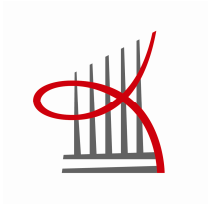


BOOTLOADER AVR-ARKKITEHTUURIN MIKROKONTROLLERILLE

Tommi Kaivola

Opinnäytetyö
Huhtikuu 2013
Tietotekniikka
Sulautetut järjestelmät ja elektroniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Sulautetut järjestelmät ja elektroniikka

TOMMI KAIVOLA:
Bootloader AVR-arkkitehtuurin mikrokontrollerille

Opinnäytetyö 27 sivua
Huhtikuu 2013

TreLab Oloni on mittalaitejärjestelmä, jota voidaan soveltaa muun muassa terveydenhuollossa, henkilökohtaisessa hyvinvoinnissa, yksilövalmennuksessa, rakennusautomaatiossa ja logistiikassa.

Järjestelmän osana olevan langattoman monitoimilaitteen tulee toimia luotettavasti 24 päivää vuorokaudessa ja seitsemän päivää viikossa pitkän aikaa. Ohjelmistolle asetettujen suurten luotettavuusvaatimusten ja runsaan toiminnallisuuden johdosta laitteeseen on haluttu mahdollisuus päivittää ohjelmisto langattomasti.

Matalan tason alustukset ja mahdollisen ohjelmistopäivityksen suorittavaa ohjelmaa kutsutaan bootloaderiksi. Bootloader ottaa vastaan ja kirjoittaa järjestelmän osana toimivalta serveriltä saapuvan ohjelmistopäivityksen. Data lähetetään ilmateitse BLE-protokollaa käyttäen. BLE on Bluetooth 4.0 -standardiin sisältyvä vähän tehoa kuluttava protokolla.

Bootloader toteutettiin osana sovellusohjelman suunnittelu- ja tuotantoprosessia. Prosessissa käytettiin projektinhallintaohjelmana Redminea. Versionhallintaohjelmana toimi GIT. Dokumentaatio toteutettiin Doxygenilla. Ohjelmoinnissa käytettiin Atmel Studio 6 -IDE:ä.

Asiasanat: AVR, BLE, ohjelmistopäivitys, sulautetut järjestelmät, TreLab, Bluegiga

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
ICT Engineering
Embedded Systems and Electronics

TOMMI KAIVOLA:
Bootloader for an AVR-Architecture Microcontroller

Bachelor's Thesis 27 pages
April 2013

TreLab Oloni is a measurement device solution which can meet the needs of industries including health care, personal wellness, personal training, building automation and logistics.

The wireless sensor tag, which is a part of the measurement solution, has to work reliably 24 hours a day and seven days a week for extended time periods. Because of the reliability requirements and copious functionality the option for wireless firmware upgrade is needed.

The program performing low level initialisations and possible firmware upgrade is called the bootloader. The bootloader receives and writes the firmware upgrade coming from the server, which functions as a part of the Oloni solution. The data is sent over the air using BLE-protocol. BLE is a low energy protocol included in Bluetooth 4.0 -standard.

The bootloader was implemented as a part of the design and production process for the firmware. The process used Redmine as a project management program. The version control program was GIT. The documentation was made using Doxygen. Atmel Studio 6 IDE was used for programming.

Key words: AVR, BLE, firmware update, embedded systems, TreLab, Bluegiga

SISÄLLYS

LYHENTEET JA TERMIT

Bootloader	Ohjelmanpätkä, joka suorittaa karkeat initialisaatiot ja lataa sovellusohjelman
AVR	Mikrokontrolleriarkkitehtuuri
BLE	Bluetooth Low Energy, Bluetooth 4.0 -standardiin sisältyvä vähän tehoa kuluttava standardi
DMA	Direct Memory Access, suora muistin hallinta
USART	Universaali synkroninen/asynkroninen sarjaliitäntä
OTA	Over the Air, langattomasti
GCC	Gnu Compiler Collection, vapaan lähdekoodin kääntäjätyökaluketju
IDE	Integrated Development Environment
API	Application Programming Interface, sovellusrajapinta
ASF	Atmel Software Framework, kokoelma matalan tason rajapintaohjelmia Atmelin mikrokontrollereille
PDU	Protocol Data Unit, viestintäprotokollan datayksikkö
GATT	Generic Attribute Profile, BLE-standardiin sisältyvä standardi laitteiden keskinäiselle kommunikaatiolle
CRC	Cyclic Redundancy Check, datalle laskettava tarkistussumma
UUID	Unique User ID, BLE-protokollan käyttämä yksilöllinen tunniste
GPIO	General Purpose Input Output, yleiskäyttöinen liitäntä
JTAG	yleisnimitys IEEE 1149.1 -standardille, jota käytetään nykyisin laajalti integroitujen piirien debug-porttina

1 JOHDANTO

Ympäristötietoa keräävän monitoimilaitteen ohjelmisto sisältää runsaasti erilaisia ominaisuuksia. Näiden ominaisuuksien tulee toimia luotettavasti 24 tuntia vuorokaudessa ja 7 päivää viikossa. Samalla laitteen tehonkulutuksen tulee olla pieni. Ominaisuuksien toteuttaminen ja yhteen sovittaminen on pitkä prosessi, ja usein käy niin, että laite on saatava markkinoille ennen kuin kaikki ominaisuudet ovat viimeistellysti asiakkaan käytettävissä. Toisinaan myös jokin ohjelmiston virhe, bugi, läpäisee tuotannon testausprosessin ja ilmenee kiusallisesti vasta asiakasympäristössä.

Uuden ominaisuuden asiakkaalle saamiseksi tai virheen korjaamiseksi pahimmassa tapauksessa olisi koko toimitettu laitekanta kerättävä takaisin tehtaalle ohjelmistopäivitystä varten. Tällaisen tilanteen välttämiseksi on monitoimilaitteeseen haluttu mahdollisuus ohjelmiston päivittämiseen langattomasti. Vastaava toiminnallisuus on arkipäivää esimerkiksi PC-tietokoneissa ja matkapuhelimissa, mutta pienemmissä sulautetuissa järjestelmissä ratkaisu ei ole vielä yleistynyt.

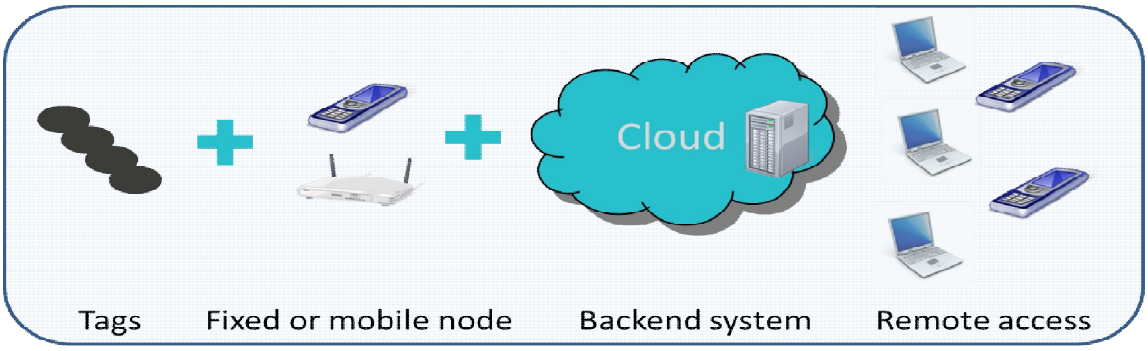
Tämä opinnäytetyö kertoo lyhyesti TreLab Oloni-järjestelmästä, jonka osa monitoimilaite on. Työssä kuvataan ohjelmistotuotannolliset työkalut aina integroidusta kehitysympäristöstä projektinhallintaohjelmistoon.

Kilpailuteknisistä syistä langattoman ohjelmistopäivityksen kulkua ei kerrota tarkkaan. Prosessista annetaan kuitenkin yleiskuva.

