Opinnäytetyö AMK

Tieto- ja viestintätekniikka

2019

Rauno Vesti

YOCTO-PROJEKTI JA OHJELMISTOKEHITYS SULAUTETUILLE JÄRJESTELMILLE

– Sovellusohjelma Raspberry Pi 3 -alustalle

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2019 | 40 sivua

Rauno Vesti

YOCTO-PROJEKTI JA OHJELMISTOKEHITYS SULAUTETUILLE JÄRJESTELMILLE

- Sovellusohjelma Raspberry Pi 3 -alustalle

Sulautettujen järjestelmien kehitys on tänä päivänä voimakkaasti kasvava tietotekniikan ala, johon erityisesti Internet of Things (IoT, esineiden Internet) avaa valtavat kehitysnäkymät. Opinnäytetyö liittyy keskeisesti juuri tähän tietotekniikan haaraan, ja aiheena on Yocto SDK (Software Development Kit) -sovelluskehitysympäristön asennus ja käyttö. Ohjelmointi ei varsinaisesti kuulunut opinnäytetyön piiriin, vaan ainoastaan hyvin pieni esimerkkiohjelma luotiin ja testattiin Raspberry Pi 3 -minitietokoneella. Työssä lähdettiin liikkeelle Yocto SDK:n asennustiedoston luomisesta Yocto-projektin työvälineillä, mutta itse Yocto-projektin asennus oletettiin jo ennalta suoritetuksi. Yocto-projekti ja Yocto SDK:n uusin versio ovat toiminnoiltaan keskenään lähes vastaavia, joskin Yocto SDK on enemmän suunniteltu nimenomaan sovellusohjelmien kehitykseen ja Yocto-projekti käyttöjärjestelmien luomiseen. Uusimmat versiot näistä järjestelmistä kykenevät molempiin tehtäviin, eikä ero näiden kahden välillä ole suuri.

Opinnäytetyössä Yocto SDK asennettiin Linux Ubuntu 18.04 -järjestelmään Yocto-projektilla luodun asennustiedoston avulla. Yocto SDK:n asentamisen jälkeen luotiin sen avulla ensin käyttöjärjestelmäydin ja tiedostojärjestelmä testiympäristöä ja kohdelaitteistoa varten, joita tässä tapauksessa olivat Quick Emulator (Qemu) ja Raspberry Pi 3 B+. Qemu on virtuaalinen, ohjelmallisesti toteutettu fyysistä laitetta jäljittelevä (emuloiva) sovellus, jota käytettiin ohjelmakoodin testaukseen.

Sovellusohjelman kehitykseen käytettiin erillistä Eclipse IDE -ohjelmointityökalua, joka integroitiin Yocto SDK -kehitysympäristöön Yocto-lisäosan (engl. plug-in) avulla. Ohjelmointivaiheessa koodi käännettiin ja ladattiin Eclipse IDE:n debug-komentoa käyttäen suoraan Qemu-järjestelmään testattavaksi. Kun testausvaihe oli suoritettu onnistuneesti, luotiin Yocto SDK:n devtool add -komennolla resepti (engl. recipe) suoraan Eclipse IDE:n projektitiedostoista. Resepti tarkoittaa tässä yhteydessä järjestelmälle annettavaa ohjeistusta siitä, miten tietty ohjelmistopaketti kootaan. Resepti on vastaava, kuin mitä Yocto-projektissakin käytetään, ja sen avulla voidaan ohjelma kääntää ja asentaa laitteelle tai liittää se muodostettavaan uuteen käyttöjärjestelmään. Viimeisenä vaiheena luotiin bitbake-ohjelmalla Yocto-kerros (layer), jonne devtool finish -komennolla lisättiin uuden ohjelman resepti lopullista varastointia ja myöhempää käyttöä varten eri projekteihin liitettäväksi.

ASIASANAT:

sovelluskehitys, sulautettu, järjestelmä, kehitystyökalu, ohjelmointi

BACHELOR´S | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree programme in Information and Communications Technology

2019 | 40 pages

Rauno Vesti

YOCTO PROJECT AND SOFTWARE DEVELOPMENT FOR EMBEDDED SYSTEMS

- Application for Raspberry Pi 3

The development of embedded systems today is a rapidly growing area of information technology and, in particular, the Internet of Things (IoT) opens up huge prospects for it. The thesis is mainly related to this branch of information technology and the objectives of the thesis were to install and use a Yocto SDK (Software Development Kit) environment. Programming was not within the scope of this thesis. Only a very small sample program was created and tested for the Raspberry Pi 3 minicomputer.

The work started with creating a Yocto SDK installation file with the tools of the Yocto project. It is assumed that Yocto project itself has been pre-installed. The Yocto project and the latest version of Yocto SDK are almost the same in terms of functionality, although the Yocto SDK is more specifically designed for software development and Yocto project for creating operating systems. Nevertheless the latest versions of both are capable of performing these tasks and the difference between them is not notable.

In the thesis, the Yocto SDK was installed on Linux Ubuntu 18.04 using the installation file created by the Yocto project. Once installed, the Yocto SDK was first used to create the operating system kernel and the file system for the test environment and the target hardware, in this case Quick Emulator (Qemu) and Raspberry Pi. Qemu is a virtual device, an application that emulates a physical device, used for testing the program.

Eclipse IDE is a programming tool used in this thesis for software development. It is separate from the Yocto SDK but a plug-in is available for integrating it to the Yocto SDK development environment. In the coding and testing phase the software was compiled and loaded directly into Qemu using the Eclipse IDE debug feature.

Once the testing was successful, a recipe was created directly from the Eclipse IDE project files with the Yocto SDK devtool add -command. Recipes are instructions for the system of how to proceed in creating a certain software package. The Yocto SDK recipe is corresponding to the Yocto project recipe, and it can be used for compiling and installing the program directly on the device or adding it to the operating system when creating it. In this project the program was installed on Raspberry Pi device by the devtool deploy-target –command as a final step to ensure that it works on a physical device as well. As a final step the devtool recipe was transformed to a Yocto recipe with the devtool finish command, and was so exported for storing it in a repository, and to be subsequently used in various projects.

KEYWORDS:

application, development, embedded, system, programming

SISÄLTÖ

ALKULAUSE

Tämä opinnäytetyö on kirjoitettu suomen kielellä, vaikka englannin kieli olisi monilta osin ollut luontevampi vaihtoehto mm. terminologian osalta. Tietotekniikan alalla ja erityisesti sulautetuissa järjestelmissä käytetty terminologia on erityislaatuista ja osin vaikeaa kääntää suomeksi. Omalla äidinkielellä kirjoittaminen taas muuten on helpompaa ja sujuvampaa, ja siksi tässä opinnäytetyössä on päädytty kielen osalta juuri suomen kieleen. Lukijoille (kuten myös tämän kirjoittajalle) saattaa kuitenkin olla vaikea ymmärtää joidenkin ilmaisujen tarkkaa merkitystä suomen kielellä, jos ne ovat pääosin tulleet tutuiksi juuri englannin kielellä.

