Tomi Lehto

Sulautetun Linux-järjestelmän käynnistys-vaiheen kehitystyö

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

30.9.2013

Metropolia

Tekijä Otsikko Sivumäärä Aika	Tomi Lehto Sulautetun Linux-järjestelmän käynnistysvaiheen kehitystyö 47 sivua + 2 liitettä 30.9.2013
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	sulautettu tietotekniikka
Ohjaaja(t)	yliopettaja Markku Nuutinen tuotekehityspäällikkö Jouni Meriläinen

Insinöörityön tavoitteena oli parantaa verkostoautomaatiossa käytettävän prosessorikortin päivitys- ja käynnistystoimintoja. Laitteen Linux-käyttöjärjestelmän etäpäivittäminen tuli saada turvalliseksi niin, etteivät päivityksen mahdolliset virhetilanteet saisi laitetta toimimattomaan tilaan vaan laite toipuisi virheistä ja pysyisi toimintakunnossa.

Työssä perehdyttiin sulautetuissa järjestelmissä yleisesti käytetyn flash-muistin ominaisuuksiin ja käyttövaatimuksiin. Lisäksi tutkittiin, miten flash-muistin käyttö sulautetuissa järjestelmissä eroaa muiden flash-muistiin perustuvien massamuistien, kuten USB-muistien, käytöstä.

Työssä suoritinkortin käynnistysominaisuuksia muutettiin siten, että käynnistymisessä tarvittaville tiedostoille luotiin kaksi erillistä käynnistysosiota, jotta voidaan tarvittaessa palata vanhaan, toimivaan kokoonpanoon, jos päivityksessä tapahtuu virhe. Erillinen käynnistysosio myös nopeuttaa järjestelmän käynnistymistä huomattavasti. Järjestelmän U-Boot-käynnistyslataajaan lisättiin toimintoja, jonka avulla järjestelmä saatiin automaattisesti valitsemaan edellinen käynnistyskokoonpano, jos käynnistyminen uusien, päivitettyjen tiedostojen kanssa epäonnistuu.

Työssä kehitettyjen toimintojen ja ominaisuuksien avulla järjestelmän päivitysominaisuuksiin saatiin haluttu toiminnallisuus ja työssä tehdyt muutokset on otettu käyttöön laitteessa.

Avainsanat	Linux, sulautettu, flash-muisti, tiedostojärjestelmä, U-Boot

Author Title Number of Pages Date	Tomi Lehto Development of embedded Linux system startup phase 47 pages + 2 appendices 30 September 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Embedded Engineering
Instructor(s)	Markku Nuutinen, Principal Lecturer Jouni Meriläinen, Product Development Manager

The aim of this thesis was to enhance the upgrade and startup features of a processor board used in energy distribution network systems. Safe remote upgrades of the Linux operating system on the device were important. The main goal was to make sure that the system would be able to recover from possible error situations during the upgrading process, instead of becoming inoperative.

The work involved studying typical features of a flash memory often used in embedded systems, and also what needs to be considered when using this type of a memory. The differences in flash memory usage between embedded systems and flash-based mass memory devices, such as USB memories, were studied.

During the process, the startup features of the processor board were changed. Two partitions for startup files were created to make it possible to fall back using old files in case the new, upgraded startup files were corrupt or unusable. Using a separate partition for startup files also made the startup process considerably faster. New functionalities were made to U-Boot, the boot loader used on the system, to have it automatically select previous startup configuration in the case of an error situation.

With the new features and functionality developed, the wanted enhancements were achieved and they are now in use in production devices.

Keywords	Linux, embedded, flash memory, file system, U-Boot

Sisällys

Lyhenteet

Bash — Bourne Again Shell. Komentotulkki, jota käytetään Unix-käyttöjärjestelmissä.

Block device — Lohkolaite. Unix-käyttöjärjestelmän laitetyyppi, jota käsitellään lohko kerrallaan, kuten esimerkiksi kiintolevy-, cd- tai dvd-asema.

BSP — Board Support Package. Sulautettujen järjestelmien kehitykseen tarkoitettu, tiettyä suoritinta ja emokorttia varten tehty ohjelmisto- ja käännösympäristö. Se sisältää usein kääntäjän ja käyttöjärjestelmäajurit kyseiselle laitteistolle.

CFI — Common Flash Memory Interface. Flash-muistin ja sitä käyttävän laitteen tunnistamis- ja kättelyprotokollan standardi.

Character device — Merkkilaite. Unix-käyttöjärjestelmän laitetyyppi, jonka kanssa kommunikoidaan lukemalla ja kirjoittamalla merkki kerrallaan, kuten esimerkiksi sarjaporttia tai näppäimistöä.

CRC — Cyclic Redundancy Check. Tarkistussumman laskentapa.

DT — Device Tree. Open Firmwaressa (avoin laiteohjelma) käytetty laitteistoa kuvaava puurakenne.

EEPROM — Electrically Erasable Programmable Read Only Memory. Muistityyppi haihtumattomalle pysyväismuistille.

Erase block — NAND-flash-muistin lohko. Muistin pienin yhtenäinen alue, joka voidaan kerrallaan tyhjentää.

ECC Error Correction Code. Virheenkorjaustekniikka, jossa datan lisäksi tallennetulla koodilla voidaan havaita ja korjata datassa olevia virheitä.

FDT Flattened Device Tree. Teknologia, jossa käynnistyksen yhteydessä tarvittavat tiedot laitteistosta listataan tiedostoon Linux-ydintä varten.

FTL Flash Translation Layer. Flash-muistipiirejä käyttävissä massamuisteissa oleva muunnostaso, joka tulkitsee käyttöjärjestelmän lohkopohjaiset kirjoitus- ja lukukäskyt NAND-flashille sopivaan muotoon. FTL voi olla toteutettu ohjelmallisesti massamuistin mikro-ohjaimella tai laitteeseen tallennetulla ohjelmalla.

GC Garbage Collection. Siivous, kuten esimerkiksi JFFS2-tiedostojärjestelmässä lohkojen vapautus uudelleen käytettäväksi etsimällä ja vapauttamalla vanhentuneita merkintöjä (nodes) uudelleen käytettäviksi.

GPIO General Purpose Input/Output. Yleiskäyttöinen portti mikropiirissä, joka voidaan ohjelmallisesti asettaa joko tulo- tai lähtöportiksi.

IMMR Internal Memory Mapped Registers. PowerPC-suorittimen sisäiset rekisterit, jotka näkyvät suorittimen muistiavaruudessa.

JFFS2 Journaling Flash File System version 2. Lokiin perustuva flash-muisteille tarkoitettu tiedostojärjestelmä.

JTAG Joint Test Action Group. Mikropiirien ohjelmointiin ja testaukseen kehitetty portti.

MAC-osoite Media Access Control -osoite. Verkkosovittimen yksilöity osoite jolla se tunnistetaan verkkosegmentissä.

Memory mapping
Muistikartoitus, muistikuvaus. Jonkin muistiosoitteen tai rekisterin asettaminen näkymään toisen muistiosoitteen kautta.

MLC Multi-Level Cell. NAND-flash-valmistusteknologia, jossa yhteen muistisoluun on tallennettuna useampi bitti.

MMC	MultiMedia Card. Vanhentunut muistikorttiformaatti.
Mount	Unix-käyttöjärjestelmissä tiedostojärjestelmän ottaminen käyttöön ja liittäminen hakemistopuuhun.
MTD	Memory Technology Device. Erilaisten muistilaitteiden, kuten flash-muistien, käyttöön tarkoitettu Linuxin alijärjestelmä.
OOB	Out Of Band. NAND-flash-muisteissa jokaisen sivun yhteydessä oleva lisäalue, jota voidaan käyttää esimerkiksi virheenkorjaustiedon tallentamiseen.
PCMCIA	Personal Computer Memory Card International Association. Tietokoneen laajennuskorttipaikan tyyppi.
SCP	Secure Copy. SSH-protokollaan perustuva suojattu tiedonsiirtotapa.
SD	Secure Digital. Muistikorttiformaatti, jota käytetään yleisesti esimerkiksi kannettavissa laitteissa.
SDHC	Secure Digital High Capacity. Secure Digital -formaatin uudempi versio, joka tukee kapasiteetiltaan aiempaa suurempia muistikortteja.
SLC	Single-Level Cell. NAND-flash-valmistusteknologia, jossa yhteen muistisoluun on tallennettuna yksi bitti.
SPI	Serial Peripheral Interface. Sarjamuotoinen kaksisuuntainen synkronoitu oheislaiteväylä piirien väliseen kommunikointiin.
SSD	Solid-State Drive. Massamuistilaite, jossa tieto tallennetaan muistipiireille. SSD-levyjä voidaan käyttää perinteisten kiintolevyjen tilalla, koska niiden sähköiset liitännät ovat samanlaiset.
TFTP	Trivial File Transfer Protocol. Yksinkertainen tiedostojen siirtoon tarkoitettu tiedonsiirtoprotokolla.

U-Boot Useita arkkitehtuureja tukeva avoimen lähdekoodin käynnistyslataaja ja monitoriohjelma.

USB Universal Serial Bus. Sarjaväylä oheislaitteiden liittämiseksi tietokoneeseen.

Volume Taltio. Massamuistissa/muisteissa oleva tallennustila. Taltio on osiointia joustavampi tapa hallita tallennustilaa massamuisteilla.

Wear leveling

Flash-muistialueiden kulumista tasaava toiminnallisuus, missä kirjoitusoperaatiot kohdistetaan koko muistitilaan siten, että muistin lohkoja käytetään tasaisesti.

xD Olympuksen ja Fujifilmin kehittämä muistikorttityyppi. Käytetään pääasiassa vanhoissa digitaalikameroissa.

XIP Execute In Place. Toiminnallisuus, jossa ohjelmakoodi luetaan ja suoritetaan suoraan pysyväismuistista, kuten NOR-flashilta, sen sijaan, että se kopioitaisiin ensin RAM-muistiin ja suoritettaisiin sieltä.

YAFFS, YAFFS2

Yet Another Flash File System. Flash-muistissa käytettäväksi kehitetty tiedostojärjestelmä.

1 Johdanto

Työn tausta

Verkostoautomaatiolla tarkoitetaan energianjakeluverkkojen ohjaukseen ja valvontaan käytettäviä tietokonejärjestelmiä. Näitä maantieteellisesti hajanaisia järjestelmiä käytetään ja ohjataan tietoliikenneyhteyksien kautta, mutta järjestelmän joidenkin osien ylläpito on pitänyt suorittaa paikan päällä. Tietoliikenneyhteyksien paraneminen ja komponenttien ja laitteiden kehittyminen on tehnyt mahdolliseksi ohjelmistopäivitysten suorittamisen etäyhteydellä verkon kautta. Koska päivitettävät tietokonejärjestelmät ovat tärkeitä osia energianjakeluverkkojen toiminnan kannalta, on päivitysten onnistuminen ja vikatilanteiden hallinta äärimmäisen tärkeää.

Työn tavoitteet

Insinöörityön tavoitteena on verkostoautomaation sulautetun järjestelmän prosessorikortin käyttöjärjestelmän etäpäivityksen turvallinen käyttöönotto ja käyttöjärjestelmän automaattinen toipuminen mahdollisista virhetilanteista. Tämän lisäksi tavoitteena on kehittää ja nopeuttaa prosessorikortin käynnistysprosessia.

Haasteet

Käyttöjärjestelmän päivittämisessä on monta häiriöille altista vaihetta, jotka on otettava huomioon etäpäivityksen yhteydessä. Ensin on varmistettava, että päivitys on siirtynyt ehjänä tietoverkon yli. Toiseksi tulee varmistua siitä, että päivitystapahtuma etenee häiriöttä ja järjestelmä käynnistyy toivotulla tavalla päivityksen jälkeen. Kolmantena haasteena on toipua päivityksen aikana mahdollisesti syntyvistä häiriö- ja virhetilanteista.

2 Sulautettujen järjestelmien komponentteja

Sulautettujen järjestelmien komponenttien valinnassa on otettava huomioon monia seikkoja. Laitevalmistajien nopea kehitystyö ja vanhojen komponenttien valmistuksen loppuminen on osattava arvioida mahdollisimman oikein. Myös yleisten standardien

puute ja toisaalta standardien kehitystyö sekä laitevalmistajien omat standardit ja niiden muutokset vaikeuttavat sulautettujen järjestelmien ylläpitoa. Oleellinen tekijä sulautettujen järjestelmien komponenttien valinnassa on niiden elinkaari suhteessa oman järjestelmän elinkaareen, jotta voitaisiin taata kriittisten komponenttien saatavuus mahdollisimman pitkään. Edellä mainittujen seikkojen huomioimiseksi on tunnettava laitteet ja niiden valmistajat sekä oltava hyvin perillä alan ja standardien kehityksestä.

2.1 Flash-muistiteknologia

Flash-muistit ovat haihtumattomia puolijohdemuisteja, joita käytetään tänä päivänä useissa eri sovelluksissa, kuten muistikorteissa ja -tikuissa sekä sulautetuissa järjestelmissä massamuistina. Flash-muisti on kehitetty EEPROM-muistista. Flash-muisti on edeltäjäänsä edullisempi, sillä on suurempi kapasiteetti, nopeampi tietosisällön tyhjennys ja sen vuoksi myös nopeampi uudelleenkirjoitus. Flash-muisti on tällä hetkellä yleisimmin käytetty haihtumaton muistityyppi, kun tarvitaan suurta tallennuskapasiteettia.

2.2 Flash-muistityypit

Flash-muistit ovat päätyypiltään joko NOR- tai NAND-flash-muisteja. Muistien nimet on johdettu siitä, että NOR- ja NAND-flash-muistien muistisolujen sisäinen rakenne muistuttaa vastaavannimisten logiikkaporttien rakennetta. Flash-muistiin kirjoitettaessa bitin tila voidaan vaihtaa vain 1:stä 0:ksi. Bitti voidaan asettaa 0:sta 1:ksi vain tyhjennysyksikön eli tyhjennyslohkon (erase block) kokoisissa osissa kerrallaan. Lohkon koko riippuu flash-muistityypistä ja vaihtelee kilotavuista satoihin kilotavuihin. Taulukossa 1 on esitetty flash-muistien ominaisuudet.

Taulukko 1. NOR- ja NAND-flash-muistien ominaisuudet. [1]

Flash-muistin tyyppi	**NOR**	**NAND**
Kirjoitusnopeus	hidas	nopea
Luentanopeus	nopea	nopea peräkkäisluenta, hidas satunnaisluenta
Tyhjennysnopeus	hidas	nopea
Kapasiteetti pinta-alaa kohden	pieni	suuri
Osoitus	luenta ja kirjoitus tavu kerrallaan, tyhjennys lohko kerrallaan	luenta ja kirjoitus sivu kerrallaan, tyhjennys lohko kerrallaan
Ohjelma suoritus muistista (XIP)	kyllä	ei

2.2.1 NOR-flash

NOR-flash tukee XIP (Execute In Place) -toimintoa, joten suoritin voi ajaa ohjelmaa suoraan NOR-flashilta kopioimatta sitä ensin flashilta RAM-muistiin.

NOR-muistipiirien tyhjennys-, ohjelmointi- yms. -komennot ovat valmistajakohtaisia. Eri valmistajien nykyisten ja tulevien muistien käyttöä helpottaa olemassa oleva CFI-standardi (Common Flash Memory Interface) [2]. CFI:n määrittelemien komentojen avulla voidaan lukea muistipiirikohtaiset tiedot itse muistipiiriltä ja näin yhdellä ajuriohjelmalla voidaan käyttää eri valmistajien muistipiirejä kaikille yhteisen rajapinnan kautta.

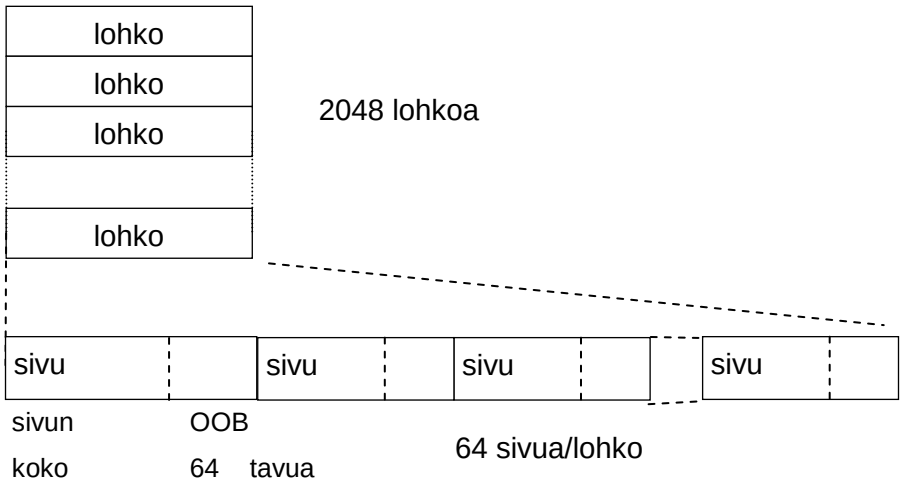
2.2.2 NAND-flash

NAND-flash-muistia valmistetaan joko SLC (Single Level Cell) tai MLC (Multi Level Cell) -tyyppisenä. SLC-tyyppisessä NAND-muistissa jokainen muistisolu sisältää vain yhden bitin ja MLC-tyyppisessä muistisolussa voi olla kaksi tai useampia bittejä.

MLC-tekniikassa solun jännite on jaettu useaan tasoon, joista jokainen vastaa tiettyä bittikuviota. Näin bittejä siis sopii samaan tilaan useampia. Haittapuolena SLC-tyyppiin verrattuna siinä on kolme kertaa hitaampi luku- ja kirjoitusnopeus sekä pienentyneestä vierekkäisten jännitetasojen erosta johtuva kasvanut bittivirheiden määrä [3, s. 3].

