Harri Jernström

Web-käyttöliittymä sulautetun tietokonelaitteiston ohjaamiseen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

10.12.2012

Metropolia

Tekijä Otsikko Sivumäärä Aika	Harri Jernström Web-käyttöliittymä sulautetun tietokonelaitteiston ohjaamiseen 33 sivua + 5 liitettä 10.12.2012
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja	yliopettaja Markku Nuutinen

Insinöörityön tavoitteena oli kehittää WWW-pohjainen käyttöliittymä ja tarvittava ohjelmisto Beagleboard-mallisen sulautetun tietokoneen laitteiston ohjaukseen. Ohjattavaksi laitteistoksi määritettiin Beagleboardilla olevat laajennusväylät ja niiltä löytyvät signaalit. Työssä oli tarkoitus valita käyttöjärjestelmä ja sopivat ohjelmistot ja asentaa ja konfiguroida nämä toimivaksi kokonaisuudeksi.

Beagleboardin laitteisto toimii ARM-prosessorilla, ja työssä perehdyttiin myös ristiinkääntämisen eri vaiheisiin ja yksityiskohtiin. Työssä asennettiin ja konfiguroitiin ristiinkääntämistä varten crosstools-ng -työkaluketju ja käytettiin sitä sekä Lighttpd WWW-palvelimen että oheiskirjastojen kääntämiseen. Kaikki kirjastot käännettiin dynaamisesti linkitettäviksi.

Laitteiston ohjaamiseksi työssä perehdyttiin Beagleboardin eri laajennusväyliin ja niiltä löytyviin signaaleihin, kuten GPIO ja I²C, ja niiden käyttöön omissa ohjelmissa. GPIO:n käyttö sysfs-tiedostorakenteen kautta osoittautui yksinkertaiseksi ja helpoksi. I²C:n käyttöön perehdyttiin lähinnä teoriassa.

Työssä kehitettiin myös WWW-pohjainen demo-ohjelma, jonka avulla voi ohjata Beagleboardissa olevia ledivaloja GPIO:n avulla. Ohjelmaa kehitettäessä ja testattaessa löytyi joitain mahdollisia ongelmakohtia ledivalojen tilatiedon synkronoinnissa, jotka kannattaa ratkaista, jos tällaista ohjausohjelmistoa halutaan käyttää jossakin todellisessa projektissa.

Avainsanat	Linux, sulautettu, ARM, www, GPIO

Author	Harri Jernström
Title	Web interface for controlling embedded computer hardware
Number of Pages	33 pages + 5 appendices
Date	10 December 2012
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Markku Nuutinen, Principal Lecturer

The goal of this thesis was to develop a web-based user interface and other needed software to control the hardware of a Beagleboard embedded computer. The expansion connectors of Beagleboard and the various signals in the pins of them were selected to be the hardware to control with this application. The project also involved installing a suitable operating system and other needed software, and configuring those to make a working environment for the software.

The processor of Beagleboard is based on ARM architecture. This project involved studying the details of cross compiling, especially for ARM architecture. During the project a crosstool-ng toolchain was configured and installed to compile a Lighttpd webserver, and a few supporting libraries such as openssl. All the software was compiled to support dynamic linking.

To control the device, this project involved studying the expansion connectors of Beagleboard, and different ways to control various signals in the individual pins. Mainly GPIO and I²C were studied to learn how to use them when creating new software. Using the GPIO through sysfs filesystem turned out to be relatively simple. Using the I²C was studied mainly in theory.

During this project a simple web-based demo program was developed to control the leds of Beagleboard with GPIO. Through the course of the development and testing of this program some problems with synchronizing the status of leds were found that may need to be solved, if this kind of software is used in any real world project.

Keywords	Linux, embedded, ARM, www, GPIO

Sisällys

1 Johdanto 1

2 Sulautetun järjestelmän tekniikkaa 2

2.1 Yleistä sulautetuista järjestelmistä 2
2.2 Linuxin käyttö sulautetuissa järjestelmissä 3
2.3 ARM-prosessori 4
2.4 Beagleboardin tekniikkaa 4
2.5 Liittymät Beagleboardin laitteistoon 6

3 Web-ohjauksen tekniikkaa 8

3.1 Web-tekniikan perusteet 8
3.1.1 HTTP-protokolla 8
3.1.2 WWW-palvelin 9
3.1.3 CGI-rajapinta 10
3.2 Verkkoliitännät 11

4 Testiympäristö 12

4.1 Käyttöjärjestelmä 12
4.2 WWW-palvelin 13
4.2.1 Crosstool-NG asennus 14
4.2.2 Openssl- ja muiden kirjastojen kääntö 16
4.2.3 Lighttpd:n ristiinkääntö 19
4.2.4 Ohjelmien asennus Beagleboardille 20
4.2.5 WWW-palvelimen asetukset 20
4.3 Verkkoasetukset 22
4.4 Laitteiston ohjaus 24

5 Testisovelluksen toteutus 26

5.1 Laitteiston ohjaus WWW-liittymän kautta 26
5.2 Käyttöliittymä 27
5.3 CGI-komentosarjan toiminta 28
5.4 Leddmn-taustaprosessin toiminta 30

6 Johtopäätökset 31

Lähteet 33

Liitteet

Liite 1. Lighttpd:n kääntämisessä tarvittava komentosarja

Liite 2. Lighttpd käynnistyskomentosarja

Liite 3. leddmn.c lähdekoodi

Liite 4. leddmn.h lähdekoodi

Liite 5. beaglecontrol.py CGI-komentosarja

Lyhenteitä ja termejä

Adhoc	WLAN-verkkotyyppi, jossa laitteet ottavat tukiaseman sijasta yhteyden toisiinsa.
ARM	ARM Holdings inc. -yhtiön piirivalmistajille lisensoima prosessoriarkkitehtuuri.
Beagleboard	Texas Instrumentsin suunnittelema OMAP3-prosessoriin perustuva pieni yhden piirilevyn tietokone.
CGI	Common gateway interface. Standardi jolla WWW-palvelin voi siirtää sivun tai tiedoston luomisen ulkoiselle ohjelmalle tai komentosarjalle.
GPIO	General purpose input/output. Digitaalinen signaali, joka on ohjelmallisesti ohjattavissa.
HTTP	Hypertext transfer protocol. WWW:n perusprotokolla.
I²C	Inter integrated circuit. Sarjaliitäntä laitteiden ja piirien kytkemiseen. Kilpaileva tuote SPI:lle.
OMAP	Texas Instrumentsin valmistama ARM-arkkitehtuuriin perustuva piiriperhe.
PoP	Package on package. Tapa paketoida prosessori ja esimerkiksi muistit päällekkäin siten, että ne käyttävät samaa kantaa piirilevyllä
SoC	System on a chip. Yhdelle piirille rakennettu prosessorin ja muiden tarvittavien oheispiirien kokonaisuus.
SPI	Serial peripheral interface. Sarjaliitäntä eri lisälaitteiden ja piirien kommunikointiin.
WLAN	Wireless local area network. Langaton tietoliikenneverkko.

1 Johdanto

Työn tarkoituksena on kehittää WWW-pohjainen käyttöliittymä ja muut tarvittavat ohjelmat joilla voi ohjata Beagleboard-mallisen sulautetun tietokoneen laitteistoa. Käyttöliittymällä on tarkoitus ohjata Beagleboardilta löytyviä laajennusväyliä ja niiden signaaleja. Helppona esimerkkinä väylien ohjaamisesta käytetään Beagleboardilta löytyviä ledejä, joiden ohjaus toimii samoilla menetelmillä kuin laajennusväylän signaalien ohjaus. Samalla on tarkoitus perehtyä koko prosessiin, jolla valitaan, asennetaan ja tarvittaessa käännetään projektin tarvitsemat ohjelmat Beagleboardin kaltaiseen sulautettuun tietokoneeseen.

Monissa nykyään käytetyissä sulautetuissa järjestelmissä, muun muassa matkapuhelimissa, käytetään jotain ARM-arkkitehtuuriin pohjautuvaa prosessoria. ARM-arkkitehtuurin tuki paranee versio versiolta nopeasti Linuxin ytimessä. Samoin lisääntyy suora tuki erilaisille sulautetuille ARM-pohjaisille laitteille eri lisälaitteineen ja piireineen. Suuri osa Linuxiin saatavissa olevista ohjelmista ja kirjastoista on kehitetty alun perin Intel-arkkitehtuuriin. Ohjelmien valinta ja ristiinkääntäminen eri arkkitehtuurille ja sulautetulle järjestelmälle tyypilliselle rajoitetulle laitteistolle aiheuttaa ylimääräisiä haasteita sulautettuja järjestelmiä kehitettäessä.

Sulautetun järjestelmän ohjaaminen jollain yleisesti käytetyllä tekniikalla, esimerkiksi WWW-selaimella, on kiinnostava aihe, koska mahdollisilla käyttäjillä on tarvittava ohjelma valmiina tai ainakin helposti saatavilla. Etuna on myös se, että sulautetussa laitteessa tarvitaan vain verkkoliitäntä ja WWW-palvelin. Muuta erillistä näyttölaitetta ei sulautettuun laitteeseen tässä tapauksessa tarvita. WWW-tekniikat ovat olleet käytössä niin pitkään, että niiden luotettavuus on riittävä tällaisen järjestelmän ohjaamiseen.

2 Sulautetun järjestelmän tekniikkaa

2.1 Yleistä sulautetuista järjestelmistä

Sulautettu tietokone on nykyisin yksi yleisimmistä tietokoneista. Sulautettuja tietokoneita esiintyy kaikkialla ympärillämme, esimerkiksi autojen moottoreiden ohjauksissa, mikroaaltouuneissa ja digibokseissa varsinaisissa tehdaslaitoksissa esiintyvästä automaatiosta puhumattakaan. Itse asiassa suuressa osassa elektroniikkalaitteista on nykyisin myös ohjelmallisesti ohjattavia osia, eli ne ovat sulautettuja tietokoneita. Yleisimpiä tavallisen kansalaisen käyttämiä sulautettuja tietokoneita ovat matkapuhelimet. Nykyinen älypuhelin vastaa teholtaan takavuosien tehotyöasemaa, lisäksi sillä voi soittaa puheluita.

Sulautettujen järjestelmien erityispiirteitä ovat muun muassa rajalliset resurssit, kiinteästi määritelty laitteisto ja pitkä käyntiaika. Lisäksi sulautetun laitteen on käynnistyttävä itsekseen käyttövalmiiseen tilaan ja toimittava, kunnes laitteisto taas sammutetaan. Sulautetun järjestelmän on kyettävä toimimaan ennalta arvaamattomassa tahdissa tulevien syötteiden perusteella. Usein syötteisiin reagoinnilla on jokin maksimivasteaika.

Järjestelmän vasteaika on yksi tärkeimmistä ominaisuuksista sulautetuissa laitteissa. Esimerkiksi polttomoottorin ohjauksessa vaaditut vasteajat ovat joitain kymmeniä millisekunteja, kun taas www-palvelin voi suorituskykynsä ylärajoilla käsitellä yhden asiakkaan sivupyyntöä sekuntikaupalla. Oikea vaste riippuu jossain tapauksissa myös siitä, onko vaste toimitettu oikeaan aikaan eli riittävän nopeasti. Hieman myöhästynyt mutta muuten odotettu vaste ei tällöin olekaan enää oikea vaste siinä tilanteessa. Esimerkkeinä tällaisista aikakriittisistä ohjauksista voi mainita ajoneuvon lukkiutumattoman jarrun ohjauksen tai lentokoneiden elektronisiin ohjausjärjestelmiin liittyvät sulautetut laitteet.

Sulautetuissa järjestelmissä signaalit tulevat satunnaisina aikoina ja joskus satunnaisista lähteistä ja niihin pitää pystyä vastaamaan silti tosiaikaisesti. Tämä johtaa systeemin optimointiin ja resurssien mahdollisimman sujuvan käytön varmistamiseen.

Laitteistoissa on yleensä rajoitetut resurssit tavalliseen käyttötietokoneeseen verrattuna. Esimerkiksi pelikonsolien kiinteänkokoinen muisti, matkapuhelinten ja muiden kannettavien laitteiden akkukestot ja monissa laitteistoissa fyysisen koon, painon tai muun käyttöympäristön rajoitukset pakottavat laitteen suunnittelijan pienempiin resursseihin. Joissain tapauksissa ohjelmisto tekee sitä, mitä muuten tehtäisiin piireillä, koska halutaan vähentää käytettyjen piirien määrää.

Sulautetuissa koneissa on harvoin mitään liittymää, jolla laitteeseen pääsee tekemään muutoksia tai sen toimintaa ohjaamaan muuten kuin laitteeseen asennetun käyttäjälle suunnitellun käyttöliittymän avulla. Esimerkkinä tästä on mikrouunin ohjaus ohjelmanvalinta- ja ajastusnappuloilla tai digiboksin ohjaus kauko-ohjaimella kuvaruutuvalikosta. Yleensä ohjelmointiin tai uuden ohjelmiston lataukseen tarvitaan jonkinlainen datakaapeli ja tarkoitukseen sopiva ohjelmisto. Joskus ohjaukseen riittää terminaaliohjelmakin, jos laitteessa on USB- tai sarjaliitäntä. [1; 3.]

2.2 Linuxin käyttö sulautetuissa järjestelmissä

Linuxin suosiota on hankala arvioida tarkalleen. Työpöytäkoneissa suosituin käyttöjärjestelmä on pitkään ollut Microsoft Windows eri versioineen. Linuxin yleisimmät käyttöalueet ovat kuitenkin palvelimissa ja sulautetuissa laitteissa. Varovaistenkin arvioiden mukaan maailmassa on kymmeniä miljoonia Linux-laitteita esimerkiksi internet-verkon palvelimina. Älypuhelinten ja tablettien yleistyessä todellinen Linux-laitteiden määrä on todennäköisesti moninkertainen. Esimerkkeinä usein Linuxin sisältävistä laitteista ovat digiboksit, verkon modeemit ja muut laitteet, verkon palvelimet sekä Android-matkapuhelimet ja -tabletit. [1.]

Hallinan on kirjassaan [1, s 12] luetellut tyypillisiä syitä Linuxin valintaan sulautettuun laitteeseen:

- Linuxissa on laaja tuki eri laitteille. Tuki löytyy monelle eri prosessoriarkkitehtuurille ja lisälaitteelle.
- Linuxissa on monipuolisesti tukea eri verkkoprotokollille ja verkkolaitteille.
- Käyttöjärjestelmän ydin skaalautuu pienistä sulautetuista kuluttajalaitteista aina suuriin verkon reitittimiin ja kytkimiin asti.

- Linux on periaatteessa ilmainen, eli ei ole hankalia ja kalliita laitekohtaisia lisenssejä ja maksuja.
- Kehittäjien määrä on suuri ja monipuolinen yksittäisistä harrastajista aina suuriin ohjelmistotaloihin.
- Jatkuvasti lisääntyvä joukko laitteisto- ja piirivalmistajia tukee omien laitteidensa osalta Linuxia.