Edellä mainituista syistä johtuen tässä työssä on monia asioita pyritty selventämään sanallisesti paitsi suomeksi, myös laittamalla tärkeiden teknisten termien ja ilmaisujen perään sulkeisiin sama asia englannin kielellä. Suomenkieliset käännökset eivät useinkaan ole vielä täysin vakiintuneita tai ovat harvoin käytettyjä. Englanninkielisen termin kautta lukija voi varmistua käytetyn suomenkielisen vastineen merkityksestä kuten esim. ristikehitys (cross-development). Niinpä tässä opinnäytetyössä ei ole nähty tarpeelliseksi koota mitään erityistä lyhenneluetteloa tai erikoissanastoa, koska asiat ja ilmaisut selvennetään tekstin yhteydessä.

Monissa kohdin englanninkielinen termi on jonkin järjestelmän tai toiminnon nimi, jolloin sen suomentaminen ei edes ole tarpeen. Tällaisia ovat esimerkiksi juuri otsikossa esiintyvä nimi Yocto tai tämän työn varsinainen aihe ”Yocto Software Development Kit”, joka on hyvin vakiintunut nimi tai käsite juuri englanninkielisissä materiaaleissa ja siksi sen käyttö sellaisenaan kertoo tarkimmin mistä puhutaan. Lyhyyden vuoksi tässä työssä käytetään pääasiassa kuitenkin muotoa Yocto SDK. Tässä työssä käytetään myös suomennosta ohjelmistokehitysjärjestelmä tai -kehitysympäristö, vaikka sana ”kit” tarkoittaakin varsinaisesti ”varusteet” tai ”tarvikkeet”. Kyse on kuitenkin hyvin kattavasta kokonaisuudesta, joten sana järjestelmä tai ympäristö on perusteltu

Vastaavia nimiksi katsottavia termejä ovat OpenEmbedded ja Poky, mutta joidenkin kohdalla rajan vetäminen ei ole niin helppoa. Qemu (Quick Emulator) voitaisiin suomentaa sanalla ”pikajäljittelijä” tai ”pikaemulaattori”, mutta se ei tunnu luontevalta, ja siksi onkin katsottu paremmaksi käyttää termiä Qemu.

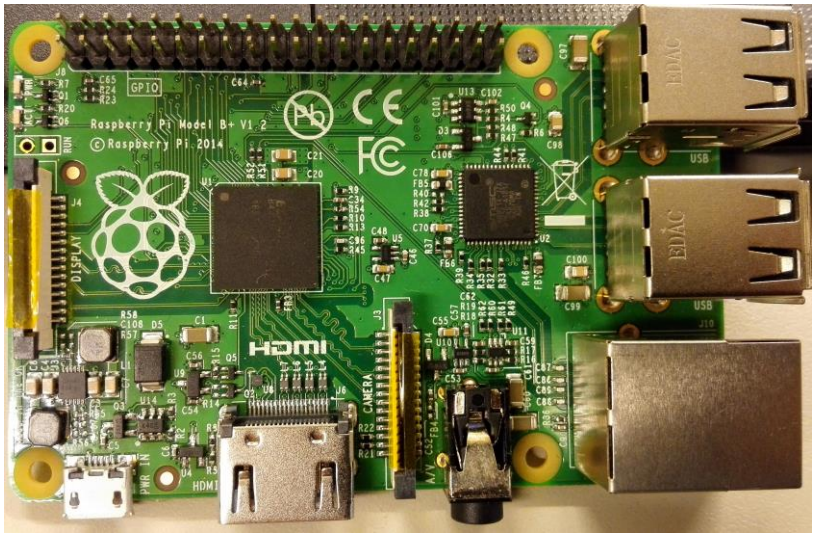
1 JOHDANTO

Tämä opinnäytetyö liittyy maailmalla valtavassa kasvuvaiheessa olevaan IoT (Internet of Things, esineiden Internet) -teknologiaan [1,2] ja sulautettujen järjestelmien kehitystyöhön. Työssä perehdytään sulautettuihin järjestelmiin asennettavien sovellusten kehitykseen ja asennukseen Yocto SDK (Yocto Software Development Kit) -kehitysympäristön avulla. Varsinainen ohjelmointi ei kuulu tämän työn tavoitteisiin, vaan ainoastaan itse kehitysympäristön asennus ja käyttöönotto. Esimerkin ja testauksen vuoksi toteutetaan kuitenkin hyvin yksinkertainen ohjelma. Tavoitteena on asentaa Yocto SDK sekä tähän integroitava erillinen ohjelmointityökalu Eclipse IDE (Eclipse Integrated Development Environment).

Yocto SDK liittyy olennaisesti laajaan sulautettujen järjestelmien kehitysympäristöön nimeltä Yocto Project, johon nimikin jo selvästi viittaa. Kyse onkin miltei samasta tai samankaltaisesta järjestelmästä, mutta nämä on silti syytä erottaa toisistaan. Tässä työssä viitataan usein Yocto-projektiin ja hieman siihen perehdytäänkin, sillä se on lähtökohta ja perusta koko opinnäytetyölle. Lisäksi käydään läpi Eclipse IDE -ohjelmointityökalun integrointi Yocto SDK -kehitysympäristöön, mikä osaltaan tuo lisäulottuvuuksia tämän järjestelmän käyttöön.

Sulautettujen järjestelmien kirjo on laaja, ja erilaisia laitteita ja laitteistoarkkitehtuureja on tarjolla moniin eri tarkoituksiin [3]. Sovellusten kohdejärjestelmä (engl. target system) on se, mihin ympäristöön kehitettävä sovellus lopulta asennetaan. Sulautetut järjestelmät ovat yleensä resursseiltaan tavallista PC-tietokonetta rajoitetumpia (työmuisti, prosessoriteho, pysyvä muisti eli ns. kovalevy) [1,4]. Nämä järjestelmät ovat usein myös suunniteltuja vain johonkin tiettyyn tarkoitukseen, eikä käyttäjä voi aina valita, miten järjestelmä toimii tai mitä ohjelmia siihen asennetaan. Myös monia muita rajoituksia voi liittyä näihin järjestelmiin [5], mutta tarkempi perehtyminen niihin ei kuulu tämän työn aihepiiriin.

Tässä työssä kohdejärjestelmänä tai -laitteena käytettiin kuvassa 1 esitettyä ja maailmalla valtaisan suosion saanutta Raspberry Pi -piiriä. Se voidaan oikeastaan määritellä jo pieneksi tietokoneeksi, sillä se on teholtaan ja ominaisuuksiltaan hyvin lähellä perinteisiä kotitietokoneita [6].



