printf("-F- Systick configuration error\n\r" ) ;
   }

  /* PIO configuration for Buttons. */
  PIO_InitializeInterrupts( IRQ_PRIOR_PIO ) ;

  printf( "Configure buttons with debouncing.\n\r" ) ;
```

```
  _ConfigureButtons() ;

  /* Initialize display*/
  LCDD_Initialize() ;
  LCDD_On();

  /* Print text on LCD */
  text();

  /* Print CD-Cover on LCD */
  cover();

  /* Clock */
  while ( 1 )
  {
  Alku:
  /* Check the status-variable*/
    if ( status == 0 ) /* if variable is 0 this means pause-state */
    {
      pause();
      sprintf(kello,"%02u:%02u",m ,s);
      gselfont(&Calibri24B);
      gsetpos(70,261);
      gputs(kello);
    }
  /* if variable is 1 this meand play-state.
   * In play state first print play icon to the display then start the
clock.*/
    if ( status == 1 )
    {
      play();
      for ( ; m <= 60 ; m++ )
      {
        if ( s == 60)
          s=0;
        if ( m == 60)
          m=0;
        sprintf(kello,"%02u:%02u",m ,s);
        gselfont(&Calibri24B);
        gsetpos(70,261);
        gputs(kello);
        for ( ; s < 60 ; s++ )
        {
          if ( status == 0 )
/* Goto has been used in this program because we need to jump out of two for
loops. */
            goto Alku;
          sprintf(kello,"%02u:%02u",m ,s);
          gselfont(&Calibri24B);
          gsetpos(70,261);
```

```
            gputs(kello);
            Wait(1000);
            if ( status == 0 )
              goto Alku;
          }
        }
      }
    }
  }
```