2.3 ARM-prosessori

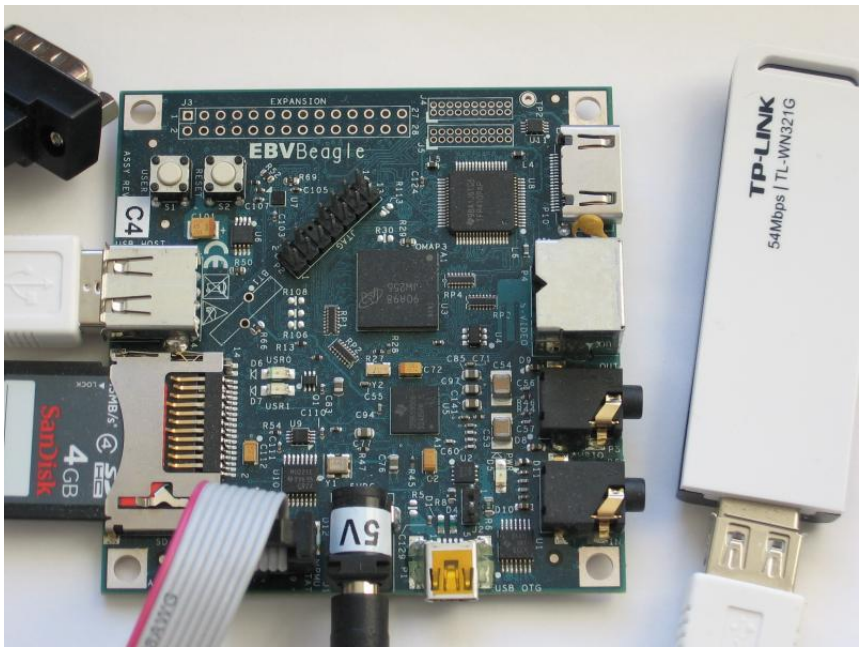
ARM-arkkitehtuuri on ARM Holdings plc. -yrityksen kehittämä ja lisensoima prosessoriarkkitehtuuri. Se on tyypiltään RISC-käskykantainen yhden tai useamman ytimen prosessori. Aiemmat ARM-versiot olivat 32-bittisiä. Vuonna 2011 julkaistu ARMv8-arkkitehtuuri tukee myös 64-bittistä käskykantaa. ARM-arkkitehtuurin kehittänyt yhtiö ainoastaan tuotekehittää uusia versioita arkkitehtuurista, varsinaiset prosessorit valmistavat lisenssin ostaneet piirivalmistajat. Eri versioita arkkitehtuurista on lisensoitu sadoille eri piirivalmistajille. Viimevuotisen tilaston mukaan ARM-arkkitehtuuriin perustuvia prosessoreita käytetään 95 %:ssa uusista matkapuhelimista ja tablettitietokoneista. Muita tyypillisiä ARM-prosessorin käyttökohteita ovat langattoman verkon laitteet, kehittyneemmät tv:t ja kannettavan elektroniikan laitteet. [6 s.12.]

Suuri syy ARM-arkkitehtuurin suosioon varsinkin kannettavissa laitteissa on pieni virrankulutus. Toisaalta sulautetuissa laitteissa toiminnallisuus ja luotettavuus on usein tärkeämpää kuin laskentateho. ARM-arkkitehtuurin lisensoineiden valmistajien suuri määrä helpottaa löytämään sopivan prosessorin sopivilla lisätoiminnoilla eri laiteprojekteihin.

2.4 Beagleboardin tekniikkaa

Beagleboard on Texas Instrumentsin OMAP3530-prosessoriin perustuva pieni yhden piirilevyn kehitystietokone. Beagleboardin piirikaaviot on julkaistu Creative Commons -lisenssillä, ja laitteita on saatavilla useilta eri valmistajilta. Tässä projektissa käytössä olevassa laitteessa on RAM-muistia 256 Mt ja flash-muistia 256 Mt. Tallennustilana voi käyttää SD-muistikorttia. Valmiina piirilevyllä on SD/MMC-korttipaikan lisäksi DVI-D-liitäntä digitaalinäytölle ja s-videoliitäntä esimerkiksi tv-näytölle. Lisäksi piirilevyllä on

kaksi kappaletta USB-liittimiä, joista toinen toimii tarvittaessa virransyöttöliitäntänä, ja sarjaportti. Kortilla on myös liitännät mikrofonille ja kaiuttimelle.



Kuva 1. EVB-Beagle, jossa on liitettynä SD-muistikortti, virransyöttö, USB-jatkojohto ja sarjaporttiliitin. Kuvan oikealla sivulla on WLAN-kortti kiinni USB-jatkojohdossa.

OMAP3530 on ARMv7 -arkkitehtuuriin perustuva ARM Cortex-a8 -tyyppinen prosessori. Texas Instrumentsin suunnitteludokumentin mukaan se on suunniteltu muun muassa tukemaan mobiilipelaamista, suoratoistovideota, videokonferenssiohjelmistoja ja suuren resoluution digikuvia. Prosessorilla on signaalin käsittelyyn DSP-piiri, PowerVR SGX 3D- grafiikkapiiri ja digitaalikuvan käsittelyyn sopiva ISP-piiri. Prosessori on koteloitu niin sanotulla package on package tyylillä (PoP), eli muistit ja prosessorin ydin on paketoitu yhteen päällekkäin ja koko paketti on kiinni piirilevyllä yhdessä kannassa. Kuvassa 1 näkyy OMAP-prosessorin ja muistipiirien paketti heti JTAG-liittimen vieressä oikealle alaviistoon, piirilevyn keskiosassa. [7, s. 175–176.]

2.5 Liittymät Beagleboardin laitteistoon

Beagleboardilla on yksi 28-pinninen laajennusväylä, johon voi liittää omia laitteita tai kaupallisesti saatavilla olevia lisälaitteita. Saatavilla on eri valmistajilta lisäkortteja, joissa on esimerkiksi ethernet-liitäntä tai lisää I²C, SPI tai muita laiteväyliä. Lisäksi piirilevyllä on kaksi kappaletta 20-pinnisiä laajennuspaikkoja, jotka on tarkoitettu LCD-näytön kytkemiseen mutta joiden pinnit voi ottaa myös GPIO- tai muuhun käyttöön. Molempien laajennusväylien liittimet näkyvät kuvassa 1 piirilevyn yläosassa. Kuvan Beagleboardista puuttuu kuitenkin liittimet näistä väylistä.

Table 20. Expansion Connector Signals

EXP	OMAP	0	1	2	3	4	5	6	7
1		VIO_1V8							
2		DC_5V							
3	AE3	MMC2_DAT7	*	*	*	GPIO_139	*	*	Z
4	AB26	UART2_CTS	McBSP3_DX	GPT9_PWMEVT	X	GPIO_144	X	X	Z
5	AF3	MMC2_DAT6	*	*	*	GPIO_138	*	X	Z
6	AA25	UART2_TX	McBSP3_CLKX	GPT11_PWMEVT	X	GPIO_146	X	X	Z
7	AH3	MMC2_DAT5	*	*	*	GPIO_137	*	X	Z
8	AE5	McBSP3_FSX	UART2_RX	X	X	GPIO_143	*	X	Z
9	AE4	MMC2_DAT4	*	X	*	GPIO_136	X	X	Z
10	AB25	UART2_RTS	McBSP3_DR	GPT10_PWMEVT	X	GPIO_145	X	X	Z
11	AF4	MMC2_DAT3	McSPI3_CS0	X	X	GPIO_135	X	X	Z
12	V21	McBSP1_DX	McSPI4_SIMO	McBSP3_DX	X	GPIO_158	X	X	Z
13	AG4	MMC2_DAT2	McSPI3_CS1	X	X	GPIO_134	X	X	Z
14	W21	McBSP1_CLKX	X	McBSP3_CLKX	X	GPIO_162	X	X	Z
15	AH4	MMC2_DAT1	X	X	X	GPIO_133	X	X	Z
16	K26	McBSP1_FSX	McSPI4_CS0	McBSP3_FSX	x	GPIO_161	X	X	Z
17	AH5	MMC2_DAT0	McSPI3_SOMI	X	X	GPIO_132	X	X	Z
18	U21	McBSP1_DR	McSPI4_SOMI	McBSP3_DR	X	GPIO_159	X	X	Z
19	AG5	MMC2_CMD	McSPI3_SIMO	X	X	GPIO_131	X	X	Z
20	Y21	McBSP1_CLKR	McSPI4_CLK	X	X	GPIO_156	X	X	Z
21	AE2	MMC2_CLKO	McSPI3_CLK	X	X	GPIO_130	X	X	Z
22	AA21	McBSP1_FSR	X	*	Z	GPIO_157	X	X	Z
23	AE15	I2C2_SDA	X	X	X	GPIO_183	X	X	Z
24	AF15	I2C2_SCL	X	X	X	GPIO_168	X	X	Z
25	25	REGEN							
26	26	nRESET							
27	27	GND							
28	28	GND							

Kuva 2. Taulukko Beagleboardin 28-pinnisen laajennusväylän pinnien signaalivaihtoehdoista [5 s. 96.]

OMAP-prosessorissa, kuten useissa System on a chip- tyyppisissä (SoC) piireissä, on osalle pinneistä valittavissa useampia eri signaaleja. Tätä kutsutaan multipleksoinniksi. OMAP-prosessorissa näillä multipleksoitavilla pinneillä voi olla kullakin enintään 8 eri funktiota. Kuvassa 2 on taulukko Beagleboardin laajennusväylän pinnien multiplekson-nilla valittavista signaaleista. Nämä signaalit eivät voi olla yhdellä pinnillä toiminnassa samaan aikaan, vaan kunkin pinnin toiminnallisuus pitää erikseen valita. Osa toiminnallisuuksista vaatii toimiakseen useamman pinnin konfiguroinnin. On täysin käyttäjän vastuulla ottaa kaikki tarpeelliset signaalit käyttöön. Kuten kuvan 2 taulukosta näkyy,

suurin osa pinneistä on valittavissa GPIO-käyttöön. Muita vaihtoehtoja ovat esimerkiksi eri I²C- tai SPI-väylien signaalit.

Käynnistettäessä Beagleboardia alkuvaiheessa suurin osa pinneistä on oletusarvossa eli multiplex-moodi 7:ssä, jossa pinnillä ei ole jännitettä tai mitään funktiota. Kunkin pinnin käytönaikainen oletusarvo asetetaan Uboot-lataajassa, joka on ensimmäinen ohjelma joka käynnistetään laitteen käynnistyksen jälkeen. Oletusarvon voi asettaa myös Linuxin ytimessä, jos ydin on käännetty optiolla joka sallii ytimen muuttavan pinnien multiplex-arvoja. Oletusarvoissaan olevassa ytimessä tämä vaihtoehto on poissa päältä, koska ideana on ollut että pinnien multiplex-arvot asetetaan käynnistyksen yhteydessä lataajassa. Oletusarvojen muuttaminen vaatii joko Uboot-lataajan tai ytimen lähdekoodin editointia ja uudelleen kääntämisen.

OMAP-prosessorissa voi myös kontrolloida pinnien multiplex-asetuksia suoraan muistin rekistereiden kautta. Kontrollirekisteri alkaa muistiosoitteesta 0x4800 2030. Suorat osoitteet kullekin pinnille voi katsoa lähteen [7] taulukosta. Kyseessä on 32-bittiset rekisterit, joista jokaisessa on kaksi 16-bittistä kontrollirekisteriä, eli jokainen rekisteri kontrolloi kahta pinniä OMAP-prosessorissa. Koska osa näistä OMAP-prosessorin pinneistä on tuotu Beagleboardin laajennusväylään, nämä prosessorin rekisterit kontrolloivat samalla laajennusväylän pinnejä. [7 s.773–783.]

Taulukko 1. GPIO kontrollirekistereiden perusosoitteet [7 s. 3378].

GPIO-1	0x48310000
GPIO-2	0x49050000
GPIO-3	0x49052000
GPIO-4	0x49054000
GPIO-5	0x49056000
GPIO-6	0x49058000

Jos halutaan kontrolloida suoraan kutakin GPIO-signaalia ohjelmallisesti ilman sysfs-väylää, pitää etsiä ensin oikea kontrollipiiri, sitten siltä piiriltä oikea GPIO-signaali ja lopuksi kirjoittaa tai lukea muistirekistereiden kautta halutut asetukset tai datat. GPIO-kontrollipiirien perusmuistiosoitteet on lueteltu taulukossa 1. Kullakin piirillä on 32 GPIO-signaalia, eli teoreettinen maksimimäärä GPIO-signaaleja OMAP3-prosessorilla on 192. Todellisuudessa Beagleboardin GPIO-numerointi nousee tämän yli, koska lisälaitteiden ja laajennuskorttien GPIO-signaalit numeroidaan myös.

3 Web-ohjauksen tekniikkaa

3.1 Web-tekniikan perusteet

3.1.1 HTTP-protokolla

HTTP-protokolla (RFC 2116, http 1.1) eli Hypertext transfer protocol on WWW-palvelimen ja selaimen väliseen tiedonsiirtoon käytettävä protokolla. Sen toiminta yksinkertaistettuna perustuu siihen, että asiakasohjelma lähettää palvelinohjelmalle pyynnön, jossa kerrotaan halutun resurssin (WWW-sivun) osoite (URL). Palvelinohjelmisto lähettää vastauksena joko halutun resurssin tai muun statustiedon, esimerkiksi: ”sivua ei löydy”, ”sivu on siirretty” tai ”käyttäjän oikeudet eivät riitä sivun näyttämiseen”. HTTP-protokolla on tekstimuotoinen ja toimii TCP-yhteyden päällä. OSI-mallissa HTTP on ylimmällä eli sovellustasolla.

HTTP-sivu koostuu HEADER- ja FOOTER-osista. HEADER-osassa on erilaisia muuttujia, esimerkiksi palvelimen osoite, evästeet ja mahdollisen dataosion (BODY) datatyyppi, koodaus ja koko jos datatyyppi on jokin muu kuin oletustyyppi. BODY-osassa taas on varsinainen käyttäjälle esitettävä tieto. Usein BODY-osa on tyhjä, ja tiedon siirtoon käytetään HEADER-osaa.

HTTP-yhteys ei ole niin sanotusti tilallinen, eli palvelinohjelmisto ei seuraa asiakasohjelmiston tilaa sivunhakujen välillä. HTTP-standardin versiossa 1.0 TCP-yhteys suljettiin jokaisen sivunhaun välillä, mutta versiossa 1.1 on herätemekanismi, jolla TCP-yhteys voidaan jättää päälle hetkeksi haun jälkeen, koska usein sivun haun yhteydessä tehdään muitakin hakuja samalta palvelimelta. Jos WWW-sovelluksen logiikka vaatii käyttäjän tilan seurantaa sivun hausta toiseen, se joudutaan tekemään sovelluksessa ohjelmallisesti. Yleisesti WWW-perustaisen ohjelman sisäisen logiikan tilan seurantaan käytettyjä tapoja ovat evästeet tai muut istuntokohtaiset muuttujat joko URL-uudelleenkirjoitusta tai piilotettuja muuttujia käyttäen.