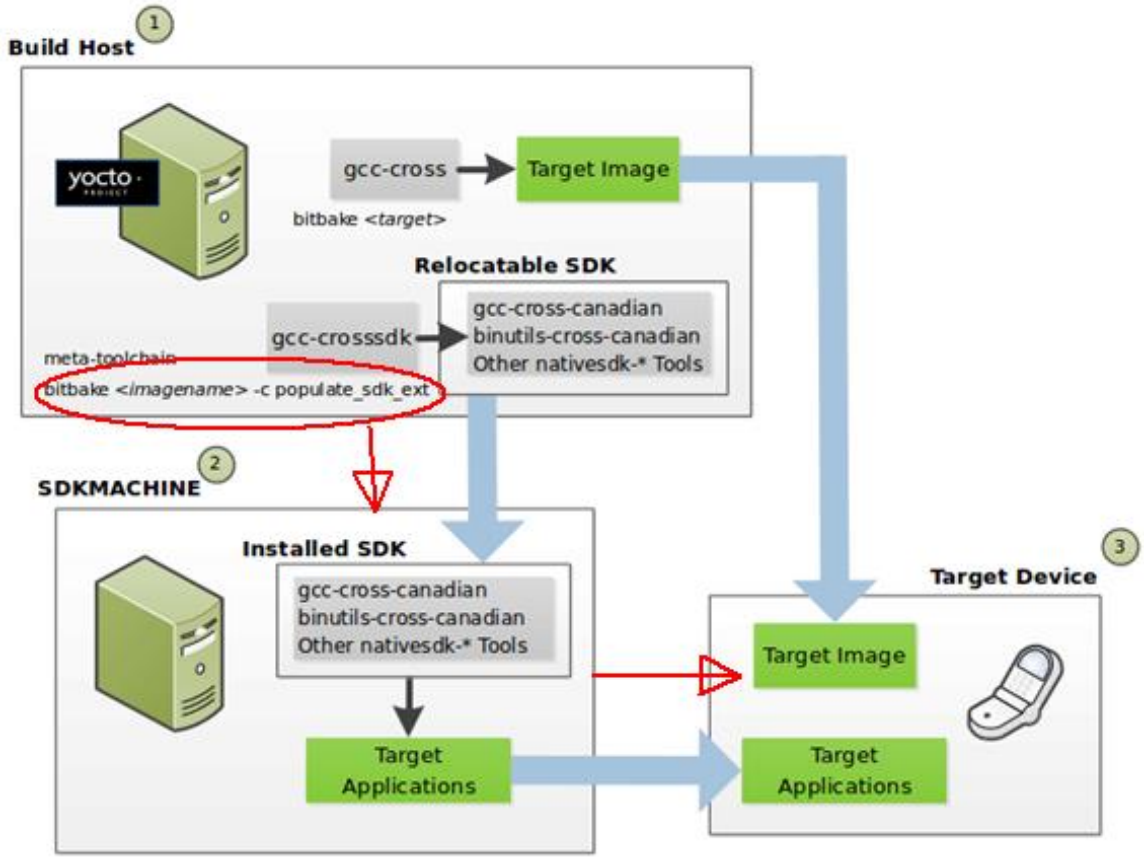
Kuva 1. Raspberry Pi -piiri (kuva R. Vesti).

Raspberry Pi on noin luottokortin kokoinen piiri [6], jossa prosessorin lisäksi on useita normaalista PC-tietokoneestakin löytyviä I/O portteja kuten USB, Ethernet, HDMI, WiFi ja Bluetooth [7]. Kyseinen laite on ensimmäisen, v. 2011 myydyn tuotantoerän jälkeen käynyt läpi muutamia kehitysvaiheita, ja uusin malli 3 B+ on tässä opinnäytetyössä käytetty versio. Raspberry Pi on monipuolisten ominaisuuksiensa vuoksi hyvin käyttökelpoinen monenlaisissa tehtävissä. Nykyään erityisesti IoT-järjestelmissä sitä käytetään mm. ohjaimena (engl. controller) sekä monissa muissa tehtävissä [8]. IoT on niin ikään erittäin voimakkaasti kasvava tietotekniikan ala, ja verkkoon liitettävien uusien laitteiden määrä kasvaa kiihtyvällä nopeudella [1]. Siksi myös sovelluskehitysjärjestelmä Yocto SDK on hyvin keskeisessä asemassa tällä kasvavalla alalla, ja sovellusten kehittäminen Raspberry Pi -piiriin laajentaa sen käyttömahdollisuuksia.

Sulautetuissa järjestelmissä on käyttöjärjestelmänä nykyään yhä useammin Linux eikä enää välttämättä laitevalmistajan itse kehittämä järjestelmä [4], kuten aiemmin oli yleistä. Raspberry Pi -piiriä myydään joko ilman mitään käyttöjärjestelmää tai siten, että mukana on esim. Raspbian-niminen versio Linux Debian -jakelusta [7]. Linux onkin ollut yksi merkittävä vaikuttaja myös Raspberry Pi -piirin suosion huimaan nousuun. Tämä johtuu paljolti Linuxin vapaasta saatavuudesta, käytöstä ja kehityksestä Gnu GPL -lisenssin alla sekä myös valtavasta maailmanlaajuisesta ohjelmistokehittäjien yhteisöstä, joka tukee Linuxin eri versioiden kehitystä.

Tässä työssä Linux-käyttöjärjestelmä Raspberry Pi -piirille luotiin samalla Yocto SDK -järjestelmällä, jolla muitakin sovelluksia kehitetään. Käytännössä Yocto SDK on omi-

naisuuksiltaan niin monipuolinen, ettei varsinaista Yocto-projektia enää välttämättä tarvita. Tämä eri järjestelmien kokonaisuus on hyvin laaja, eikä sitä voida tässä yksityiskohtaisesti käydä läpi. Lyhyesti sanottuna tämä opinnäytetyö lähtee siitä, että Yocto-projekti on valmiiksi asennettuna Linux Ubuntu 18.04 -käyttöjärjestelmään, ja siinä ympäristössä luodaan erillinen asennustiedosto (asennusskripti) Yocto SDK -kehitysympäristön asentamista varten. Tämä asennustiedosto siirretään ja suoritetaan toisessa tietokoneessa (tässä tapauksessa Oracle VirtualBox), jossa on sama käyttöjärjestelmä ja jolla sitten varsinaisesti työskennellään siitä eteenpäin. Yocto-projektia ei enää tämän jälkeen käytetä. Kuvan 2 kaaviossa on hahmoteltu pääpiirteittäin miten Yocto-projektin sisältävällä järjestelmällä luodaan SDK-kehitysympäristö.



Kuva 2. Kuvassa on hahmoteltu järjestelmän ”työnjako” eri yksiköiden välillä. Kohta 1 kuvaa järjestelmää, johon on asennettu Yocto-projekti, ja jossa luodaan asennustiedosto SDK -kehitysympäristöä (kohta 2) varten. Kohdassa 3 (Target Device) on kohdelaite, mihin kehitettävät sovellukset (Target Applications) lopulta asennetaan [9].

Yocto-projektilla luodaan ensin asennustiedosto komennolla bitbake <imagename> -c populate_sdk_ext, ja sitten kyseinen tiedosto siirretään ja asennetaan kokonaan eri laitteelle (SDKMACHINE). Tämän jälkeen SDK:n kehitystyökaluilla luodaan sovelluksia

(Target Applications) kohdelaitteelle (Target Device). Kuvassa 2 esitetään, miten varsinainen käyttöjärjestelmä (Target Image) kohdelaitteelle luodaan Yocto-projektilla. Tuo käyttöjärjestelmä voidaan kyllä luoda myös SDK:n työvälineillä ilman Yocto-projektia (punainen nuoli), ja näin tässä opinnäytetyössä meneteltiinkin. Tämä pätee tosin vain uudempaan Extensible SDK (eSDK) -versioon, jolla voidaan luoda sekä sovelluksia että koko käyttöjärjestelmä (ks. luku 2.2). Kuvassa 2 käytetäänkin nimenomaan uudempaa eSDK-versiota, sillä bitbake-komennon lopussa on pääte _ext, joka viittaa juuri siihen. Näillä työkaluilla voidaan tietenkin toimia hyvin monella tavalla, joko yhdessä tai erikseen, riippuen käyttäjän tarpeista ja mahdollisuuksista.