2 TRELAB OLONI

TreLab Oloni on monitoimimittalaitejärjestelmä, joka palvelee niin ympäristösuureiden, läsnäolon kuin liikkeenkin mittauksessa. Järjestelmän kyvykkyyttä tullaan jatkossa laajentamaan. Järjestelmä sopii muun muassa terveydenhuoltoon, henkilökohtaiseen hyvinvointiin, yksilövalmennukseen, rakennusautomaatioon ja logistiikkaan.

TreLab Oloni -järjestelmä koostuu langattomasta ympäristötietoja keräävästä monitoimilaitteesta, kiinteästä tai mobiilista Internet-yhteyden tarjoavasta päätelaitteesta sekä serveripilvestä. Kuvassa 1 on havainnollistus Oloni-järjestelmästä.



Kuva 1: TreLab Oloni-järjestelmä (lähde: TreLab Oy)

TreLab Olonin mittasuureita keräävä monitoimilaite, tagi, on halkaisijaltaan 40 mm ja paksuudeltaan 12 mm. Tagin teholähteenä on kertakäyttöinen 550 mAh:n litiumioniparisto, jolla on tarkoitus taata noin vuoden toiminta-aika [1].

Tagi kerää tietoa muun muassa läsnäolosta, ilmankosteudesta, ilmanpaineesta, lämpötilasta, asennosta, liikevoimasta ja liikesuunnasta. Tagilla voidaan myös mitata biometristä tietoa.

3 LAITTEISTO

Vähän tehoa kuluttavan sulautetun järjestelmän ohjelmistoa toteutettaessa laitteistolla on suurempi vaikutus ohjelmistosuunnittelussa kuin esimerkiksi tehokkaalle PC-tietokoneelle ohjelmistoa toteutettaessa. Toisin kuin nykyaikaisessa PC:ssä, sulautetussa järjestelmässä sekä ohjelmamuisti että ajon aikana käytettävä RAM-muisti ovat pieniä. Tämän johdosta sovelluksen tulee olla pieni ja käyttää mahdollisimman vähän RAM-muistia.

3.1 Mikrokontrolleri

Tämän työn suorittamiseen käytettävän monitoimilaiteversion mikrokontrolleri on Atmelin RISC-käskykantainen AT32UC3L0256. Kontrolleri edustaa Atmelin AVR-arkkitehtuurin 2006 julkistettua 32-bittistä perhettä. AVR-arkkitehtuuri on alkujaan norjalaisessa yliopistossa suunniteltu mikrokontrolleriarkkitehtuuri. Mikrokontrolleri, toisinaan myös mikro-ohjain, on laite, joka sisältää prosessorin lisäksi muita mikrotietokoneen tarvitsemia oheispiirejä [2]. Oheispiirejä ovat muun muassa erilaiset väylämuuntimet, muistit sekä A/D-muuntimet.

AT32UC3L-sarjan mikrokontrollerit käyttävät Atmelin picoPower-tekniikkaa, jonka ansiosta kontrollerin tehon kulutus on alimmillaan 174 µW/MHz. Ohjelmamuistia on käytössä 256 kilotavua ja SRAM-muistia 32 kilotavua [3]. Ohjelmamuistiin tulee mahtua sekä sovellus että bootloader.

AT32UC3L0256-kontrollerissa on 15 yleiskäyttöistä 32-bittistä rekisteriä. Kuten muissakin nykyaikaisissa mikrokontrollerissa, AT32UC3L0256-kontrollerissa on useille toiminnoille, kuten flash-muistin hallinnalle, oma keskitetty moduulinsa rekistereineen ja toimintoineen. [3]

3.2 BLE-moduuli

Monitoimilaitteen langaton viestintä on toteutettu BLE-protokollalla. BLE on Bluetooth 4.0-standardiin sisältyvä vähän tehoa käyttävä langaton viestintäprotokolla. Oloni-

monitoimilaitteessa käytetty BLE-moduuli on suomalaisen Bluegiga-yrityksen valmistama BLE112.

Moduulin perustana on yhdysvaltalaisen Texas Instruments -yrityksen CC2540-mikrokontrolleri. CC2540 sisältää mikrokontrollerin ja BLE-protokollan mukaisen fyysisen kerroksen komponentit. CC2540:n voidaan liittää sensoreita, ja se tukee monia erilaisia kommunikaatioväyliä. [4]

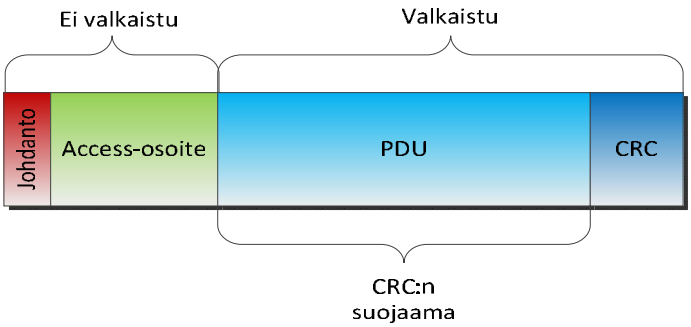
BLE-moduulin datansiirtoprotokollan fyysisen kerroksen ominaisuudet ovat taulukossa 1.

Taulukko 1: BLE-protokollan fyysisen kerroksen ominaisuudet [5]

Ominaisuus	Arvo
Taajuuskaista	2,4 GHz (2420 MHz - 2480 MHz)
Modulaatio	GFSK, 1 Mbps
Modulaatioindeksi	0,5
Kanavaväli	2 MHz
Mainostuskanavat	3
Datakanavat	37
Taajuushyppely	Adaptiivinen FHSS

Moduuli voidaan ohjelmoida joko Basic-ohjelmointikieltä muistuttavalla BGScript-skriptauskielellä tai Bluetooth Low Energy Profile Toolkitilla [5]. Bluetooth Low Energy Profile Toolkit on työkalukokoelma, jolla luodaan GATT-pohjaisia palveluita ja protokollia. GATT on lyhenne sanoista Generic Attribute profile, ja tarkoittaa spesifikaatiota, joka täsmentää kahden langattoman Bluetooth-laitteen kommunikaation yksityiskohdat.

Moduuli huolehtii BLE-standardin vaatimasta paketin muodostamisesta, datan valkaisusta sekä modulaatiosta. Kuvassa 2 on geneerisen BLE-paketin rakenne. Samaa rakennetta käytetään sekä mainostus- että datapaketeissa [5].



Kuva 2: BLE-paketin rakenne

Datapaketin alussa on johdanto, joka on aina joko 010101010 tai 101010101. Johdantoa seuraa Access-osoite. Mainostuspaketeille Access-osoite on aina 0x8E89BED6, kun taas datapaketit käyttävät sattumanvaraista, yhteydestä riippuvaa Access-osoitetta. PDU on viestistä riippuva dataosio, ja CRC on 24-bittinen tarkistussumma. [5]

Datan valkaisulla tarkoitetaan matemaattista prosessia, jossa algoritmille syötetty vektori muutetaan luonteeltaan valkoisen kohinan kaltaiseksi [6].

Mainostusviesti on mainostustilassa olevan BLE-laitteen lähettämä broadcast-viesti, jollaisia skannaustilassa oleva laite aktiivisesti kuuntelee. Mikäli skannaustilassa oleva laite niin pyytää, voi mainostustilassa oleva laite lähettää lisätietoa. [5].

Moduuli kommunikoi AT32UC3L0256-mikrokontrollerin kanssa USART-väylällä mikrokontrollerin suoraa muistinkäyttöä (DMA) hyödyntäen. Suora muistinkäyttö tekee tiedonsiirrosta nopeampaa.