NAND-flash-muisti on jaettu lohkoihin, jotka jakautuvat sivuihin ja jokaiseen sivuun kuuluvaan lisäalueeseen (OOB, Out Of Band). Lisäalueelle voidaan tallentaa sivuun liittyvää lisätietoa, kuten tiedostojärjestelmän tietoja, virheenkorjausdataa (ECC, Error Correction Code) tai tietoa viallisista lohkoista (bad block). NAND-flashia voidaan lukea ja kirjoittaa sivu kerrallaan, mutta tyhjennys voidaan tehdä vain lohko kerrallaan.

Kuvassa 1 on esitetty erään 256 Mt:n NAND-flash-muistin rakenne. Muistissa on yhteensä 2048 lohkoa, ja yhden lohkon koko on 128 kt. Kukin lohko on jaettu 64 sivuun. Yhden sivun koko on 2048 tavua ja sivun lisäalueen koko 64 tavua.



Kuva 1. 256 Mt:n kokoisen NAND-muistin jakautuminen osiin.

2.2.3 Flash-muistin vialliset lohkot

NAND-flash-muisti suunniteltiin edulliseksi puolijohdemuistiksi sellaisiin sovelluksiin, joissa tarvitaan runsaasti massamuistitilaa [1, s. 20]. Jotta voitaisiin valmistaa paljon muisteja halpaan hintaan, sallii standardi muisteissa tietyn määrän viallisia lohkoja. NAND-flashissa voi siis olla jo tehtaalta tullessa viallisia lohkoja. Nämä tulevat ilmi valmistajan testauksessa [1, s. 20]. Kullakin valmistajalla on oma tapansa merkitä

vialliset lohkot. Joissain tapauksissa tieto on kirjoitettu OOB-alueelle. Käyttäjän täytyy huomioida vialliset lohkot muistia käyttäessään, koska NAND-flashin lohkoja vioittuu käytön aikana. Tämä tarkoittaa sitä, että valmistajan ilmoittamien viallisten lohkojen lisäksi käytön aikana vikaantuvat lohkot merkitään viallisiksi. Koska NAND-flashissa käytetään lohkokohtaista osoitustapaa, on viallisten lohkojen huomioiminen ja luettelointi helppo toteuttaa.

NOR-flashissa ei perinteisesti ole ollut viallisia lohkoja, koska NOR-flash-muistia ei ole tehty yhtä pieneen tilaan kuin NAND-muisti. Näin on voitu valmistaa virheettömiä muisteja. Käytön aikana tulleet virheet täytyy kuitenkin havaita ja ohittaa ohjelmallisesti muistia käytettäessä.

2.3 Flash-muistiin perustuvat massamuistit

Nykyisin useimmat siirrettävät massamuistit perustuvat haihtumattomiin puolijohdemuisteihin. Nämä solid-state-massamuistit toteutetaan flash-muistilla. Esimerkkeinä tällaisista massamuisteista ovat SD-, SDHC-, MMC- ja CompactFlash-muistikortit, vanhat PCMCIA-flash-kiintolevyt, USB-muistitikut sekä viime vuosina yleistyneet SSD-kiintolevyt (Solid-State Drive). Muistien pohjana olevasta muistiteknologiasta huolimatta nämä massamuistivälineet eivät näy niitä käyttäville laitteille flash-muisteina kaikkine kyseisen muistityypin ominaispiirteineen. Ne näkyvät kiintolevyn tapaisina tallennuslaitteina, joita käyttöjärjestelmä käsittelee sektori kerrallaan normaalin kiintolevyn tapaan [4, s. 5; 5, s. 2]. Flash Translation Layer -niminen rajapinta huolehtii flash-muistin lohko-osoituksen muuntamisesta kiintolevyn sektoriosoitukseen.

2.3.1 FTL, Flash Translation Layer

Flash-muistiin perustuvissa siirrettävissä massamuisteissa, kuten muistikorteissa ja USB-muistitikuissa, muistivälineeseen on sisällytetty muistiohjain, joka toimii varsinaisten flash-muistipiirien sekä muistilaitteen ulkoisen rajapinnan välillä. Sen sijaan monissa sulautetuissa järjestelmissä, kuten tässä insinöörityössä käytetyssä prosessorikortissa, flash-muistia käsitellään suoraan ilman erillistä flash-muistiohjaintasoa.

Muistiohjain hoitaa flash-muistiin kirjoittamisen ja sieltä lukemisen sekä kulumisen tasaamisen (wear leveling) ja tarjoaa ulospäin standardin rajapinnan. Muistiohjain toimii FTL-tasona muistipiirin tai -piirien ja muistia käyttävän järjestelmän välillä [6, s. 262]. Valmistajan toteutuksesta riippuen FTL voi olla toteutettu joko ohjelmatasolla tai laitetasolla [5, s. 1]. Tiedon lukeminen suoraan flash-muistipiiriltä ymmärrettävässä muodossa voi olla hankalaa ja aikaa vievää [7], koska valmistajat eivät ole halukkaita paljastamaan flash-massamuisteissaan käyttämiään tallennus- ja muistipiirien kulutuksen tasaamisalgoritmeja. Käytettyjä algoritmeja voi yrittää päätellä esimerkiksi tutkimalla kirjoitus- ja lukuoperaatioihin kuluvaa aikaa [5, s. 3–4].

2.3.2 Muistikortit ja USB-muistitikut

Muistikorteissa ja USB-muisteissa on sisäänrakennettuna edellä kuvattu FTL-taso, joka hoitaa flash-muistipiirin käsittelyn. Käyttöjärjestelmälle tai laitteelle muistikortti tai -tikku näkyy kiintolevyn tapaisena sektoreittain osoitettavana laitteena. Flash-massamuistia voi lukea ja sille voi kirjoittaa kuin kiintolevylle ja sitä voi myös osioida ja alustaa haluamalleen tiedostojärjestelmälle. Usein muistit on tehtaalla valmiiksi alustettu FAT-tiedostojärjestelmälle [8; 9], jota käytetään useimmissa kameroissa. Flash-massamuisteissa voidaan käyttää muitakin tiedostojärjestelmiä.

USB-flash-muisti näkyy käyttöjärjestelmälle USB-massamuistilaitteena [10]. Käyttöjärjestelmän USB-ajurit ja USB:n sähköiset liitännät toimivat niin kuin muillakin USB-laitteilla, joten esimerkiksi tietokoneen kannalta on samantekevää millä muistitekniikalla USB-liitäntäinen massamuistilaite on toteutettu. Tämä massamuistilaite voi yhtä hyvin olla siis toteutettu joko flash-muistilla, perinteisellä pyörivällä kiintolevyllä tai DVD-ROM-asemana. Koska flash-muisteilla on rajallinen uudelleenkirjoitusmäärä, ne eivät kestä toistuvaa kirjoittamista ja muuttamista, kuten pyörivät kiintolevyt. Sellainen massamuistin käyttö, jossa tiedostoja muutetaan jatkuvasti (kuten swap-muisti), kuluttaa flash-muistia runsaasti.

Flash-muistikortteja voidaan lukea ja kirjoittaa kyseisen tekniikan kortinlukijoilla. Sulautetuissa järjestelmissä esimerkiksi SD- tai MMC-korttia voidaan käsitellä perus-SPI-tilassa, jolloin kortinlukija voidaan liittää 4-johtimiseen SPI-väylään [11, s. 11]. Muistikortilla itsellään oleva muistiohjain tulkitsee liitäntäväylältä tulleet komennot ja käsittelee varsinaisen NAND-flash-muistipiirin tai -piirien kirjoituksen ja lukemisen sekä

huolehtii myös muistipiirien kulumisen tasauksesta ja virheidenkorjauksesta virheenkorjaustietojen avulla.

Muistikorteista poikkeuksena mainittakoon vanhat SmartMedia- ja xD-muistikortit, joissa fyysistä NAND-muistipiiriä voi käsitellä suoraan muistikortin ulkoisen rajapinnan kautta [12]. Näissä muistikorteissa FTL puuttuu kokonaan ja tarvittava tekniikka ja logiikka on rakennettu suoraan muistikortinlukijaan. Jos halutaan lukea NAND-flash-muistipiiriä suoraan, voidaan tämä suorittaa modifioimalla xD-kortinlukijaa, koska xD-kortin sähköiset kontaktit vastaavat melko tarkasti NAND-flash-muistipiirin kontakteja [12].

2.3.3 SSD-kiintolevyt

Nykyään kokonaan flash-muistiin perustuvat SSD-kiintolevyt ovat korvanneet tai korvaamassa perinteiset magneettiseen tallennukseen perustuvat kiintolevyt useissa kohteissa. Esimerkiksi tärinälle alttiissa järjestelmissä SSD-levyjen etuna on, ettei niissä ole liikkuvia osia, kuten perinteisissä kiintolevyissä. Myös tietyntyyppisissä sovelluksissa käyttöjärjestelmälevyinä SSD-levyt ovat huomattavasti suorituskykyisempiä kuin normaalit kiintolevyt. Sen sijaan peräkkäisluku- tai kirjoituskäytössä, kuten suurten mediatiedostojen lukemisessa, perinteinen kiintolevy saattaa vielä olla järkevämpi vaihtoehto, kun verrataan suorituskykyä, nopeutta ja hintaa. [13, s. 8]

SSD-levyssä massamuisti koostuu useasta fyysisestä muistipiiristä, jotka ovat yhteydessä muistiohjaimeen väylien kautta. Muistiohjaimen tehtävänä on muistipiirien hallinta ja se hoitaa myös mahdollisesti luku- ja kirjoitusoperaatioiden hajauttamisen useammalle muistipiirille suorituskyvyn parantamiseksi. [14]

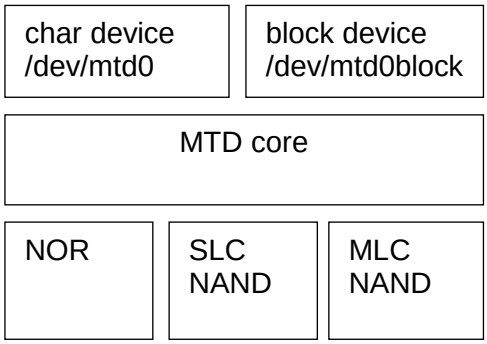
2.3.4 Hybridilevy

Hybridilevyt ovat massamuisteja, joissa normaalin kiintolevyn yhteyteen on lisätty haihtumattomalla puolijohdemuistilla toteutettu välimuisti. Välimuisti voi olla muutaman gigatavun kokoinen ja toimii usein lukuvälimuistina. Kun kiintolevyltä on luettu välimuistiin oikeaa tietoa, tiedon lukeminen hybridilevyltä ohjelmien käyttöön tapahtuu

huomattavasti nopeammin, koska tieto tulee nopeasta välimuistista levyltä lukemisen sijaan.

2.4 Flash-muistit Linuxissa

Flash-muistit näkyvät Linuxissa MTD-laitteina (Memory Technology Device). Flash-muistiajurit rekisteröityvät Linuxin MTD-alijärjestelmään ja flash-muisteja käsitellään alijärjestelmän rajapintojen kautta. Kuvassa 2 on esitetty flash-muistien, MTD-alijärjestelmän ja MTD-laitteiden välinen hierarkia. MTD-järjestelmä luo jokaista flash-muistia varten merkkilaitteen (character device) sekä lohkolaitteen (block device). Nämä näkyvät laitetiedostoina kuten /dev/mtd0 ja /dev/mtd0block.



Kuva 2. MTD-laitehierarkia Linuxissa [15, s. 8].

2.4.1 Linux FTL – mtdblock

Linuxissa on yksinkertainen mtdblock-niminen FTL-taso [16]. Se on toteutettu ajurina, joka ladattaessa luo jokaista järjestelmässä olevaa mtd-laitetta vastaavan mtdblock-laitteen. Kun lohkoa kirjoitetaan muistiin, ajuri lukee ensin keskusmuistiin sen NAND-flashin tyhjennyslohkon, missä lohko on. Sen jälkeen ajuri tekee muutokset lohkoon keskusmuistissa, tyhjentää flash-muistin kyseisen lohkon ja kirjoittaa uuden, muuttuneen lohkon keskusmuistista flash-muistiin. Linuxin mtdblock-ajuri on yksinkertainen työkalu. Sen kirjoitustoimintoja ei ole optimoitu ottamaan huomioon vikatilanteita, sillä se ei huomioi viallisia lohkoja eikä se tee flash-muistin kulumisen tasausta. Tästä johtuen ajuria ei suositella käytettäväksi normaalitilanteessa.

2.4.2 Flash-muistin kirjoittaminen ja lukeminen Linuxissa

Flash-muistien muokkaaminen Linuxissa toteutetaan eri tavoilla, riippuen siitä, käsitelläänkö kyseistä muistia pelkkänä muistina ilman tiedostojärjestelmää vai onko muistille tallennettu oma tiedostojärjestelmänsä, jonka kautta muistia käsitellään.

Jos flash-muistille on tehty tiedostojärjestelmä ja kyseinen tiedostojärjestelmä on liitetty käyttöjärjestelmän hakemistopuuhun, voidaan flash-muistiin kirjoittaa ja sieltä lukea tiedostoja niin kuin muistakin tiedostojärjestelmistä.

Jos flash-muistia käytetään puhtaana lohkomuistilaitteena, ja jos siinä ei ole tiedostojärjestelmää, siihen voidaan kirjoittaa Linuxin ’dd’-komennolla. ’dd’-komentoa käytettäessä flash-muistiin täytyy kirjoittaa mtdblock-ajurin muistille luoman laitenimen kautta. Esimerkiksi seuraavalla komennolla voidaan kopioida file.ext-niminen n-pituinen tiedosto flash-muistilaitteen mtd0 osoitteeseen x:

```
# dd if=file.ext of=/dev/mtd0block count=n obs=1 seek=x
```

Tällä tavalla flash-muistiin kirjoitettaessa viallisten lohkojen käsittelyä ei kuitenkaan tapahdu automaattisesti, vaan lähdetiedosto kirjoitetaan viallisia lohkoja huomioon ottamatta ja näin kaikki tieto ei tallennu flash-muistiin. Jotta tämä voidaan välttää, on otettava käyttöön Linuxin mtd-utils-paketti ja sen ’nandwrite’-niminen ohjelma. Tämä ohjelma ohittaa vialliset lohkot ja kirjoittaa sellaisen löytäessään seuraavaan ehjään lohkoon. Tämä pitää ottaa huomioon myös flash-muistia luettaessa niin, että vialliset lohkot ohitetaan. NOR-flash-muistiin voidaan kirjoittaa ’flashcp’-nimisellä ohjelmalla. Tällä ohjelmalla voi myös kirjoittaa NAND-flash-muistiin, mutta ohjelma ei osaa huomioida ja ohittaa NAND-flashin viallisia lohkoja. Tämä pitää muistaa jos ’flashcp’-ohjelmaa käytetään NAND-flash-muistin kanssa.

2.5 Flash-muistin simulointi Linuxissa

Linux tarjoaa ohjelmat NAND-flash-muistin simulointiin. Flash-tiedostojärjestelmiin tutustuminen kannattaa aloittaa peruskomentojen ja niiden parametrien kokeilemisesta. Koska normaalissa työpöytä-PC:ssä harvoin on flash-muistia käytettävissä (tähän ei lueta mukaan flash-muistia sisältäviä muistitikkuja tai muistikortteja), tarvitaan testaukseen laite, jossa on käytettävissä sopivaa flash-muistia. Insinöörityössäni olen

käyttänyt prosessorikorttia, jolla pystyi käsittelemään NOR- ja NAND-flash-muisteja todellisella alustalla. Testauksessa ei aina ole käytettävissä mitään konkreettista laitetta eikä sellaista edes tarvita. Esimerkiksi PC-testeissä voidaan käyttää Linuxin simuloimaa flash-muistia.

2.5.1 NANDSim-ajuri NAND-flashin simulointiin

NAND-flash-muistia ja siinä käytettäviä tiedostojärjestelmiä voidaan tutkia ja kehittää ilman fyysistä flash-muistia käyttämällä Linuxin nandsim-nimistä ajuria muistin simulointiin [17]. Linuxin ytimen mukana tuleva nandsim simuloi NAND-flash-muistia joko keskusmuistissa tai levytiedostossa.

Ajuria ladattaessa sille voidaan antaa parametreina lista simuloiduista viallisista lohkoista sekä luku-, kirjoitus- tai muista viiveistä. Lisäksi ajurille voidaan kertoa, miten ja milloin sekä millainen vikatilanne flash-muistiin tulee. Jokin lohko voidaan esimerkiksi määritellä kuluneeksi (weak) ja määritellä sen vikaantuminen tiettyjen tyhjennyskertojen jälkeen.

Alla olevalla esimerkin 1 komennolla luodaan simuloitu 256 Mt:n kokoinen NAND-flash-muisti, jonka lohkon koko on 128 kt. Muisti on jaettu kahteen 128 Mt:n osioon. NANDsim-ajuri luo kaksi mtd-laitetiedostoa, /dev/mtd0 ja /dev/mtd1 (numerot alkavat järjestelmän ensimmäisestä vapaasta mtd-laitteen numerosta). Luonnin jälkeen uusien mtd-laitteiden tiedot näkyvät cat-komennon avulla /proc/mtd-tiedostossa.

```
$ sudo modprobe nandsim first_id_byte=0xec second_id_byte=0xda ↵
third_id_byte=00 fourth_id_byte=0x15 parts=1024,1024
$ cat /proc/mtd
dev:    size   erasesize  name
mtd0: 08000000 00020000 "NAND simulator partition 0"
mtd1: 08000000 00020000 "NAND simulator partition 1"
```

Esimerkki 1. Linuxin nandsim-ajurin lataaminen ja näyttäminen.