Tämän opinnäytetyön tarkoituksena on käydä läpi ohjelmistotuotannon eri vaiheet:

- Yocto SDK -järjestelmän asennustiedoston luominen
- Yocto SDK -järjestelmän asennus
- Eclipse IDE:n asennus ja integrointi Yocto SDK -järjestelmään
- pienen esimerkkiohjelman luominen Eclipse IDE:n avulla
- ohjelmakoodin testaus Qemu-virtuaalilaitteella ohjelmoinnin yhteydessä
- ohjelman asennus kohdelaitteelle (Raspberry Pi) sekä toiminnan testaus
- työn viimeistely luomalla ohjelmaprojektista Yocto-resepti (engl. recipe) myöhempää käyttöä varten.

2 TAUSTAA

Johdannossa mainittujen asioiden ja tekniikoiden tarkempi esittely on tarpeen kokonaiskuvan saamiseksi tämän opinnäytetyön päämäärästä ja menetelmistä. Sulautettujen järjestelmien ala on maailmanlaajuinen ja voimakkaasti kehittyvä niin fyysisten laitteiden kuin ohjelmistojenkin osalta. Uusia yhä pienempiä ja tehokkaammilla prosessoreilla varustettuja laitteita kehitetään ja ohjelmia niiden hyödyntämiseksi tarvitaan. Linux käyttöjärjestelmän synty ja kehitys (alkaen vuodesta 1991) ajoittuu sopivasti tähän kokonaisuuteen, sillä Linux on osoittautunut hyvin mukautuvaksi erilaisiin laitearkkitehtuureihin ja -ympäristöihin. Sen käyttö sulautetuissa järjestelmissä on kasvattanut suosiotaan, ja yhä useammassa laitteessa tänä päivänä toimii Linux [4]. Linux-järjestelmästä on useita eri versioita, samoin kuin niiden kehitykseen on myös monenlaisia työkaluja.

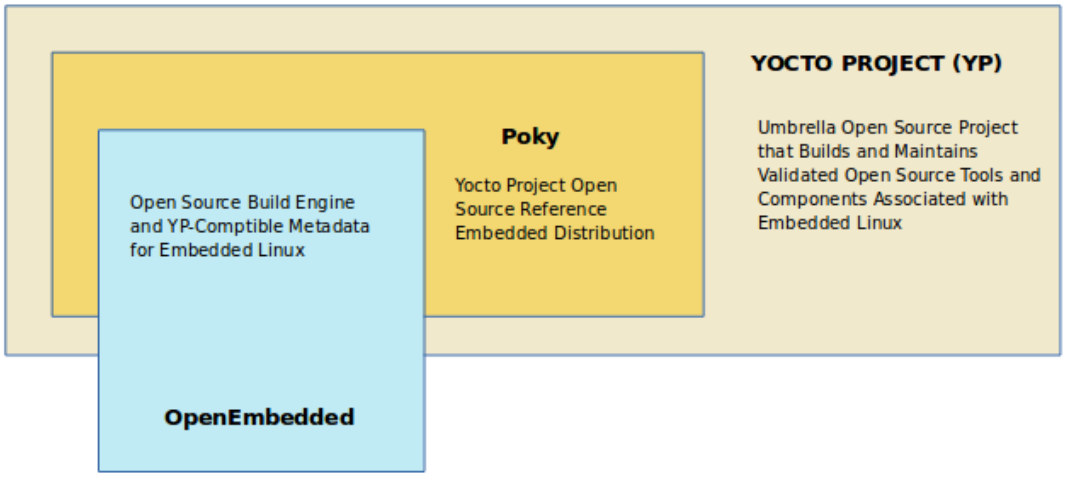
2.1 Yocto-projekti kokoaa yhteen Linuxin kehitystyövälineet

Yocto-projekti on avoimen lähdekoodin laaja yhteistyöhanke, joka sisältää paljon erilaisia valmiita malleja, työkaluja ja menetelmiä Linuxiin pohjautuvien sulautettujen järjestelmien luomiseen riippumatta laitteiston arkkitehtuurista [9,10]. Yocto-projekti on eräänlainen kattonimike tai sateenvarjo kokoelmalle erilaisia tekniikoita ja työvälineitä. Projektia johtaa ja tukee Linux Foundation, jonka toimesta se myös perustettiin v. 2010 [10]. Se on enemmän kuin vain ohjelmakoodin käännöstyökalu, sillä se mahdollistaa sekä käyttöjärjestelmän että siihen liittyvien sovellusten kehittämisen, testaamisen ja virheenkorjauksen (engl. debug). Yocto-projektin kolme tärkeintä osa-aluetta ovat ohjelmakoodin käännösjärjestelmä (engl. build system), ohjelmapakettien käännösohjeistot (engl. package meta-data) sekä kehitystyökalut (engl. developer tools). Yocto -projektin kaksi merkittävintä komponenttia ovat Poky ja Bitbake [10].

- Poky (ohjelmakoodin käännösjärjestelmä)
- Bitbake (käännös- ja koontiprosessin hallintaohjelma, ”Scheduler”)

Yocto-projektin syntymisen taustalla on n. vuonna 2003 alkanut OpenEmbedded -projekti, joka sulautui Yocto-projektiin, kun Linux Foundation lokakuussa 2010 julisti Yocto-projektin syntyneeksi [11]. Poky oli hieman vastaava mutta oma kehityshaaransa, joka myös liitettiin Yocto-projektiin. Useita Linuxin kehitykseen liittyviä työvälineitä ja ohjelmakirjastoja ym. (kuva 3), on sulautettu yhdeksi kehitysympäristöksi nimeltä Yocto

Project, joka on aktiivisen kehitystyön kohteena ja johon tulee uusia lisäyksiä ja muutoksia säännöllisesti.



Kuva 3. Yocto-projektiin sulautetut eri osa-alueet menevät osittain päällekkäin [12].

2.1.1 Yocto-projektin tärkeimmät kehitystyökalut

Yocto-projektin tärkeimmät kehitystyökalut ja niiden käyttötarkoitukset [12] ovat seuraavat:

- CROPS sisältää kehitysympäristön binääritiedostojen luomiseen erilaisille laitearkkitehtuureille ja käyttöjärjestelmille (Windows, Linux, Mac OS)

- Devtool on komentoriviltä käytettävä työkalu ohjelmien kääntämiseen, testaamiseen ja kokoamiseen. Se on hyvin keskeinen osa Extensible SDK (eSDK) -järjestelmää, jota myös tässä opinnäytetyössä käytettiin.

- Extensible Software Development Kit (eSDK) sisältää työkalukokonaisuuden mm. ohjelmien ristikehitykseen (engl. cross-development).

- Eclipse IDE -lisäosa mahdollistaa Yocto-projektin hyödyntämisen Eclipse IDE -kehitystyökalusta käsin suoraan. Sovelluksia voidaan ristikääntää ja testata esim. Qemussa tai fyysisessä laitteessa (esim. Raspberry Pi). Qemu (Quick Emulator) on ohjelmallisesti toteutettu virtuaalinen laitepohja, joka jäljittelee (emuloi) jotain haluttua laitearkkitehtuuria.