DMA:n tehokkuus pohjautuu siihen, että datan siirtäminen tapahtuu ilman mikrokontrollerin osana olevan prosessorin osallistumista muistiin kirjoittamiseen. Dataa voidaan siirtää oheislaitteelta muistiin tai muistista oheislaitteelle. Muistit tai oheislaitteet voivat olla mikrokontrollerin sisäisiä tai ulkoisia. Kontrollerin sisäiset oheislaitemoduulit voivat olla esimerkiksi USART tai SPI. DMA:ta ohjaa mikrokontrollerilla PDCA (Peripheral DMA Controller). [3]

PDCA kommunikoi oheislaitteiden kanssa kättelyliittymien kautta. Oheislaite ilmoittaa PDCA:lle, kun se on valmis lähettämään tai vastaanottamaan dataa. PDCA ilmoittaa

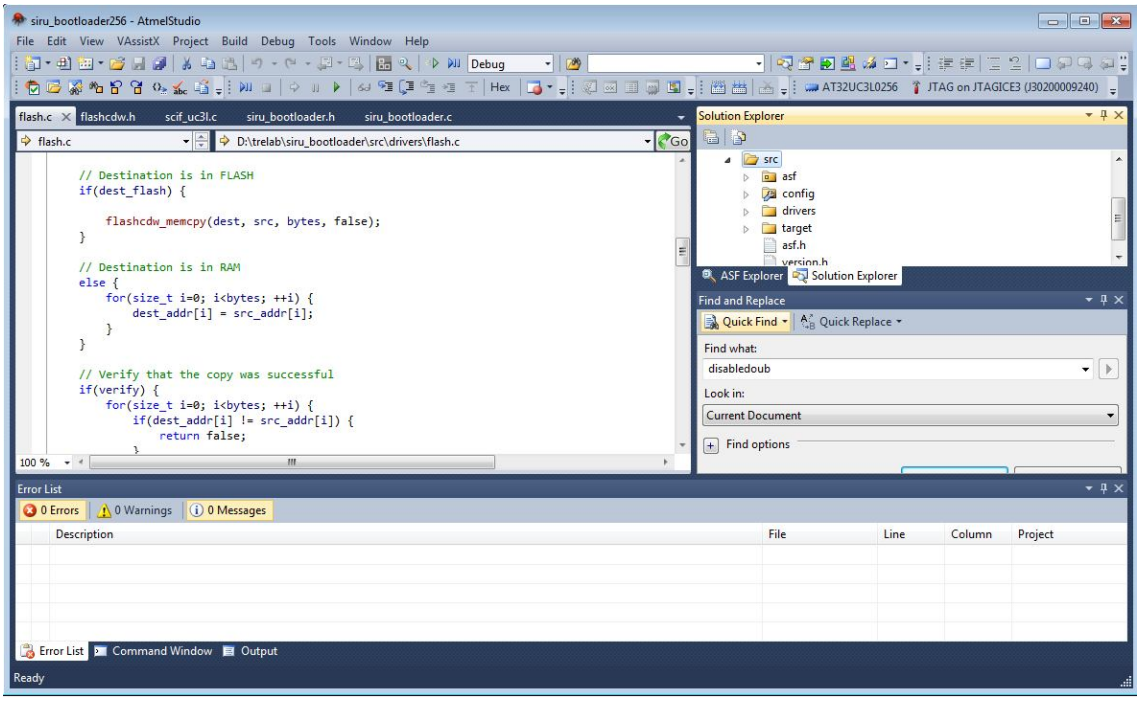
vastaanottaneensa pyynnön, kun tiedonsiirto on alkanut. Halutessaan käyttäjä voi ohjelmoida PDCA:n antamaan keskeytys, kun lähetyspuskuri on tyhjä tai vastaanottopuskuri on täynnä. [3]

4 OHJELMISTOTYÖKALUT

Avoimen lähdekoodin ohjelmistoprojektit ovat tuoneet kaikkien saataville paljon erilaisia ilmaisia integroituja kehitysympäristöjä ja versionhallintaohjelmia. Koska monien tällaisten suunnittelu- ja toteutusprosessissa käytettävien ohjelmistojen käytöstä ei veloiteta erikseen edes kaupallisissa projekteissa, kannattaa ohjelmistosuunnittelijan ainakin tutustua nykyisin saatavilla oleviin työkaluihin. Koska bootloader toteutettiin osana isompaa projektia, työn suorittamisen apuna käytettiin projektin- ja versionhallintaohjelmia jäljitettävyyden parantamiseksi ja dokumentoinnin tueksi.

4.1 Atmel Studio 6

Atmel Studio 6 on yhdysvaltalaisen Atmel-yrityksen tuottama ilmainen muttei vapaan lähdekoodin IDE, integroitu kehitysympäristö. Atmel Studio 6 yhdistää monta erillistä työkaluketjua yhdeksi kokonaisuudeksi. Kuvassa 3 on Atmel Studio 6:n työnäkymä.



Kuva 3: Atmel Studio 6 IDE

Atmel Studio 6:n tekstieditori käyttää hyödykseen Microsftin Visual Studio

-ohjelmistoa ja Whole Tomato Softwaren kehittämää Visual Assist X -älykästä tekstinkäsittelyä. Kehittynyt editori auttaa ohjelmakoodin kirjoittamisessa ja hallitsemisessa. Esimerkiksi hakutoiminto, joka löytää ja halutessa korvaa muulla sanalla kaikki valitun muuttujan tai funktion instanssit, on hyvin käytännöllinen. Automaattinen täydennys nopeuttaa työtä ja vähentää kirjoitusvirheitä.

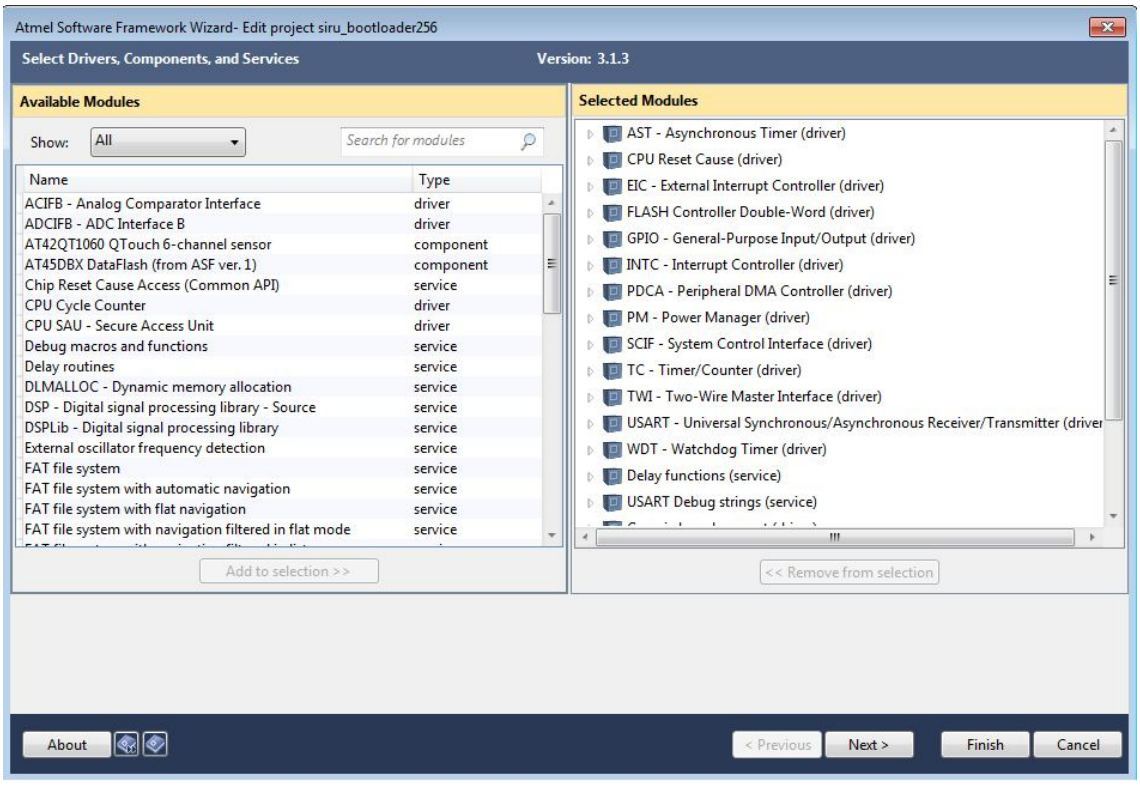
Kääntämiseen ja debuggaamiseen Atmel Studio 6 käyttää yhdysvaltalaisen voittoa tavoittelemattoman GNU-säätiön työkaluja. Debuggaustyökaluna on AVR-GDB. Debugatessa voidaan askeltaa C-kielistä lähdekoodia rivi kerrallaan ja lukea rekistereiden sekä SRAM:in sisältöä.