Komennon jälkeen mtd-laitteita käsitellään samalla tavalla kuin fyysistä NAND-muistia. Oletuksena on, että nandsim-muistit luodaan tietokoneen keskusmuistiin. Sopivilla parametreilla ne voidaan luoda myös tiedostoon esimerkiksi tilanteessa, jossa ei haluta tai voida käyttää keskusmuistia suurikokoisen NAND-flashin simulointiin.

NAND-flash-muisteista saadaan ’Read ID’ -komennolla [17] luettua muistista tietoja, kuten muistin valmistaja, muistin koko, lohkokoko sekä ajoitukset. Nandsim-ajurin simuloiman muistin tyyppi voidaan asettaa antamalla sille parametreina todellisen flash-muistin ’Read ID’ -komennolla palauttamat tavut. Lisätietoa komennon luomasta simuloidusta nand-muistista saa komennon antamisen jälkeen ’dmesg’-komennolla. Esimerkki ’dmesg’-komennon tulosteesta on liitteessä 1, jossa luodaan simuloitu NAND-flash-muisti erään Samsungin muistipiirin asetuksilla.

2.5.2 block2mtd – mtd-laitteen emulointi lohkolaitteen päällä

Linuxin ytimen mukana tulee block2mtd-ajuri, jolla voidaan emuloida mtd-laitetta lohkolaitteen (block device) päällä. Ajuria voidaan käyttää esimerkiksi silloin, kun halutaan lukea tiedostona olevan JFFS2-levykuvan sisältöä. Kun block2mtd-ajuri ladataan, se luo /dev/mtdblock-laitteen, jonka avulla voidaan esimerkiksi liittää tiedostona oleva JFFS2-levykuva johonkin hakemistoon. Tällä tavalla hakemistoon liitetty levykuva on pelkästään lukutilassa eikä sen sisältöä voi muuttaa. Jos levykuvan sisältöä halutaan muuttaa, se tehdään kopioimalla kyseisen hakemiston sisältö muualle. Sisältömuutokset tehdään uudessa sijainnissa ja siitä tehdään uusi levykuva, jolla korvataan vanha, muutettava levykuva. Esimerkissä 2 esitetään miten tiedostojärjestelmän levykuva liitetään hakemistoon.

```
$ losetup /dev/loop0 rootfs.jffs2
$ mknod /dev/mtdblock0 b 31 3
$ modprobe block2mtd block2mtd=/dev/loop0,0x20000
$ modprobe mtdblock
$ mount -t jffs2 /dev/mtdblock0 /mnt/rootfs/
```

Esimerkki 2. JFFS2-tiedostojärjestelmä-levykuvan liittäminen /mnt/rootfs/-hakemistoon.

2.5.3 Levykuvien tavujärjestys

JFFS2-levykuvien tavujärjestys riippuu prosessoriarkkitehtuurista. Tämä on otettava huomioon levykuvan luontivaiheessa tai ainakin ennen sen käyttämistä kohdejärjestelmässä. Oletuksena ’mkfs.jffs2’-komento käyttää luotavalle levykuvalle sitä tavujärjestystä, mikä on käytössä järjestelmässä, jossa komento ajetaan. Levykuvan luontivaiheessa tavujärjestyksen voi valita käyttämällä sopivaa parametria ’mkfs.jffs2’-komennolle. Jos levykuvaa tullaan käyttämään PowerPC-arkkitehtuurissa,

on Intel-pohjaisessa (little-endian) järjestelmässä (oikea tai virtuaalikone) käytettävä '--big-endian' (tai '-b') -parametria. Jos taas halutaan luoda little-endian-levykuva PowerPC-arkkitehtuurissa, pitää käyttää '--little-endian' (tai '-l') -parametria.

Esimerkissä 3 on luotu 'mkfs.jffs2'-komennolla rootfs-big.jffs-niminen levykuva rootdir-hakemistosta. Levykuvan tavujärjestys on big-endian, lohkokoko on 128 kt (heksadesimaalina 20000) ja sivukoko 2048 tavua.

```
$ mkfs.jffs2 -b -e 0x20000 -n -p 2048 -o rootfs-big.jffs2 -v -q -s ↵
2048 -r rootdir
```

Esimerkki 3. Big-endian-levykuvan luonti.

Valmiin levykuvan tavujärjestyksen muuttamiseen voidaan käyttää 'jffs2dump'-komentoa. Kun käännösympäristön tuottamaa, PowerPC-kohdejärjestelmässä käytettävää big-endian-levykuvaa halutaan tarkastella toisessa, eri tavujärjestystä käyttävässä järjestelmässä (Intel), täytyy levykuvalle tehdä muutos big-endian-muodosta little-endian-muotoon esimerkin 4 komennolla.

```
$ jffs2dump -b -c -e rootfs-little.jffs2 rootfs-big.jffs2
```

Esimerkki 4. Big-endian-jffs2-levykuvasta muunnos little-endian-muotoon.

Tämän jälkeen tuloksena saatu levykuva rootfs-little.jffs2 voidaan liittää käyttöön esimerkiksi block2mtd-ajurilla (esitelty kappaleessa 2.5.2) Intel-pohjaisessa järjestelmässä.

3 Massamuistien tiedostojärjestelmät

Eri tallennusvälineille, kuten optisille levyille, nauhoille, levymuisteille ja flash-muisteille on omat tiedostojärjestelmänsä. Koska tiedon tallennus- ja lukutapa muistiin ja muistista riippuvat muistin tyypistä, ei ole järkevää käyttää sektoripohjaisen levymuistin tiedostojärjestelmiä flash-muisteilla tai toisinpäin. Tiedostojärjestelmässä voi olla kohtia, jotka muuttuvat usein, kuten hakemiston sisältämät tiedostot. Kiintolevyille tarkoitetulla tiedostojärjestelmällä tämä saattaa tarkoittaa sitä, että levyn samaa kohtaa muutetaan aina, kun hakemistossa tapahtuu muutoksia. Koska flash-muisteilla on

rajattu kirjoitusmäärä, tällaista samaan kohtaan toistuvaa kirjoittamista täytyy välttää ja muutokset tulee kirjoittaa tasaisesti joka puolelle muistipiiriä sen kulumisen tasaamiseksi (wear leveling) [18, s. 89]. Sulautetuissa järjestelmissä käytetään levy- ja flash-muisteja sekä niiden tiedostojärjestelmiä.

3.1 Levytiedostojärjestelmät

Perinteiset levymuisteille tarkoitetut tiedostojärjestelmät on suunniteltu tietojen tallentamiseen pyöriville levyille. Nämä tiedostojärjestelmät näkevät levyn peräkkäisinä sektoreina, joiden koko on yleensä 512 tavua (tosin uusissa suurikapasiteettisissa kiintolevyissä käytetään 4096 tavun sektorikokoa). Kirjoitus- ja lukuoperaatioita voidaan osoittaa yksittäisiin sektoreihin ja satunnaisiin sijainteihin [19, s. 235]. Esimerkkeinä levytiedostojärjestelmistä on Windowsissa käytetyt FAT32 ja NTFS sekä Linuxin ext2, ext3, ext4 sekä ReiserFS ja btrfs.

3.2 Flash-tiedostojärjestelmät

Flash-muisteja varten on kehitetty tiedostojärjestelmiä, jotka ottavat huomioon flash-muistin erityispiirteet, kuten kulumisen tasaamisen, lohkojen vapauttamisen uudelleen käyttöön siivoustoiminnolla (garbage collection), flash-muistin kirjoitus- ja lukutavat sekä virheenkorjaustiedon (ECC) ylläpitämisen [20, s. 41].

Flash-muisteille tarkoitettuja tiedostojärjestelmiä, kuten JFFS2, voi periaatteessa käyttää myös flash-muistiin perustuvilla, mutta ulospäin tavalliselta levymuistilta näyttävillä muistikorteilla, USB-muisteilla ja tavallisilla kiintolevyillä [21]. Flash-tiedostojärjestelmien käytöstä muistikorteilla ja USB-muisteilla seuraa kuitenkin suorituskyvyn lasku, koska välissä on silloin enemmän tiedon osoitus- ja muunnostasoja. Tällöin muistilaitteessa tapahtuu ensin osoitteen muunnos sisäisesti flashista FTL:n avulla levytyyppiseen sektoriosoitukseen ja sen jälkeen muunnos Linuxissa ohjelmallisesti takaisin levytyyppisestä sektoriosoituksesta flash-tyyppiseen lohko-osoitukseen. Linuxissa flash-tiedostojärjestelmää voidaan käyttää sektorityyppisellä levymassamuistilla alaluvussa 2.5.2 esitetyllä block2mtd-ajurilla.

3.2.1 Tiedostojärjestelmät JFFS, YAFFS, SquashFS, CRAMFS

JFFS-tiedostojärjestelmä kehitettiin NOR-flash-muisteja varten, jotka olivat siihen aikaan yleisesti saatavilla oleva flash-muistityyppi [6, s. 239]. JFFS-tiedostojärjestelmästä on sittemmin kehitetty NAND-muistia tukeva, seuraavassa kappaleessa kuvattu JFFS2-tiedostojärjestelmä.

Ensimmäinen NAND-flash-muistia varten kehitetty tiedostojärjestelmä oli nimeltään YAFFS. YAFFS tuki sillä hetkellä käytössä olevia NAND-muisteja, joissa oli 512 tavun kokoiset sivut. YAFFS2-tiedostojärjestelmä kehitettiin tukemaan NAND-muisteja, joiden sivukoko on tätä suurempi.

SquashFS on lohkolaitteissa käytettävä tiedostojärjestelmä. Sillä ei voi kirjoittaa, vaan se on pelkästään luettava järjestelmä. Tästä johtuen SquashFS:ää voidaan käyttää myös flash-muistilla, kun käytetään lisäksi mtdblock-ajuria [22, s. 26]. Koska mtdblock-ajurin luomat laitteet eivät hallitse viallisia lohkoja, virheet flash-muistissa näkyvät myös luettaessa SquashFS-osiolta. Tämä otetaan huomioon tarkastamalla luetun tiedon oikeellisuus.

CRAMFS on samantyyppinen kuin SquashFS eli se on pelkästään luettava tiedostojärjestelmä. CRAMFS:llä on rajoituksia, kuten tiedostojen aikaleimojen ja tiedostolinkkien puute sekä 16 Mt:n tiedostokoko, joten se sopii rajoitettuun ja pienikokoiseen käyttöön esimerkiksi sulautetussa järjestelmässä. [23]

3.2.2 UBIFS-tiedostojärjestelmä

UBIFS on JFFS2:ta uudempi flash-muisteille tarkoitettu tiedostojärjestelmä. UBIFS koostuu kahdesta tasosta mtd-laitteen päällä. Ensimmäinen taso on UBI-alijärjestelmä, joka on kerros flash-muistin päällä ja joka huolehtii mtd-laitteiden ominaisuuksista, kuten kulumisen tasaamisesta ja viallisten lohkojen käsittelystä. Toisen tason muodostavat UBI-alijärjestelmän päällä oleva yksi tai useampi taltio (volume), jonka päälle tiedostojärjestelmä voidaan luoda. UBI-tiedostojärjestelmä on UBI-alijärjestelmän luomaan taltioon luotu tiedostojärjestelmä. UBIFS:n etuna esimerkiksi JFFS2:een verrattuna on sen skaalautuvuus (UBIFS ei hidastu osion koon kasvaessa) ja nopea UBIFS-tiedostojärjestelmän liittäminen hakemistopuuhun. Lisäksi se sietää

paremmin epäpuhtaita sammutuksia, joita tapahtuu esimerkiksi sähköjen katketessa, kun ajuri ei pääse sulkemaan tiedostojärjestelmää oikein. [24]

3.2.3 JFFS2-tiedostojärjestelmä

JFFS2 on lokiin perustuva tiedostojärjestelmä, jossa muutokset tiedostojärjestelmään kirjoitetaan flash-muistiin uutena merkintänä (node). Jokainen merkintä sisältää tietoa tiedostosta, johon se kuuluu, kuten tiedoston nimen tai tiedoston sisältöä. Kun tiedostoa muutetaan, muutos kirjoitetaan flash-muistiin uutena merkintänä. JFFS2:a käytettäessä flash-muistilla ei ole tallennettuna varsinaista tiedostojärjestelmän hakemistorakennetta samassa merkityksessä kuin esimerkiksi FAT- tai ext3-tiedostojärjestelmässä. Tästä johtuen JFFS2-osion hakemistopuu täytyy luoda keskusmuistiin ennen kuin sen sisältämiä tiedostoja voidaan käyttää. Tämä tehdään lukemalla JFFS2-osion merkinnät alusta loppuun ja luomalla hakemistorakenne sen avulla keskusmuistiin. Hakemiston rakentaminen muistiin tekee JFFS2:n liittämisnopeuden riippuvaiseksi osion koosta, sillä mitä suurempi osio on, sitä enemmän siihen voidaan kirjoittaa merkintöjä. Merkintöjen määrä lisääntyy myös, kun tiedostojärjestelmästä poistetaan tiedostoja, sillä jokainen tiedoston poisto kirjoitetaan uutena merkintänä tiedostojärjestelmään. Näin JFFS2-osion liittäminen hidastuu myös, kun siltä on poistettu ja sille on lisätty paljon tiedostoja.

JFFS2-tiedostojärjestelmässä on alhaisella prioriteetillä pyörivä automaattinen siivoustoiminto [25]. Siivoustoiminto siirtää ja yhdistelee käytössä olevien lohkojen dataa, kunnes vapautuu kokonainen lohko, jossa on pelkästään vanhentunutta, ei käytettävää dataa. Tämä lohko voidaan sen jälkeen tyhjentää ja ottaa uudelleen käyttöön datan tallentamiseen. Kun vapaiden lohkojen määrä JFFS2-osiolla alkaa loppua, automaattisen siivoustoiminnon prioriteetti nousee ja se vapauttaa tilaa uutta dataa varten.

JFFS2-osion liittämisen nopeuttamiseksi JFFS2:een on Linux-ytimen versiosta 2.6.28 alkaen lisätty JFFS2 Summary -niminen ominaisuus. Summary- eli yhteenveto-ominaisuutta käytettäessä kaikkia JFFS2-merkintöjä ei tarvitse liitettäessä lukea muistiin, vaan pelkästään flash-lohkokohtaiset yhteenvetomerkinnät luetaan. Tämä nopeuttaa osion liittämistä.

Kun yhteenveto-ominaisuus on käännetty mukaan Linuxin JFFS2-tiedostojärjestelmä-ajuriin, ajuri kirjoittaa jokaisen täyden lohkon loppuun merkinnän, joka sisältää yhteenvedon kyseisen lohkon sisällöstä ja siten myös kaikista lohkon sivujen sisällöistä. Tämän jälkeen kyseisestä lohkosta ei osion hakemistopuuta luotaessa tarvitse lukea jokaista sivua, vaan riittää, että luetaan pelkät yhteenvetotiedot [26].

Liitettäessä osiota JFFS2-ajuri etsii sieltä yhteenvetotietoja, joita käyttäen liittäminen on nopeaa. Jos tietoja ei ole, JFFS2 siirtyy käyttämään alkuperäistä tapaa, jolloin se lukee koko lohkon alusta loppuun ja luo tiedot luetuista tiedoista. Pelkkä yhteenvetotuen mukanaolo Linuxin ytimessä ei vielä nopeuta tiedostojärjestelmän liittämistä, vaan yhteenvetotiedot pitää olla myös tallennettuna osiolla.

Yhteenvetotiedot voidaan kirjoittaa tiedostojärjestelmän levykuvaan sumtool-nimisellä työkalulla, joka tulee mtd-utils-paketin mukana. Tällöin levykuva luodaan ensin hakemistosta ’mkfs.jffs2’-komennolla, jonka jälkeen ajetaan ’sumtool’-komento, joka tekee levykuvasta yhteenvetotiedolla varustetun kopion. Luotu kopio voidaan kirjoittaa kohdejärjestelmän flash-muistiin.

Riippumatta siitä, onko käytössä olevalla osiolla yhteenvetotietoja, Linux-ydin kirjoittaa yhteenvetotietoja levylle käytön aikana, jolloin kukin tieto tallentuu omaan lohkoonsa. Jossain vaiheessa kaikki lohkot voivat sisältää yhteenvetotietoja riippumatta siitä, oliko tietoja alkuperäisessä tiedostojärjestelmässä.

Jos Linux-ytimeen ei ole käännetty yhteenvetotietojen tukea, ydin ohittaa tällaiset merkinnät osiota liitettäessä. JFFS2-tiedostojärjestelmän merkinnöissä on kenttä, johon on tallennettuna kentän yhteensopivuuslippu (compatibility flag) [27]. Kun tiedostojärjestelmäajuri lukee merkinnän, se päättää yhteensopivuuslipun avulla, mitä kyseisen merkinnän kanssa tehdään. Esimerkiksi tilanteessa, jossa löytyy yhteenvetomerkintä, mutta Linux-ytimessä ei ole yhteenvetotietojen tukea, merkinnän yhteensopivuuslippu nimeltään JFFS2_FEATURE_RWCOMPAT_DELETE [28] tarkoittaa sitä, että kyseinen yhteenvetomerkintä poistetaan tiedostojärjestelmän siivouksen yhteydessä. Näin käytettävä Linux-ydin poistaa automaattisesti tiedostojärjestelmästä merkintöjä, joille ytimessä ei ole tukea.

4 Käynnistyslataaja

Kun laite käynnistetään, suoritin alkaa ajaa käskyjä tietystä muistiosoitteesta. Monet laitteiston komponentit täytyy alustaa ennen niiden käyttöä ja ennen kuin varsinainen käyttöjärjestelmä voidaan ladata muistiin ja käynnistää. Käynnistyslataajan tehtävänä on alustaa laitteisto käyttökuntoon niin, että sillä voidaan ajaa muita ohjelmia, ladata käyttöjärjestelmän tarvittavat osat muistiin ja lopuksi siirtää suoritus käyttöjärjestelmälle.