- Toaster on graafinen käyttöliittymä Yocto-projektin toimintojen hyödyntämiseksi selaimen avulla.

2.1.2 OpenEmbedded luo perustan Yocto-projektille

OpenEmbedded-projekti sulautettiin Yocto-projektiin maaliskuussa 2011. Se tukee hyvin monenlaisia laitearkkitehtuureja, ja sen työkaluilla voidaan luoda vaikka kokonaan uusi Linux-jakelu (engl. distro), sillä se kykenee ristikääntämään tuhansia erilaisia ohjelmistopaketteja. OpenEmbedded mahdollistaa myös käännös- ja koontiprosessin nopeuttamisen, kun jokin projekti täytyy suorittaa uudelleen asetuksiin tai koodiin tehtyjen muutosten vuoksi [13].

2.1.3 Poky tekee raskaan työn

Poky on Yocto-projektin varsinainen käännösjärjestelmä, joka muodostaa tarvittavat Linux-järjestelmätiedostot kuten ytimen (engl. kernel) ja muita sovelluskokonaisuuksia halutulle kohdearkkitehtuurille [14].

Alkuaan Poky oli OpenEmbedded-projektin rinnakkainen kehityshaara mutta pienempi ja rajoittuneempi. Sitä tuki voimakkaasti sen pääkehittäjä Richard Burdie, ja projekti keskittyi vain joihinkin erityisiin tehtäviin toisin kuin OpenEmbedded. Sen sanottiin yleisesti olevan itsenäinen ja oma käännösjärjestelmänsä, mutta tosiasiallisesti Poky käytti aina bitbake ja OpenEmbedded -arkkitehtuureja, ja oli täysin vastaava sen kanssa [15].

Yocto-projektin luomisen yhteydessä OpenEmbedded jaettiin useisiin eri komponentteihin, kuten OpenEmbedded-Core ja Meta-OpenEmbedded. Samassa yhteydessä Pokysta tuli Yocto-projektin viitejakelu [15]. Nyt kun eri järjestelmiä on liitetty Yocto -projektiin, saattaa asiaa koskevista lähdemateriaaleista huomata, miten termien merkitys ja käyttö vaihtelee jonkin verran riippuen siitä, milloin dokumentti on kirjoitettu tai mistä näkökulmasta asiaa esitellään.

Poky sisältää myös paljon tehtävien toteutuksessa ja kääntämisessä tarvittavia asetusmäärittelyjä, joita kutsutaan nimellä metadata tai meta-poky. Meta-poky tarkoittaa Poky-järjestelmän erityistä määrittelyosiota (engl. Poky-specific metadata).

Metadata tarkoittaa tehtävien määrittelyjä (engl. task definitions) eli ”mitä käännetään ja miten”. Se sisältää mm. seuraavia asioita [14]:

- konfiguraatiot (muuttujien määrittelyt ym.)

- luokat (engl. classes, tiedostot joiden päätteenä on bbclass), jotka sisältävät tietoa ja määrittelyjä ominaisuuksien periytymisestä, käännöslogiikasta ja ohjelmapakettien koostamisesta. Luokat sisältävät suuren osan ”raskaista” tehtävistä, kuten esim. määrittelyt siitä, miten Linux-ydin käännetään tai miten tiedostojärjestelmä rakennetaan

- reseptit (engl. recipes), jotka ovat loogisia ohjeistuksia (”reseptejä”) siitä, miten jokin ohjelmiston osa-alue muodostetaan. Reseptit sisältävät myös yksittäisiä käännettäviä ohjelmakoodeja sekä ohjeita siitä, mitä paketteja asennetaan muodostettavaan Linux-järjestelmään.

2.2 Ohjelmistokehitysjärjestelmää on uudistettu

Yocto-projekti on syntyvaiheidensa jälkeen muuttunut ja kehittynyt, ja siitä on julkaistu useita versioita. Tällä hetkellä (huhtikuu 2019) uusin versio on 2.7.0 (Warrior). Ennen version 2.0 julkaisua sovellusten kehitysjärjestelmä oli nimeltään ADT (Application Development Toolkit), jonka lisäksi saatavilla oli ristikehitystyökalu sekä muutamia muita työkaluja [16]. Yocto-projektin version 2.1 julkaisun myötä sovelluskehitys muuttui nimelle SDK (Software Development Kit), josta nykyään on jo kaksi eri versiota. Näistä varhaisempi Standard SDK on toiminnoiltaan rajoitetumpi kuin uudempi ja monipuolisempi Extensible SDK (eSDK) [16], jota myös tässä työssä käytetään. Yleisnimityksenä käytetään usein kuitenkin vain lyhennettä SDK. Tarkempia nimityksiä käytetään, kun on tarve tehdä ero eri versioiden välillä tai jos on epäselvää kumpaa tarkoitetaan.

Yocto eSDK on monilta osin parempi ja monipuolisempi kuin edeltäjänsä, joten sen käyttö on ehdottomasti järkevä valinta. Erittäin tärkeä ja keskeinen työkalu on devtool (ks. luku 2.1.1), joka ei sisälly vanhempaan versioon. Taulukossa 1 on vertailtu näitä kahta järjestelmää tärkeimpien ominaisuuksiensa osalta [16]. Taulukko on esitetty alkuperäisessä muodossaan englannin kielellä ilmaisujen tiiviyden ja tarkkuuden vuoksi.

Taulukko 1. Taulukosta voidaan havaita, että eSDK sisältää useita ominaisuuksia, joita Standard -versiossa ei vielä ollut. Devtool-työkalun lisäksi on huomattavana parannuksena mahdollisuus luoda kokonainen käyttöjärjestelmä (engl. build images). Tällöin Yocto-projektia ei välttämättä tarvita, vaan kehitystyön kaikki osa-alueet voidaan hoitaa eSDK-järjestelmällä [16].

Feature	***Standard SDK***	***Extensible SDK***
Toolchain	Yes	Yes (only full installation)
Debugger	Yes	Yes (only full installation)
Size	100+ MBytes	1+ GBytes (or 300+ MBytes for minimal w/toolchain)
devtool	No	Yes
Build Images	No	Yes
Updateable	No	Yes
Managed Sysroot	No	Yes (through the use of devtool)
Installed Packages	No	Yes
Construction	Packages	Shared State

3.2 Pieni mutta pippurinen Raspberry Pi

Tässä työssä kohdelaitteena käytetyn Raspberry Pi:n ”tarina” ansaitsee pienen ”ekskursion”, sillä se on mielenkiintoinen kertomus siitä, miten jostain pienestä alusta voi räjähdysmäisesti kasvaa jotain yllättävää ja suurta. Raspberry Pi on tänä päivänä tullut suosituksi paljolti muussakin käytössä, kuin mihin se alun perin suunniteltiin [17]. Sen syntyhistoria liittyy Englantiin ja Cambridgen yliopistoon, missä Eben Upton työskenteli tietotekniikan opetuksen johtajana v. 2006 [6,18]. Hän huomasi, kuinka monilla opiskelijoilla oli vain vähän kokemusta ohjelmoinnista ja että 1980-luvulla tehdyt kotitietokoneet eivät olleet kovin miellyttäviä käyttää. Kuitenkin monet IT-alan työntekijät opiskelivat ohjelmoimaan näillä vanhoilla kotitietokoneilla. Kun tietokoneet myöhemmin kehittyivät paremmiksi, monet osasivat kyllä käyttää niitä, mutta todellinen ohjelmointitaito ja tietokoneen tuntemus puuttuivat edelleen.