Kääntäjänä toimii AVR-GCC:n AVR32-arkkitehtuurille tehty versio.

GCC:n eduksi voidaan ilmaisuuden lisäksi mainita lähdekoodin avoimuus. Tämä tarkoittaa sitä, että kuka tahansa voi halutessaan saada nähtäväkseen ja muokattavakseen ohjelman lähdekoodin. Lähdekoodin vapauden ansiosta GCC-kääntäjäkokoelmasta on olemassa versioita monille mikrokontrollereille.

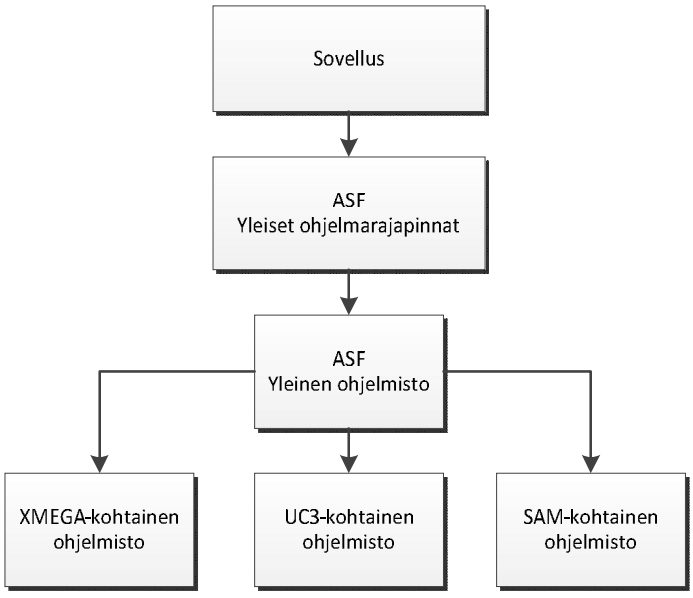
4.2 Atmel Software Framework

Atmel Software Framework helpottaa ylhäältä päin lähtevää suunnitteluprosessia yhdessä Atmel Studio 6:n kanssa. Sovellusten yhteiskäyttö mahdollistaa sovelluskehitysmallin, jossa merkittäviä osia ohjelmakoodista ei tarvitse kirjoittaa uudestaan joka mikrokontrollerivariantille. Tämä mahdollistaa sen, että sovelluskehittäjät voivat keskittyä matalan tason rajapintojen sijasta sovelluksen kirjoittamiseen. [7]



Kuva 4: Atmel Studio 6:n ASF Wizard

Kuvassa 4 on Atmel Studio 6:n ASF Wizard. ASF Wizardilla eri moduulien lisääminen projektiin onnistuu vaivattomasti. Moduuleita lisättäessä tulee kiinnittää huomiota lisensseihin: jotkut moduulit ovat GPL-lisensoituja, ja niiden käyttäminen asettaa vaatimuksia valmiin sovelluksen lähdekoodin avoimuuden suhteen.



Kuva 5: Tavallinen ASF-arkkitehtuurirakenne

Kuvassa 5 on esimerkki ASF-arkkitehtuuria käyttämällä useille eri mikrokontrollereille toteutetusta ohjelmistoarkkitehtuurista. Kuvassa 5 sovellus on ohjelmoijan oma sovellus, joka käyttää ASF:n yleisiä ohjelmarajapintoja. Ohjelmarajapinnat käyttävät Atmelin yleisiä rutiineja ja ohjelmia, joista kustakin on olemassa versiot eri mikrokontrollereille. [7]

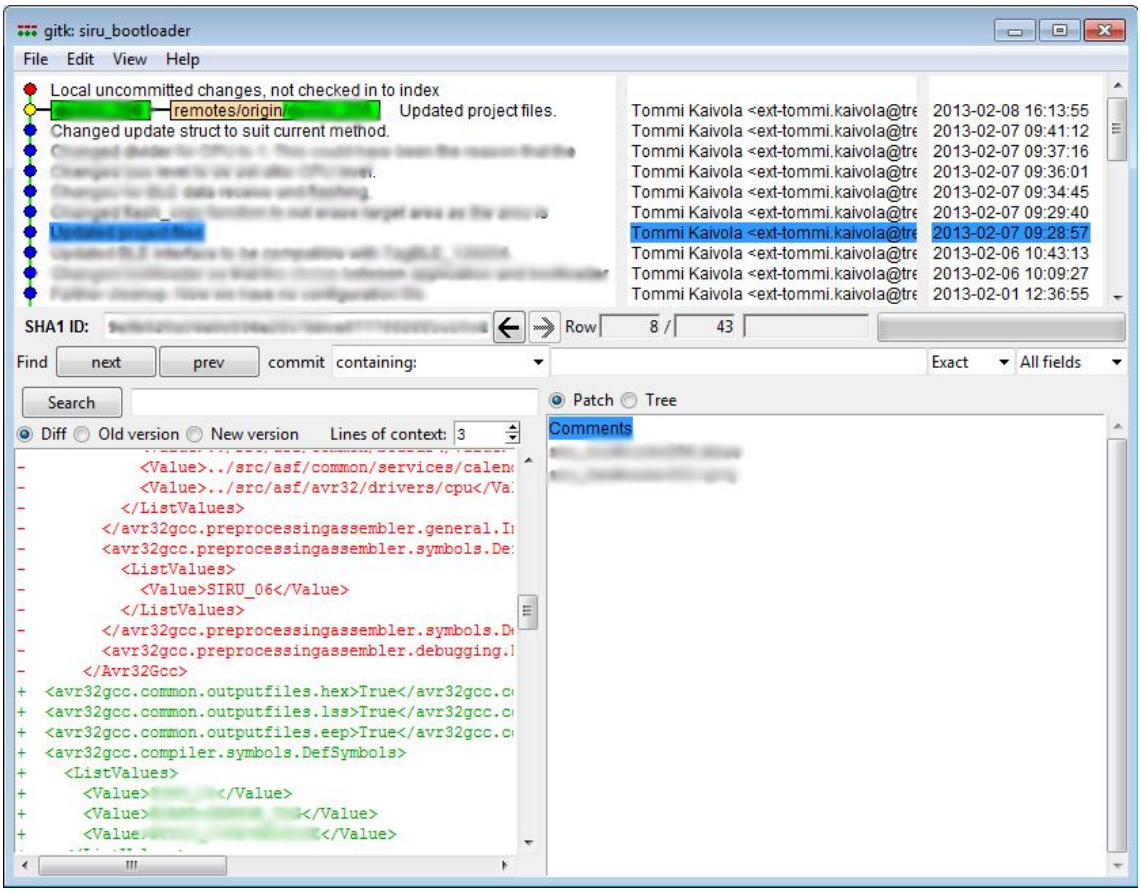
4.3 GIT

GIT on suomalaisen Linus Torvaldsin kehittämä versionhallintaohjelmisto. GIT kehitettiin alun alkaen Linux-käyttöjärjestelmäytimen kehityksen versionhallintaan, kun aiemmin käytössä ollut BitKeeper-ohjelmisto muuttui lisenssiltään rajoittuneemmaksi [8].