Resurssien hakemiseen HTTP-protokollassa on GET- ja POST-metodit. HEAD-metodilla haetaan ns. HTTP-otsikkotiedot, eli esimerkiksi tutkitaan, onko sivu muuttunut sen jälkeen kun asiakasohjelma talletti sen välimuistiinsa. GET-metodilla haetaan tietoa.

Standardin mukaan jokaisen peräkkäisen samoilla parametreilla tehtävän GET-haun pitää tuottaa tuloksena sama sivu. POST-metodilla taas yleensä lähetetään palvelimelle tietoa, esimerkiksi jonkin lomakkeen tiedot, joiden perusteella palvelin muokkaa sivun halutuksi. Muitakin HTTP-metodeita löytyy, ja tarkemmin niihin voi perehtyä rfc-2116 lukemisella.

Salattuun yhteyteen käytetään HTTPS-protokollaa, joka on HTTP-protokolla SSL- tai TLS-tiedonsalausprotokollalla salattuna (rfc-2128). Yhteyteen vaaditaan WWW-palvelimeen asennettu sertifikaatti, jonka myöntäjän asiakasohjelmisto tunnistaa. WWW-selaimien mukana toimitetaan yleisimpien sertifikaattivalmistajien juurisertifikaatit. Ongelmaton toiminta vaatii, että myös WWW-palvelimen sertifikaatti on näiden samojen myöntäjien myöntämä tai ainakin allekirjoittama. Tiedon salaus lisää laitteiston kuormitusta normaaliin WWW-palvelimeen verrattuna, joten aika usein salataan ainoastaan kriittisin osa tiedosta. Esimerkiksi käyttäjän kirjautuessa WWW-sovellukseen sisään salataan se yhteys, jossa lähetetään käyttäjän tunnus ja salasana.[4.]

3.1.2 WWW-palvelin

WWW-palvelin on ohjelma, joka tarjoaa HTTP-protokollaa käyttäen asiakasohjelmalle erilaisia resursseja. Asiakasohjelma lähettää WWW-palvelimelle URL:n, jonka perusteella WWW-palvelin etsii oikean tiedoston tai käynnistää ohjelman, joka luo asiakasohjelmalle lähetettävän vastauksen. Resurssit voivat olla tyypillisesti HTML-koodattua tekstiä, kuvia, selaimessa suoritettavia komentosarjoja, tyylitiedostoja tai ohjelmatiedostoja, jotka suoritetaan asiakaskoneen selaimen lisäosassa, esimerkiksi Java-appletit tai Adobe Flash -ohjelmat. Muitakin tiedostotyyppejä voidaan WWW-palvelimen avulla siirtää, ja on asiakasohjelmaan asennetuista lisäosista kiinni, että mitä asiakasohjelma näille tiedostoille tekee. Viime kädessä on mahdollista tallettaa WWW-palvelimen lähettämät tiedostot levylle.

Tarjottu resurssi voi olla staattinen, eli jokaisella hakukerralla samanlainen, tai dynaaminen, eli erilaisten parametrien perusteella hakuhetkellä luotava sivu. Staattisten sivujen tuottaminen on luonnollisesti paljon www-ohjelmistoa ja palvelinlaitteistoa vähemmän kuormittavaa kuin dynaamisten sivujen. Tyypillisessä dynaamisessa sivustossa on www-palvelimen lisäksi jonkinlainen tietokanta, ja sivusto rakennetaan jollain, yleensä

WWW-sivustojen rakentamiseen optimoidulla ohjelmointikielellä tai yleiskäyttöisemmällä komentosarjakielellä. Esimerkkejä suosituista komentosarjakielistä ovat php, java, python ja perl.

Suosittuja WWW-palvelinohjelmistoja ovat nykyään Apache, Microsoftin IIS, nginx ja erilaiset javaan pohjautuvat sovelluspalvelimet. Sulautetuissa laitteissa ovat suosiossa vähemmän muistia käyttävät WWW-ohjelmistot, esimerkiksi Lighttpd ja tHttpd.

3.1.3 CGI-rajapinta

Common gateway interface on WWW-palvelimissa käytetty standardoitu tapa, jolla sivuja voidaan luoda dynaamisesti palvelinohjelman ulkopuoleisella ohjelmalla. CGI version 1.1 -standardi on määritetty rfc-3875 -dokumentissa. Tyypillinen CGI-ohjelma on perl- tai python-kielellä kirjoitettu komentosarja, joka joko saamiensa parametrien tai muun muuttuvan tiedon perusteella tuottaa html-sivun, joka lähetetään asiakkaan www-selaimelle. CGI-komentosarja voi toki tuottaa minkälaisen tiedoston tahansa.

Komentosarjakielellä tehtyjen ohjelmien ongelma on WWW-palvelimessa se, että niiden suorittamiseksi pitää yleensä käynnistää vastaava suorittava ohjelma, esimerkiksi perl- tai java-tulkki. Tämä taas lisää WWW-palvelimen muistinkäyttöä, ja jokaisen uuden prosessin luontikerta kestää suhteellisen pitkään. Käynnistystä tai uusien prosessien luontia voi nopeuttaa käyttämällä esimerkiksi C- tai C++-kielellä kirjoitettuja valmiiksi käännettyjä CGI-ohjelmia tai käyttämällä WWW-palvelimen moduuliksi ohjelmoitua komentosarjakielen kääntäjää, esimerkiksi mod_perl Apacheen. FastCGI-tekniikalla voidaan pitää CGI:tä suorittava ohjelma käynnissä palvelemassa useampia CGI-kutsuja.

Vaihtoehtoja CGI-komentosarjoille ovat WWW-palvelinohjelmiston ohjelmointirajapinnan avulla luotavat lisämoduulit. Näillä moduuleilla on etuna se, että yleensä toiminta on paremmin integroitu WWW-palvelimen toimintaan. Ohjelman suoritus on nopeampaa ja muistinkäytön kannalta tehokkaampaa, jos ladataan kirjastomuotoinen käännetty moduuli tarvittaessa sen sijaan että käynnistettäisiin uusi ohjelma joka kerran kun komentosarjaa kutsutaan. Esimerkkejä yleisesti käytetyistä WWW-palvelimen moduu-

leista ovat PHP ja SSL, lisäksi esimerkiksi perl- ja python-komentosarjoja voidaan käyttää omien vastaavien moduuliensa kautta Apachessa. [2.]

3.2 Verkkoliitännät

WWW-pohjainen käyttöliittymä tarvitsee tietysti myös yhteyden verkkoon joko langattoman WLAN-verkon kautta tai ethernet-yhteyden. Perusmallin Beagleboardilta puuttuu verkkoliitäntä. Nykyisin on saatavilla useita erilaisia USB-liitäntäisiä verkkokortteja sekä ethernet- että WLAN-verkkoon. Jos kortti tukee USB 2.0 –väylää, sen voi liittää Beagleboardin isompaan USB-liittimeen, jos taas ei, tarvitaan väliin USB-keskitin. Linuxin ytimestä löytyy tuki usealle saatavilla olevalle USB-verkkokortille.

WLAN-verkkokortin avulla Beagleboard voidaan konfiguroida liittymään johonkin tiettyyn WLAN-verkkoon ja hakemaan DHCP-protokollalla osoite. Tällöin kannattaa verkon DHCP-palvelimelle konfiguroida Beagleboardia varten pysyvä IP-osoite, jolla laitteeseen saa helposti yhteyden. Voidaan myös sopia jokin kiinteä IP-osoiteverkko Beagleboardin käyttöön DHCP:n käytön sijaan.

Toinen vaihtoehto on konfiguroida WLAN-adhoc -tyyppinen verkko. Adhoc-verkossa Beagleboard toimii tukiasemana, johon voi ottaa yhteyden sovituilla parametreilla ja salasanoilla. Tällöin pitää joko Beagleboardilla olla käynnissä DHCP-palvelinohjelma josta liittyvä kone saa osoitteen, tai sitten pitää käyttää jotain sovittua kiinteää IP-verkkonumerointia. Tietoturvan vuoksi WLAN-verkon tulisi olla salattu Wpa2-salauksella sekä adhoc- että muussa tapauksessa.

Kolmas vaihtoehto on määritellä Beagleboard oman WLAN-verkkonsa tukiasemaksi. Erona tällä menetelmällä ja adhoc-verkolla on se, että adhoc-verkossa laitteet keskustelevat keskenään, tukiasemaverkossa taas laitteet keskustelevat tukiaseman kautta. Muitakin eroja näillä verkkotyypeillä toki on mutta tässä niihin ei sen tarkemmin syvennytä. Myös tukiasemana Beagleboardiin pitää joko asentaa DHCP-palvelinohjelma tai määritellä kiinteä verkko ja ennalta sovitut IP-osoitteet.

Palomuurista ei tarvitse avata kuin TCP-portit 80 HTTP:lle, 443 SSL:le ja haluttaessa 22 SSH:lle. Lisäksi tarvitaan UDP-portti 67 DHCP-viesteille. Palomuuriohjelmisto ufw (un-

complicated firewall) on Ubuntussa mukana tai ainakin saatavana valmiiksi käännettynä pakettivarastosta. Ufw on yksinkertaistettu käyttöliittymä Linuxin normaaleille iptables- ja netfilter-palomuurisäännöille.

4 Testiympäristö

4.1 Käyttöjärjestelmä

Käyttöjärjestelmäksi Beagleboardille asennettiin Ubuntu Linux 11.10. Aluksi haettiin asennuspaketti Ubuntu ARM -projektin verkkosivuilta [8]. Alasivuilta löytyi OMAP3-prosessorille käännetty versio, jonka voi asentaa SD-muistikortille. Minimaalisessa palvelinversiossa on mukana tarvittavat perusohjelmat, kuten ytimen laiteajurit verkkokorteille ja tietoliikenneohjelmia, kuten SSH-palvelin. Lisää valmiiksi käännettyjä ohjelmia löytyy Ubuntun normaalien päivitystyökalujen avulla heti, kun verkkoyhteys on saatu toimintaan. Kaikkia Intel-prosessoria käyttävän työpöytä-Ubuntun ohjelmia ei ole saatavilla, mutta melko kattava valikoima kuitenkin. Ubuntu ARM -projektin alasivuilta voi tutkia, mikä on jonkin yksittäisen ohjelman käännöksen status ja toimivuus. Muita vaihtoehtoja Ubuntulle löytyisi useampia kuin tässä voi luetella. Esimerkkeinä suosituista vaihtoehdoista sulautettuihin Linux-projekteihin olisivat Ångström-Linux tai buildrootin tai openembeddedin avulla käännetty oma Linux-versio.

Taulukko 2. Beagleboardille asennettujen ohjelmien versioita.

Libc	Eglibc-2.13
Linux kernel	3.1.6-x6
Gcc – kääntäjä	4.61

Ristiinkääntäjää varten tarkistettiin Beagleboardilta joidenkin Ubuntussa mukana olevien kirjastojen ja ohjelmien versioita (taulukko 2). Joissakin tapauksissa käännetyn ohjelman toiminta ei onnistu jos se linkitetään liian vanhan tai liian uuden kirjastoversion kanssa, tästä syystä kannattaa valita ristiinkääntäjäympäristöön suunnilleen samoja versioita peruskirjastoista, kuin on siinä käyttöjärjestelmässä, jossa ohjelmaa on tarkoitus käyttää. Toinen vaihtoehto olisi linkittää kirjastot staattisesti, mutta tämä suurentaa ohjelmapakettien kokoa ja muistin käyttöä.

Seuraavaksi selvitettiin millä kääntäjän optimoinneilla Ubuntu 11.10 on käännetty. Ubuntun wikistä ARM-sivuilta [8] löytyy aiheesta tietoa, mutta melko yleisellä tasolla. Tarkemmin tieto löytyi tässä tapauksessa asentamalla gcc-kääntäjä Beagleboardille Ubuntun pakettivarastosta ja käynnistämällä se sopivalla parametrilla. Normaalisti tuotantopalvelimiin tai -järjestelmiin ei asenneta kääntäjäympäristöä tai työkaluja. Tärkein osa kääntäjän tulostamasta tiedosta on esitetty alla.

```
bash$ gcc –v
…
--with-arch=armv7-a --with-float=softfp –with-fpu=vfpv3-d16
…
```

Arkkitehtuuriversio on siis armv7-a, liukulukujärjestelmä (softfp) on ohjelmistopohjainen, mutta voi käyttää liukulukuprosessoria laskuissa, jos laitteistossa sellainen on. Ohjelman kääntäjä valitsee arkkitehtuurin perusteella, käytetäänkö ohjelmistoa vai laitteistoa liukulukujen laskentaan. Tämä tila on yhteensopiva täysin ohjelmistopohjaista (soft) liukulukulaskentaa käyttäville ohjelmille. Liukulukuoptimoinnit ovat tärkeitä lähinnä niissä ohjelmissa, joissa lasketaan runsaasti liukuluvuilla. Lisää dokumentointia ARM-prosessorin eri liukulukuoptimoinneista löytyy Debianin wikistä [9].

4.2 WWW-palvelin

WWW-palvelinohjelmistoksi valittiin Lighttpd, koska se on nopea ja muistin käytöltään pieni. Se on myös helposti ristiinkäännettävissä ARM-prosessorille. Käännöstyötä varten työasemaan asennettiin crosstools-ng -ohjelma, jolla saa valittua ja optimoitua tämän projektin ristiinkääntöä varten käännöstyökalut hieman vapaammin kuin joillakin muilla vastaavilla ohjelmistoilla. Kirjoitushetkellä uusin versio crosstool-ng:stä on 1.16.0.

Lighttpd:n voi kääntää, ja siinä on perustoiminnallisuus ilman yhtäkään näistä lisäkirjastoista, joita tässä työssä käännettiin, mutta harjoituksen vuoksi otettiin muutama tyypillinen ohjelmakirjasto ja käännettiin ne. Kirjastot käännettiin dynaamista linkitystä varten.

4.2.1 Crosstool-NG asennus

Käännöskoneena toimi Ubuntu 12.04 työasema, jossa on AMD x2-64 -prosessori ja 6 Gt RAM-muistia. Crosstool-NG on yksi monista eri prosessoriarkkitehtuureille riistiinkääntävistä ohjelmistoista. Erona joihinkin muihin yleisesti käytettyihin vastaaviin ohjelmistoihin on se, että crosstool-ng:ssä voi hieman vapaammin valita eri versiot työkaluketjuun tulevista ohjelmista ja kirjastoista, esimerkiksi ytimestä, c-kääntäjästä tai c-kirjastosta (libc). Täysin vapaa ei valinta ole tässäkään ohjelmistossa, koska kirjastojen eri versiot eivät välttämättä täysin toimi ristiinkäännettynä ohjelmointivirheiden ja muiden ongelmien vuoksi. Monet vastaavat ristiinkääntävät ohjelmistot ovat raskaita jonkin yksittäisen ohjelmapaketin kääntämiseen. Esimerkiksi buildroot ja openembedded on molemmat suunniteltu kokonaisen tiedostojärjestelmän osien valitsemiseen ja kääntämiseen.