Upton työskenteli myös Broadcomilla integroitujen piirien kehittäjänä [6,18] haluten kehittää tietokoneen, johon kaikilla olisi varaa ja jota opiskelijat voisivat käyttää tutkimustarkoituksiin. Hän tutki matkapuhelimissa käytettyä pienikokoista mutta tehokasta huipputekniikkaa ja loi sen pohjalta useita prototyyppejä. Upton perusti myös Raspberry Pi Foundationin marraskuussa 2008 yhdessä muutaman asiantuntijan kanssa [17]. Raspberry Pi Foundation on Iso-Britanniassa toimiva hyväntekeväisyysjärjestö, joka

pyrkii saattamaan tietojenkäsittely- ja digitaalitekniikan mahdollisuudet edullisesti kaikkien ihmisten ulottuville ympäri maailmaa [10].

Pitkän kehitystyön ja useiden prototyyppien luomisen jälkeen toukokuussa 2011 tapahtui merkittävä käänne, kun eräs BBC:n toimittaja (Rory Cellan-Jones) laittoi blogiinsa videon Raspberry Pi:n protoyypistä [17]. Tieto tästä projektista levisi välittömästi kulovalkean tavoin, ja Raspberry Pi:n kehitystyö jatkui yhä kiihkeämmin. Lukuisia prototyyppejä rakennettiin, ja tuotteen lopullinen julkistus oli suunniteltu helmikuulle 2012. Kävi kuitenkin ilmi, ettei suunniteltu 10 000 kappaleen erä riitä, vaan tarve ja odotus markkinoilla olivat ainakin 10-kertaisia. Tuotteen suosio kasvoi lopulta nopeasti niin suureksi, että Raspberry Pi Foundation joutui ulkoistamaan tuotannon ja jakelun eri yrityksille, kuten Premier Farnell Ltd ja RS Components. Kolmessa kuukaudessa laitetta myytiin yli puoli miljoonaa kappaletta ja v. 2013 loppuun mennessä jo kaksi miljoonaa [17].

Laitteen suosio ei ole laskenut alkuinnostuksen jälkeen, vaan pikemmin kasvanut vakaasti. Edullinen ja pienikokoinen mutta kuitenkin kokoonsa nähden tehokas ja monipuolinen minitietokone antaa mahdollisuuksia monenlaiseen käyttöön erilaisissa ympäristöissä [17]. Raspberry Pi:sta on kehitetty jo useita versioita, ja kehitystyö jatkuu edelleen. Huhtikuussa 2019 uusin versio on 3 B+, jota tässä opinnäytetyössä käytetään.

3 YOCTO SDK -SOVELLUSKEHITYSYMPÄRISTÖN ASENNUS JA TESTAUS

Yocto SDK:n asennus toimivaksi sovelluskehitysympäristöksi kaikkine työkaluineen on hyvin monivaiheinen sarja erilaisia toimintoja. Järjestelmää on kehitetty paljon ja siitä on kaksi versiota. Standard SDK on vanhempi ”malli” ja siinä asennustiedosto on lähtökohtaisesti isompi kuin uudemmassa Extensible SDK (eSDK) -versiossa. Standard -versiossa kaikki työkalut tulevat automaattisesti mukana, kun taas eSDK:ssa voi asennustiedoston luomisvaiheessa valita myös pienemmän (minimal) version. Minimal -versiossa tarpeettomat työkalut karsitaan pois niin, että tiedoston kooksi tulee vain n. 30 Mt. Täydessä versiossa (full) kaikki työkalut ovat mukana, ja tiedoston koko on lähes 1 Gt. Muitakin variaatioita näiden kahden ääripään väliltä voidaan luoda, mikäli mukaan halutaan vain joitain tiettyjä työkaluja [19].

Eri versioita vertaillessa voi taulukon 1 perusteella havaita monia syitä, miksi eSDK on suositeltavampi vaihtoehto. Monien uusien ominaisuuksiensa lisäksi se on myös päivitettävissä toisin kuin Standard SDK sekä myös laajennettavissa erilaisten lisäpakettien asennuksen kautta [19, 20].

3.1 Asennusympäristön selvittäminen

Asennusta varten on ensin selvitettävä kohdejärjestelmän (engl. target) ja isäntäjärjestelmän (engl. host) prosessoriarkkitehtuurit. Kohdejärjestelmä on se, mihin varsinaisia sovelluksia tai ohjelmistoja kehitetään ja isäntäjärjestelmä puolestaan se, missä ympäristössä ohjelmistokehitysjärjestelmä itse toimii. Koska sulautettujen järjestelmien kehitystyössä isäntä- ja kohdejärjestelmä ovat usein erilaisia, täytyy ohjelmistokehitysjärjestelmän kyetä kääntämään sovellus kohdejärjestelmään sopivaksi. Tällaisesta ohjelmistokehityksestä käytetään termiä ristikehitys (engl. cross-development) [10].

Ohjelmiston kehitys- ja käännöstyössä tarvitaan aina monenlaisia työkaluja, jotka yleensä sisältyvät automaattisesti käännöstyön suorittavaan ohjelmistokokonaisuuteen. Niitä ovat mm. kohdejärjestelmän kirjastotiedostot (engl. libraries), kääntäjä (engl. compiler), linkitin (engl. linker) ja virheenjäljitin (engl. debugger). Ristikehityksessä tarvitaan vastaavia, mutta juuri tätä erityistä tarkoitusta varten suunniteltuja työkaluja

(engl. cross-compiler jne.). Nämä kaikki sisältyvät Yocto-projektin avulla luotavan eSDK:n asennustiedoston täyteen versioon. Asennustiedostoa luotaessa täytyy määritellä kaikki tarvittavat muuttujat local.conf -tiedostossa. Muuttujan SDK_EXT_TYPE -arvo ”full” määrittelee laajimman työkaluvalikoiman sisällytettäväksi asennustiedostoon. Oletusarvona tosin on ”full”, vaikkei sitä huomaisi erikseen määritelläkään [19].

SDK_EXT_TYPE = "full"

Mikäli Yocto SDK asennetaan toiselle laitteelle, jossa on erilainen prosessoriarkkitehtuuri kuin missä Yocto-projekti toimii, täytyy määritellä muuttuja SDKMACHINE [16]. Oletusarvona tällä muuttujalla oli i686, joka tarkoittaa 32-bittistä x86-järjestelmää. Kysymysmerkki (ks. alla) ennen yhtäläisyysmerkkiä tarkoittaa oletusarvoa. Ellei muuta arvoa määritellä, kuten tässä seuraavalla rivillä, oletusarvo jää voimaan. Jos järjestelmä halutaan muuttaa 64-bittiseksi, täytyy arvoksi antaa x86_64.

#SDKMACHINE?="i686"
SDKMACHINE = ”x86_64”

Kohdelaitteen arkkitehtuuri määritellään muuttujalla MACHINE (tässä Raspberry Pi 3). Asennustiedostoon sisällytetään kaikki tarvittavat työkalut ja kaikki kääntäjän tarvitsema tieto kohdearkkitehtuurista [16].