GIT mainostaa olevansa ennen kaikkea sisällönhallintaohjelma. GITin peruskäsite on ”repository”. Repository on yksinkertainen tietokanta, joka sisältää kaiken tarvittavan tiedon projektin historian ja revisioiden hallitsemiseen [8].

GIT laskee jokaiselle seurattavalle tiedostolle SHA-1 -hashin. SHA-1 on kryptografiassa käytettävä tiivistefunktio, joka laskee mille tahansa syötteelle 160 merkin hashin [9]. Mikäli kahden tiedoston SHA-1 -hashin arvot eroavat, on niiden sisältö erilainen. Tämä johtaa siihen, että mikäli kaksi kehittäjää kehittää toisistaan riippumatta täysin identtiset tiedostot, on niiden SHA-hashit samat [8].

GITin käyttöön on olemassa erilaisia liittymiä. GIT GUI on graafinen käyttöliittymä. GIT Bash taas tuo Windows-koneeseen GNU/Linux-ympäristöstä tutun BASH-shellin ja minimaalisen Unix-yhteensopivan komentokannan. On käyttäjän mieltymysten varassa, mitä liittymää käytetään. Kuvassa 6 on kuva gitk-ohjelmasta.



Kuva 6: Gitk-ohjelman näkymä

Kuvassa 6 ylimpänä on lista commiteista. Commit on tietue, joka sisältää tiedon repositoryyn tehdyistä muutoksista [8]. Ylhäällä keskellä on commitin tekijän nimi ja sähköpostiosoite. Oikealla ylhäällä on commitin suorittamisen ajankohta. Alhaalla oikealla on lista commitissa muutetuista tiedostoista. Alhaalla vasemmalla on kaikki muutokset. Poistetut rivit ovat punaisella ja lisätyt vihreällä. Mikäli riviä ei ole muutettu, on se mustalla tekstillä. Muuttamattomia rivejä on näkyvillä vain commitissa muutettujen rivien ympäriltä ja välistä.

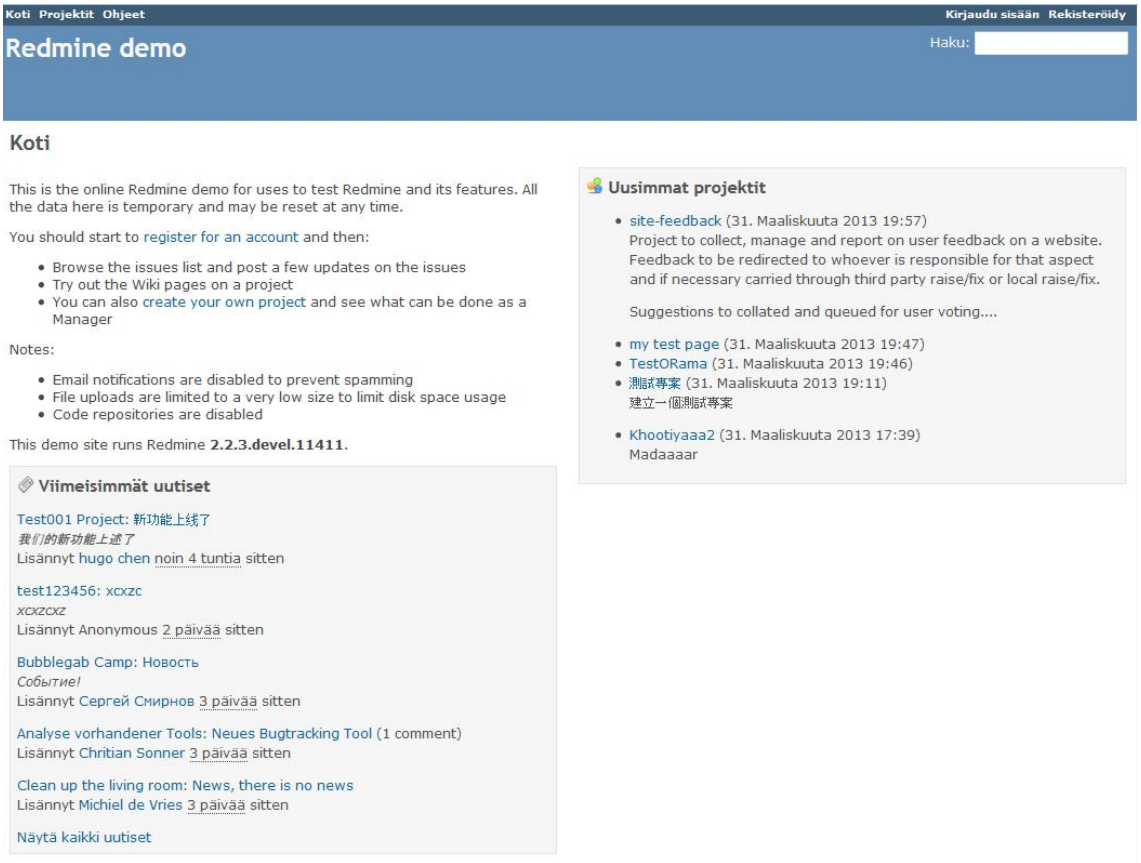
Yksittäisiä commiteja voidaan poistaa jälkeenpäin, mikäli niissä tehdyt muutokset osoittautuvat turmiollisiksi. Commitin yhteydessä tallentuu myös tieto commitin tekijästä.

4.4 Redmine

Ohjelmistoprojekteissa esiin nousee usein projektin hallinnan vaikeus. Viikottaiset tehtävät saattavat vaihdella, ja useita havaintoja ja ratkaisuja on vaikeaa muistaa. Ohjelmiston toteuttajan elämän helpottamiseksi on olemassa projektinhallintaohjelmia. Yksi niis-

tä on Redmine, ilmainen ja vapaan lähdekoodin Ruby On Rails -arkkitehtuurilla toteutettu projektinhallinta- ja buginraportointiohjelmisto [10].

Redmine on suunniteltu käytettäväksi SCRUM-pohjaisen ohjelmistoprojektin hallintaan. SCRUM on yhdenlainen ketterä kehitysmenetelmä.



Kuva 7: Redmine-näkymä (lähde: redmine.org)

Redmine-projektiin voidaan lisätä erilaisia aliprojekteja ja kuhunkin projektiin liittyviä taskeja ja bugiraportteja. Sovelluskehittäjä sitten työn edetessä päivittää kunkin hänelle osoitetun taskin tilaa ja halutessaan lisää kommentteja, kysymyksiä ja kuvia. Projektia ja sen taskeja voidaan tarkastella erilaisissa näkymissä, kuten esimerkiksi kalenterinäkymässä tai Ganttin kaaviossa.

Redmine voidaan myös integroida GIT:iin, jolloin taskeihin voidaan lisätä GIT-commitin ID. Näin taskien suorittamisen ja bugien korjaamisen voi tarkistaa suoraan Redmine-näkymästä ilman erillistä GIT-repositoryn selaamista.