Asennustyö aloitettiin perinteiseen Linux-malliin hakemalla ohjelmiston lähdekoodi ja purkamalla se sopivaan hakemistoon. Seuraavaksi siirryttiin kyseiseen hakemistoon ja konfiguroitiin sekä asennettiin ohjelma samaan hakemistoon. Tässä vaiheessa piti asentaa käännöstyöasemaan pakettienhallinnan kautta joitakin puuttuvia ohjelmia tai kirjastoja, jotta configure-vaihe menee virheittä läpi. Tarvittaessa ensimmäiset vaiheet voi käynnistää uudestaan. Alla on esitetty käskyt näiden vaiheiden suorittamiseksi.

```
bash$ ./configure --enable-local
bash$ make
bash$ make install
```

Kun asennus oli mennyt virheittä läpi, voitiin käynnistää varsinainen ristiinkääntäjän konfigurointi ja asennus. Aluksi listattiin kääntäjässä olevat esiasetukset, jotka tekevät joitain perusasetuksia valmiiksi. Tarjolla olevista vaihtoehdoista valittiin *arm-cortex_a8-linux-gnueabi*, joka on lähinnä haluttua prosessorin kannalta. Varsinainen konfigurointiohjelma on ulkoasultaan tuttu Linux-ydintä kääntäneille. Alla on esitetty osa kääntäjän esiasetuksista kääntäjän tulostamana sekä valitun esiasetuksen käyttöönottokäsky.

```
bash$ ./ct-ng list-samples
…
…
[L.X]   arm-cortex_a15-linux-gnueabi
[L..]   arm-cortex_a8-linux-gnueabi
```

```
[L..]   arm-davinci-linux-gnueabi
…
…
bash$ ./ct-ng arm-cortex_a8-linux-gnueabi
bash$ ./ct-ng menuconfig
```

Ohjelman valikkojen avulla tehtiin joitain tarpeellisia muutoksia. Taulukossa 3 on luetel-tu tähän projektiin tehdyt muutokset edellä valittuihin perusasetuksiin.

Taulukko 3. Crosstool-ng:n konfigurointivaiheessa menujen kautta tehtäviä muutoksia.

Menu	**Kohta**	**Muutos**
Paths and misc options	Prefix path	/usr/local/xtools/${CT_TARGET}
Target options	Floating point	softfp
Target options	Use specific fpu	vfpv3
Toolchain options	Tuple's alias	arm-linux
Operating system	Linux kernel version	3.1.10
Binary utilities	Binutils version	2.20.1a(oletus) tai 2.21.1a
C compiler	Gcc version	4.6.1
c-library	C library	Eglibc
c-library	Eglibc version	2_13
Debug facilities	Oman maun mukaan, ainakin gdb kannattaa ottaa	
Companion libraries	ppl version	0.11.2 valitun gcc-version kään-tämiseksi
Companion libraries	Cloog/ppl version	1.15.11

Valikko-ohjelmassa tehdyt muutokset tallentuvat valitsemalla päävalikosta kohta exit. Jos käännöstyö tehdään normaalikäyttäjän oikeuksilla ja asennushakemistoksi valitaan esimerkiksi /usr/local -hakemiston alla oleva hakemisto, hakemisto kannattaa luoda etukäteen ja sille käyttäjätunnukselle antaa omistusoikeus, jolla käännöstyö tehdään. Lopuksi käynnistettiin varsinainen ristiinkääntäjän käännösohjelma, joka valituista vaih-toehdoista ja käännöskoneen nopeudesta ja verkkoliitännästä riippuen, saattaa kestää tunnin tai kauemminkin. Alla on esitetty komennot, joilla luodaan tarvittava hakemisto oikeuksineen ja käynnistetään ristiinkääntäjä.

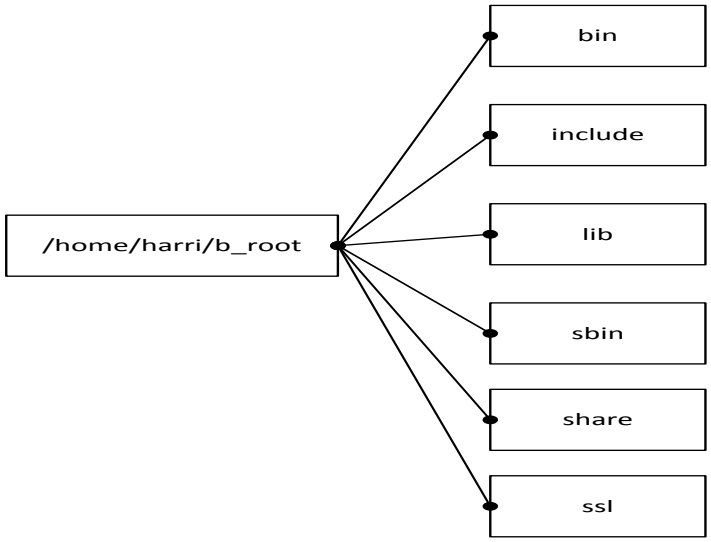
```
bash$ sudo mkdir /usr/local/xtools && sudo chown harri.harri \
/usr/local/xtools
bash$ ./ct-ng build
```

Käännöksen mentyä virheettä läpi lisättiin PATH -ympäristömuuttujaan ristiinkääntäjän bin-hakemisto. Esimerkkikomento on esitetty alla. Pysyvästi tämän asetuksen saa käyttöön lisäämällä vastaavan tiedon .profile -tiedostoon kotihakemistossa.

```
bash$ export PATH=/usr/local/xtools/arm-cortex_a8-linux-gnueabi/bin:$PATH
```

4.2.2 Openssl- ja muiden kirjastojen kääntö

Zlib on pakkausaliohjelmia sisältävä kirjasto, jota tarvitaan muun muassa openssl-kirjaston ja Lighttpd-ohjelmiston kääntämisessä. Muutkin Ubuntun ohjelmistot käyttävät zlib:iä. Pcre (perl compatible regular expressions) taas mahdollistaa Lighttpd:n konfiguroinnin perl-tyylisillä lausekkeilla. Openssl on Linuxin de-fakto -kirjasto ssl-, tls- ja muille salausalgoritmeille. Openssl:n mukana on muun muassa md5-, idea-, blowfish-, dsa-, rsa-, aes- ja monia muita salausalgoritmeja. Näistä kirjastoista ja Lighttpd:stä käytettiin uusinta versiota, joka käännöshetkellä oli saatavilla.

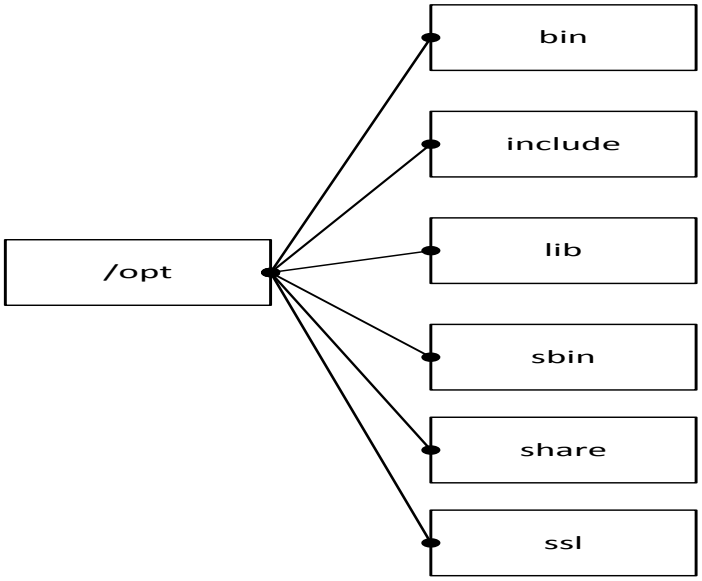


Kuva 3. Lighttpd:n kääntämisessä tarvittavat kirjastot konfiguroidaan ja asennetaan aluksi väliaikaiseen hakemistoon.

Asennusprojekti aloitettiin kirjastoista, joita tarvittiin myöhempien ohjelmien kääntämisen ja linkittämisen aikana. Koska näitä kirjastoja tarvittiin sekä toistensa että Lighttpd:n kääntämisen ja linkitysvaiheen aikana, oli selkeintä tehdä käännöstyö kahdessa vaiheessa. Ensimmäisessä vaiheessa käännettiin ohjelmat ja kirjastot ja asennettiin ne väliaikaiseen hakemistopuuhun (kuva 3). Puun juurihakemistona käytettiin

/home/harri/b_root -hakemistoa. Hakemiston alle syntyi käännöksen aikana eri hakemistoja, joiden sijainti piti kertoa kääntäjälle ja linkittäjälle, jotta eri kirjastot osattiin linkittää oikein käännöksen aikana.

Kun kirjastot oli käännetty tähän väliaikaiseen hakemistoon, ne voitiin helposti kääntää uudelleen, mutta tällä kertaa konfiguroituna lopullisen asennushakemistonsa mukaan. Lopullinen kunkin ohjelman käännös asennettiin johonkin sopivaan väliaikaishakemistoon, josta ohjelma oli helppo pakata ja siirtää Beagleboardille lopulliseen asennushakemistoonsa. Syy siihen, että käännöstä ei tehty yhdessä vaiheessa, oli se, että Linuxissa jaettuihin kirjastoihin kirjoitetaan konfigurointitietoihin asennushakemisto. Tämä olisi pitänyt muuttaa sitten joka ainoaan kirjastotiedostoon, jos olisi haluttu käyttää kerran käännettyjä kirjastoja useammassa eri hakemistossa.



Kuva 4. Lopullinen asennushakemisto Beagleboardilla.

Omat versiot näistä ohjelmista, paitsi zlib:stä, Beagleboardille asennettiin hakemiston /opt alle (kuva 4). Kaikki nämä ohjelmat on myös saatavilla Ubuntulle valmiiksi käännettynä, mutta projekti tehtiin harjoituksen vuoksi manuaalisesti ja poistettiin Beagleboardilta sinne mahdollisesti Ubuntun mukana asentuneet versiot, paitsi zlib. Monessa muussa Ubuntun mukana toimitetussa paketissa oletetaan zlib-kirjaston asennus pakettienhallinnan kautta, joten tämän paketin poistaminen rikkoo aika pahasti käyttöjärjestelmän. Toisaalta työn aihealueeseen ei suoranaisesti kuulu se, miten oman version zlib:stä saa mukaan pakettienhallintaan.

Aluksi haettiin zlib-kirjastosta uusimman version 1.2.7 lähdekoodi ja purettiin se sopivaan hakemistoon. Seuraavaksi siirryttiin tähän luotuun hakemistoon, ja konfiguroitiin kirjasto kääntämistä varten. Tämän jälkeen ohjelma käännettiin ja asennettiin syntyneet ohjelmat kuvan 3 tilapäishakemistoon. Käskyt konfigurointiin ja kääntämiseen on esitetty alla.

```
bash$ CC=arm-linux-gcc AR=arm-linux-ar RANLIB=arm-linux-ranlib \
./configure --prefix=/home/harri/b_root
bash$ make
bash$ make install
```

Seuraavaksi asennettiin vastaavalla tavalla pcre-kirjastot. Uusimman version (8.20) lähdekoodi haettiin ja purettiin se johonkin sopivaan hakemistoon. Tähän hakemistoon siirryttyä konfiguroitiin, käännettiin ja asennettiin ohjelma kuvan 3 hakemistoon. Komennot näihin toimiin on esitetty alla.

```
bash$ ./configure --host=arm-linux --disable-static \
--enable-shared --prefix=/home/harri/b_root
bash$ make
bash$ make install
```

Openssl-kirjaston kääntäminen sujui samaan malliin kuin edellistenkin kirjastojen, eli haettiin uusimman version (1.0.1c) lähdekoodi ja purettiin se sopivaan hakemistoon. Alla esitetyssä konfigurointikäskyssä kannattaa huomata lopussa oleva *linux-armv4*, joka konfiguroi kääntäjän tuottamaan optimoitua koodia arm-v4 arkkitehtuurille.

```
bash$ CC=arm-linux-gcc RANLIB=arm-linux-ranlib AR=arm-linux-ar \
NM=arm-linux-nm ./Configure --prefix=/home/harri/b_root -shared linux-armv4
```

Beagleboardilla on kuitenkin prosessori, joka käyttää hieman edistyneempää arkkitehtuuria. Tiedostoa Makefile editoitiin ja lisättiin muutama koodin optimointikäsky. Riville, joka alkaa CFLAG, lisättiin loppuun seuraavat termit:

```
-mtune=cortex-a8 -march=armv7-a -mabi=aapcs-linux -mfloat-abi=softfp
-mfpu=vfpv3
```

Makefile tallennettiin, ja alla olevilla käskyillä openssl käännettiin ja asennettiin kuten edellä olevat kirjastot samaan tilapäishakemistoon (kuva 3).

```
bash$ make
bash$ make install
```

Seuraavaksi käännettiin asennuskelpoiset versiot pcre- ja openssl-kirjastoista. Tämä tapahtui muuten samoilla käskyillä kuin aiemmin tässä luvussa, paitsi että *./configure* -vaiheessa –prefix=/home/harri/b_root korvattiin –prefix=/opt ja *make install* -vaiheessa annettiin ympäristömuuttujassa erillinen asennushakemisto, josta ohjelmat oli helppo pakata Beagleboardille siirtoa varten.