Joissain yhteyksissä käytetään nimitystä monitoriohjelma. Tällöin minimaalinen käynnistyslataaja käynnistää pelkän laitteen ja lataa varsinaisen käyttöjärjestelmän. Monitoriohjelma sisältää käynnistämisen ja ohjelman latauksen lisäksi käyttöliittymän, jossa voidaan tutkia ja muokata laitteen muistin sisältöä, tyhjentää ja kirjoittaa tietoja pysyväismuistiin (esimerkiksi flash) ja suorittaa muita matalan tason operaatioita [18, s. 247]. Esimerkkinä monitoriohjelmassa käytettävistä toiminnoista ovat flash-muistin tyhjennys ja sille kirjoitus sekä järjestelmän kokoonpanoa kuvaavan, muistissa sijaitsevan, laitepuun muokkaus ennen sen antamista Linuxin käyttöön.

Tässä insinöörityössä käytetyssä prosessorikortissa käynnistyslataaja (U-Boot) on tallennettu NOR-flash-muistiin. Flash-muisti on konfiguroitu näkymään suorittimen muistiavaruudessa niin, että alustuksen jälkeen suoritin ajaa flash-muistiin tallennetun käynnistyslatausohjelman.

4.1 U-Boot-käynnistyslataaja

U-Boot (viralliselta, pidemmältä nimeltään Das U-Boot) on vuonna 1999 PPCBoot-nimellä aloitettu projekti, joka perustui alun perin PowerPC 8xx-prosessorialustalle tehtyyn 8xxROM-nimiseen käynnistyslataajaan. Nykyään U-Boot sisältää tuen yli 460 kortille ja vajaalle 50 prosessorikonfiguraatiolle [6, s. 174].

Yhtenä syynä U-Bootin suosiolle on se, että siihen on suhteellisen helppo lisätä tuki uudelle laitealustalle. Uutta laitetta kehitettäessä voidaan perustaksi ottaa jo olemassa oleva, toiminnallisesti mahdollisimman samankaltainen laitealusta. Sama pätee suorittimen konfigurointiin. Jos valittua suoritinta ei löydy tuetuista suorittimista, voidaan

lähtökohdaksi kopioida jokin toiminnallisesti lähellä oleva tai muuten saman sukuinen suoritin ja tehdä siihen tarvittavat muutokset.

4.2 U-Bootin ympäristömuuttujat

U-Bootin toimintaa ohjataan komennoilla ja ympäristömuuttujilla tai niiden yhdistelmillä. Ympäristömuuttujia on olemassa kahta laatua, ajonaikaiset muuttujat ja tallennetut muuttujat, jotka tallennetaan pysyvästi erilaisiin pysyväismuisteihin. Tallennettujen muuttujien sijainti (osoite muistissa) määritellään jo käännösvaiheessa. Samalla voidaan määritellä oletusmuuttujia ja niiden arvoja. Tässä insinöörityössä käytetyllä prosessorikortilla ympäristömuuttujat tallennetaan NOR-flashiin. Tällöin U-Boot lataa muuttujat flash-muistista RAM-muistiin käynnistyksen yhteydessä. Muuttujien arvoja voidaan käsitellä sekä luoda uusia muuttujia tai poistaa olemassa olevia muuttujia U-Bootin komennoilla. Muutokset keskusmuistissa oleviin ympäristömuuttujiin katoavat käyttöjärjestelmän käynnistyksen tai laitteen uudelleenkäynnistyksen yhteydessä. Uudet muuttujien arvot saadaan kuitenkin talletettua pysyvästi U-Bootin ’saveenv’-komennolla, jolloin uudet ympäristömuuttujat tallentuvat U-Bootissa konfiguroituun pysyväismuistiin ja korvaavat vanhat muuttujat.

U-Bootin ympäristömuuttujia voi lukea ja kirjoittaa Linuxissa ’fw_printenv’/’fw_setenv’-ohjelmalla, joka sijaitsee U-Bootin lähdekoodipuun hakemistossa tools/env/. U-Boot:ia käännettäessä edellä mainitut ohjelmat kääntyvät mukana Linuxissa käyttöä varten. Kun halutaan ottaa käyttöön sekä kirjoitus että luenta, on Linuxissa tehtävä ’fw_setenv’-niminen linkki, joka osoittaa ’fw_printenv’-ohjelmaan. Tämä ohjelma tarvitsee konfiguraatiotiedoston (/etc/fw_env.config), jossa kerrotaan, mille mtd-laitteelle U-Bootin ympäristömuuttujat ovat tallennettuina. Lisäksi tiedostossa kerrotaan ympäristömuuttujien alkukohta flash-muistissa, muuttujatiedolle varatun muistialueen koko sekä flash-muistin lohkokoko. Esimerkissä 5 on listattu /etc/fw_env.config-konfiguraatiotiedosto, jossa on määritelty ympäristömuuttujien osoitteeksi 0x80000 ja niille on varattu muistia 64 kt. Muuttujat on tallennettu /dev/mtd0-laitteeseen, jonka lohkokoko on 64 kt.

```
Configuration file for fw_(printenv/saveenv) utility.
# Up to two entries are valid, in this case the redundant
# environment sector is assumed present.
# Notice, that the "Number of sectors" is ignored on NOR.
# MTD device name   Device offset   Env. size   Flash sector size ↲
```

```
No of sectors
/dev/mtd0           0x80000         0x10000     0x10000
```

Esimerkki 5. Konfiguraatiotiedosto U-Bootin ympäristömuuttujien käsittelyyn Linuxissa.

U-Boot varmistaa ympäristömuuttujiensa oikeellisuuden ennen niiden käyttöä CRC-tarkistussumman avulla, joka sijaitsee muistissa ennen varsinaisia ympäristömuuttujia. Tarkistussumma lasketaan ja tallennetaan joka kerta, kun ympäristömuuttujat tallennetaan ’saveenv’-komennnolla U-Bootin kehotteessa tai ’fw_setenv’-ohjelmalla Linuxissa. Jos järjestelmän käynnistyessä pysyväismuistista luetuista ympäristömuuttujista laskettu tarkistussumma ei vastaa muistiin tallennettua, muuttujien tallennuksen yhteydessä laskettua tarkistussummaa, U-Boot hylkää luetun tiedon viallisena (konsolille tulostuu virheilmoitus ”*** Warning - bad CRC, using default environment”) ja U-Boot ottaa käyttöön ohjelman käännösvaiheessa määritellyt kiinteät oletusmuuttujat. Oletusmuuttujat ovat tämän jälkeen olemassa vain RAM-muistissa. Muuttujat pitää tallentaa pysyväismuistiin ’saveenv’-komennolla, jotta ne tulisivat käyttöön seuraavalla käynnistyskerralla.

U-Bootissa on myös mahdollista ottaa käännösvaiheessa käyttöön ympäristömuuttujien varmuuskopiointiominaisuus. Kun ominaisuus on käytössä, U-Bootille määritellään kaksi muistialuetta kaksille ympäristömuuttujille. U-Bootilla on oma osoittimensa, joka osoittaa aina kulloinkin käytössä olevaan muistialueeseen. Aina kun ajetaan ’saveenv’-komento U-Bootin kehotteessa (tai ’fw_setenv’ Linuxissa), vaihdetaan ensin osoitin toiseen muistialueeseen ja kirjoitetaan uudet muuttujat sinne. Jos kirjoituksessa tapahtuu virhe, esimerkiksi sähköt katkeavat kesken kirjoituksen, seuraavalla tietojen lukukerralla vaihdetaan käyttöön edellinen, aikaisemmin käytössä ollut muistialue. Uusimpia muuttujia ei siis ole tallennettuna kahteen paikkaan, vaan ei-aktiiviseen muistialueeseen on tallennettuna edelliset ympäristömuuttujat. Identtinen kopio muuttujista molemmille alueille saadaan ajamalla ’saveenv’-komento kahteen kertaan.

Sen lisäksi, että ympäristömuuttujat ovat muuttujia, jolla on arvo, U-Bootissa muuttujien arvoksi voidaan antaa komento tai komentoja ja kyseinen muuttuja voidaan suorittaa ’run’-komennolla. Tällä tavalla U-Bootiin saa helposti lisättyä toiminnallisuutta ja logiikkaa. Tästä on hyötyä erityisesti käynnistysvalintoja tehdessä. U-Bootissa on myös mukana käännettävissä oleva bash-tyylinen komentokieli nimeltään hush, jossa on käytettävissä monia bashin rakenteita, kuten silmukoita, ehtolauseita ja testejä.

Raskaampaan U-Bootin ohjelmointiin ja automatisointiin on käytettävissä U-Bootin lähdekoodin mukana sisäisiksi komennoiksi käännettävät omat komennot, erikseen käännettävät ja ajon aikana flash-muistilta, verkosta, sarjaportista tai muista lähteistä ladattavat C-kieliset ohjelmat sekä suoritettavaksi tiedostoksi muutetut hush-komentosarjat.

Esimerkissä 6 on esitetty, kuinka U-Bootin ympäristömuuttujaan tallennetaan sarja komentoja ja sen jälkeen komentosarja suoritetaan kutsumalla kyseessä olevaa muuttujaa. Esimerkissä luodaan muuttuja ’loadcheck’, joka sisältää seuraavat komennot:

- Linuxin ytimen sisältävän tiedoston (uImage) lataaminen (’fsload’) muistiin. Muistiosoite tallennettu muuttujaan ’loadaddr’.
- Komento (’iminfo’), jolla U-Boot tarkistaa ladatun tiedoston eheyden.
- Informatiivisen viestin tulostaminen käyttäjälle tiedoston tarkastuksen lopputuloksesta.

```
=> setenv loadcheck 'fsload $loadaddr uImage; if iminfo $loadaddr; ↵
then echo "image ok"; else echo "image not ok"; fi'
=> run loadcheck
### JFFS2 loading 'uImage' to 0x800000
### JFFS2 load complete: 2242412 bytes loaded to 0x800000

## Checking Image at 00800000 ...
   Legacy image found
   Image Name:   Linux-board2
   Created:      2011-10-20   6:29:06 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    2242348 Bytes =  2.1 MB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
image ok
=>
```

Esimerkki 6. U-Bootin ympäristömuuttujaan tallennetun komentosarjan suorittaminen.

4.3 Linuxin käynnistäminen

Sulautetussa järjestelmässä käynnistyslataaja alustaa järjestelmän toiminnalle olennaiset laitteiston osat, kuten muistit ja keskeytykset sekä massamuistin tai verkkoyhteyden käyttöjärjestelmän lataamista varten. Tämän jälkeen käynnistyslataaja lukee tarvittavat tiedostot ja käynnistää käyttöjärjestelmän. Käynnistyessään Linux-

ytimen täytyy saada tietoa järjestelmästä löytyvistä väylistä ja niihin liittyvistä laitteista, kuten muistit, massamuistit, keskeytysohjaimet ja oheislaitteet, jotta se voi ladata ja ottaa käyttöön laitteiden toimintaa ohjaavat ja niitä käyttävät ajurit.

PC-tietokoneen laitekonfiguraation tieto saatiin ennen BIOSista, myöhemmin EFI:n ja siitä edelleen kehitetyn UEFI-rajapinnan kautta [29]. Yleiskäyttöisillä PowerPC-alustoilla suosituin rajapinta on Open Firmware (IEEE 1275) [30], jossa järjestelmän laitteisto kuvataan laitepuuna (device tree). Laitepuussa on solmuja, jotka kuvaavat laitetta, kuten prosessoria ja sen ominaisuuksia. Solmu voi olla myös virtuaalinen solmu, joka sisältää yhden tai useampia laitesolmuja, esimerkiksi localbus-solmu voi sisältää NOR- ja NAND-flash-laitteen, UARTin tai ethernet-ohjaimen. Kuvassa 3 on esimerkki laitepuusta.

```
/ o device-tree
      |- name = "device-tree"
      |- model = "MyBoardName"
      |- compatible = "MyBoardFamilyName"
      |- #address-cells = <2>
      |- #size-cells = <2>
      |- linux,phandle = <0>
      |
      o cpus
      | | - name = "cpus"
      | | - linux,phandle = <1>
      | | - #address-cells = <1>
      | | - #size-cells = <0>
      | |
      | o PowerPC,970@0
      |   |- name = "PowerPC,970"
      |   |- device_type = "cpu"
      |   |- reg = <0>
      |   |- clock-frequency = <5f5e1000>
      |   |- 64-bit
      |   |- linux,phandle = <2>
      |
      o memory@0
      | |- name = "memory"
      | |- device_type = "memory"
      | |- reg = <00000000 00000000 00000000 20000000>
      | |- linux,phandle = <3>
      |
      o chosen
        |- name = "chosen"
        |- bootargs = "root=/dev/sda2"
        |- linux,phandle = <4>
```

Kuva 3. Esimerkki laitepuusta [31].

Sulautetuissa PowerPC-järjestelmissä yleisimmin käytössä oleva käynnistyslataaja on U-Boot. U-Boot ei tarjoa Open Firmwaren mukaista rajapintaa järjestelmän laitteiston tunnistamiseen, mutta sen kanssa käytetään Open Firmwaresta johdettua laitepuuta, josta käytetään nimitystä Flattened Device Tree (FDT) [6, s. 187; 31].

PowerPC-alustalla tiedot laitteistosta voidaan viedä Linux-ytimelle kahdella tavalla. Jos järjestelmässä on firmware, joka on Open Firmware -yhteensopiva tai toteuttaa tarvittavat Open Firmware -rajapinnat, ytimelle voidaan antaa osoite rajapintaan ja ydin saa haettua kuvauksen laitteistosta firmwaren Open Firmware-laitepuusta. Jos taas järjestelmässä ei ole Open Firmwarea tai sen kanssa yhteensopivaa käynnistysohjelmistoa, kuten käytettäessä U-Boot-käynnistyslataajaa, käynnistyslataaja kopioi laitepuun muistiin lataamalla sen tiedostosta ja antaa tämän muistiosoitteen Linux-ytimelle, joka luo käyttöönsä tiedostona olevasta puusta Open Firmwaren mukaisen laitepuun.

Laitepuu käännetään ladattavaan binäärimuotoon dtc-kääntäjällä (device tree compiler), joka tulee Linuxin lähdekoodin mukana. Laitepuun tekstimuotoinen lähdekoodi, päätteeltään .dts (device tree source), sijaitsee Linux-ytimen lähdekoodissa PowerPC-suorittimen hakemistossa arch/powerpc/boot/dts/.

Sulautettu käännösympäristö ajaa yleensä dtc-kääntäjän Linux-ytimen kääntämisen yhteydessä. Tuloksena saatu binäärimuotoinen laitepuutiedosto on yleensä päätteeltään .dtb (device tree blob). Laitepuutiedosto voidaan kääntämisen jälkeen siirtää käynnistyslataajan saataville tallentamalla tiedosto flash-muistiin tai lataamalla se keskusmuistiin TFTP:llä tai sarjaportin kautta.

Peruskonfiguraatiossa U-Boot suorittaa alkuviiveen jälkeen bootcmd-nimisen komentojonon. Mikäli järjestelmää halutaan konfiguroida, U-Bootissa on monipuoliset mahdollisuudet muuttaa käynnistymiseen ja käyttöjärjestelmän lataamiseen liittyviä ominaisuuksia ja toimintoja. Konfiguroitavia toimintoja ovat muun muassa seuraavat:

- viive, jonka aikana automaattinen käyttöjärjestelmän lataaminen voidaan keskeyttää ja siirtyä komentokehotteeseen,
- komento, joka voidaan suorittaa ennen viivettä ja
- varsinainen käynnistyskomento ja vaihtoehtoinen käynnistystapa, kuten käynnistyslaskuri (ks. 6.7 Käynnistyslaskuri).

5 Prosessorikortti

Insinöörityöni käsittelee Netcontrol Oy:n kehittämää sulautetun järjestelmän prosessorikorttia ja sen hallittua etäpäivittämistä. Prosessorikortti on tarkoitettu liitettäväksi Netcontrolin eri tuotteiden emokortteihin. Emokortit eroavat toisistaan käyttötarkoituksen mukaan, ja niiden eroja ovat on mm.

- sarjaporttien lukumäärä
- SPI-väylien laitteet
- GPIO-kytkennät sekä
- ethernet-porttien ja -kytkinpiirien konfiguraatiot.

Korttien erot voidaan ottaa huomioon konfiguroimalla Linux-ydin ja oheisohjelmat jokaista emokorttia varten, jolloin sama prosessorikortti käy kaikkiin emokortteihin.

5.1 Prosessorikortin rakenne

Prosessorikortin hallittuun etäpäivittämiseen tarvitaan kortilla olevista piireistä ja komponenteista vain seuraavia:

- Freescale PowerQUICC II Pro-perheen PowerPC-suoritin
- 256 Mt RAM-muistia ja
- massamuisteina kortilla käytetty 4 Mt:n NOR-flash sekä 256 Mt:n NAND-flash.

Prosessorikortti liitetään varsinaiseen emokorttiin siinä olevalla liittimellä. Insinöörityössäni olen käyttänyt Netcontrolin kehittämää emokorttia, johon saa järjestelmän ulkopuolelta yhteyden sarjakonsoliportin sekä ethernet-verkkoliitännän kautta.

Prosessorikortilla käytetään käynnistyslataajana U-Bootin versiota 2010.03, joka on valittu kortin aikaisemman kehitystyön yhteydessä. Käyttöjärjestelmänä perustuu 2.6.x-sarjan Linux-ytimeen. Sekä käynnistyslataajaan että Linux-ytimeen on tuotu ominaisuuksia ja korjauspäivityksiä ohjelmistojen uudemmista versioista. Käytetty

Linux-jakelu on nimeltään LTIB (Linux Target Image Builder), joka on BSP-kehitystä (Board Support Package) varten tehty työkalu.