4.5 Doxygen

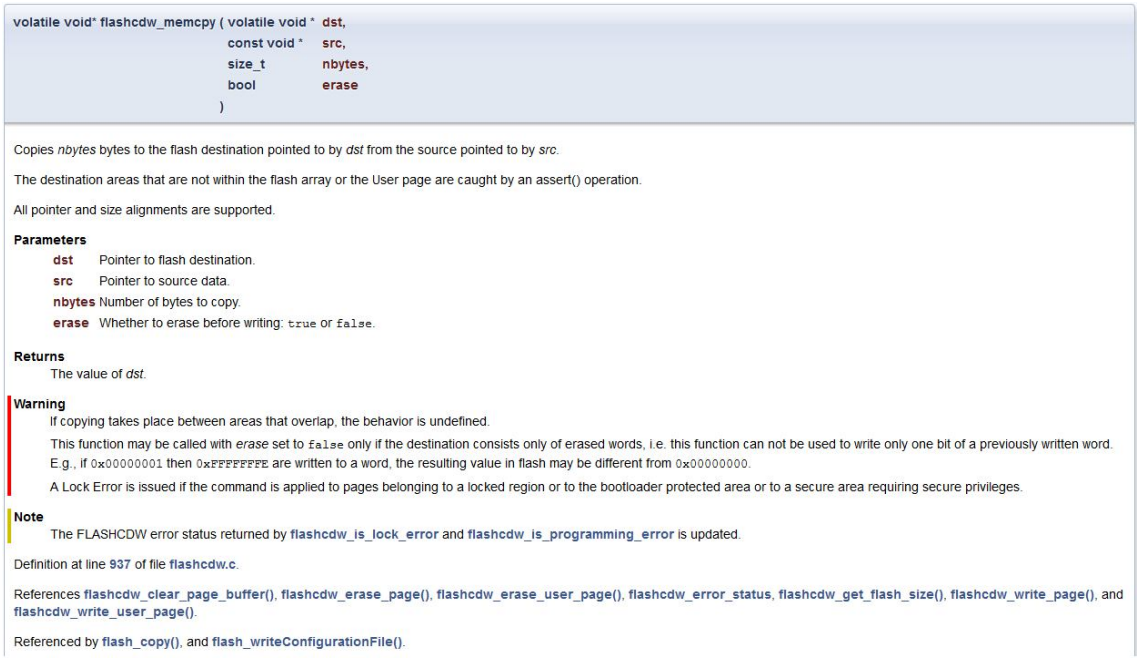
Oli sitten kyseessä ohjelmisto- tai elektroniikkasuunnittelu, dokumentointi on sekä tärkein että eniten laiminlyöty osa projekteja. Tämä johtaa usein turhaan työhön ja ylläpidettävyyden hankaluuteen. Ohjelmoinnissa dokumentointia on syytä harjoittaa edes ohjelmakoodin sekaan kirjoitettavilla kommenteilla. Usein myös erillinen dokumentaatio on tarpeen.

Doxygen on ilmainen ja vapaan lähdekoodin dokumentointityökalu. Doxygen kehittää dokumentoinnin kutsuhierarkioineen ohjelmakoodin kommentteihin sijoitettujen käskymerkkien perusteella. Dokumentaatio voi olla esimerkiksi PDF-dokumentti tai HTML-sivusto.

Doxygen kykenee myös kehittämään LaTeX-tyyppisistä kaavamerkinnöistä matemaattisia kaavoja. Tämä on avuksi erityisesti runsaasti algoritmeja sisältävissä ohjelmissa. Esimerkki Doxygen-tyyliin kommentoidusta funktion esittelystä:

```
/*! \brief Copies \a nbytes bytes to the flash destination pointed to by \a dst
 *         from the source pointed to by \a src.
 *
 * The destination areas that are not within the flash
 * array or the User page are caught by an assert() operation.
 *
 * All pointer and size alignments are supported.
 *
 * \param dst Pointer to flash destination.
 * \param src Pointer to source data.
 * \param nbytes Number of bytes to copy.
 * \param erase Whether to erase before writing: \c true or \c false.
 *
 * \return The value of \a dst.
 *
 * \warning If copying takes place between areas that overlap, the behavior is
 *          undefined.
 *
 * \warning This function may be called with \a erase set to \c false only if
 *          the destination consists only of erased words, i.e. this function
 *          can not be used to write only one bit of a previously written word.
 *          E.g., if \c 0x00000001 then \c 0xFFFFFFFE are written to a word, the
 *          resulting value in flash may be different from \c 0x00000000.
 *
 * \warning A Lock Error is issued if the command is applied to pages belonging
 *          to a locked region or to the bootloader protected area or to a secure
 *          area requiring secure privileges.
 *
 * \note The FLASHCDW error status returned by \ref flashcdw_is_lock_error and
 *       \ref flashcdw_is_programming_error is updated.
 */
extern volatile void *flashcdw_memcpy(volatile void *dst, const void *src, size_t
nbytes, bool erase);
```

Esimerkki on ASF:n AT32-arkkitehtuurin flash-muistin hallintaan käytetyn flashcdw-moduulin muistin kopiointifunktion esittelystä. Esimerkin kaltainen kommentointi tuottaa Doxygenillä kuvan 8 kaltaisen HTML-dokumentin.



volatile void* flashcdw_memcpy (volatile void * dst,
const void * src,
size_t nbytes,
bool erase
)

Copies *nbytes* bytes to the flash destination pointed to by *dst* from the source pointed to by *src*.

The destination areas that are not within the flash array or the User page are caught by an assert() operation.

All pointer and size alignments are supported.

Parameters

dst Pointer to flash destination.
src Pointer to source data.
nbytes Number of bytes to copy.
erase Whether to erase before writing: `true` or `false`.

Returns

The value of *dst*.

Warning

If copying takes place between areas that overlap, the behavior is undefined.

This function may be called with *erase* set to `false` only if the destination consists only of erased words, i.e. this function can not be used to write only one bit of a previously written word. E.g., if `0x00000001` then `0xFFFFFFFE` are written to a word, the resulting value in flash may be different from `0x00000000`.

A Lock Error is issued if the command is applied to pages belonging to a locked region or to the bootloader protected area or to a secure area requiring secure privileges.

Note

The FLASHCDW error status returned by **flashcdw_is_lock_error** and **flashcdw_is_programming_error** is updated.

Definition at line **937** of file **flashcdw.c**.

References **flashcdw_clear_page_buffer()**, **flashcdw_erase_page()**, **flashcdw_erase_user_page()**, **flashcdw_error_status**, **flashcdw_get_flash_size()**, **flashcdw_write_page()**, and **flashcdw_write_user_page()**.

Referenced by **flash_copy()**, and **flash_writeConfigurationFile()**.

Kuva 8: Doxygenin kehittämä HTML-dokumentaatio

Kuvasta 8 voidaan nähdä, että Doxygen kertoo myös, mitä funktioita dokumentoitu funktio kutsuu ja mitkä funktiot kutsuvat dokumentoitua funktiota.

Koodin ulkoisen dokumentaation tarve on aina tapauskohtaista: monet ohjelmoijat lukevat mieluummin itse ohjelmakoodia, mutta monia moduuleita ja useita abstraktiorajapintoja sisältävissä projekteissa on usein helpompi lukea esimerkiksi Doxygen-dokumentaatiota. Dokumentaation tuottaminen ei vaadi oikeastaan muuta kuin erityisen kommenttien muotoilun, joten se kyllä kannattaa.

5 TOTEUTUS

Työ toteutettiin sovellusohjelman suunnittelu- ja toteutustyön ohella. Koska bootloaderin GIT-repositoryyn saa yhteyden SSH:lla mistä tahansa, voitiin työtä tehdä sekä kotitoimistolla että työpaikalla. Tämän kaltainen nykyaikainen etätyöskentely toimii hyvin ohjelmistosuunnittelun kaltaisella alalla, jossa työkalut ovat helposti kotiin hankittavissa, eikä prosessi vaadi kaikkien toteuttajien jatkuvaa fyysistä läsnäoloa.

5.1 Työprosessi

Ylläpidettävyyden helpottamiseksi bootloader haluttiin toteuttaa eri projektissa kuin varsinainen sovellusohjelma. Bootloaderia varten perustettiin erillinen GIT-repository ja lähdekoodipuu. Osa lähdekoodipuusta kopioitiin sovellusohjelman lähdekoodeista. Varsinainen bootloader-toiminnallisuus toteutettiin erikseen.

Sovellusohjelma oli työn suorituksen aikana nopean tuotekehityksen alla. Sen johdosta bootloaderin lähdekoodipuu tuli päivittää esimerkiksi BLE-moduulin ohjelmakoodin osalta aina kun sovellusta oli merkittävästi muutettu.