Seuraavassa on esitettynä aluksi käskyt pcre-asennuspaketin kääntöön ja sen jälkeen vastaavat käskyt openssl-kirjastolle. Ennen openssl:n kääntämistä kannattaa muistaa editoida samat optimoinnit Makefile-tiedostoon kuin aiemmin. Pakkaamis- ja asennusvalmiit tiedostot löytyvät siis b_root hakemiston alta nimillä pcreinst ja osslinst. Näissä molemmissa hakemistoissa on aluksi opt-hakemisto ja sen alla muut tarpeelliset hakemistot.

```
pcre:
bash$ ./configure --host=arm-linux --disable-static \
--enable-shared --prefix=/opt
bash$ make
bash$ DESTDIR=/home/harri/b_root/pcreinst make install

openssl:
bash$ CC=arm-linux-gcc RANLIB=arm-linux-ranlib AR=arm-linux-ar \
NM=arm-linux-nm ./Configure --prefix=/opt -shared linux-armv4
bash$ make
bash$ make INSTALL_PREFIX=/home/harri/b_root/osslinst install
```

4.2.3 Lighttpd:n ristiinkääntö

Aluksi haettiin lighttpd-1.4.30 asennuspaketti ja purettiin se sopivaan hakemistoon. Lighttpd:n konfigurointi ja kääntäminen vaati hieman enemmän polkuja eri kirjastoihin ja include-tiedostoihin, joten eri asennusparametreja varten kirjoitettiin shell-komentosarja. Komentosarja on esitetty kokonaisuudessaan liitteessä 2. Konfigurointi

sujui siis käynnistämällä tämä komentosarja lähdekoodihakemistossa. Lighttpd:tä ei kuitenkaan tarvitse konfiguroida kuvan 3 tilapäishakemistoon, joten tehtiin suoraan asennuskelpoinen paketti hakemistoon b_root/lightinst. Tähän hakemistoon syntyi opt-hakemisto ja sen alle Lighttpd:n ohjelmatiedosto ja kirjastot omiin hakemistoihinsa. Käskyt kääntämiseen ja asennuspaketin asennukseen on esitetty alla.

```
bash$ make
bash$ DESTDIR=/home/harri/b_root/lightinst make install
```

4.2.4 Ohjelmien asennus Beagleboardille

Edellisissä luvuissa käännetyt ja luodut ohjelmapaketit kopioitiin Beagleboardin /opt -hakemistoon ja purettiin ne sinne. Seuraavaksi esitetyllä käskyllä varmistettiin että ohjelmilla on oikeat omistajat.

```
bash$ sudo chown –R root.root /opt/*
```

/opt/lib -hakemisto lisättiin Linuxin kirjastotiedostojen hakupolkuun lisäämällä /etc/ld.conf.d -hakemistoon uusi tekstitiedosto. Tälle tiedostolle annettiin nimeksi oma_opt_lib.conf ja kirjoitettiin tiedostoon alla esitetty yksi rivi tekstiä.

```
/opt/lib
```

Lopuksi kirjastojen automaattiset hakupolut päivitettiin seuraavalla käskyllä.

```
bash$ sudo ldconfig
```

Ohjelmien hakupolkuun lisättiin /opt/bin -hakemisto. Käyttäjälle tämä tapahtuu helpoimmin lisäämällä tieto .profile -tiedostoon kotihakemistossa.

4.2.5 WWW-palvelimen asetukset

Lighttpd:n konfigurointitiedostot asennettiin /etc/lighttpd -hakemistoon Beagleboardille. Käännöksen aikainen asennusohjelma ei kopioi näitä tiedostoja asennushakemistoihin, joten ne piti hakea lähdekoodihakemiston alaisesta doc/config -hakemistosta. Tästä

hakemistosta voi alihakemistoineen kopioida kaiken muun paitsi Makefile-alkuiset tiedostot ja siirtää ne Beagleboardille /etc/lighttpd -hakemistoon.

Lighttpd konfiguroitiin editoimalla /etc/lighttpd/lighttpd.conf -tiedostoa. Alla on esitetty tärkeimmät muutokset tähän tiedostoon. Kohdassa index-file.names varmistetaan, että käytetty CGI-komentosarja käynnistetään, vaikka käyttäjä tulisi sivulle pelkällä palvelimen nimellä tai tässä tapauksessa IP-osoitteella.

```
var.server_root = "/var/www"
server.username  = "www-data"
server.groupname = "www-data"
server.document-root = "/var/www"
server.network-backend = "writev"
index-file.names += ("beaglecontrol.py", "index.html", …
```

Jos halutaan käyttöön SSL-salaus, pitää vastaava moduuli ottaa käyttöön /etc/lighttpd/lighttpd.conf -tiedostossa. Alla olevassa asetusesimerkissä on tehty aivan perusasetukset SSL-protokollan käyttöön. Loppupuolen ”scheme” -kohdasta lähtien ohjataan kaikki HTTP-protokollalla tulevat kutsut käyttämään salattua HTTPS-protokollaa. Lighttpd.pem -tiedosto sisältää palvelimen sertifikaatin ja avaimen. Se pitää luoda erikseen sopivaan paikkaan esimerkiksi openssl:n avulla. Itse luoduissa ja allekirjoitetuissa sertifikaateissa on se huono puoli, että selaimet varoittavat tietoturvan puutteesta kun sertifikaatin allekirjoittajaa ei tunneta. Yhteys on kuitenkin salattu, jos ohittaa selaimen varoitukset ja lataa sivun.

```
$SERVER["socket"] == ":443" {
  ssl.engine = "enable"
  ssl.pemfile = "/opt/ssl/certs/lighttpd.pem"
}

$HTTP["scheme"] == "http" {
    $HTTP["host"] =~ ".*" {
        url.redirect = (".*" => "https://%0$0")
    }
}
```

Muutama moduli otettiin käyttöön editoimalla /etc/lighttpd/modules.conf -tiedostoa. Poistettiin #-kommettimerkki alla esitettyjen rivien alusta. Mod_aliasta tarvitaan cgi-bin

-hakemiston käyttöönotossa alempana ja mod_redirectiä tarvitaan SSL-moduulin konfiguroinnissa yllä.

```
  "mod_alias",
  "mod_redirect",
include "conf.d/cgi.conf"
```

Seuraavaksi editoitiin conf.d/cgi.conf -tiedostoa ja poistettiin kommettimerkit muutaman rivin alusta. Nämä editoidut rivit on esitetty alla. Tämä tosin ei ole välttämätön vaihe, jos ei käytetä cgi-bin hakemistoa. CGI-komentosarjat toimivat muissakin hakemistoissa, jos *cgi.assign = (".py" => "/usr/bin/python")* tai vastaava määrittely muille tiedostoille löytyy.

```
alias.url += ( "/cgi-bin" => "/usr/lib/cgi-bin" )
$HTTP["url"] =~ "^/cgi-bin" {
   cgi.assign = ( "" => "" )
}
```

Lopuksi piti vielä varmistaa, että WWW-palvelin käynnistyy kun Beagleboard käynnistetään. Tämä tapahtui helpoimmin siten, että laitettiin /etc/init.d -hakemistoon käynnistyskomentosarja. Liitteessä 3 on esitetty käytetty käynnistyskomentosarja, joka on luotu Lighttpd:n wikin [10] esimerkin pohjalta. Tämä kopioitiin /etc/init.d -hakemistoon nimelle omalighttpd. Alla esitetyillä käskyillä varmistettiin että komentosarjalla on oikea omistaja ja käynnistysoikeus. Lopuksi lisättiin linkit käynnistysvalikoihin.

```
bash$ sudo chown root.root /etc/init.d/omalighttpd
bash$ sudo chmod a+x /etc/init.d/omalighttpd
bash$ sudo update-rc.d omalighttpd defaults
```

4.3 Verkkoasetukset

Beagleboardissa ei ole suoraan ethernet- tai WLAN-verkkoliittymää. TP-Link TL-WN231G WLAN-adapteri liitettiin USB-porttiin, ja tehtiin tarvittavat asetukset Ubuntuun. Ensimmäisenä ongelmana oli määritellä sellainen verkkoyhteys, jolla Beagleboard voi käydä hakemassa internetistä päivityksiä ja lisäpaketteja. Nämä päivitykset olisi voitu tietysti tuoda jollekin sisäverkon palvelimelle tai vaikka verkkolevylle ja asentaa

sieltä, mutta kehitys- ja testivaiheessa suora verkkoliittymä säästää aikaa. Asetusten jälkeen Beagleboard osasi liittyä WLAN-verkkoon ja hakea DHCP:llä IP-osoitteen. Verkossa oli päällä wpa2-salaus, ja kirjautuminen verkkoon tapahtui psk:n (passkey) avulla. Tämän jälkeen laitteeseen sai yhteyden vaikka ssh-ohjelmalla.

Lopuksi Beagleboardiin tehtiin asetukset joilla Beagleboard toimii tukiasemana. Toinen vaihtoehto, eli adhoc-verkko, jouduttiin hylkäämään, koska kirjoitushetkellä adhoc-verkkoa ei saatu käyttämään wpa2-salausta joidenkin verkkokortin ajureiden ongelmien vuoksi. Beagleboardille haettiin siis DHCP-palvelinohjelma ja tukiaseman asetuksia varten hostapd-ohjelma. Alla esitetyillä parametreilla DHCP-palvelimelle määriteltiin yksinkertainen verkko, josta palvelin jakaa osoitteita. Määritykset tehtiin /etc/dhcp/dhcpd.conf -tiedostoon.

```
subnet 10.1.1.0 netmask 255.255.255.0{
  range 10.1.1.2 10.1.1.10;
  option routers 10.1.1.1;
}
```

Alla on esitetty verkkoasetukset, jotka tehtiin sekä /etc/network/interfaces-tiedostoon että /etc/hostapd/hostapd.conf-tiedostoon. Interfaces-tiedostoon määriteltiin käytetty kiinteä IP-osoite ja verkko. Hostapd.conf-tiedostoon taas tulivat kaikki WLAN-asetukset. Näillä asetuksilla verkko käytti pelkkää wpa2-protokollaa. Salasanana kannattaa käyttää jotain hieman monimutkaisempaa merkkijonoa kuin mitä alla olevaan esimerkkiin on kirjoitettu.

tiedosto **/etc/network/interfaces**:

```
auto wlan0
iface wlan0 inet static
address 10.1.1.1
netmask 255.255.255.0
```

tiedosto **hostapd.conf**:

```
interface=wlan0
driver=nl80211
ssid=beagle
hw_mode=g
channel=6
```

```
wpa=2
wpa_passphrase=abcd12345wasd****
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
auth_algs=1
macaddr_acl=0
```

4.4 Laitteiston ohjaus

GPIO eli general purpose input/output on ohjelmallisesti ohjattava digitaalinen signaali. Kutakin GPIO-signaalia vastaa yksi pinni OMAP-prosessorissa, ja vastaavasti Beagleboardin laajennusväylässä. Beagleboardilla suuri osa laajennusporttien pinneistä on otettavissa GPIO-käyttöön asettamalla kyseisen pinnin multiplex-arvo sopivasti. Kun halutut pinnit on multipleksoitu GPIO-käyttöön, niitä voidaan käyttää esimerkiksi sysfs-tiedostojärjestelmän kautta. Tämä on käyttäjän ohjelmille (userspace) suositeltava tapa käyttää GPIO:ta. Osaa GPIO-signaaleista ei kuitenkaan voi tätä kautta tuoda sysfs-järjestelmään, koska käyttöjärjestelmän ydin on jo ottanut ne muuhun käyttöön tai mahdollisesti tuonut ne käytettäväksi jo eri systeemin kautta. Esimerkkinä tästä voi mainita Beagleboardin usr-ledit, jotka löytyvät GPIO-porteista 149 ja 150 ja joita toisaalta voi ohjata /sys/class/leds -hakemiston kautta. Yleisen GPIO-liittymän kautta niitä ei siis voi enää ottaa näillä numeroilla käyttöön. [11; 12.]

Jokin pinni saadaan GPIO-sysfs-käyttöön etsimällä ensin oikean pinnin numeroa vastaava GPIO-numero. Esimerkiksi laajennusväylän pinniä 21 vastaa GPIO-numero 130 (kuvasta 2). Hakemistosta /sys/class/gpio löytyy muun muassa tiedostot export ja unexport. Kun kirjoitetaan pinnin numero tekstinä tiedostoon export, ydin liittää kyseisen GPIO-signaalin sysfs-järjestelmään. Samaan hakemistoon ilmestyy uusi GPIO-signaalin numeroa vastaava hakemisto, tässä tapauksessa nimeltään gpio130. Tästä alihakemistosta löytyy muutama vastaavalla tavalla toimiva tiedosto. Osaa näistä tiedostoista voi lukea tai kirjoittaa, kuten value-tiedostoon voi kirjoittaa tekstinä 1 tai 0. Arvot muuttavat pinnin loogisen arvon (jännitteen) vastaavasti. Direction-tiedostolla voi valita onko väylän tarkoitus olla sisään- vai ulosmeno eli sallitaanko vain value-tiedoston luku vai myös kirjoitus. Edge-tiedostolla taas saadaan mahdollinen keskeytys kytkettyä luettavaan signaaliin, eli valitaan, tuleeko keskeytys signaalin muuttuessa

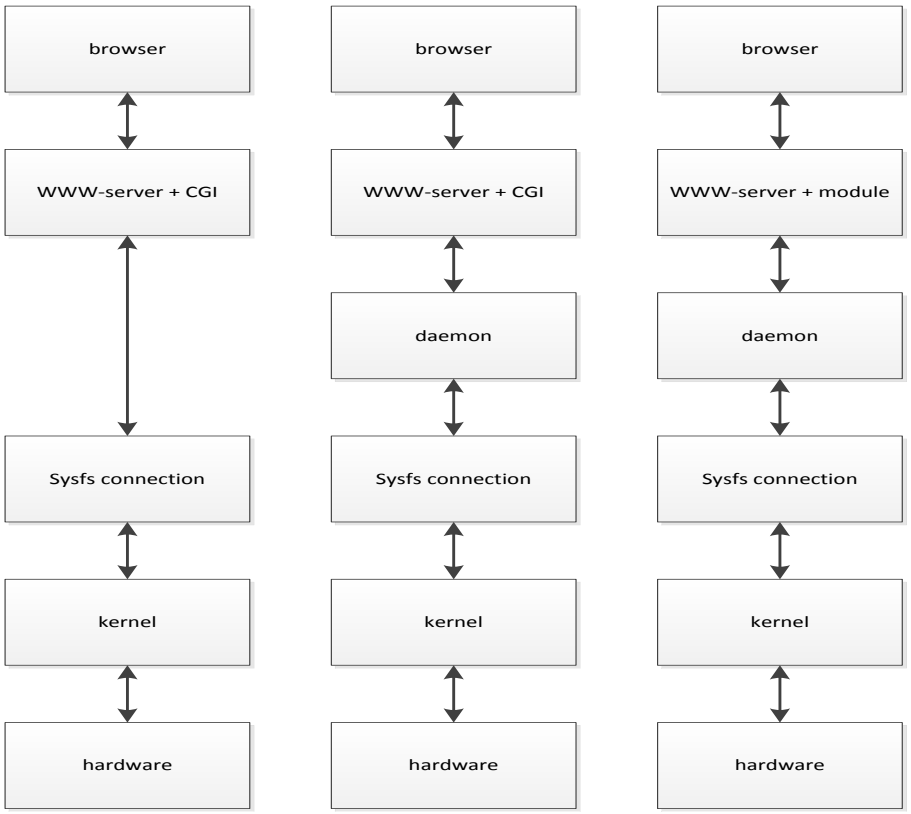
nousevalla, laskevalla tai molemmilla reunoilla. Oletusarvona keskeytys ei ole kytkettynä. [11; 12.]