5.2 Flash-muistien käyttö prosessorikortilla

NOR-flash-muistissa on tallennettuna käynnistyslataaja sekä sen käyttämä ympäristömuuttujadata. NOR-muistia käytetään formatoimattomana, joten siihen ei ole tallennettuna tiedostojärjestelmää. NOR-flash on konfiguroitu suorittimen osoiteavaruuteen siten, että suoritin alkaa käynnistyessään ajaa käynnistyslataajaa NOR-flashilta. Kaikille emokorteille yhteisillä perusasetuksilla varustettu U-Boot tallennetaan tyhjälle NOR-flashille JTAG-liitynnän kautta. U-Boot voidaan päivittää käynnistämällä kortti U-Bootin kehotteeseen ja antamalla U-Bootissa komentorivillä komennot, joilla keskusmuistiin ladattu uusi U-Boot kirjoitetaan NOR-flash-muistiin nykyisen päälle. Tiedosto voidaan siirtää keskusmuistiin joko TFTP:llä, sarjaportin kautta tai se voidaan ladata NAND-flashilla olevasta tiedostojärjestelmästä. Jos päivityksen yhteydessä komentoriviltä kirjoitettaessa tapahtuu virhe tai ladattu U-Boot-binääri on muuten viallinen niin, ettei U-Boot enää käynnisty, on U-Boot palautettava flash-muistille JTAG:n kautta.

Kun prosessorikortti otetaan käyttöön, se käynnistetään U-Bootin komentokehotteeseen. Sen jälkeen kehitysympäristön luoma käyttöjärjestelmän sisältävä JFFS2-levykuvatiedosto ladataan TFTP:llä lähiverkon palvelimelta keskusmuistiin, ja sen jälkeen levykuva kirjoitetaan NAND-flashille. Lataamiseen ja levykuvan kirjoitukseen on U-Bootissa omat komennot. Prosessorikortin NAND-flash-muisti on jaettu kahteen osioon, joista ensimmäiseen on kirjoitettu levykuva, josta käyttöjärjestelmä ladataan.

NOR- ja NAND-flash-muistit näkyvät Linuxissa mtd-laitteina, NOR-flash mtd0-nimisenä laitteena ja kahteen osioon jaettu NAND-flash mtd1- ja mtd2-nimisinä laitteina. Koska NAND-flash-muistissa on käytössä JFFS2-tiedostojärjestelmä, sen sisältöä voidaan muokata niin kuin mitä tahansa tiedostoja Linuxin komennoilla ja järjestelmäkutsuilla. NOR-flash-muisti sen sijaan näkyy vain 4 Mt:n yhtenäisenä muistialueena ilman erityisempää rakennetta. NOR-flash-muistin sisältöä ei siis voi muokata Linuxista tavallisilla tiedostojen käsittelyyn tarkoitetuilla komennoilla. Insinöörityöni aluksi muutin

NAND-flash-muistin konfiguraatiota siten, että jaoin muistin neljään osioon, joista kaksi varattiin Linuxin käynnistystiedostoja varten.

5.3 Linuxin käynnistyminen

Kun käynnistyslataaja on käynnistynyt, se lukee ’fsload’-komennolla NAND-flash-muistissa olevasta JFFS2-tiedostojärjestelmästä Linux-ytimen sekä laitepuutiedoston. Tämän jälkeen U-Boot käynnistää ytimen ’bootm’-komennolla antaen komennolle parametreiksi muistiosoitteet, joihin Linux-ydin ja laitepuu on ladattu. Linux saa komentoriville parametreinaan laitenimen, jossa juurihakemisto sijaitsee, konsolisarjaportin asetukset, NAND-flash-muistin osioinnin sekä tiedot käynnistystyypistä. Esimerkissä 7 on lueteltu Linux-ytimelle käynnistyksessä annetut komentoriviparametrit.

```
root=mtd1 rw console=ttyS0,115200 ↲
mtdparts=nand:128M(rootfs),12M(bootfs),12M(boot2),-(rest) ↲
rootfstype=jffs2 bootpart=1 bootmode=normal
```

Esimerkki 7. Linuxin ytimelle sen käynnistyksessä annetut komentoriviparametrit.

5.4 Osioiden määrittäminen

Toisin kuin esimerkiksi tavallisilla kiintolevyillä, flash-muistiin ei ole tallennettuna osiotaulukkoa, josta muistissa sijaitsevien osioiden alku- ja loppukohdat voitaisiin lukea. Muistia käyttävällä järjestelmällä tai ohjelmistolla, kuten U-Boot ja Linux, on oltava osiointitieto itsellään. Ohjelmistolle pitää esimerkiksi kertoa, että käytössä on 256 Mt:n NAND-flash, jossa on kaksi 128 Mt:n osiota. Poikkeuksen tekee muun muassa RedBoot-käynnistyslataaja, joka varaa flash-muistista yhden tai useampia lohkoja, joihin osiotiedot tallennetaan ja mistä käyttöjärjestelmä ne lukee.

Tiedot flash-muistin (mtd-laitteiden) osioista voi välittää Linuxille usealla eri tavalla [6, s. 67]. Insinöörityössäni käyttämälläni prosessorikortilla mtd-laitteiden osiot määritellään laitepuussa, jonka käynnistyslataaja lataa RAM-muistiin ja välittää Linux-ytimelle sen käynnistämisen yhteydessä. U-Boot lataa laitepuutiedoston ja Linux-

ytimen muistiin ennen Linuxin käynnistämistä ja käynnistyskomennon parametrina annetaan muistiosoite, johon laitepuu on ladattu.

Linux-ytimelle voidaan antaa tieto flash-muistin osioinnista laitepuun sijasta myös komentoriviparametrina, esimerkiksi ”`mtdparts=nand:128M(rootfs),12M(boot),12M(boot2),-(rest)`”. Jos tiedot mtd-laitteiden osioista välitetään Linux-ytimelle komentorivillä, ominaisuus täytyy kääntää Linux-ytimeen. Ominaisuus on nimeltään MTD_CMDLINE_PARTS. Tämän lisäksi käytettävän flash-muistin ajurin täytyy lukea osiotiedot komentoriviltä siihen tarkoitetulla funktiolla.

On tärkeää huomata, että edellä oleva koskee vain Linux-ytimen komentoriviparametria. U-Boot säilyttää flash-muistin osiotiedot samannimisessä ’mtdparts’-ympäristömuuttujassa. Muuttujassa määriteltyjä osiotietoja käyttävät ne U-Bootin komennot, jotka käsittelevät flash-muistia. Tällaisia komentoja ovat esimerkiksi ’fsload’ (tiedoston luku muistiin flashilla olevalta tiedostojärjestelmältä) sekä ’nand’ (muistin tyhjennys ja siihen kirjoitus).

Insinöörityössäni käytetty NAND-flash on jaettu neljään osioon taulukon 2 mukaisesti.

Taulukko 2. NAND-flash-muistin osiojako.

Osion nimi	**Osion koko**	**Käyttötarkoitus**
rootfs	128 Mt	päätiedostojärjestelmä
boot	12 Mt	1. käynnistystiedosto-osio
boot2	12 Mt	2. käynnistystiedosto-osio
rest	104 Mt	vapaa tila

Esimerkissä 8 on esitetty osiotietojen näkyminen U-Bootin ’mtdparts’-ympäristömuuttujassa.

```
=> printenv mtdparts
mtdparts=mtdparts=nand:128M(rootfs),12M(bootfs),12M(boot2),-rest
```

Esimerkki 8. Osiointien katsominen U-Bootissa.

Linuxin rekisteröimät mtd-laitteet voidaan lukea /proc/mtd-tiedostosta esimerkin 9 mukaisesti. Tulostuksesta nähdään mtd-laitteiden numerot, osioiden koot ja laitteiden lohkokoot sekä osioille annetut nimet.

```
$ cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00400000 00010000 "bootflash"
mtd1: 08000000 00020000 "rootfs"
mtd2: 00c00000 00020000 "boot"
mtd3: 00c00000 00020000 "boot2"
mtd4: 06800000 00020000 "rest"
```

Esimerkki 9. Linuxin rekisteröimien mtd-laitteiden luettelointi.

Kaikki flash-muistien osiot luetellaan Linuxin rekisteröimässä järjestyksessä. Yksittäisten mtd-laitteiden muistityyppiä ei näe vain katsomalla /proc/mtd-tiedoston sisältöä. Tiedostosta ei myöskään selviä eri osioiden sijainti muisteissa, jos järjestelmässä on useampia kuin yksi flash-muisti. Osionti täytyy olla tiedossa ennen kuin mtd-laitetta voi käyttää, koska esimerkiksi NOR- ja NAND-flashiin kirjoitus tapahtuu eri komennoilla. NOR-muistiin kirjoitetaan ’flashcp’-komennolla ja NAND-muistiin kirjoitetaan ’flash_write’-komennolla.

Esimerkin 9 komento on ajettu insinöörityöni prosessorikortilla. Kun kortin konfiguraatio on tiedossa (NOR 4 Mt ja NAND 256 Mt), listauksesta nähdään, että mtd0 on NOR-flash, jonka lohkokoko on 64 kt. Laitteet mtd1—mtd4 sijaitsevat NAND-flashilla, jonka lohkokoko on 128 kt, ja NAND-flashin eri osioiden koot ovat järjestyksessä 128 Mt, 12 Mt, 12 Mt ja 104 Mt.

Ennen kuin U-Bootissa voidaan ajaa ’fsload’-komento, käytettävä osio täytyy valita ’chpart’-komennolla, jos se ei ole oletuksena oleva ensimmäinen osio. Komennolle annetaan parametreina muistin tunnus ja osion numero, esimerkiksi ’chpart nand0,0’. Jokaisen NAND-flash-muistin osioiden numerointi alkaa nollasta. Tämä ero täytyy muistaa verrattuna Linuxin mtd-laitteiden numerointiin, jossa kaikki flash-muistit (NOR ja NAND) sekä niiden osiot luetellaan peräkkäin juoksevana numerosarjana. Esimerkiksi edellä olevassa muistikonfiguraatiossa U-Bootissa NAND-muistin ensimmäinen osio on ’nand0,0’, mutta Linuxissa sama osio on mtd1, koska mtd0 on NOR-flash. Insinöörityössä käytetyllä prosessorikortilla NAND-flashin neljä osioita (rootfs, boot, boot2 ja rest) on määritelty laitepuun oksina esimerkin 10 mukaisesti.

```
nand@1,0 {
    #address-cells = <0x1>;
    #size-cells = <0x1>;
    device_type = "board1-nand";
    reg = <0x1 0x0 0x10000>;
    nandready-gpiochip = <0x2>;
    nandready-pin = <0x7>;
    rootfs@0 {
        reg = <0x0 0x8000000>; /* 128 MB */
    };
    boot@8000000 {
        reg = <0x8000000 0xc00000>; /* 12 MB */
    };
    boot2@8c00000 {
        reg = <0x8c00000 0xc00000>; /* 12 MB */
    };
    rest@8000000 {
        reg = <0x9800000 0x6800000>; /* 104 MB */
    }
}
```

Esimerkki 10. NAND-flash-muistin osioiden esitys laitepuun oksina.

6 Prosessorikortin käynnistyksen ja päivityksen kehitys

Insinöörityössäni oli tavoitteena kehittää prosessorikortin käynnistystoimintoja siten, että käyttöjärjestelmää olisi mahdollista etäpäivittää ja varmistaa päivityksen vikasietoisuus. Tavoitteena oli myös kortin käynnistysprosessin kehittäminen ja nopeuttaminen.

6.1 Kehitystyön osa-alueet

Kortin toiminnan voi ajatella jakautuvan kahteen osaan: käynnistyslataajaan sekä käyttöjärjestelmään. Osilla on toimintoja, jotka ovat yhteisiä ja muutokset toisessa vaikuttavat myös toiseen osaan. Tämän lisäksi jotkut toiminnot koskevat vain toista osaa.

Kehitystyön kohteet voidaan jakaa seuraavasti:

- käynnistystiedostojen lataaminen ja päivittäminen
- käynnistysprosessin nopeuttaminen
- virhetilanteiden hallinta käynnistyksessä

- käynnistyslataajan perusasetukset.

6.2 Käynnistystiedostojen lataaminen ja päivittäminen

Käynnistystiedostojen päivittämisessä on tärkeää, että mahdollisen häiriön sattuessa voidaan palata edelliseen, toimivaan kokoonpanoon. Lähtötilanteessa tällaista mahdollisuutta ei ollut, vaan päivittäminen on tehty ylikirjoittamalla käynnistystiedostot. Alkuperäisessä konfiguraatiossa NAND-flash-muisti oli jaettu puoliksi. Ensimmäisessä osiossa oli Linuxin koko tiedostojärjestelmä, mukaan lukien /boot/-hakemisto, jossa Linuxin ydin ja laitepuutiedosto sijaitsivat. Jälkimmäinen osio oli käyttämättömänä ja alustamattomana.

Turvallista päivittämistä lähdettiin ratkaisemaan siten, että tiedostoista pidetään kahta versiota, joista toisen voi päivittää niin, että on mahdollisuus vielä palata edelliseen versioon. Päivitys täytyy pystyä tekemään Linuxissa, jotta uusien tiedostojen kopiointi ja asennus voidaan tehdä myös etäyhteyden kautta.

Päivityksen kannalta erillisen käynnistysosion tai -osioiden tekeminen käynnistystiedostoille on järkevää kahdesta syystä:

- Järjestelmän käynnistymisen nopeutuu, koska käynnistyslataaja pystyy lukemaan Linuxin ytimen ja laitepuutiedoston pieneltä (n. 10 Mt) JFFS2-osiolta nopeammin kuin suurelta osiolta (128 Mt).
- Yksittäisten tiedostojen korvaamisen sijasta voidaan kirjoittaa käynnistysosion päälle valmis levykuva, jossa on uudet käynnistystiedostot.

Ensimmäinen pienen käynnistysosion etu, nopea käynnistyminen, kävi ilmi tehdyssä kokeessa, jossa Linuxin ytimen lataus noin 65 Mt:n kokoiselta rootfs-osiolta vei U-Bootissa noin minuutin ja 50 sekuntia, kun taas erilliseltä boot-osiolta lataaminen tapahtui 6 sekunnissa.

Toinen etu pienestä käynnistysosiosta saadaan käynnistystiedostojen päivittämisessä. Tiedostot voidaan päivittää kirjoittamalla joko uudet käynnistystiedostot sisältävä levykuva tai yksittäiset uudet käynnistystiedostot käynnistysosiolle. Kun käytetään levykuvan päivitystapaa, levykuva voidaan siirtää laitteelle joko kopioimalla se verkon kautta tai lukemalla paikan päällä muistikortilta. Tämän jälkeen levykuva saadaan

kirjoitettua flash-muistiin Linuxin standardityökaluilla ’flash_eraseall’ ja ’nandwrite’. Yksittäistiedostojen päivitystapa vaatii lisäksi osion tyhjennyksen Linuxissa ’flash_eraseall’-komennolla, ennen kuin uudet tiedostot voidaan kopioida osiolle. Pelkästään uusien tiedostojen kopiointi vanhojen päälle, osiota ensin tyhjentämättä, ei toimi. Tämä johtuu siitä, että käytetty U-Boot ei osaa lukea JFFS2-osiolta virheettömästi tiedostoja, joiden päälle on tallennettu samalla nimellä uusi tiedosto. Kääntämällä U-Boot "CONFIG_SYS_JFFS2_SORT_FRAGMENTS"-käännösoptiolla [32] tiedostojen luku onnistuisi. Tämä kuitenkin hidastaa JFFS2-osioiden lukemista niin paljon, että käännösoption käyttö ei kannata [33].

U-Bootin ’fsload’-komento lataa NAND-flash-muistilla olevasta tiedostojärjestelmästä tiedoston keskusmuistiin. Oletuksena U-Boot käyttää NAND-flash-muistin ensimmäistä osiota. Jos flash-muistilla on useita osioita ja tiedostot sijaitsevat muulla kuin oletusosiolla, täytyy käytettävä osio valita ennen ’fsload’-komennon ajamista. Osion valinta tehdään ’chpart’-komennolla. Komennolle annetaan parametrina NAND-muistin tunnus ja osion numero. Jos ’chpart’-komentoa ei ajeta, on oletuksena ensimmäisen NAND-flash-muistin (nand0) ensimmäinen osio (0). Esimerkissä 11 määritellään ’chpart’-komennossa ensimmäisen NAND-flash-muistin toinen osio ja ladataan sieltä keskusmuistiin ’fsload’-komennolla /boot/uImage-niminen tiedosto.

```
=> chpart nand0,1
=> fsload 0x800000 /boot/uImage
```

Esimerkki 11. ’chpart’- ja ’fsload’-komentojen käyttö Linuxin ytimen lataamisessa tiedostosta.