MACHINE = "raspberrypi3"

3.2 Asennustiedoston luominen

Kun kaikki muuttujat on määritelty, Yocto SDK:n asennustiedosto luodaan bitbake -ohjelmalla. Asennustiedoston voi myös ladata Internetistä, mikäli tarvittava tiedosto löytyy osoitteesta http://downloads.yoctoproject.org/releases. Tässä opinnäytetyössä asennustiedosto luotiin kuitenkin Yocto-projektin avulla, kun luvussa 3.1 mainitut muuttujat oli asetettu. Yocto-projektin asennus oli jo ennalta suoritettu. Asennustiedostoa luotaessa on huomattava, että eSDK:ta varten komennon lopussa täytyy olla pääte _ext (extensible). Tarvittava komento on Yocto-projektissa hyvin yksinkertainen [21].

$ bitbake <image> -c populate_sdk_ext

Image-määrite voi olla esim. ”core-image-minimal”, joka on pienin versio käynnistymään (engl. boot) kykenevästä Linux-järjestelmästä. Monipuolisempi (mutta myös

isompi) versio olisi ”core-image-full-cmdline”, mutta tässä työssä käytettiin pienempää versiota. Asennustiedoston luominen saattaa ensimmäisellä kerralla kestää koneen tehoista riippuen jopa useita tunteja, joten se on kätevää laittaa ”valmistumaan” vaikka yöaikaan, kuten itse Yocto-projektin asennuskin. Kun asennustiedosto on valmis, se löytyy Yocto-projektin kansiosta ~build/tmp/deploy/sdk ja sen tunnistaa tiedostopäätteestä sh. Nimi on yleensä pitkä sisältäen sekä isäntä- että kohdearkkitehtuurin määritelmät. Tässä työssä tiedoston nimi oli seuraava:

poky-glibc-x86_64-core-image-minimal-cortexa7hf-neon-vfpv4-toolchain-ext-2.5.1.sh

jossa ”x86_64” tarkoittaa Isäntäarkkitehtuuria ja ”cortexa7hf” kohdearkkitehtuuria.

3.3 Yocto SDK:n asennus ja tarvittavat lisätyökalut

Yocto SDK:n isäntäjärjestelmänä tässä työssä oli Linux Ubuntu 18.04.2 (Bionic Beaver) asennettuna Oracle VirtualBox:iin. Aluksi järjestelmään täytyi asentaa erilaisia käännös- ja koontiprosessissa tarvittavia työkaluja kuten Python ja gawk sekä grafiikkaa ja Eclipse IDE -lisäosaa varten xterm [22]. Nämä asennettiin seuraavilla komennoilla:

$ sudo apt-get update

$ sudo apt-get install -y gawk wget git-core diffstat unzip texinfo gcc-multilib \

build-essential chrpath socat cpio python python3 python3-pip python3-pexpect \

xz-utils debianutils iputils-ping libsdl1.2-dev xterm

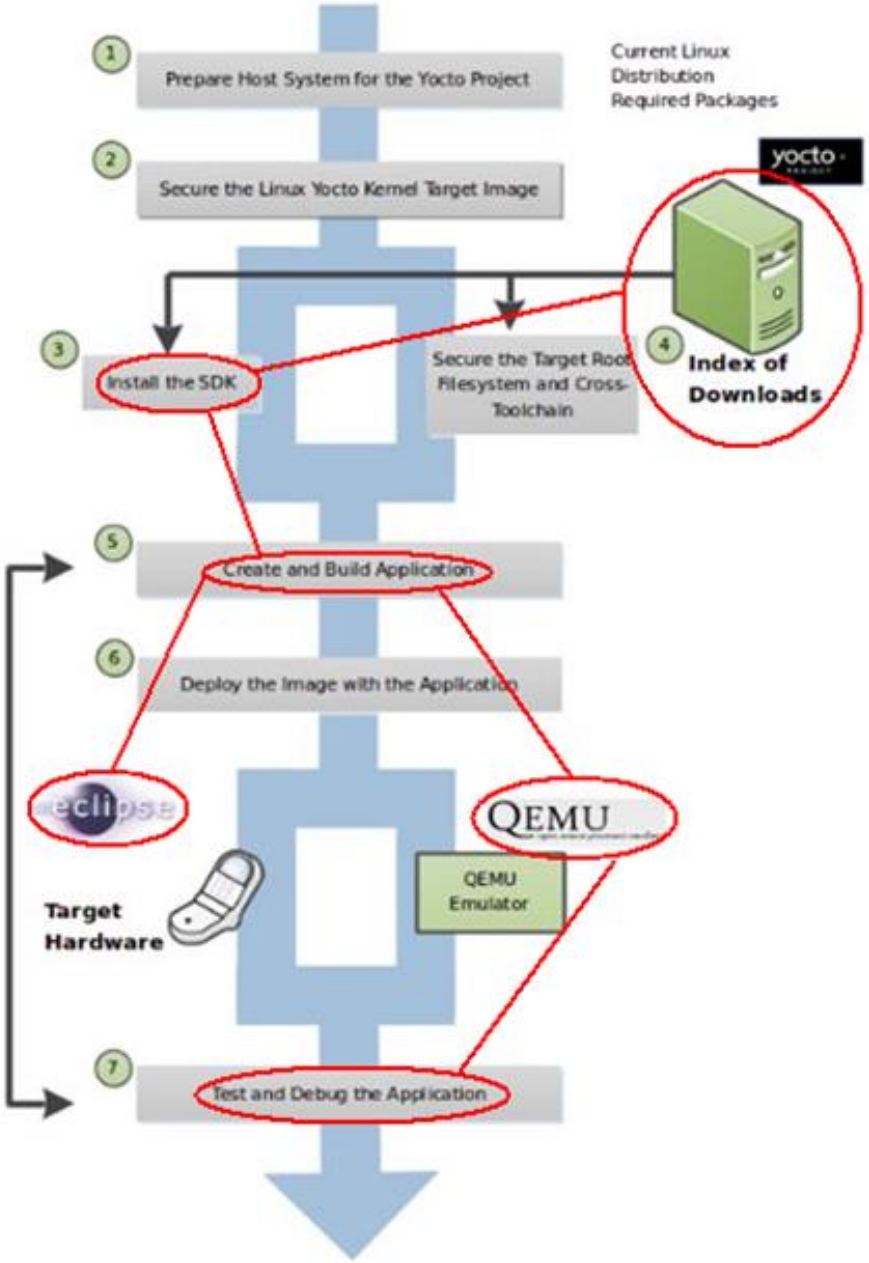
Tämän jälkeen eSDK:n asennustiedosto kopioitiin isäntäjärjestelmään, lisättiin oikeus suorittamiseen sekä suoritettiin asennus.

$./poky-glibc-x86_64-core-image-minimal-cortexa7hf-neon-vfpv4-toolchain-ext-2.5.1.sh

Asennuksen sijainniksi valittiin oletuskansio poky_skd, ja prosessin kesto oli vähän alle tunnin. Komentoriville tuli varoituksia siitä, ettei isäntäjärjestelmän yhteensopivuutta ole täysin testattu. Kaikki toimi kuitenkin hyvin, sillä varsinaisia virheilmoituksia ei tullut.