Beagleboardin ledien ohjaus tapahtuu helpoimmin käyttämällä ytimen /sys/class/leds-rajapintaa, johon Beagleboardin ledit on valmiiksi tuotu. Tämän hakemiston alta löytyy molemmille ledeille oma hakemistonsa, nimettynä beagleboard::usr0 ja beaglebo ard::usr1. Näistä hakemistoista löytyvät ledien kontrollitiedostot. Tärkeä tiedosto on nimeltään trigger. Tähän tiedostoon kirjoittamalla valitaan ledin kytkentä esimerkiksi kiintolevyn aktiviteettilediksi tai heartbeat-lediksi, jonka välkyntänopeus on verrannollinen käyttöjärjestelmän kuormitukseen. Kirjoittamalla trigger-tiedostoon ”none” saadaan ledi käyttäjän hallintaan. Toinen tärkeä tiedosto on brightness, jolla valitaan ledin palaminen tai sammuksissaolo. Arvolla 0 ledi on sammuksissa ja arvolla 1 tai enemmän se palaa. Eri positiivisilla arvoilla ei ole eroa ledin toiminnassa. Tiedostossa max_brightness on kerrottu maksimiarvo, joka tiedostossa brightness voi lukea. Jos brightness-tiedostoon yrittää kirjoittaa suuremman luvun kuin tämä maksimiarvo, sinne kirjautuu arvoksi tämä maksimiarvo. [11; 12.]

SPI- ja I²C-väyliä kontrolloidaan Linuxissa normaalisti /dev -tiedostojärjestelmän avulla. Siinä vaiheessa kun kukin väylä on oikein määritelty käyttöjärjestelmälle, pitäisi kyseisestä hakemistosta löytyä väylää vastaava tiedosto. Testatussa Ubuntu-versiossa oli I²C-väylät luotu valmiiksi ja tiedostot /dev/i2c-1, /dev/i2c-2 ja /dev/i2c-3 löytyvät. SPI-väyliä ei ollut vastaavalla tavalla konfiguroitu valmiiksi.

5 Testisovelluksen toteutus

5.1 Laitteiston ohjaus WWW-liittymän kautta



Kuva 5. Kolme eri tapaa ohjata laitteistoa www-selaimella.

Kuvassa 5 on esitetty yleisellä tasolla kolme eri vaihtoehtoa välittää viesti WWW-palvelimen kautta laitteistolle sysfs-tiedostojärjestelmän kautta. Yksinkertaisimmassa vasemman puoleisessa tapauksessa selaimella otetaan yhteys www-palvelimeen, joka käynnistää CGI-komentosarjan. Tämä komentosarja avaa sysfs:n kautta yhteyden Beagleboardissa oleviin ledeihin ja lukee tai kirjoittaa halutun muutoksen. Ongelmana tässä keinossa on käytännössä se, että WWW-palvelinta pitää käyttää root-oikeuksilla, jotta CGI-komentosarjalla on riittävät oikeudet avata sysfs-tiedostot. Tämä taas on yleensä tietoturvasyistä huono idea. Lisäksi joissain palvelinohjelmistoissa tämä on pyritty jo ohjelmoinnissa tekemään hankalaksi. Esimerkiksi Lighttpd:ssa pitää editoida lähdekoodia ja kääntää palvelinohjelma uudestaan, jos haluaa käyttää palvelinta root-oikeuksilla.

Kakkosvaihtoehdossa, eli kuvassa keskimmäisessä, CGI-komentosarjan ja sysfs-liittymän väliin laitetaan prosessi, joka pitää huolen sysfs-tiedostojen avauksista ja käsittelystä. CGI-komentosarja viestii tämän prosessin kanssa jollain prosessien välisen viestinnän (IPC) menetelmällä, tässä tapauksesssa unix-soketeilla. Etuna on se, että WWW-palvelin voi toimia normaaleilla www-palvelimen käyttäjän oikeuksilla, mutta taustaprosessi toimii root-oikeuksilla.

Kolmannessa eli kuvan oikeanpuoleisessa vaihtoehdossa WWW-palvelimeen ohjelmoidaan moduuli, joka huolehtii CGI-komentosarjan toiminnallisuudesta. Muuten ohjelmisto on sama. Erona CGI:tä käyttävään versioon on käytännössä se, että WWW-palvelimen moduuli on ensimmäisestä kutsusta lähtien jatkuvasti ladattuna muistiin, eikä sitä tarvitse ladata uudestaan pahimmillaan jokaista uutta yhteyttä varten. Toisaalta joillekin CGI-komentosarjakielille on saatavissa vastaava toiminnallisuus kielen oman WWW-palvelinmoduulin avulla. Esimerkiksi mod_perl Apachessa on tällainen moduuli. Myös fastcgi:tä tai vastaavaa toiminnallisuutta voi käyttää optimoimaan ja vauhdittamaan CGI-komentosarjojen käyttöä, jos palvelinmoduulia ei ole saatavilla.

Testisovellus tehtiin käyttämään kuvan 5 keskimmäistä vaihtoehtoa. CGI-komentosarjana toimii pythonilla kirjoitettu ohjelma beaglecontrol.py. Taustaprosessina toimii c-kielellä kirjoitettu ja ARM-arkkitehtuurille käännetty ohjelma leddmn. Viestinvälitys CGI-komentosarjan ja taustaprosessin välillä toimii yhdellä unix-soketilla. CGI-komentosarjan suoritusta varten piti Ubuntun pakettivarastosta asentaa python-ohjelmisto Beagleboardille. Asennettu versio oli Python-2.7.2. CGI-komentosarja voidaan toki kirjoittaa esimerkiksi c-kielellä ja kääntää. Tämä on aika yleinen tapa sulautetuissa järjestelmissä, joissa halutaan säästää levytilaa ja muistitilaa.

5.2 Käyttöliittymä

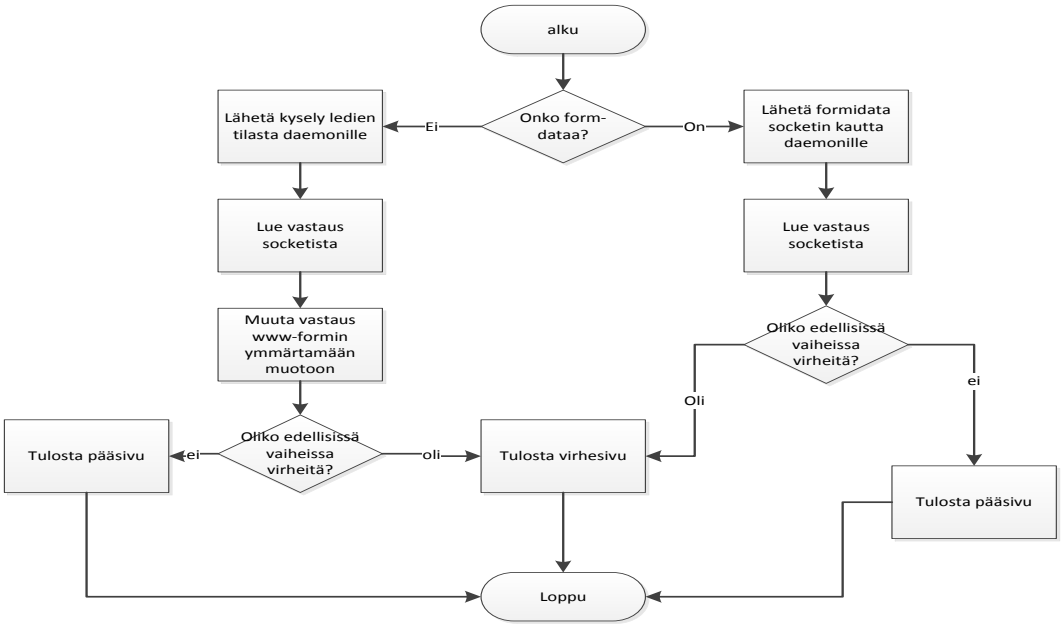
Käyttöliittymänä toimii yksinkertainen WWW-lomake, joka on esitetty kuvassa 6. Molemmilla ledeillä on oma asetusrivinsä. Ledin tilan sivun hakuhetkellä näkee asetusrivin lopusta, ja muutoksia tähän tilaan voi tehdä saman rivin radionapeilla. Painettaessa nappia Update radionappien senhetkiset arvot lähetetään käsiteltäväksi python-komentosarjalle, joka vastaa myös WWW-sivun tuottamisesta. Virheen sattuessa esitetään yleinen virheilmoitus erillisellä virhesivulla. Kuvasta 6 kannattaa huomata että

Google-Chrome selain näyttää ongelman sivun sertifikaatissa punaisena yliviivattuna https-tekstinä. Varoitus johtuu itse luodusta palvelinsertifikaatista, jonka allekirjoittajaa ei ole selaimen sertifikaattivarastossa. HTTPS-protokollaa kuitenkin käytetään yhteyden aikana, jos selaimen varoitus ohitetaan sivua ladattaessa.

Kuva 6. Käyttöliittymän tämänhetkinen versio www-selaimessa.

5.3 CGI-komentosarjan toiminta

Pythonilla kirjoitettu beaglecontrol.py -komentosarja luo sovelluksen käyttöliittymän. Komentosarjan päätoiminnot ovat WWW-lomakkeen luku ja tulkinta, ja toisaalta local-soketin kautta viestintä leddmn-prosessin kanssa. Leddmn-prosessi luo soketin, komentosarja taas avaa sen lukemista ja kirjoittamista varten. Tästä syystä soket-tiedostolle on annettava luku- ja kirjoitusoikeus kaikille käyttäjille. Kuvassa 7 on esitetty komentosarjan toiminta periaatteellisella tasolla.



Kuva 7. Vuokaavio CGI-komentosarjan toiminnasta periaatetasolla.

Ensimmäisellä kutsulla, kun WWW-lomakkeen dataa ei ole kutsun mukana, komentosarja kysyy ledien tilan leddmn-prosessilta ja laittaa lomakkeelle statustiedon vastaavaksi. Virheen sattuessa komentosarja tulostaa virhesivun. Seuraavilla kutsuilla, kun mukana on käyttäjän web-lomakeelle kirjoittamat tiedot, komentosarja lähettää ledien statuksen päivitetyn tiedon leddmn-prosessille, joka päivittää tiedot Beagleboardin laitteistolle. Tässä vaiheessa luetaan vastaus jälleen beagleboardilta. Taulukossa 4 on kerrottu mahdolliset viestit joita CGI-komentosarja lähettää ja vastaanottaa taustaprosessin kanssa. Jälleen virheen sattuessa tulostetaan virhesivu. Beaglecontrol.py CGI-komentosarjan lähdekoodi on esitetty liitteessä 5.

Taulukko 4. CGI-komentosarjan ja leddmn-prosessin välillä soketissa välitettävät viestit.

Suunta	**viesti**	**selitys**
cgi-> daemon	Q	kysyy ledien statuksen
cgi->daemon	S12	Asettaa uuden tilan ledeille
daemon->cgi	A01	Vastaus queryyn, ledien tila
daemon->cgi	O	Vastaus asetukseen, ok

5.4 Leddmn-taustaprosessin toiminta

Leddmn-prosessi on c-kielellä kirjoitettu ja ARM-arkkitehtuurille käännetty ohjelma, joka toimii taustaprosessina Beagleboardilla. Prosessin lähdekoodi on esitetty liitteissä 3 ja 4.

Leddmn-prosessin tehtävät on lueteltu seuraavassa listassa.

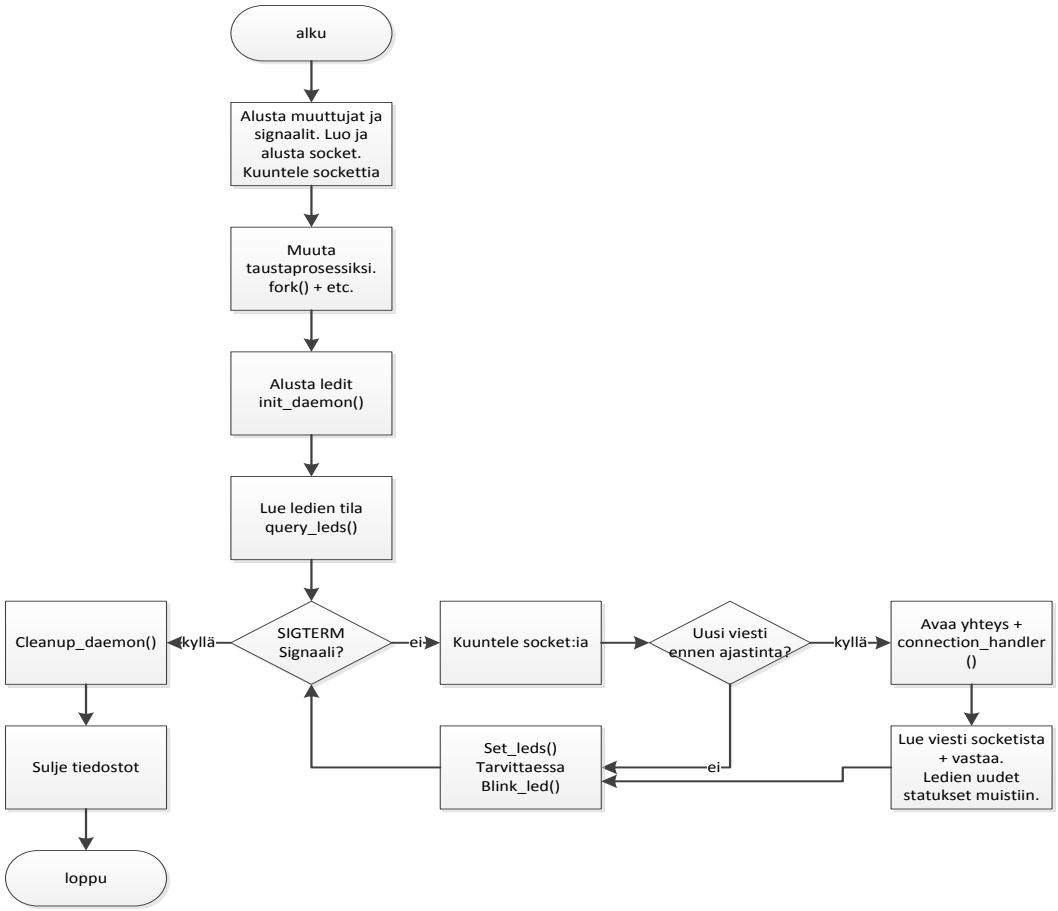
- Ottaa käyttöön tarvittavat osat sysfs-tiedostojärjestelmästä.
- Luo soketin CGI-komentosarjoille viestintää varten.
- Ottaa vastaan soketin kautta kyselyitä ledien tilasta ja vastata niihin.
- Ottaa vastaan soketin kautta uudet asetukset ledien tilaan, ja asettaa ledit näihin tiloihin sysfs-tiedostojärjestelmän kautta.
- Tarvittaessa vilkuttaa lediä sytyttämällä ja sammuttamalla säännöllisin väliajoin.

Yleisen tason vuokaavio taustaprosessin toiminnasta on esitetty kuvassa 8. Prosessi siis toimii taustaprosessina eli ilman kontrolloivaa terminaalia. Soketin luku tapahtuu select-systeemikutsun avulla. Timeout-arvoksi on asetettu yksi sekunti. Jos ennen ajastimen loppuunkulumista soketissa ei ole luettavissa olevaa tietoa, ohjelma jatkaa eteenpäin ja tarvittaessa vilkuttaa ledejä, eli vaihtaa tilan päällä->pois tai päinvastoin. Tässä vaiheessa myös ledien tilat kirjoitetaan sysfs-tiedostojärjestelmään, olivat ne muuttuneet tai eivät. Näiden toimien jälkeen ohjelma alustaa select-kutsun uudelleen ja palaa odottamaan soketista luettavaa tietoa, kunnes ajastin nollautuu tai tulee keskeytys.