Jotta vanha ja uusi versio käynnistystiedostoista voisivat olla yhtä aikaa tallennettuna flash-muistiin, tein flash-muistiin kaksi osiota käynnistystiedostoja varten. Osioista toinen on aina aktiivinen, ja U-Boot lataa järjestelmän käynnistyksessä käynnistystiedostot tältä aktiiviselta osiolta. Aktiivisen käynnistysosion tallentaminen U-Bootissa on tehty käyttämällä ympäristömuuttujaa, johon tallennetaan käytettävän osion numero. Ympäristömuuttujan nimeksi on valittu ’bootpart’. Muokattavuuden ja joustavan käytön takia myös muut tiedot, kuten käynnistystiedostojen nimet, hakemistot ja muistiosoitteet on tallennettu ympäristömuuttujiin. U-bootin komennot ja komentosarjat käyttävät näitä muuttujia kiinteästi määrättyjen arvojen sijasta. Näin U-Bootin toimintaa voidaan muuttaa antamalla muuttujille uusia arvoja. Kun muuttujille

annetaan arvot ’bootpart=1’, ’loadaddr=0x800000’, ’bootdir=/boot/’ ja ’bootfile=uImage’, voidaan Linuxin ydin ladata komennoilla

```
=> chpart nand0,$bootpart
=> fsload $loadaddr $bootdir$bootfile
```

Sen jälkeen, kun käynnistystiedostot on päivitetty Linuxissa uudelle käynnistysosiolle, uusi osio valitaan käyttöön tallentamalla U-Bootin ’bootpart’-ympäristömuuttujalle uusi arvo Linuxin ’fw_setenv’-ohjelmalla. Käynnistysosio saadaan vaihdettua 2:ksi Linuxissa komennolla

```
$ fw_setenv bootpart 2
```

Tällöin seuraavassa käynnistyksessä U-Boot lataakin käyttöjärjestelmän uudelta osiolta. Osion vaihtoa varten kirjoitettiin Linuxiin ’chbootpart.sh’-niminen bash-skripti. Skriptillä voidaan vaihtaa käynnistysosio toiseksi (1:stä 2:ksi tai 2:sta 1:ksi), asettaa haluttu osio parametrilla tai katsoa mikä on nykyinen valinta:

`chbootpart.sh change`
vaihtaa nykyisen osion. Jos nykyinen osio on 1, vaihdetaan se 2:ksi ja päinvastoin.

`chbootpart.sh set <part>`
asettaa <part>-määrityksen aktiiviksi käynnistysosioksi.

`chbootpart.sh show`
näyttää nykyisen käynnistysosion numeron.

6.3 JFFS2 yhteenvetotietojen vaikutus käynnistykseen

Alkuperäisessä konfiguraatiossa suurin yksittäinen aikaa vievä osa käynnistyksessä on käynnistyslataajan Linuxin käynnistystiedostojen lukemiseen kulunut aika. Tämä johtuu suuresta tiedostojärjestelmästä, jolta tiedostot ladataan. Käyttämällä pienikokoista, vain käynnistystiedostoille tarkoitettua osiota, saadaan lataamiseen kuluvaa, häiriöaltista aikaa lyhennettyä huomattavasti.

Alaluvussa 3.2.3 on kuvattu JFFS2-tiedostojärjestelmän yhteenvetotieto-ominaisuus, joka nopeuttaa JFFS2-tiedostojärjestelmän liittämistä Linuxissa.

Käynnistyksen yhteydessä tiedostojärjestelmää luetaan kahdesti. Ensin käynnistyslataaja lukee käynnistystiedostot keskusmuistiin NAND-flash-muistin tiedostojärjestelmästä jonka jälkeen se käynnistää Linuxin. Toisen kerran tiedostojärjestelmää luetaan, kun Linux käynnistyessään ottaa tiedostojärjestelmän käyttöönsä ja liittää sen juurihakemistoon.

6.3.1 JFFS2-yhteenvetotiedot U-Bootissa

Prosessorikortilla käytettävään U-Bootin versioon 2010.03 voidaan kääntää mukaan tuki JFFS2-yhteenvetotiedoille. Tämän käyttöönotto vaatii kuitenkin, että myös toinen käännösvalinta ”CONFIG_SYS_JFFS2_SORT_FRAGMENTS” otetaan käyttöön. Tämä saattaa hidastaa käynnistystiedostojen lukemista huomattavasti etenkin suurelta JFFS2-osiolta, koska käännösvalinnan kanssa JFFS2-tiedostojärjestelmän läpikäyntiaika kasvaa eksponentiaalisesti luettavien tietojen määrän kasvaessa [32]. Koska kortilla on käytetty erillisiä pienikokoisia osioita käynnistystiedostoja varten, on käynnistystiedostojen luku nopeampaa. Edellä kerrotun perusteella yhteenvetotietoominaisuuden käyttöönottaminen U-Bootissa ei kannata.

6.3.2 JFFS2-yhteenvetotiedot Linuxissa

Yhteenvetotietojen tuki otetaan käyttöön Linuxissa valitsemalla se JFFS2-tiedostojärjestelmäajuriin. Kun tuki on mukana, se nopeuttaa JFFS2- yhteenvetomerkinnöillä varustetun tiedostojärjestelmän liittämistä hakemistopuuhun. Käytetyllä prosessorikortilla Linux ottaa NAND-flashilla sijaitsevan tiedostojärjestelmän käyttöön käynnistyksessä.

Yhteenvetotiedon vaikutusta tiedostojärjestelmän liittämisnopeuteen mitattiin Linuxissa erikokoisten ja sisällöltään erilaisten tiedostojärjestelmien kanssa. Vertailtavat tiedostojärjestelmät tehtiin ajamalla työpöytäkoneen Linuxissa ohjelma, joka loi hakemistopuun ja sen sisälle tiedostoja, kunnes hakemistopuu oli halutun kokoinen. Tämän jälkeen puusta luotiin JFFS2-levykuva ’mkfs.jffs2’-komennolla. Seuraavaksi levykuvasta tehtiin kopio, johon kirjoitettiin ’jffs2dump’-komennolla yhteenvetotiedot alaluvun 2.5.3 esimerkin 4 mukaisesti.

Prosessorikortilla oli käytössä Linux, johon oli otettu mukaan tuki JFFS2-yhteenvetotiedoille. Flash-muistin rest-osiolle kirjoitettiin vuorollaan kukin edellä tehty

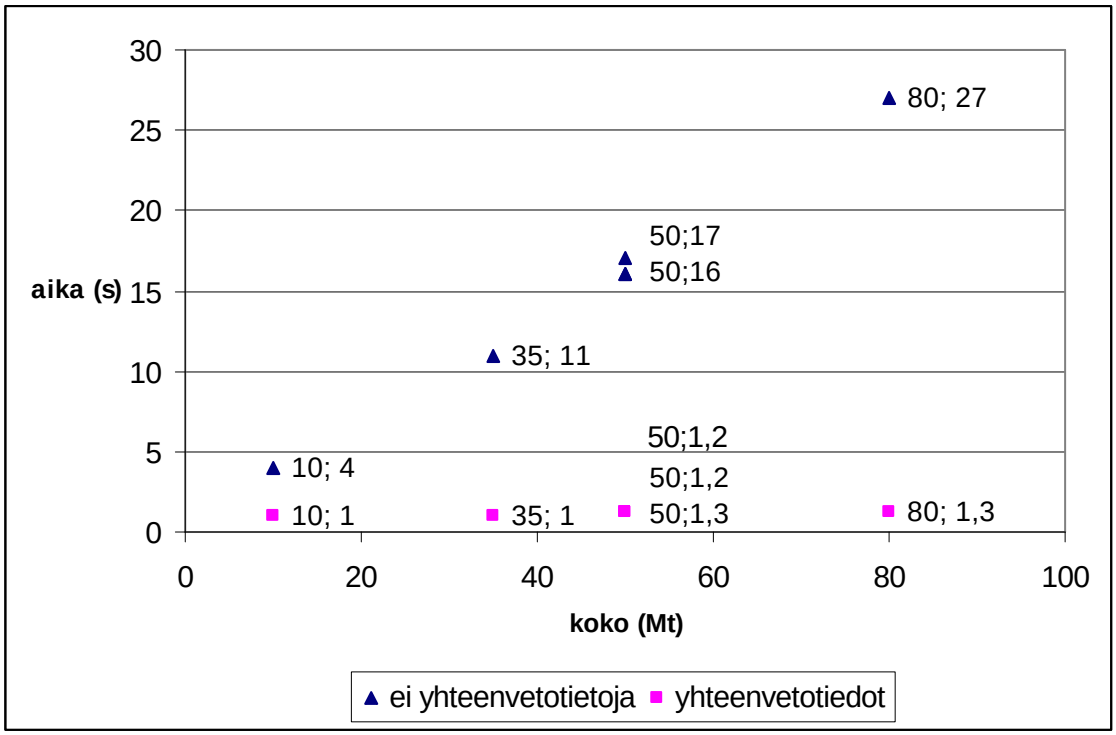
levykuva Linuxin ’nandwrite’-ohjelmalla. Tämän jälkeen mitattiin, kauanko aikaa kului, kun rest-osio liitetään /mnt/rest-hakemistoon. Tämä tehtiin ajamalla komento ’time mount -t jffs2 mtd:rest /mnt/rest’. Tulokset on kerätty taulukkoon 3.

Taulukko 3. Yhteenvetotiedolla ja ilman tietoja olevan JFFS2-osion liittämiseen kulunut aika.

Tiedostojärjestelmän koko (lisäksi mahdollista lisätietoa rakenteesta)	**liittämiseen kulunut aika**	
	ilman yhteenvetotietoa	**yhteenvetotiedon kanssa**
10 Mt	4 s	1 s
35 Mt (219 tiedostoa, 230 hakemistoa, tiedostojen koko 0—240 kt)	11	1 s
50 Mt (422 tiedostoa, 301 hakemistoa, tiedostojen koko 0—240 kt)	16 s	1,2 s
50 Mt (50 kpl 1 Mt:n tiedostoja)	17 s	1,2 s
50 Mt (500 kpl 100 kt:n tiedostoja)	16 s	1,3 s
80 Mt	27 s	1,3 s

Tuloksista havaitaan, että ilman yhteenvetotietoa tiedostojärjestelmän liittämiseen kuluva aika kasvaa tiedostojärjestelmän koon mukaan lineaarisesti. Jos taas yhteenvetotieto on tallennettuna tiedostojärjestelmään, liittämisnopeus pysyy tasaisesti 1—1,5 sekunnin tuntumassa.

Yhteenvetotietojen vaikutusta koko järjestelmän käynnistymiseen mitattiin tekemällä kaksi identtistä 42 Mt:n kokoista JFFS2-levykuvaa, joista toiseen lisättiin yhteenvetotiedot. Sen jälkeen levykuvat kirjoitettiin vuorollaan NAND-flashin rootfs-osiolle ja otettiin aika siitä, kun U-Boot alkaa ladata Linuxin ydintä siihen, kun Linuxin Login-kehote tuli näyttöön. Ilman yhteenvetotietoa aikaa kului 43 sekuntia, kun taas yhteenvetotiedon kanssa latauksen alusta kehotteeseen aikaa kului 29 sekuntia. Eroa ajassa, joka kuluu tiedostojärjestelmän liittämiseen Linuxin käynnistymisessä, on 14 sekuntia. Kuvaan 4 sijoitettuna tiedostojärjestelmän koko 42 Mt ja sen liittämiseen kulunut aika 14 sekuntia osuu samalle suoralle kuin sarja ilman yhteenvetotietoja.



Kuva 4. Erikokoisten JFFS2-osioiden liittämisaika ilman yhteenvetotietoa (kolmio) ja yhteenvetotietojen kanssa (neliö).

6.4 Virhetilanteiden käsittely päivityksessä

Jos käynnistystiedostojen lataaminen epäonnistuu, voidaan virheentarkistusta lisäämällä käyttää mahdollista vaihtoehtoista käynnistystä ja tehdä ennalta määriteltyjä toimenpiteitä tilanteen korjaamiseksi.

Kun järjestelmän käynnistystiedostoja päivitetään, on tärkeää että, järjestelmä ei virheen sattuessa jää tilaan, josta sitä ei enää saa käyntiin. Tämän takia on otettu käyttöön kaksi erillistä osiota flash-muistissa. Toisella osiolla on käytössä olevat käynnistystiedostot ja päivityksessä uudet tiedot kirjoitetaan toiselle, ei käytössä olevalle osiolle. Päivityksen jälkeen käynnistyslataaja vaihdetaan käyttämään uusia, päivitettyjä tiedostoja ja aikaisemmin käytössä olleet vanhat tiedostot jäävät talteen. Näin on mahdollista palata käyttämään vanhoja tiedostoja, jos päivitys jostain syystä epäonnistuu.

Päivityksen ensimmäisessä vaiheessa uudet tiedostot kopioidaan ei-aktiiviselle käynnistysosiolle. Jos tässä kirjoitusvaiheessa tapahtuu sähkökatkos, josta johtuen järjestelmä käynnistyy äkillisesti uudelleen, käynnistyslataaja käynnistää järjestelmän vanhoilla käynnistystiedostoilla. Keskeytynyt päivitys täytyy havainnoida ohjelmallisesti, jotta päivitys voidaan tehdä uudelleen.

Kun tiedostot on onnistuneesti kopioitu käyttämättömälle käynnistysosiolle, käynnistyslataajan käyttämä osio vaihdetaan osoittamaan uusiin tiedostoihin.

6.5 Käynnistyslataajan perusasetukset

Koska kehityskohteena olevaa prosessorikorttia käytetään useassa eri tuotteessa, prosessorikortin käynnistyslataajaan tulisi saada kaikille tuotteille soveltuvat perusasetukset niin, että vasta asennettaessa käyttöjärjestelmää flash-muistiin tehdään tuotekohtaiset muutokset.

Prosessorikortille ladataan tuotantovaiheessa käynnistyslataaja, joka toimii prosessorikortin ollessa asennettuna minkä tahansa tuotteen emokorttiin. Käynnistyslataaja ei käsittele prosessorikortin ulkopuolella olevaa emokorttia mitenkään, joten prosessorikortin toiminnan kannalta laitteistot ovat identtiset. Vasta käyttöjärjestelmän pitää ottaa huomioon emokorttien erot.

Ensimmäisellä käynnistyskerralla käynnistyslataajan ympäristömuuttujia ei ole tallennettuna pysyväismuistiin, joten käynnistyslataaja ottaa käyttöön käännösvaiheessa konfiguroidut muuttujien oletusarvot. Oletusmuuttujat on valittu siten, että käyttöjärjestelmää asennettaessa käynnistyslataajan asetuksiin täytyy tehdä mahdollisimman vähän muutoksia. Käyttöjärjestelmän asennuksen yhteydessä muutokset tehdään käynnistyslataajan muuttujiin ja lopuksi kaikki muuttujat tallennetaan ’saveenv’-komennolla pysyväismuistiin, josta ne latautuvat seuraavassa käynnistyksessä. Tarvittavat korttikohtaiset muutokset ovat:

- Prosessorikortin ethernet-MAC-osoitteiden asetus. Suorittimessa on kaksi ethernet-porttia ja niille annetaan yksilölliset MAC-osoitteet.
- Käynnistystiedostojen nimien muuttaminen. Käyttöjärjestelmän laitepuutiedoston nimi on joko board1.dtb, board2.dtb jne. riippuen siitä, minkä tuotteen emokorttiin prosessorikortti liitetään.

6.6 Linuxin uudelleen käynnistyminen virhetilanteessa

Linux-ydin voi joutua virhetilaan, josta suorituksen jatkaminen ei ole mahdollista. Tällaisesta virhetilanteesta käytetään nimeä kernel panic [18, s. 168]. Tällöin ydin pysäyttää kaikkien ohjelmien suorituksen ja tulostaa virheilmoituksen konsolille. Näin vakava virhe voi johtua esimerkiksi laitteistoviasta tai virheellisesti toimivasta laiteajurista. Vikatilanteessa ohjelmien suorituksen jatkaminen voi aiheuttaa enemmän vahinkoa kuin se, että suoritus pysäytetään. Esimerkiksi massamuistin käytön jatkaminen vakavan virheen jälkeen voi johtaa tietojen menettämiseen.

Linux-ytimessä on asetus, jolla voidaan valita, halutaanko kernel panic -tilanteessa järjestelmä jättää pysäytettyyn tilaan vai käynnistää se tietyn ajan kuluttua automaattisesti uudelleen. Tälle asetukselle, nimeltään panic_timeout, annetaan numeerinen arvo, jolla valitaan haluttu toiminto. Arvolla 0 (nolla) järjestelmä jää kernel panic -tilanteessa pysäytettyyn tilaan. Nollaa suurempi arvo tarkoittaa sitä, kuinka monen sekunnin kuluttua virhetilanteesta ydin käynnistää järjestelmän uudelleen. Oletuksena arvo on 180 sekuntia. Asetusta voidaan muuttaa kirjoittamalla uusi arvo /proc/sys/kernel/panic-tiedostoon komennolla ’sysctl -e kernel.panic=<arvo>’ tai pysyvästi muokkaamalla /etc/sysctl.conf-tiedostoa, josta asetus luetaan järjestelmän käynnistyessä. Esimerkissä 12 on listattu Linux-ytimen tulostama kernel panic -virheilmoitus tilanteessa, jossa se ei järjestelmää käynnistettäessä löydä ’init’-nimistä ohjelmaa. Syynä voi olla esimerkiksi se, että ytimen käynnistysparametrina annettu juuritiedostojärjestelmän laitenimi on väärä tai tiedostojärjestelmä on vahingoittunut niin, ettei sen lukeminen onnistu.

```
Freeing unused kernel memory: 196k init
Warning: unable to open an initial console.
Kernel panic - not syncing: No init found.  Try passing init= option ↲
to kernel.
Rebooting in 180 seconds..
```