Bootloaderin toimintaa kehitettiin rinnan sovelluksen kanssa. Serveripuolen tuen kehittämisestä vastasi serveripuolen toiminnallisuutta toteuttava tiimin osa.

5.2 Testaus

Bootloaderin testaaminen oli työlästä. Koska sovellusohjelma haluttiin kehittää erillisessä projektissa, tuli bootloaderin ja applikaation yhteistoimintaa testaava ohjelmakoodi kääntää kummallakin sovelluksella erikseen, minkä jälkeen erilliset ohjelmakoodit poimittiin ihex-tiedostoista ja liitettiin yhteen notepad++-ohjelmalla. Saatu ihex-tiedosto ladattiin mikrokontrolleriin. Tällä tavoin saatiin selville, toimiiko hyppy bootloaderista applikaatioon. Myös bootloaderissa tapahtuva kellojen ja GPIO-moduulin alustuksen toimivuus saatiin varmistettua.

Bootloaderin toimintaa erikseen testattaessa käytettiin perinteisempiä metodeja. Koska käytössä oli testausasetelma, jossa mikrokontrollerin UART-väylästä sai kaapelin PC:lle, voitiin käyttää debug-tulosteita ratkaisevissa kohdissa ohjelmaa. Atmel Studio 6:n debug-työkaluja käytettiin myös niissä alustuksen vaiheissa, joissa debug-UART:ia ei ollut vielä alustettu.

Flash-muistiin kirjoittamisen toimivuus testattiin kirjoittamalla selkeää heksadesimaalilukua (esimerkiksi 0xdeadbeef) koko ohjelmamuistin bootloaderin ulkopuoleiselle alueelle ja lukemalla mikrokontrollerin ohjelmamuisti. Mikäli luettu ohjelmamuisti vastasi tavoiteltua, oli kirjoitus onnistunut. Tällä tavoin testattiin myös useiden puskureiden käyttämistä ja pointtereiden vaihtoa. Pointterin vaihto on menetelmä, jossa taulukoiden lisäksi käytetään osoitinmuuttujaa, joka laitetaan aina taulukon täytyttyä osoittamaan täyttä taulukkoa. Näin kirjoitusfunktiolle voidaan lähettää aina sama muuttuja, ja taulukoiden täyttymisestä vastaava funktio huolehtii siitä, mitä taulukkoa kulloinkin täytetään.

Varsinaista langatonta ohjelmistopäivitystä ei serveripuolen tuen keskeneräisyyden johdosta saatu tämän opinnäytetyön kirjallisen osan valmistumiseen mennessä testattua, mutta itsessään bootloader on toimiva, ja serveripuolen toteutuksen valmistumisen jälkeen suoritettavan integraatiotestaamisen ei uskota kestävän kauaa.

6 BOOTLOADER

Boottaaminen on termi, jonka lähes jokainen työssään tai vapaa-ajallaan tietokonetta käyttänyt on todennäköisesti joskus kuullut. Termiä käytetään usein PC-tietokoneen tai sulautetun järjestelmän käynnistyessään suorittamasta sovellusohjelman lataamisesta ja oheislaitteiden alustamisesta.

Ohjelmaa, joka suorittaa nämä alkurutiinit kutsutaan bootloaderiksi. Nimitys tulee ilmaisusta, jossa henkilö nostaa itsensä ylös omista saappaanrakseistaan, englanniksi bootstraps. [11]

Varhaiset tietokonejärjestelmät käyttivät erilaisia ad-hoc-menetelmiä sovellusohjelman lataamiseen. ROM-muistien keksimisen myötä laitteet voitiin toimittaa asiakkaille tehtailta asiakkaan sovelluskoodin ohjelmoimisen mahdollistava ohjelmisto asennettuna. [11]

Esimerkiksi nykyaikaiset PC-tietokoneet sisältävät BIOSin, joka on eräänlainen bootloader. BIOS suorittaa käynnistyksen yhteydessä itsediagnostiikan ja lataa käynnistyslevyltä 512 tavun mittaisen boot sectorin RAM-muistiin ja aloittaa sen suorittamisen.

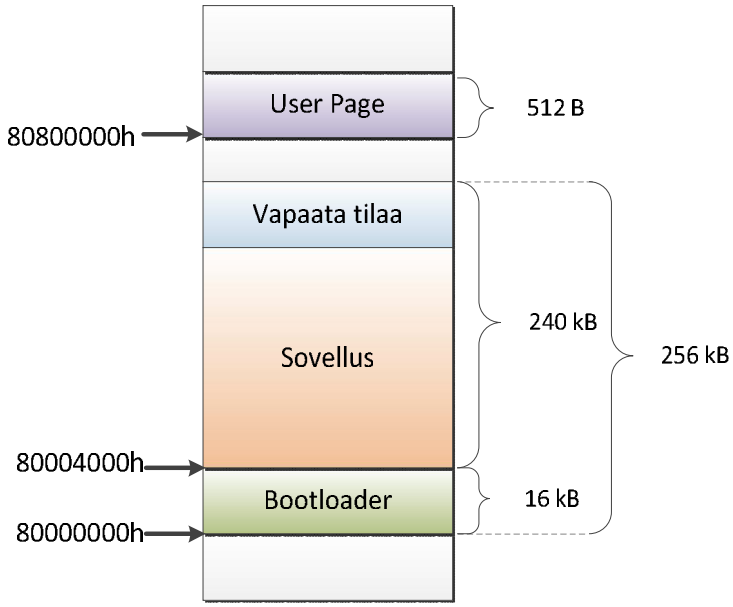
Sulautetuista järjestelmistä voidaan esimerkkinä käyttää Arduino Nano -alustaa jonka bootloader mahdollistaa sovellusohjelman lataamisen USB-väylää käyttäen. Näin satunnaisen ohjelmoijan ei tarvitse ostaa kallista JTAG-ohjelmointilaitetta.

6.1 Toimintalogiikka

Kun sovellusohjelman käydessä BLE-moduulilta saadaan tieto tulevasta ohjelmistopäivityksestä, suoritetaan sarja toimintoja, joiden tarkoituksena on kertoa bootloaderille tulevasta ohjelmapäivityksestä. Tämän jälkeen laite resetoidaan.

Ohjelmakoodin suoritus aloitetaan aina reset-vektorista. AT32UC3L0256:n reset-vektori on osoitteessa 0x80000000 [3]. Koska bootloader halutaan suorittaa joka käynnistyksen yhteydessä, on bootloader järkevintä sijoittaa muistialueen alkuun. Vaihtoeh-

tona olisi sijoittaa reset-vektoriin ehdoton hyppykäsky bootloaderin alkuun. Laitteen muistirakenne on kuvassa 6.



Kuva 9: Bootloaderin, sovelluksen ja user pagen sijainti muistissa

AT32UC3L0256-kontrollerin flash-ohjainmoduuli sisältää toiminnot ohjelmamuistin suojaamiseen. Suojaamisella tarkoitetaan kirjoitus- ja pyyhintätoimintojen estämistä halutulla alueella. Bootloaderin suojaamiseen on oma rekisterinsä, jonne kirjoitetulla arvolla voidaan valita suojattavan muistialueen koko kahdeksasta eri kokovaihtoehdosta. Kun suojaus on asetettu voidaan suojatun alueen ohjelmamuistia muuttaa vain JTAG-liitännän kautta suoritettavan pyyhinnän jälkeen. [3].

Bootloader suorittaa ensin perusalustuksen (muun muassa kello, GPIO-moduuli ja flash-ohjain) ja tarkistaa, ovatko ohjelmistopäivityksen saapumista tarkoittavat ehdot täyttyneet. Mikäli ehdot täyttyvät, alustetaan ja käynnistetään BLE-moduuli ja odotetaan serveriltä tietoa ohjelmistopäivityksen aloittamisesta.