Prosessi käsittelee omalla keskeytyskäsittelijällä SIGTERM-keskeytyksen. Tällä pyritään varmistamaan se, että ohjelma voidaan sulkea hallitusti ja ledien tilat palauttaa oletusarvoihinsa sysfs-tiedostojärjestelmään. Käyttäjä voi toki komentoriviltä lähettää prosessille minkä tahansa keskeytyksen, ja koska osalla eri keskeytyksistä on oletustoimintona ohjelman lopetus, ohjelman kyllä halutessaan saa sammumaan vähemmän hallitusti.

Prosessi kirjoittaa logitiedostoon /home/ubuntu/beagledaemon.log mahdolliset käynnistyksessä tai käytön aikana tapahtuneet virheet. Tämän hetkinen versio käsittelee yh-

den yhteyden kerrallaan. Jos ohjelman yhtäaikaisia käyttäjiä on useampia, muut joutuvat odottamaan.



Kuva 8. leddmn-taustaprosessin yleisen tason toiminta vuokaaviona.

6 Johtopäätökset

Beagleboardilla on monipuolisesti väyliä ja laajennusportteja, joiden signaaleja voi käyttää hyväksi omien laitteiden kytkemiseksi. GPIO:n käyttäminen on erityisen helppoa, koska signaalit on pääsääntöisesti tuotu tai voidaan tuoda sysfs-tiedostojärjestelmään käyttäjien ohjelmien käytettäväksi. Sysfs:ää käytettäessä ollaan kuitenkin niiden ominaisuuksien varassa, jotka käyttöjärjestelmän ytimen tai laiteajurin kehittäjä on päättänyt siihen toteuttaa. Vastaavan toiminnallisuuden ottaminen käyttöön suoraan muistirekistereihin kirjoittamalla taas on työläämpi ja virhealttiimpi vaihtoehto.

Linuxin kehitys varsinkin ARM-arkkitehtuurin osalta on tällä hetkellä niin nopeaa, että melkein aina kannattaisi ottaa ainakin kokeiluun uusin löytyvä versio ytimestä. Beagleboardilla Ubuntun mukana tullut ytimen versio helpotti kehitystyötä, koska esimerkiksi multipleksointi ja debugfs-tiedostojärjestelmä oli otettu valmiiksi käyttöön. Valmiin käyttöjärjestelmän lataamista sulautettuun kannattaa ainakin harkita koska saatavilla on huomattava määrä valmiiksi käännettyjä ohjelmia, joita voi omaan projektiin testata. Toisaalta paremman kontrollin käytettyihin optimointeihin ja eri ohjelmien ja kirjastojen versioihin saa, jos kääntää ja valitsee kaiken itse alusta lähtien, esimerkiksi buildrootin avulla.

Ristiinkääntäminen ja eri kirjastojen ja ohjelmaversioiden kanssa toimiminen on ohjelmakohtaisesti aina hieman käsityötä vaativa prosessi, koska eri ohjelmat eivät välttämättä noudata kaikkia yleisiä standardeja tai käytäntöjä. Joissain tapauksissa ei kaikista säädöistä huolimatta saada jotain toiminnallisuutta toimimaan johtuen ohjelmointivirheistä tai muista yllättävistä yhteensopivuusongelmista. Esimerkiksi projektin aikana WLAN-adhoc -verkon määrittelyssä tuli ongelma siinä, että kunnollista salausta ei saatu toimimaan eri laiteajureiden ongelmista johtuen. Toisaalta tukiasemana toimiminen ei aivan suoraan onnistunut luotettavasti joidenkin käynnistyskomentosarjojen käynnistysjärjestykseen liittyvien ongelmien vuoksi.

Demo-ohjelman toteutukseen jäi jotain ongelmia. Beagleboardin laitteiston tietoja voi muuttaa käyttämättä tätä kehitettyä ohjelmaa esimerkiksi kirjoittamalla suoraan komentoriviltä sopiviin sysfs-tiedostoihin. Toisaalta tätä ei oikein voi mitenkään myöskään estää. Tuotantokäytössä olevassa laitteessa pitäisi siis sopia käytännöt, joilla yritetään pitää huoli siitä, että kyseisiä arvoja muutetaan vain yhden ohjelman kautta. CGI-komentosarjan ja WWW:n toiminnan tilattomasta luonteesta johtuen käyttäjän selaimelle ei päivitetä ledien tilatietoa, jos ne muuttuvat jotain muuta kautta kuin käyttäjän itse CGI-komentosarjalla muuttamalla. Esimerkkitapaus voisi olla vaikka leddmn-taustaprosessin uudelleenkäynnistys, jolloin prosessi ei käynnistyessään lue ledeiltä muuta kuin sen, onko ledi palamassa vai sammuksissa. Ledien vilkutus on täysin ohjelmallinen toiminto, joten sitä tilaa ei voi ledeille tallettaa.

Lähteet

1 Hallinan, Christopher. 2010. Embedded Linux primer, second edition. Prentice Hall.

2 rfc3875 CGI-standardi. 2012. Verkkodokumentti. Internet engineering task force. <http://tools.ietf.org/html/rfc3875>. Luettu 17.11.2012.

3 Embedded system. 2012. Verkkodokumentti. wikipedia.org. <http://en.wikipedia.org/wiki/Embedded_system>. Luettu 18.11.2012.

4 HTTP-standardi. 2012. Verkkodokumentti. wikipedia.org. <http://en.wikipedia.org/wiki/Http>. Luettu 17.11.2012.

5 Beagleboard system reference manual, revision C4. 2012. Verkkodokumentti. Beagleboard.org. <http://beagleboard.org/static/BBSRM_latest.pdf>. Luettu 15.10.2012

6 ARM annual report 2011-1. 2012. Verkkodokumentti. Arm holdings plc. <http://www.arm.com/>. Luettu 10.10.2012.

7 OMAP 35x Applications Processor Technical Reference manual. 2012. Verkkodokumentti. Texas Instruments. <http://www.ti.com/lit/ug/spruf98x/spruf98x.pdf>. Luettu 10.10.2012.

8 Ubuntu wiki, ARM/OMAP sivu. 2012. Verkkodokumentti. ubuntu.org. <Https://wiki.ubuntu.com/ARM/OMAP> Luettu 2.11.2012.

9 Debian wiki. 2012. Verkkodokumentti. debian.org. <http://wiki.debian.org/ArmHardFloatPort> . Luettu 8.11.2012.

10 Lighttpd wiki. 2012. Verkkodokumentti. lighttpd.net. <http://redmine.lighttpd.net/projects/1/wiki/scriptsubuntu> . Luettu 11.11.2012.

11 Linux ytimen dokumentti GPIO:sta. 2012. Verkkodokumentti. kernel.org. <http://www.kernel.org/doc/Documentation/gpio.txt>. Luettu 5.9.2012.

12 Elinux.org sivu Beagleboardin multipleksauksesta. 2012. Verkkodokumentti. elinux.org. <http://elinux.org/BeagleBoardPinMux>. Luettu 14.10.2012.

Lighttpd:n kääntämisessä tarvittava komentosarja

```
#!/bin/bash
export PATH=/home/harri/b_root/bin:$PATH
export CC=arm-linux-gcc
export RANLIB=arm-linux-ranlib
export STRIP=arm-linux-strip
export AR=arm-linux-ar
export LD=arm-linux-ld
export CPPFLAGS=-I/home/harri/b_root/include
export LDFLAGS="-L/usr/local/xtools/arm-cortex_a8-linux-gnueabi/arm-
cortex_a8-linux-gnueabi/sysroot/usr/lib -L/home/harri/b_root/lib"
./configure --host=arm-linux \
--prefix=/opt \
--disable-static \
--enable-shared \
--with-zlib \
--without-bzip2 \
--with-pcre \
--with-openssl
```

Lighttpd käynnistyskomentosarja

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:          lighttpd
# Required-Start:    $syslog $remote_fs $network
# Required-Stop:     $syslog $remote_fs $network
# Should-Start:      fam
# Should-Stop:       fam
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start the lighttpd web server.
### END INIT INFO

USER=www-user
GROUP=www-user
PATH=/opt/sbin:/sbin:/bin:/usr/sbin:/usr/bin
LIGHTY_DAEMON=/opt/sbin/lighttpd
LIGHTY_OPTS="-f /etc/lighttpd/lighttpd.conf"
LIGHTY_NAME=lighttpd
LIGHTY_PIDFILE=/var/run/$LIGHTY_NAME.pid
SCRIPTNAME=/etc/init.d/omalighttpd
SSD="/sbin/start-stop-daemon"
RETVAL=0

test -x $LIGHTY_DAEMON || exit 0

set -e

. /lib/lsb/init-functions

case "$1" in
  start)
        log_daemon_msg "Starting $LIGHTY_NAME"
        if ! $SSD --quiet --start --pidfile $LIGHTY_PIDFILE --exec
$LIGHTY_DAEMON -- $LIGHTY_OPTS 2> /dev/null; then
            log_end_msg 1
        else
            log_end_msg 0
        fi
        RETVAL=$?
  ;;
```

```
  stop)
        log_daemon_msg "Stopping $LIGHTY_NAME"
        if $SSD --quiet --stop --oknodo --retry=0/30/KILL/5 --exec
$LIGHTY_DAEMON; then
            rm -f $LIGHTY_PIDFILE
            log_end_msg 0
        else
            log_end_msg 1
        fi
        RETVAL=$?
  ;;
reload)
        log_daemon_msg "Reloading $LIGHTY_NAME configuration"
        if $SSD --stop --signal 2 --oknodo --quiet --pidfile $LIGHTY_PIDFILE
--exec $LIGHTY_DAEMON; then
            if $SSD --start --quiet --pidfile $LIGHTY_PIDFILE --exec
$LIGHTY_DAEMON -- $LIGHTY_OPTS ; then
                log_end_msg 0
            else
                log_end_msg 1
            fi
        else
            log_end_msg 1
        fi
        RETVAL=$?
  ;;
  restart|force-reload)
        $0 stop
        [ -r  $LIGHTY_PIDFILE ] && while pidof lighttpd |\
                 grep -q `cat $LIGHTY_PIDFILE 2>/dev/null` 2>/dev/null ; do
sleep 1; done
        $0 start
  ;;
  *)
        echo "Usage: $SCRIPTNAME {start|stop|restart|reload|force-reload}"
>&2
        exit 1
  ;;
esac