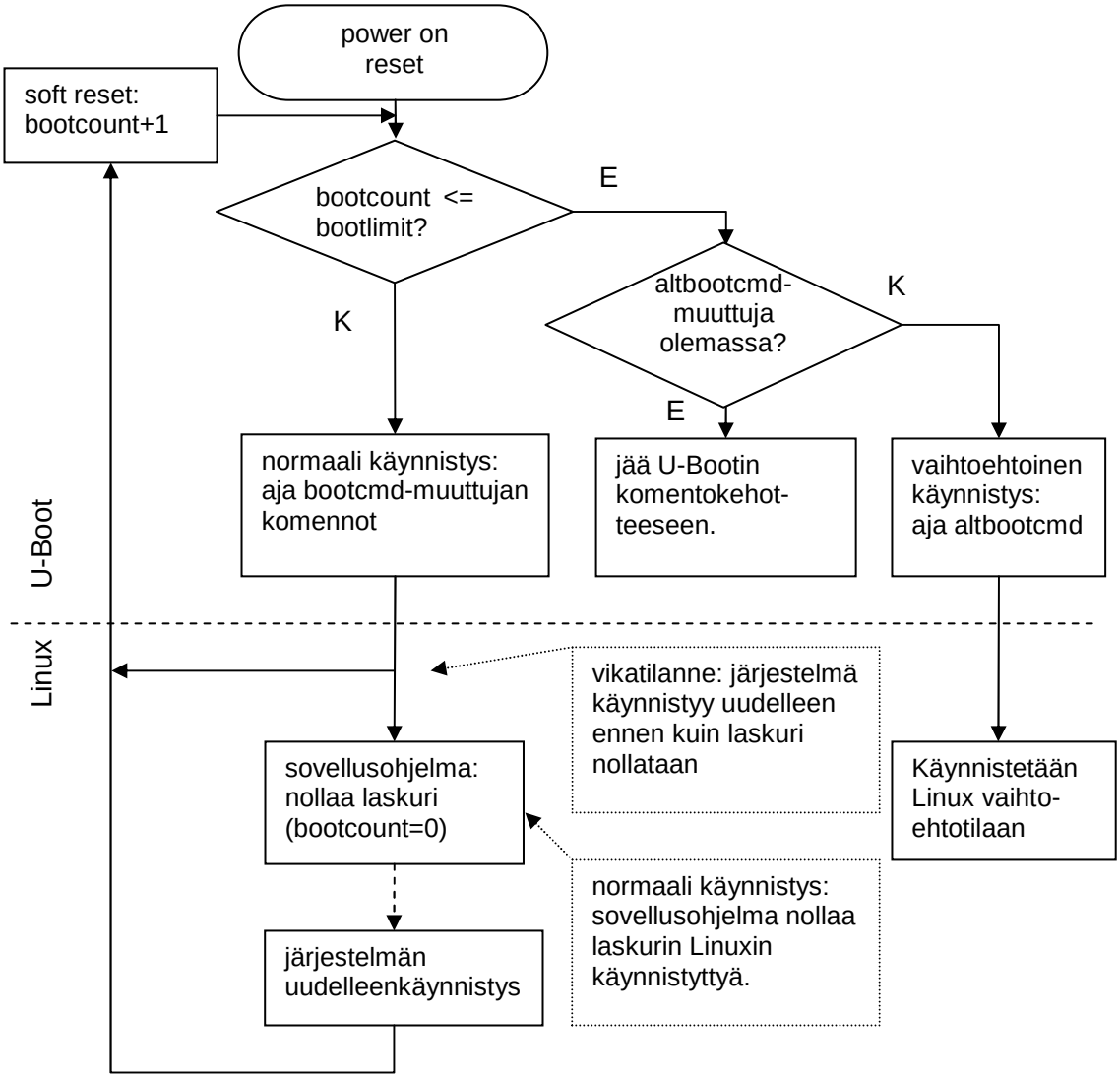
Esimerkki 12. Ytimen virheilmoitus tilanteessa, jossa ydin ei löydä ’init’-nimistä ohjelmaa.

6.7 Käynnistyslaskuri

U-Bootissa on virhetilanteista johtuvien toistuvien uudelleenkäynnistysten havaitsemiseen tarkoitettu Boot Count Limit -niminen toiminto. Tämän avulla U-Boot osaa suorittaa vaihtoehtoisen käynnistystoiminnon, jos se huomaa, että normaalin käynnistyksen jälkeen järjestelmä käynnistyy toistuvasti uudelleen virhetilanteesta johtuen.

Toiminto nimensä mukaisesti pitää laskurilla kirjaa siitä, montako kertaa järjestelmä on käynnistetty. Kun järjestelmään kytketään sähkö (power on reset), laskuriin tallennetaan arvo yksi. Jokaisessa suorittimen alustuksessa (reset) laskurin arvoa kasvatetaan yhdellä. Laskuri nollataan onnistuneen käyttöjärjestelmän käynnistyksen jälkeen. Nollaaminen tehdään ulkopuolisen sovelluksen avulla, esimerkiksi Linuxin käynnistämällä ohjelmalla.

Toiminto käyttää U-Bootissa ’bootcount’-nimistä ympäristömuuttujaa, joka sisältää käynnistyslaskurin arvon. Tämän lisäksi toiminto käyttää kahta muuta U-Bootin ympäristömuuttujaa, ’bootlimit’ ja ’altbootcmd’. Käyttäjä tallentaa ’bootlimit’-nimiseen muuttujaan tiedon, kuinka monta kertaa käynnistys saa tapahtua uudelleen eli kuinka suuren arvon ’bootcount’ saa saada. Jos ’bootcount’ on suurempi kuin ’bootlimit’, U-Boot ei tee normaalia käynnistystä, vaan vaihtoehtoisen käynnistyksen. Tässä tapauksessa, jos ’altbootcmd’-nimistä muuttujaa ei ole määritetty, U-Boot jää komentotilaan odottamaan käyttäjän toimenpiteitä. Jos taas U-Bootissa on ’altbootcmd’-muuttuja, U-Boot suorittaa muuttujan sisältämät komennot, jotka käynnistävät vaihtoehtoisen Linux-ytimen tai tiedostojärjestelmän. Käynnistyslaskurin toiminta on esitetty kuvassa 5.



Kuva 5. Käynnistyslaskurin toiminta.

Käynnistyslaskuriominaisuus valitaan U-Bootin käännösvaiheessa esikääntäjän '#define CONFIG_BOOTCOUNT_LIMIT'-määrittelyllä. Käytössä oleva U-Boot 2010.03 tukee laskuriominaisuutta vain MPC8xx-, MPC82xx-, MPC8360- ja MPC5200-suorittimissa. Näissä laskurin arvo tallennetaan suorittimesta riippuen joko muistikartoitettuun suorittimen sisäiseen muistiin tai rekisteriin, josta U-Boot lukee ja johon se kirjoittaa laskurin arvoa. Insinöörityössä käytetyllä prosessorikortilla olevassa prosessorissa U-Boot ei suoraan tue käynnistyslaskuria. Koska käytetty prosessori kuluu samaan 83xx-sarjaan kuin U-Bootissa tuettu MPC8360, käynnistyslaskuri on otettavissa käyttöön myös prosessorikortissa seuraavilla muutoksilla U-Bootin lähdekoodiin:

- cpu/mpc83xx/cpu.c: Sallitaan laskuri, jos suoritin on MPC8360 tai kuuluu MPC83xx-sarjaan.
- cpu/mpc83xx/fdt.c: Tehdään tarvittavat lisämäärittelyt, jos laskuri on otettu käyttöön ja valitussa suorittimessa on ”QUICC Engine” -lohko, joka on sekä MPC8360:ssä että MPC83xx:ssä.
- include/configs/board2.h: Otetaan laskuri käyttöön prosessorikortin konfiguraatiotiedostossa.

Käytetty patch-muutostiedosto on listattu liitteessä 2.

Prosessorikortin U-Bootissa laskuri toteutetaan käyttämällä prosessorin sisäistä muistia muistikartoituksen kautta (IMMR-Internal Memory Mapped Registers). Tällöin määritelty osoite suorittimen muistiavaruudessa osoittaa prosessorin sisäiseen muistiin, josta laskurin arvoa voidaan lukea ja kirjoittaa. Laskurin arvo säilyy suorittimen alustuksen (soft boot) jälkeen ja laskuria voidaan käsitellä sekä U-Bootista että Linuxista.

Linuxin puolella laskurin lukemisen ja kirjoittamisen voi tehdä kahdella tavalla: joko erillisellä ohjelmalla tai Linuxin ytimeen lisätyllä ajurilla, jonka proc-tiedostojärjestelmään luoman tiedoston kautta laskurin arvoon päästään käsiksi.

Käynnistyslaskurin käsittelyyn Linuxissa kirjoitettiin erikseen käännettävä apuohjelma, joka lukee muistista laskurin arvon tai nollaa sen. Ohjelma kopioidaan kohdelaitteelle asennuksen yhteydessä (’/usr/sbin/bootcount’) ja se ajetaan Linuxin käynnistysskriptistä /etc/rc.d/rc.local järjestelmän käynnistysvaiheessa. Jälkimmäiseen vaihtoehtoon eli ytimen ajuriin löytyy esimerkkilähdekoodi U-Bootin kotisivulta. Ajuri luo tiedoston /proc/driver/bootcount, josta voi lukea laskurin nykyisen arvon tai johon kirjoittamalla ”0” (nolla) laskuri nollataan.

6.8 Käytetyn käynnistysvaihtoehdon välittäminen Linuxille

Normaalitilanteessa käynnistyslataaja lataa valitulta käynnistysosiolta Linux-ytimen ja laitepuun keskusmuistiin ja siirtää suorituksen Linuxille. Poikkeustilanteissa, kuten käynnistyslaskurin ylittäessä sallitun ylärajan tai jos käynnistystiedostojen lataaminen valitulta käynnistysosiolta epäonnistuu, U-Boot lataa keskusmuistiin vaihtoehtoiset käynnistystiedostot ja käynnistää Linuxin niillä.

Vaihtoehtoisessa käynnistyksessä U-Bootilta pitää saada välitettyä tieto vaihtoehtoisesta käynnistymisestä Linuxille. Näin Linux voi käynnistyttyään suorittaa ohjelmoidut toimenpiteet: ilmoittaa käyttäjälle ja yrittää käynnistystiedostojen korjausta tai päivitystä uudelleen. Koska tämä tieto ei ole pysyvää eikä sitä tarvitse saada pysyvästi talteen, se voidaan välittää käynnistyksen yhteydessä Linux-ytimen komentoriviparametrina. U-Bootissa parametrit tallennetaan ’bootargs’-nimiseen ympäristömuuttujaan, josta ne automaattisesti välitetään Linuxille ytimen käynnistysparametreina. Esimerkissä 13 Linuxin puolella ytimen käynnistysparametrit luetaan /proc/cmdline-tiedostosta.

```
# cat /proc/cmdline
root=mtd1 rw console=ttyS0,115200 ↲
mtdparts=nand:128M(rootfs),12M(bootfs),12M(boot2),-(rest)↲
rootfstype=jffs2 bootpart=1 bootmode=normal
```

Esimerkki 13. Linuxin ytimen komentoriviparametrien lukeminen ./proc/cmdline-tiedostosta.

Linuxille välitettäviä tietoja ovat käynnistysosio sekä käynnistystyyppi. Komentorivimuuttujien nimiksi on valittu ’bootpart’ ja ’bootmode’. Käynnistysosio ilmoitetaan ytimelle ’bootpart’-nimisellä muuttujalla. Se kertoo, miltä käynnistysosiolta käynnissä oleva Linux-ydin ja käytetty laitepuutiedosto on ladattu. Käynnistystyyppi ilmoitetaan ’bootmode’-nimisellä muuttujalla. Se kertoo, onko käynnistys normaali (’bootmode=normal’), vaihtoehtoinen, jos ensisijaiset käynnistystiedostot eivät toimi (’bootmode=alt’), vai onko käynnistyslaskurin arvo ylittänyt asetetun rajan (’bootmode=limit’).

6.9 Käynnistyminen U-Bootissa

Kun järjestelmään kytketään sähkö, suoritin käynnistää käynnistyslataajan NOR-flash-muistista. Käynnistyslataaja puolestaan lataa ’fsload’-komennolla NAND-flash-muistin käynnistysosiolta keskusmuistiin .dtb-päätteisen laitepuutiedoston ja uImage-nimisen tiedoston, joka on U-Bootin formaatissa oleva pakattu Linux-ydin. Tämän jälkeen U-Boot ajaa ’bootm’-komennon, joka käynnistää muistiin ladatun Linux-ytimen. Käynnistyksessä ’bootm’-komennolle annetaan parametrina ytimen osoite keskusmuistissa, mahdollisesti initrd-kuvan osoite ja laitepuutiedoston osoite keskusmuistissa.

Käyttöjärjestelmän lataamisessa tai käynnistyksessä voi tapahtua seuraavia kolmenlaisia virhetilanteita.

Ensimmäinen virhetilanne voi tapahtua, jos JFFS2-tiedostojärjestelmässä on liian paljon merkintöjä U-Bootin käytössä olevaan keskusmuistiin nähden. U-Bootilla ei siis riitä muisti tiedostojärjestelmän luomiseksi keskusmuistiin Linuxin ytimen tai laitepuutiedoston lataamista varten. Tällöin tulostuu seuraavanlainen virheilmoitus:

```
=> chpart nand0,0
partition changed to nand0,0
=> fsload /boot/uImage
### JFFS2 loading '/boot/uImage' to 0x780000
Scanning JFFS2 FS:
.....................................................................
.............................. add_node: malloc failed
add_node failed!
load: Failed to scan JFFSv2 file structure
### JFFS2 LOAD ERROR<0> for /boot/uImage!
```

Esimerkki 14. U-Bootin muistin loppuminen tiedostoa ladattaessa.

U-Bootin konfiguroinnissa voidaan kasvattaa malloc()-kutsulle varattua muistialuetta, mutta muutos vaatii U-Bootin uudelleen kääntämisen ja kirjoittamisen flash-muistiin. Linuxissa tiedostojen kirjoittaminen ja poistaminen tiedostojärjestelmästä kasvattaa tiedostojärjestelmän kokoa ja sitä kautta lisää muistintarvetta. Näin kasvavan tiedostojärjestelmän lukeminen hidastuu. Kun käytetään pienikokoista, vain käynnistystiedostoille tarkoitettua osiota, tarvittava muistin määrä ja lukunopeus pysyy hallinnassa.

Toinen mahdollinen virhetilanne Linuxin lataamisessa on käynnistystiedostojen puuttuminen. Tiedoston lataamisen epäonnistumista voi seurata ’fsload’-komennon paluuarvosta esimerkin 15 mukaisesti:

```
=> if fsload 0x800000 somefile ; then echo "OK, somefile loaded"; ↵
else echo "Error: not loaded"; fi
### JFFS2 loading 'somefile' to 0x800000
find_inode failed for name=somefile
load: Failed to find inode
### JFFS2 LOAD ERROR<0> for somefile!
Error: not loaded
```

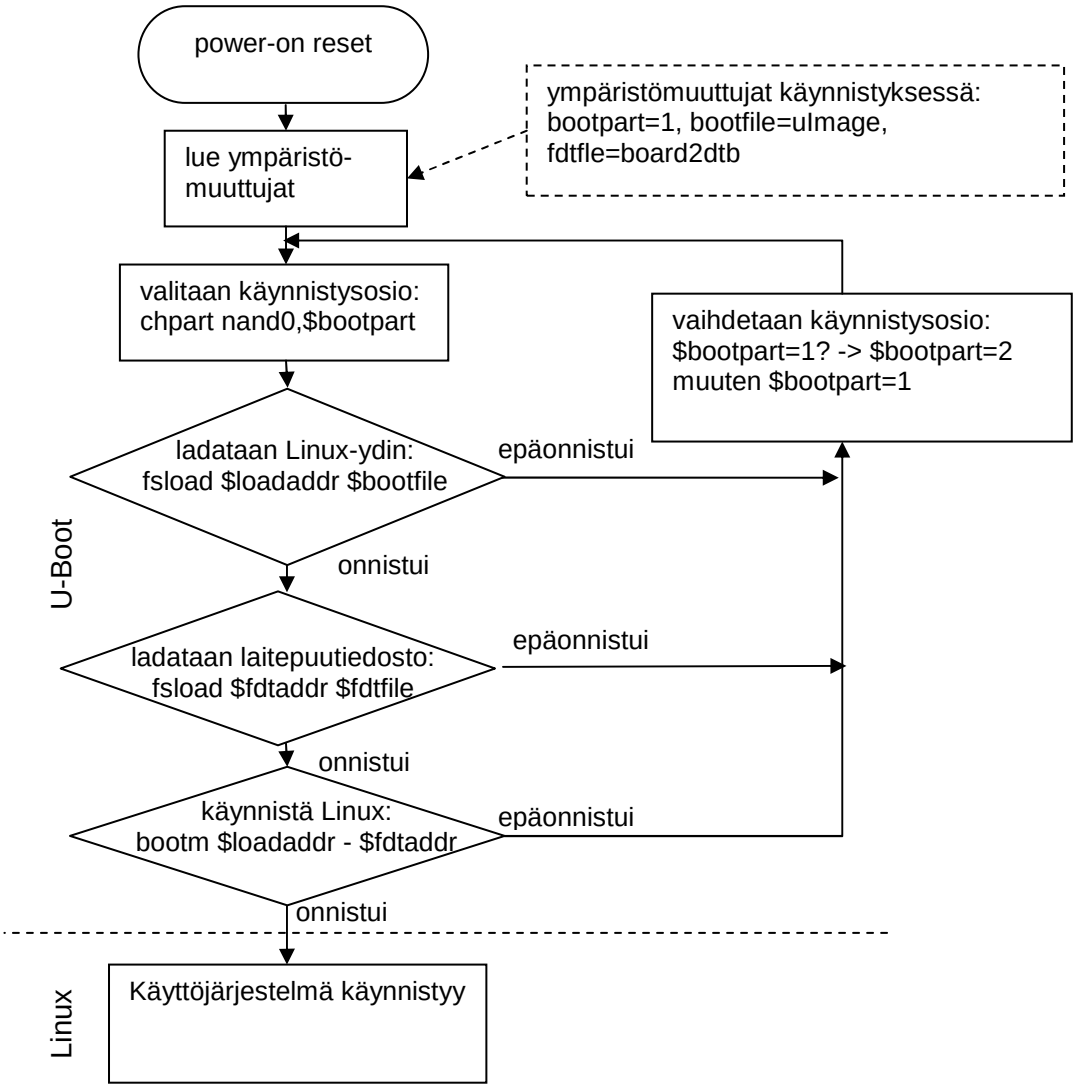
Esimerkki 15. Ladattavaa tiedostoa ei löydy.