Mikäli ohjelmistopäivitys suoritetaan, siirrytään ajamaan pääsilmukkaa ja annetaan ohjelmakoodin päivittyä BLE-moduulin USART-keskeytyspalveluohjelmassa. Mikäli ehdot ovat täyttyneet jonkin virheen seurauksena, ilmoitetaan tästä serverille ja suoritetaan virheenkäsittelyrutiinit.

Pääsilmukassa suoritetaan mahdolliset bottom half -taskit ja kirjoitetaan säännöllisesti watchdog-rekisteriin watchdog-resetin estämiseksi. Bottom half -taskit ovat keskeytyspalveluissa tapahtuvien toimintojen perusteella suoritettavaksi määriteltyjä ei-aikakriittisiä toimintoja, jotka suoritetaan pääsilmukassa jotta keskeytyspalvelut pysyisivät mahdollisimman nopeina. Pääsilmukan ohjelmakoodi näyttää seuraavalta:

```
/* updating -> loop and do bottom half tasks and let the
            BLE data receive function do the job */
while(1) {

            print_dbg("loop forever\r\n");

            /* Run BLE bottom half tasks */
            BLE_bottomHalf();

            /* clear watchdog */
            wdt_clear();

            print_dbg("tickle the dog\r\n");

            /** @todo decide whether it would be necessary to sleep
            here (IDLE sleep state perhaps) */
}
```

Debug-tulosteita käytettiin kehitysvaiheessa ohjelman toiminnan analysoimiseen.

6.2 Päivityksen vastaanotto ja kirjoitus

AT32UC3L0256-mikrokontrollerin flash-muisti on NAND-tyyppinen. NAND-flashilla on tietty kirjoitusgranulariteetti [12]. AT32UC3L0256-mikrokontrollerin flash-ohjelmamuistin sivun koko, ja näin ollen myös kirjoitusgranulariteetti, on 512 tavua [3].

Oloni-sovelluksen BLE-protokollan data-PDU:n hyötykuorma on 250 tavua. Koska flash-sivun koko on 512 tavua, täytyy vastaanotettu tieto kerätä puskuriin ja kirjoittaa sivu kerrallaan. Aina kun sovellusohjelmasta on saapunut sivun kokoinen osa, voidaan sivu kirjoittaa flash-muistiin.

Kilpailuteknisistä syistä tässä dokumentissa ei kerrota enempää päivityksen tarkemmasta toiminnasta ja käytetyistä salausmenetelmistä.

Kun ohjelmistopäivitys on vastaanotettu kokonaisuudessaan, käynnistetään laite uudelleen. Bootloader suorittaa jälleen alustukset ja toteaa, että laitteessa on validi sovellusohjelma. Tämän jälkeen hypätään sovellusohjelmaan.

Sovellusohjelman ensimmäisen käskyn on sijaittava aina samassa osoitteessa, jotta bootloaderista osattaisiin hypätä siihen. Hyppykäsky on toteutettu funktiopointterilla. Funktiopointterin tyyppimäärittely näyttää tältä:

```
/* Function pointer type definition for jump to firmware start address */
typedef void (*fPtr_t)(void)__attribute__((noreturn));
```

Määrittelyn loppuun lisätty attribuutti noreturn tarkoittaa sitä, että kyseisestä aliohjelmakutsusta ei palata, ja näin ollen kääntäjän ei tarvitse lisätä paluuseen liittyviä rutiineja käännöstulokseen. Tyyppimääritellyn muuttujan nimi fPtr on lyhenne sanoista "function pointer". Määrittelyn loppuun lisätty "_t" on sovittu merkintä, jolla erotetaan tyyppimääritellyt muuttujat.

Esitellään tyyppimääritelty muuttuja ja annetaan sille absoluuttinen osoitearvo:

```
fPtr_t start_fw = (fPtr_t) BOOT_PROG_START;
```

Funktiota kutsutaan normaalisti:

```
start_fw();
```

Kääntäjä sijoittaa funktiokutsun tilalle ehdottoman hypyn sovellusohjelman ensimmäiseen käskyyn. Sovellusohjelma pystyttää oman ajonaikaisen ympäristönsä ja jatkaa suorittamistaan aina virran katkeamiseen tai seuraavaan ohjelmistopäivitykseen saakka.

7 YHTEENVETO

Näennäisen yksinkertaisenkin ohjelmiston toteuttaminen muuttaa luonnettaan, kun se toteutetaan osana suurempaa projektia. Tuotekehitysvaiheessa oleva laitteisto tuo myös lisähaasteita.

Myös osana suurempaa projektia toimiminen version- ja projektinhallitsemisineen lisää projektin kompleksisuutta. Muun muassa ylläpidettävyys, luotettavuus ja yhteistoiminta muiden ohjelmistokomponenttien kanssa tuovat kaikki oman lisäulottuvuutensa muuten suoraviivaiseen toteutukseen.

Yhteensopivuutta serverin sovelluksen kanssa ei saatu testattua tämän dokumentin valmistumiseen mennessä, mutta bootloaderin voidaan olettaa olevan toimiva.

Työn suurimpana antina voidaan pitää sekä sujuvan ohjelmointirutiinin syntymistä että projektin ja tiimin osana toimimista. On täysin eri asia kirjoitella ohjelmistoja tai tehdä harraste-elektroniikkaa itsekseen kotona kuin toimia osana tuotekehitysprosessia tekevää tiimiä.

Loppujen lopuksi ohjelmoinnista tai elektroniikasta ei tullut opittua paljoakaan uutta, mutta kummankin suunnittelusta ja toteuttamisesta osana dynaamista kehitystiimiä tuli.

LÄHTEET

[1] TreLab Oy, ”TreLab Data Sheet,” Tampere, 2013.

[2] J. Koskinen, Mikrotietokonetekniikka; sulautetut järjestelmät, 2. painos toim., Keuruu: Otava, 2006.

[3] Atmel, ”32-bit Atmel AVR Microcontroller,” 2012. [Verkkolähde] http://www.atmel.com/Images/doc32145.pdf. [Haettu 11.3.2013].

[4] Texas Instruments, ”2.4-GHz Bluetooth® low energy System On Chip,” 12 2012. [Verkkolähde] http://www.ti.com/lit/ds/symlink/cc2540.pdf. [Haettu 11.3.2013].

[5] Bluegiga Technologies, ”Bluegiga Bluetooth Smart Software,” 2012.

[6] Wikipedia, ”Whitening transformation,” The Wikipedia Foundation, [Verkkolähde] http://en.wikipedia.org/wiki/Whitening_transformation. [Haettu 18.3.2013].

[7] J. Bethold, ”Next-Generation IDE: Maximizing IP Reuse,” Atmel Corporation, 2012. [Verkkolähde] http://www.atmel.fi/Images/WP_Studio6_July2012_FINAL.pdf. [Haettu 12.03.2013].

[8] J. Loeliger, Version Control with GIT, Yhdysvallat: O'Reilly Media Inc, 2009.

[9] Wikipedia, ”SHA,” The Wikipedia Foundation, [Verkkolähde] http://fi.wikipedia.org/wiki/SHA-1. [Haettu 13.03.2013].

[10] Wikipedia, ”Redmine,” The Wikipedia Foundation, [Verkkolähde] http://en.wikipedia.org/wiki/Redmine. [Haettu 12.03.2013].

[11] Wikipedia, ”Booting,” The Wikipedia Foundation, [Verkkolähde] http://en.wikipedia.org/wiki/Booting. [Haettu 12.3.2013].

[12] Wikipedia, ”Flash Memory,” The Wikipedia Foundation, [Verkkolähde] http://en.wikipedia.org/wiki/Flash_memory#NAND_flash. [Haettu 11.3.2013].

[13] Atmel, ”AVR32,” 2011. [Verkkolähde] http://www.atmel.com/Images/doc32000.pdf. [Haettu 13.3.2013].