exit $RETVAL
```

leddmn.c lähdekoodi

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include <signal.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <linux/un.h>
#include <errno.h>
#include "leddmn.h"

/*
 *Käsitellään itse signaali SIGTERM jolla
 *normaalisti lopetetaan ohjelman suoritus KILL komennolla
 *Linuxissa. Koitetaan varmistaa se että ledit voidaan
 *palauttaa oletustilaansa ohjelman sammuessa.
*/
static void signal_handler(int signal)
{
    switch(signal)
    {
    case SIGTERM:
           gotterm = 1;
           break;
    }
}

FILE *log_fp = NULL; //Logitiedoston kirjoittamiseen

int main(void)
{
    FILE *leds[MAXLEDS];      //Ledien brightness-tiedostot. Sytyttää tai
sammuttaa ledin
    FILE *led0_fp = NULL;
    FILE *led1_fp = NULL;

    enum ledstatus led0_stat, led1_stat;      //pidetään muistissa ledien
status ohjelman sisällä
```

```
pid_t pid = 0;
pid_t sid = 0;

//soketin avaukseen muuttujat
struct sockaddr_un addr;
int sockfd, connfd;
socklen_t addrln;

//selectille muuttujat
struct timeval tv;
fd_set read_fds, orig_fds;
int numfds, res;

signal(SIGTERM, signal_handler);

leds[0] = led0_fp;
leds[1] = led1_fp;

FD_ZERO(&read_fds);
FD_ZERO(&orig_fds);
tv.tv_sec = TIMEOUTSEC;
tv.tv_usec = 0;

log_fp = fopen (LOGFILE, "w+");
fprintf(log_fp, "Ohjelman käynnistys\n");
fflush(log_fp);

/*
 *CGI-skripti toimii www-data käyttäjän oikeuksilla ja tämä prosessi
 *root-oikeuksilla, joten soketille pitää antaa riittävät
 *luku+kirjoitus oikeudet. Luodaan soket.
*/
umask(0);
if((sockfd = socket(PF_UNIX, SOCK_STREAM, 0)) < 0) {
      fprintf(log_fp, "Soketin luonti epäonnistui, lopetetaan\n");
      fflush(log_fp);
      exit(-1);
}
unlink(SOCKETNAME);
memset(&addr, 0, sizeof(struct sockaddr_un));

addr.sun_family = AF_UNIX;
snprintf(addr.sun_path, UNIX_PATH_MAX, SOCKETNAME);
```

```
    if(bind(sockfd, (struct sockaddr *) &addr, sizeof(struct sockaddr_un))
!= 0) {
          fprintf(log_fp, "bind() epäonnistui, lopetetaan\n");
          fflush(log_fp);
          exit(-1);
    }

    if(listen(sockfd, 5) != 0) {
          fprintf(log_fp, "listen() epäonnistui, lopetetaan\n");
          fflush(log_fp);
          exit(-1);
    }

    /*
     *Lisätään soket selektiä varten fileset:iin.
     *Kopioidaan talteen orig_fds uutta select-kutsua varten koska
     *select muuttaa fileset:iä kutsun aikana.
    */
    FD_SET(sockfd, &read_fds);
    orig_fds = read_fds;
    numfds = sockfd + 1;

    /*
     *Taustaprosessin luonti alkaa tästä. Forkataan,
     *suljetaan turhat äitiprosessit, laitetaan lapsiprosessi
     *sessionleader asemaan (ei terminaalia, taustalle) ja suljetaan
     *turhat luku+kirjoitus tiedosto-osoittimet.
    */
    pid = fork();

    if (pid < 0) {
          fprintf(log_fp, "fork epäonnistui, lopetetaan\n");
          fflush(log_fp);
          exit(-1);
    }
    if (pid > 0) {
          printf("Lapsiprosessin id: %d \n", pid);
          exit(0);
    } //pid == 0

    sid = setsid();
    if(sid < 0) {
          exit(1);
```

```
    }

    close(STDIN_FILENO);
    close(STDOUT_FILENO);
    close(STDERR_FILENO);

    //alustetaan ledien käsittely. Aliohjelma avaa sysfs:n kautta tarpeelli-
set tiedostot
    if(init_daemon(leds) == -1) {
          fprintf(log_fp, "init_daemon() epäonnistui, lopetetaan\n");
          fflush(log_fp);
          exit(-1);
    }

    //luetaan ledien tämänhetkinen tila ohjelman muistiin
    if(query_leds(leds, &led0_stat, &led1_stat) == -1){
          fprintf(log_fp, "query_leds() epäonnistui, lopetetaan\n");
          fflush(log_fp);
          exit(-1);
    }

    /*
     *Pääsilmukka. Jatketaan kunnes tulee joku virhe tai keskeytys.
     *gotterm asetetaan keskeytyskäsittelijässä arvoon 1 (true)
     *keskeytyksen tullessa
    */
    while (!gotterm) {
          read_fds = orig_fds;
          if ((res = select(numfds, &read_fds, NULL, NULL, &tv)) == -1) {
                //keskeytys keskeytti selectin, ei data soketista tai timer
                if (errno == EINTR) {
                      break;
                }
                //joku muu virhe selectissä
                fprintf(log_fp, "Select epäonnistui, lopetetaan");
                fflush(log_fp);
                return -1;
          } else if (res > 0) {
                //jotain dataa soketissa luettavissa
                if ((connfd = accept(sockfd, (struct sockaddr *) &addr,
&addrln)) == -1) {
                      fprintf(log_fp, "accept() epäonnistui, lopetetaan");
                      fflush(log_fp);
                      return -1;
```

```
                }
                connection_handler(connfd, leds, &led0_stat, &led1_stat);
          } //connection_handler lopetti tai selektin timeout, eli ei dataa
soketissa

          /*
           *vilkutellaan ledejä tarvittaessa. Ledien tila myös kirjoitetaan
sysfs:n
           *tiedostoihin joka kierroksella uudestaan
          */
          if (led0_stat == BLINK) blink_led(leds, 0);
          if (led1_stat == BLINK) blink_led(leds, 1);
          set_leds(leds, led0_stat, led1_stat);
          tv.tv_sec = TIMEOUTSEC;
          tv.tv_usec = 0;
    }
    fprintf(log_fp, "SIGTERM, suljetaan ohjelma\n");
    fflush(log_fp);
    close(sockfd);
    unlink(SOCKETNAME);
    if(cleanup_daemon(leds) == -1) {
          fprintf(log_fp, "cleanup_daemon() epäonnistui, lopetetaan\n");
          fflush(log_fp);
          exit(-1);
    }
    fclose(log_fp);
    return (0);
}

/*
 *Tämä aliohjelma alustaa sysfs-tiedostojen kautta ledit ohjelman käyttöön.
 *trigger-tiedostoon kirjoittamalla otetaan kontrolli ledien sytyttelystä ja
 *brightness-tiedostoon kirjoittamalla sytytetään tai sammutetaan ledi.
 *Molemmille ledeille on omat tiedostonsa.
 *Onnistuneessa tapauksessa leds-taulukossa on brightness-tiedostojen osoit-
timet,
 *virheen sattuessa palautetaan -1, muutoin 0.
*/
int init_daemon(FILE **leds)
{
    FILE *trig_fp = NULL;

    trig_fp = fopen(LED0TRIGGER, "w+");
    if(trig_fp == NULL)
```

```
		return -1;
	fprintf(trig_fp, DISABLETRIG);
	fclose(trig_fp);

	trig_fp = fopen(LED1TRIGGER, "w+");
	if(trig_fp == NULL)
		return -1;
	fprintf(trig_fp, DISABLETRIG);
	fclose(trig_fp);

	leds[0] = fopen(LED0BRIGHT, "w+");
	if(leds[0] == NULL)
		return -1;
	leds[1] = fopen(LED1BRIGHT, "w+");
	if(leds[1] == NULL)
		return -1;
	return 0;
}

/*
 *Tämä aliohjelma palauttaa sysfs-tiedostot oletusarvoihinsa.
 *trigger-tiedostojen avulla palautetaan ledien
 *kontrolli käyttöjärjestelmälle
 *lisäksi suljetaan tiedosto-osoittimet.
 *virheen sattuessa palautetaan -1
*/
int cleanup_daemon(FILE **leds)
{
	FILE *trig_fp = NULL;

	set_leds(leds, OFF, OFF);
	trig_fp = fopen(LED0TRIGGER, "w+");
	if(trig_fp == NULL)
		return -1;
	fprintf(trig_fp, LED0DEFAULT);
	fclose(trig_fp);

	trig_fp = fopen(LED1TRIGGER, "w+");
	if(trig_fp == NULL)
		return -1;
	fprintf(trig_fp, LED1DEFAULT);
	fclose(trig_fp);
	fclose(leds[0]);
	fclose(leds[1]);
```

```
    return 0;
}

/*
 *Asetetaan ledeille sysfs:n kautta statuksen mukaiset arvot.
 *Jos ledin status on BLINK, ei tehdä mitään koska lediä vilkutetaan erik-
seen
 *toisessa aliohjelmassa. Virheen sattuessa palauttaa -1, muuten 0.
*/
int set_leds(FILE **leds, enum ledstatus led0, enum ledstatus led1)
{
    if(led0 == ON) {
          if(fprintf(leds[0], "1") == -1)
                  return -1;
    }
    else if(led0 == OFF) {
          if(fprintf(leds[0], "0") == -1)
                return -1;
    }
    fflush(leds[0]);

    if(led1 == ON) {
          if(fprintf(leds[1], "1") == -1)
                return -1;
    }
    else if(led1 == OFF) {
          if(fprintf(leds[1], "0") == -1)
                return -1;
    }
    fflush(leds[1]);
    return 0;
}

/*
 *Luetaan ledien tila sysfs tiedostoista. Tiedostossa arvo
 *0 = OFF ja >0 tarkoittaa ON.
 *Tiedostossa numero on kuitenkin tekstinä.
 *Virheen sattuessa palauttaa -1, muutoin 0.
*/
int query_leds(FILE **leds, enum ledstatus *led0, enum ledstatus *led1)
{
    int n;
    rewind(leds[0]);
    if(fscanf(leds[0], "%d", &n) == -1)
```

```
            return -1;
    if(n == 0)
            *led0 = OFF;
    else
            *led0 = ON;

    rewind(leds[1]);
    if(fscanf(leds[1], "%d", &n) == -1)
            return -1;
    if(n == 0)
            *led1 = OFF;
    else
            *led1 = ON;
    return 0;
}

/*
 *Välkytetään yhtä lediä, eli vaihdetaan tila 0->1 tai päinvastoin.
 *Virheen sattuessa palauttaa -1, muutoin 0.
*/
int blink_led(FILE **leds, int led)
{
    int n, newstat;
    rewind(leds[led]);
    if(fscanf(leds[led], "%d", &n) == -1)
            return -1;
    if(n == 0)
            newstat = 1;
    else
            newstat = 0;
    if(fprintf(leds[led], "%d", newstat) == -1)
            return -1;
    fflush(leds[led]);
    return 0;
}

/*
 *Tämä aliohjelma lukee ja kirjoittaa soketin kautta tulleet viestit.
 *Vastaukset kirjoitetaan ledien tilan perusteella.
 *jos viesti on asetusviesti niin ledien tila muutetaan sitä vastaavaksi.
 *Mahdolliset viestit soketista:
 *Q = query eli paluuarvo sokettiin A + ledien arvot numeroina
 *S = set eli uudet arvot ledeille. paluuarvoksi kirjoitetaan O.
 *Virheen sattuessa aliohjelma palauttaa -1, muutoin 0.
```

```
*/
int connection_handler(int connfd, FILE **leds, enum ledstatus *led0, enum
ledstatus *led1)
{

    int nbytes;
    char buffer[MAXBUF], ch;

    if ((nbytes = read(connfd, buffer, MAXBUF)) == -1) {
          return -1;
    }
    buffer[nbytes] = '\0';

    switch (buffer[0])
    {
    case 'Q':
          nbytes = snprintf(buffer, MAXBUF, "A%d%d", *led0, *led1);
          if (write(connfd, buffer, nbytes) == -1)
                return -1;
          break;
    case 'S':
          ch = buffer[1];
          if(isdigit(ch))
                *led0 = atoi(&ch);
          else
                return -1;
          ch = buffer[2];
          if (isdigit(ch))
                *led1 = atoi(&ch);
          else
                return -1;
          nbytes = snprintf(buffer, MAXBUF, "O");
          if (write(connfd, buffer, nbytes) == -1)
                return -1;
          break;
    }
    return 0;
}
```

leddmn.h lähdekoodi

```
#ifndef LEDDMN_H
#define LEDDMN_H

#define LED0TRIGGER "/sys/class/leds/beagleboard::usr0/trigger"
#define LED1TRIGGER "/sys/class/leds/beagleboard::usr1/trigger"
#define LED0BRIGHT "/sys/class/leds/beagleboard::usr0/brightness"
#define LED1BRIGHT "/sys/class/leds/beagleboard::usr1/brightness"
#define LOGFILE "/home/ubuntu/beaglecontrol.log"
#define DISABLETRIG "none"
#define LED0DEFAULT "heartbeat"
#define LED1DEFAULT "mmc0"
#define MAXLEDS 2
#define MAXBUF 500
#define TIMEOUTSEC 1
#define SOCKETNAME "/home/ubuntu/temp_socket"

enum ledstatus { OFF, ON, BLINK }; //0, 1, 2

int init_daemon(FILE **);
int cleanup_daemon(FILE **);
int query_leds(FILE **, enum ledstatus *, enum ledstatus *);
int set_leds(FILE **, enum ledstatus , enum ledstatus);
int blink_led(FILE **, int);
int connection_handler(int, FILE **, enum ledstatus *, enum ledstatus *);

volatile sig_atomic_t gotterm = 0;

#endif
```

beaglecontrol.py CGI-komentosarja

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-

import socket
import cgi
import cgitb; cgitb.enable()
import os,  os.path

#Tulostaa yleisen virheilmoituksen html-sivuna

def error_page():
    print "Content-type: text/html"
    print

    print """
    <html>
    <head>
    <meta content="text/html; charset=utf-8"
    http-equiv="content-type">
    <title>Error page</title>
    </head>
    <body>
    <h2>Beagleboard ohjausdemo</h2>
    <br>
    Virhe ohjelman suorituksessa<br>
    Yritä myöhemmin uudelleen<br>
    </body>
    </html>
    """

#Tulostaa pääsivun. Parametreina led0 ja led1 joilla voi olla
#tekstiarvot "off", "on" tai "blink"

def print_mainpage(led0,  led1):
    print "Content-type: text/html"
    print

    print """
    <html>
    <head>
    <meta content="text/html; charset=utf-8"
```

```
    http-equiv="content-type">
    <title>Beaglecontrol</title>
    </head>
    <body>
    <h2>Beagleboard ohjausdemo</h2>
    <br>
    <form method="post" action="beaglecontrol.py" name="beagle_ohjaus"><br>
    """

#field1-3 tulostuvat weblomakkeeseen. Niillä valitaan ledikohtaisesti
#oikea radionappi ledin statuksen perusteella

    field1 = ""
    field2 = ""
    field3 = ""

    if led0 == "off":
        field2 = 'checked="checked"'
    elif led0 == "on":
        field1 = 'checked="checked"'
    else:
        field3 = 'checked="checked"'

    print """
    Led 0 ohjaus: <input name="led0" value="on" %s type="radio">On<input
    name="led0" value="off" %s type="radio">Off<input
    name="led0" value="blink" %s type="radio">Blink    Led 0
    status: %s<br>
    """ % (field1, field2, field3, cgi.escape(led0))

    field1 = ""
    field2 = ""
    field3 = ""

    if led1 == "off":
        field2 = 'checked="checked"'
    elif led1 == "on":
        field1 = 'checked="checked"'
    else:
        field3 = 'checked="checked"'

    print """
    Led 1 ohjaus: <input name="led1" value="on" %s type="radio">On<input
    name="led1" value="off" %s type="radio">Off<input
```

```
    name="led1" value="blink" %s type="radio">Blink    Led 1
    status: %s<br>
    """ % (field1, field2, field3, cgi.escape(led1))

    print """
    <br>
    <input value="Update" type="submit"><br>
    </form>
    <br>
    <br>
    </body>
    </html>
    """

form = cgi.FieldStorage()
err = 'no'  # käytetään muuttujaa err yksinkertaiseen virheseurantaan

#Ensimmäisellä käynnistyksellä ei ole form dataa,
#eli tehdään ledien tilasta kysely soketin kautta

if not form:
    if os.path.exists( "/home/ubuntu/temp_socket" ):
        try:
            client = socket.socket( socket.AF_UNIX, socket.SOCK_STREAM )
            client.connect( "/home/ubuntu/temp_socket" )

            client.send( 'Q' )
            repl = client.recv( 1024 )
        except socket.error :
            err = "yes"

        #jotain dataa soketista ja ei virhettä soketissa
        if (repl != "") and (err == "no"):
            if repl[0] == 'A':
                if repl[1] == '0':
                    led0 = 'off'
                elif repl[1] == '1':
                    led0 = 'on'
                elif repl[1] == '2':
                    led0 = 'blink'
                else:
                    # led0 ei sallittu arvo
                    err = 'yes'
                if repl[2] == '0':
```

```
                    led1 = 'off'
                elif repl[2] == '1':
                    led1 = 'on'
                elif repl[2] == '2':
                    led1 = 'blink'
                else:
                    # led1 ei sallittu arvo
                    err = 'yes'
        else: #ei dataa soketista tai virhe soketissa
            err = 'yes'
        client.close()
    else: # soket tiedosto puuttuu
        err = "yes"
    if err == 'yes':
        error_page()
    else:
        #   kaikki ok eli tulostetaan varsinainen pääsivu
        print_mainpage(led0,  led1)

#web-lomakkeella dataa, luetaan muuttujiin ja tulkitaan
#soket-viestiä varten
#st0 ja st1  sisältävät numerona ledin statustiedon, Sallitut arvot:
#off = 0, on = 1 ja blink = 2
else:
    led0 = form.getvalue("led0", "off")
    led1 = form.getvalue("led1", "off")
    if led0 == 'off':
        st0 = '0'
    elif led0 == 'on':
        st0 = '1'
    elif led0 == 'blink':
        st0 = '2'
    else: # mahdoton arvo muuttujassa led0
        err = 'yes'

    if led1 == 'off':
        st1 = '0'
    elif led1 == 'on':
        st1 = '1'
    elif led1 == 'blink':
        st1 = '2'
    else: # mahdoton arvo muuttujassa led1
        err = 'yes'
```

```
if os.path.exists( "/home/ubuntu/temp_socket" ) and (err == "no"):
    try:
        client = socket.socket( socket.AF_UNIX, socket.SOCK_STREAM )
        client.connect( "/home/ubuntu/temp_socket" )
        buff = 'S' + st0 + st1
        client.send( buff )
        repl = client.recv( 1024 )
    except socket.error:
        err = "yes"
    if (repl == ''):
        err = 'yes'
    client.close()
else:
    #Soket tiedostoa ei löydy
    err = 'yes'

if err == 'yes':
    error_page()
else:
    #kaikki ok, pääsivun tulostus
    print_mainpage(led0,  led1)
```