Kolmas virhetilanne voi syntyä Linuxin käynnistävän ’bootm’-komennon ajamisessa. Sen seurauksena Linuxin käynnistyminen voi epäonnistua, jos joko Linux-ydin tai laitepuutiedosto on viallinen. Tällöin U-Boot tulostaa konsolille virheilmoituksen ja ’bootm’-komento keskeytyy. Tällaiseen virhetilanteeseen voidaan reagoida ohjelmallisesti lisäämällä lisäkomentoja ’bootm’-komennon jälkeen. Jos Linux käynnistyy, U-Boot ei koskaan jatka lisäkomentojen suorittamista, vaan suoritus siirtyy Linuxille. Jos Linux ei käynnisty, U-Boot jatkaa ’bootm’-komentoa seuraavasta komennosta. Esimerkiksi, jos laitepuutiedosto on viallinen tai sitä ei ole ladattu muistiin, esimerkin 16 komento tulostaa ilmoituksen konsolille:

```
=> bootm $loadaddr - $fdtfile; echo "Virhe bootm-komennossa"
## Booting kernel from Legacy Image at 00800000 ...
  Image Name:   Linux-board2
  Created:      2012-03-06  11:44:25 UTC
  Image Type:   PowerPC Linux Kernel Image (gzip compressed)
  Data Size:    2243242 Bytes =  2.1 MB
  Load Address: 00000000
  Entry Point:  00000000
  Verifying Checksum ... OK
ERROR: Did not find a cmdline Flattened Device Tree
Could not find a valid device tree
Virhe bootm-komennnossa
```

Esimerkki 16. Ladattu tiedosto on viallinen.

Lueteltuihin virhetilanteisiin voidaan kuitenkin varautua. Käyttämällä pienikokoista käynnistysosiota voidaan estää muistin loppuminen tiedostojen lataamisessa. Käynnistystiedostojen latausvirheet ’fsload’-komennon suorituksessa tai vialliset tiedostot ’bootm’-komennon yhteydessä voidaan huomioida käyttämällä U-Bootissa sopivia komentosarjoja, jotka vaihtavat aktiivisen käynnistysosion numeron. Tämän jälkeen käynnistystä voidaan yrittää uudelleen uudelta osiolta. Kuvassa 6 on esitetty käynnistyssekvenssi virhetilanteineen. Sellaista tilannetta, jossa molemmat käynnistysosiot menisivät epäkelpoon tilaan eikä järjestelmää saada käyntiin, ei ole otettu huomioon, koska silloin järjestelmä on sellaisessa tilassa, että se vaatii manuaalista korjausta paikan päällä.



Kuva 6. Käynnistysosion vaihtaminen virhetilanteissa.

7 Yhteenveto

Tässä insinöörityössäni kehitin ohjelmiston päivitysprosessia prosessorikortille, jolla oli valmiina käynnistyslataaja ja käyttöjärjestelmä. Ennen työn aloittamista järjestelmä oli kyllä päivitettävissä, mutta siitä puuttui päivityksen kannalta oleellisia turvallisuusominaisuuksia eikä sitä pystytty päivittämään etäyhteyden kautta. Lisäksi tiedostojärjestelmän rakenteesta johtuen järjestelmä käynnistyi hitaasti. Insinöörityöni tavoitteena parantaa etäpäivitettävyyttä sekä yksinkertaistaa tiedostojärjestelmää.

Sulautetuissa järjestelmissä käytetään yleisesti massamuistina erilaisia flash-muisteja. Flash-muisteja käytetään siirrettävissä massamuisteissa ja sulautetuissa järjestelmissä eri tavoin. Siirrettävissä massamuisteissa on varsinaisen flash-muistin lisäksi elektroniikkaa ja ohjelmallisia ratkaisuja, jotka piilottavat muistin teknisen toteutuksen niin tietokoneelta, kameralta kuin laitteen käyttäjältäkin. Sulautetussa järjestelmässä taas flash-muistia käytetään suoraan, jolloin sen ominaispiirteet täytyy ottaa huomioon muistin käytössä. Flash-muisteille on tehty omia, erityisesti niille suunniteltuja tiedostojärjestelmiä, kuten CRAMFS, JFFS2 ja UBIFS.

Sulautetuissa järjestelmissä tarvitaan erilaisia käynnistyslataajia, joita käytetään laiteympäristön alustamiseen ja käyttöjärjestelmän käynnistämiseen sekä käyttöjärjestelmän päivittämiseen.

Insinöörityössäni kehitin järjestelmän päivitykseen toiminnallisuuksia siten, että ne mahdollistavat käyttöjärjestelmän turvallisen etäpäivittämisen ja järjestelmän automaattisen palautumisen edelliseen konfiguraatioon päivityksen epäonnistuessa. Ottamalla käyttöön pienikokoiset osiot käynnistystiedostoja varten sain käynnistysaikaa huomattavasti pienennettyä. Linuxille käännetyt apuohjelmat, joilla voidaan muokata käynnistyslataajan ympäristömuuttujia, tuovat lisää monipuolisuutta ja joustavuutta, kun käynnistyslataajaan toimintaa voidaan muuttaa turvallisesti Linuxista käsin. Prosessorikortin päivitykseen ja ylläpitoon tekemäni muutokset on otettu tuotantokäyttöön ja ne selkiyttävät ja helpottavat kyseisten järjestelmien ylläpitoa.

Mahdollisena jatkokehityksenä on käynnistyslataajan ja käytetyn tiedostojärjestelmän yhteistoiminta. Tällä hetkellä käytössä oleva U-Bootin versio tukee rajoitetusti JFFS2-yhteenvetotietoja ja käynnistysosiossa tapahtuvaa tiedostojen päällekirjoitusta. U-Bootia päivitetään säännöllisesti, ja todennäköisesti seuraavissa versioissa tulee uusia ominaisuuksia, jotka tukevat kattavammin JFFS2:n ominaisuuksia. Toisaalta UBIFS sietää JFFS2:ta paremmin odottamattomia sammutuksia [24]. Näin ollen muiden flash-tiedostojärjestelmien käytön tutkiminen voi tuoda parannuksia järjestelmän luotettavuuteen ja etäpäivityksen turvallisuuteen.

Lähteet

1 NAND Flash Applications Design Guide. Toshiba. Verkkodokumentti. < http://139.138.48.19/pdf/NAND/Toshiba/NandDesignGuide.pdf.pdf >. Luettu 6.2.2013.

2 Common Flash Interface (CFI Specification). 2003. Verkkodokumentti. Solid State Technology Association. < http://www.jedec.org/sites/default/files/docs/jesd68-01.pdf >. Luettu 27.10.2011.

3 Cactus Technologies Application Note: CTAN010: SLC vs. MLC NAND. 2008. Cactus Technologies Limited.

4 Understanding the Flash Translation Layer (FTL) Specification. 1998. Verkkodokumentti. Intel Corporation < http://staff.ustc.edu.cn/~jpq/paper/flash/2006-Intel%20TR-Understanding%20the%20flash%20translation%20layer%20(FTL)%20specification.pdf >. Luettu 8.10.2011.

5 Sim, HyoGi; Jung, HoYoung; Park, SungMin; Kang, Sooyong; Cha, JaeHyuk. Identifying the FTL Mapping Schemes of USB Flash Drives. 2011. Department of Electrical and Computer Engineering Hanyang University Seoul, Korea

6 Hallinan, Christopher. 2010. Embedded Linux Primer: A Practical, Real-World Approach, Second Edition: Prentice Hall.

7 Breeuwsma, M., de Jongh, M., Klaver, C., van der Knjiff, R., & Roeloffs, M. 2007. Forensic Data Recovery from Flash Memory. Small Scale Digital Device Forensics Journal. < www.ssddfj.org/papers/SSDDFJ_V1_1_Breeuwsma_et_al.pdf >. Luettu 6.2.2013.

8 Formatting a memory card, flash drive or device using a PC. 2012. Verkkodokumentti. SanDisk Corporation. < http://kb.sandisk.com/app/answers/detail/a_id/312/~/formatting-a-memory-card,-flash-drive-or-device-using-a-pc >. Luettu 6.2.2013.

9 Formatting disks and drives: frequently asked questions. 2013. Verkkodokumentti. Microsoft. < http://windows.microsoft.com/en-us/windows7/Formatting-disks-and-drives-frequently-asked-questions >. Luettu 6.2.2013.

10 Universal Serial Bus Mass Storage Class Specification Overview. 2010. Verkkodokumentti. USB Implementers Forum, Inc. < http://www.usb.org/developers/devclass_docs/Mass_Storage_Specification_Overview_v1.4_2-19-2010.pdf >. Luettu 6.2.2013.

11 SD Specifications Part E1 SDIO Simplified Specification. 2007. Verkkodokumentti. SD Card Association. < https://www.sdcard.org/developers/overview/sdio/sdio_spec/Simplified_SDIO_Card_Spec.pdf >. Luettu 18.7.2012.

12 xD picture card pinout. Verkkodokumentti. < http://pinouts.ru/Memory/xd_card_pinout.shtml >. Luettu 6.2.2013.

13 Kasavajhala, Vamsee. 2011. Solid State Drive vs. Hard Disk Drive Price and Performance Study. Verkkodokumentti. < http://www.dell.com/downloads/global/products/pvaul/en/ssd_vs_hdd_price_and_performance_study.pdf >. Luettu 20.7.2012.

14 Park, Seon-yeong; Seo, Euiseong; Shin, Ji-Yong, Maeng, Seungryoul; Lee, Joonwon. 2010. IEEE COMPUTER ARCHITECTURE LETTERS, VOL. 9. Exploiting Internal Parallelism of Flash-based SSDs. IEEE Computer Society

15 Chris Simmons. 2009. Linux flash file systems JFFS2 vs UBIFS. 2net Limited. Verkkodokumentti. < http://www.embedded-linux.co.uk/downloads/ESC-5.4-flash_filesystems-slides.pdf >. Luettu 31.8.2012.

16 The mtdblock driver. Verkkodokumentti. < http://www.linux-mtd.infradead.org/doc/general.html#L_mtdblock > Luettu 24.1.2013.

17 How do I use NAND simulator? 2008. Verkkodokumentti. < http://www.linux-mtd.infradead.org/faq/nand.html#L_nand_nandsim >. Luettu 6.2.2013.

18 Yaghmour, Karim. 2003. Building Embedded Linux systems. Sebastopol: O'Reilly Media.

19 Love, Robert. 2005. Linux Kernel Development, Second Edition: Pearson Education, Inc.

20 R Micheloni et al. 2010. Inside NAND Flash Memories. Springer Science+Business Media B.V.

21 I am going to use JFFS2 on top of my USB stick/CF card/etc, is it OK? Verkkodokumentti. < http://www.linux-mtd.infradead.org/faq/jffs2.html#L_stick_jffs2> Luettu 1.1.2013.

22 Opdenacker, Michael; Petazzoni, Thomas; Clement, Gregory. 2010. Flash filesystems. Free Electrons. Verkkodokumentti. < free-electrons.com/doc/flash-filesystems.odp >. Luettu 2.1.2013.

23 DENX Software Engineering. Compressed ROM Filesystem. Verkkodokumentti. < http://www.denx.de/wiki/DULG/FlashFilesystemsCRAMFS >. Luettu 7.4.2013.

24 UBIFS - UBI File-System. Verkkodokumentti. < http://www.linux-mtd.infradead.org/doc/ubifs.html >. Luettu 6.2.2013.

25 JFFS2 Gargabe Collection. Verkkodokumentti. < http://sourceware.org/jffs2/jffs2-html/node3.html#SECTION00036000000000000000 >. Luettu 7.4.2013.

26 JFFS2 improvement project. University of Szeged. Verkkodokumentti. < http://www.inf.u-szeged.hu/jffs2/mount.php >. Luettu 6.2.2013.

27 JFFS2 Node Format and Compatibility. Verkkodokumentti. < http://sourceware.org/jffs2/jffs2-html/node3.html#SECTION00031 000000000000000 > Luettu 24.1.2013.

28 Linuxin jffs2-otsikkotiedosto. Verkkodokumentti. < http://lxr.linux.no/#linux+v3.4.6/include/linux/jffs2.h > Luettu 24.1.2013.

29 Logic Technology. Traditional BIOS versus UEFI/EFI Framework - an overview. Verkkodokumentti. < http://www.logic.nl/Products/Technology/BIOS-and-EFI.aspx >. Luettu 7.4.2013.

30 Open Firmware, IEEE 1275. 2005. The Open Firmware Working Group. Verkkodokumentti. < http://www.openfirmware.org/1275/home.html >. Luettu 20.1.2013.

31 Booting the Linux/ppc kernel without Open Firmware. Linux kernel versio <2.6.38 lähdekoodi /Documentation/powerpc/booting-without-of.txt >

32 [PATCH V2] JFFS2: bug fix for summary support. Verkkodokumentti. < http://comments.gmane.org/gmane.comp.boot-loaders.u-boot/97890 >. Luettu 31.1.2013.

33 U-Boot JFFS2 options and usage. Verkkodokumentti. < http://git.denx.de/cgi-bin/gitweb.cgi?p=u-boot.git;a=blob;f=doc/README.JFFS2;h=f0e9bc1b37 aacf8ebb8b0ddc50d61e577b1f9399;hb=HEAD >. Luettu 31.1.2013.

Linuxin nandsim-ajurin lataaminen

Linuxin nandsim-ajurin lataaminen niin, että parametrina annetaan todellisen NAND-flash-muistin tunnistustavut. Lataamisen jälkeen dmesg-komento tulostaa ajurin simuloidun muistin tiedot sekä sen luomat mtd-laitteet.

```
$ sudo modprobe nandsim first_id_byte=0xec second_id_byte=0xda ↲
third_id_byte=00 fourth_id_byte=0x15 parts=1024,1024
$ dmesg
```

```
[505454.546443] [nandsim] warning: write_byte: command (0x90) wasn't ↲
expected, expected state is STATE_READY, ignore previous states
[505454.546502] [nandsim] warning: read_byte: unexpected data output ↲
cycle, state is STATE_READY return 0x0
[505454.546508] [nandsim] warning: read_byte: unexpected data output ↲
cycle, state is STATE_READY return 0x0
[505454.546511] [nandsim] warning: read_byte: unexpected data output ↲
cycle, state is STATE_READY return 0x0
[505454.546513] [nandsim] warning: read_byte: unexpected data output ↲
cycle, state is STATE_READY return 0x0
[505454.546516] NAND device: Manufacturer ID: 0xec, Chip ID: 0xda ↲
(Samsung NAND 256MiB 3,3V 8-bit)
[505454.546676] flash size: 256 MiB
[505454.546752] page size: 2048 bytes
[505454.546761] OOB area size: 64 bytes
[505454.546769] sector size: 128 KiB
[505454.546778] pages number: 131072
[505454.546785] pages per sector: 64
[505454.546793] bus width: 8
[505454.546800] bits in sector size: 17
[505454.546808] bits in page size: 11
[505454.546815] bits in OOB size: 6
[505454.546822] flash size with OOB: 270336 KiB
[505454.546830] page address bytes: 5
[505454.546838] sector address bytes: 3
[505454.546848] options: 0x8
[505454.548955] Scanning device for bad blocks
[505454.552777] Creating 2 MTD partitions on "NAND 256MiB 3,3V 8-bit":
[505454.553302] 0x000000000000-0x000008000000 : "NAND simulator ↲
partition 0"
[505454.557290] 0x000008000000-0x000010000000 : "NAND simulator ↲
partition 1"
```

Käynnistyslaskurin lisäämisen vaatimat muutokset U-Bootin lähdekoodiin

```
diff --exclude '*.[ao]' --exclude '.??*' -uNr u-boot-2010.03/cpu/mpc83xx/cpu.c
u-boot-2010.03.modified/cpu/mpc83xx/cpu.c
--- u-boot-2010.03/cpu/mpc83xx/cpu.c 2010-04-01 00:54:39.000000000 +0300
+++ u-boot-2010.03.modified/cpu/mpc83xx/cpu.c   2011-10-03 12:48:20.000000000
+0300
@@ -305,8 +305,8 @@

 #ifdef CONFIG_BOOTCOUNT_LIMIT
-#if !defined(CONFIG_MPC8360)
-#error "CONFIG_BOOTCOUNT_LIMIT only for MPC8360 implemented"
+#if !defined(CONFIG_MPC8360) && !defined(CONFIG_MPC832x)
+#error "CONFIG_BOOTCOUNT_LIMIT only for MPC8360/MPC832x implemented"
 #endif

 #if !defined(CONFIG_BOOTCOUNT_ADDR)
diff --exclude '*.[ao]' --exclude '.??*' -uNr u-boot-2010.03/cpu/mpc83xx/fdt.c
u-boot-2010.03.modified/cpu/mpc83xx/fdt.c
--- u-boot-2010.03/cpu/mpc83xx/fdt.c 2010-04-01 00:54:39.000000000 +0300
+++ u-boot-2010.03.modified/cpu/mpc83xx/fdt.c   2011-10-03 12:49:09.000000000
+0300
@@ -32,7 +32,7 @@


 DECLARE_GLOBAL_DATA_PTR;

-#if defined(CONFIG_BOOTCOUNT_LIMIT) && defined(CONFIG_MPC8360)
+#if defined(CONFIG_BOOTCOUNT_LIMIT) && (defined(CONFIG_QE))
 #include <asm/immap_qe.h>

 void fdt_fixup_muram (void *blob)
diff --exclude '*.[ao]' --exclude '.??*' -uNr u-boot-
2010.03/include/configs/board2.h u-boot-
2010.03.modified/include/configs/board2.h
--- u-boot-2010.03/include/configs/board2.h     2011-10-03 12:44:44.000000000
+0300
+++ u-boot-2010.03.modified/include/configs/board2.h        2011-09-28
11:12:12.000000000 +0300
@@ -28,6 +28,8 @@

 #undef CONFIG_PCI

+#define CONFIG_BOOTCOUNT_LIMIT
+
 /*
  * System Clock Setup
  */
```