3.4 Qemu ja Raspberry Pi testaus- ja käyttöympäristöinä

Kun Yocto SDK asennetaan ja sovelluksia aletaan kehittää, tarvitaan testiympäristöksi joko fyysinen laite kuten Raspberry Pi tai virtuaalinen laite Qemu. Molemmat tarvitsevat toimiakseen käyttöjärjestelmän, joka voidaan luoda Yocto SDK:lla. Kuvassa 4 on hahmoteltu erilaisia Yocto SDK:n asennus-, ohjelmointi- ja testausvaiheen etenemistapoja.



Kuva 4. Yocto SDK -järjestelmän asennus, käyttöönotto, sovellusten kehitys ja testaus on kaikkineen monivaiheinen projekti ja siinä on useita vaihtoehtoisia etenemistapoja. Tässä työssä edettiin punaisen viivan osoittamaa linjaa myöden. Ohjelmakoodia testattiin työstämisvaiheessa Qemussa Eclipse IDE:n debug-komennon avulla [23].

Qemu (Quick Emulator) on ohjelmallisesti toteutettu virtuaalinen laitepohja, joka nimensä mukaisesti emuloi (jäljittelee) jotain tiettyä laitearkkitehtuuria. Tässä tapauksessa emuloitava arkkitehtuuri on arm, jota käytetään juuri Raspberry Pi -piireissä. Ohjelmakoodia on toki tarkoituksenmukaista testata myös fyysisessä kohdelaitteessa, ja niin tässä opinnäytetyössä tehdäänkin, mutta esimerkin vuoksi on haluttu käyttää testialustana myös Qemua. Tämän vaihtoehdon esille tuominen on hyödyllistä myös siksi, että samaa menetelmää voidaan käyttää monelle muullekin laitearkkitehtuurille. Jos kohdelaitetta ei ole fyysisesti saatavilla testaushetkellä, mutta kyseiselle laitearkkitehtuurille on olemassa emulaattori, voidaan ohjelmia testata siinä. Tarvittavat työkalut ja järjestelmätiedostot on tietysti luotava ja asennettava ensin.

Eclipse IDE on mahdollista integroida järjestelmään osaksi kehitysprosessia, ja tämä esitelläänkin tarkemmin kappaleessa 3.5.

3.4.1 Qemun järjestelmätiedostot

Kun eSDK-kehitysympäristö on asennettu ja Qemun käyttöönottoa valmistellaan, on halutulle arkkitehtuurille ensin luotava Linux-ydin (engl. kernel) ja tiedostojärjestelmä (engl. root filesystem). Näiden avulla Qemu kykenee käynnistymään ja ohjelmistoja voidaan testata siinä. Tämä Linux-järjestelmätiedostojen (engl. image) luominen tapahtuu samalla tavalla kuin Yocto-projektissakin. Itse asiassa eSDK toimii samalla tavalla kuin Yocto-projekti ja kykenee kääntämään Linux-ytimen ja muut tarvittavat tiedostot yhtäläisesti.

Ennen Linux-ytimen ja tiedostojärjestelmän luomista täytyy asettaa tarvittavat muuttujat local.conf -tiedostoon [24].

MACHINE = ”qemuarm”

- ❖ tämä määrittelee halutun kohdelaitteen ja -arkkitehtuurin, jotka tässä tapauksessa ovat Qemu ja arm. Local.conf -tiedostossa on valmiina automaattisesti luotu lista tarjolla olevista Qemu-laitteista ja muuttuja-arvoista, joista valitaan sopiva.

EXTRA_IMAGE_FEATURES += " debug-tweaks eclipse-debug tools-debug tools-sdk ssh-server-openssh "

- tähän muuttujaan voidaan asettaa Linux-tiedostojärjestelmään asennettavat lisätyökalut, -ominaisuudet ja palvelut. Ssh-palvelinta tarvitaan yhteyden luomiseen isäntäjärjestelmästä (engl. host) Qemuun (engl. guest) ja sitä kautta testiohjelmien asennusta varten mm. devtool deploy-target -komennolla.

IMAGE_INSTALL_append = " kernel-image kernel-devicetree tcf-agent openssh-sftp-server"

- TCF (Target Communication Framework) on verkkoprotokolla sulautettujen kohdejärjestelmien ajamiseksi. Eclipse IDE tarvitsee sitä debug-toiminnon suorittamiseksi.

Kun edellä mainitut muuttujat on asetettu, voidaan prosessi käynnistää bitbake -komennolla. Komentorivillä siirrytään eSDK:n asennuskansioon, joka tässä työssä on oletuskansio poky_sdk. Ennen bitbake-ohjelman käyttöä täytyy kuitenkin asettaa ympäristömuuttujat source-komennolla.

§ source layers/poky/oe-init-build-env

§ bitbake core-image-minimal

Tämä komento luo sekä Linux-ytimen että tiedostojärjestelmän ja muutamia muita tiedostoja. Kun tämä toteutetaan ensimmäistä kertaa, voi koko prosessiin kulua useita tunteja riippuen isäntäjärjestelmän tehosta.

Kun prosessi on valmis, löytyvät halutut tiedostot kansiosta ~poky_sdk/build/tmp /deploy/images/gemuarm. Linux-ydin on n. 5 megatavun kokoinen tiedosto ja tässä työssä sen nimi oli ”zImage-qemuarm.bin”. Tätä tietoa tarvitaan myöhemmin Eclipsen ja Yocto SDK:n yhteistoimintaa konfiguroitaessa. Toinen tarvittava tiedosto on tiedostojärjestelmä, joka tässä työssä oli nimeltään ”core-image-minimal-qemuarm-20190416203228.rootfs.tar.bz2”. Se on pakattu tiedosto, joka sisältää mm. ristikääntäjän (engl. cross compiler) tarvitsemat tärkeät järjestelmäkirjastot yms.

Tiedostojärjestelmä täytyy purkaa, jotta Eclipse IDE löytää kohdelaitteen sysroot -järjestelmän ja pystyy käyttämään ristikääntäjää [23]. Tässä työssä noudatettiin suosituksia purkamalla se kansioon ~build/MY_QEMU_ROOTFS. Tiedoston purkamiseen on olemassa erityinen työkalu runqemu-extract-sdk kansiossa poky_sdk/sysroots/x86_64-pokysdk-linux/usr/bin. Komento, jolla purku suoritetaan, on

varsin pitkä ja sisältää polut suoritettavaan ohjelmaan, purettavaan tiedostoon sekä kohdekansioon. Kokonaisuudessaan se oli seuraava.

$ /home/rauno/poky_sdk/sysroots/x86_64-pokysdk-linux/usr/bin/runqemu-extract-sdk core-image-minimal-qemuarm-20190416203228.rootfs.tar.bz2 /home/rauno/poky_sdk/build/MY_QEMU_ROOTFS

Qemun toiminta testattiin tässä vaiheessa käynnistämällä se komennolla runqemu <target>. Komennossa target on polku kansioon, jossa ovat Qemun tarvitsemat järjestelmätiedostot (ydin ja tiedostojärjestelmä). Tämä suoritettiin eri terminaalissa kuin bitbake, sillä tarvittavat ympäristömuuttujat ovat erilaiset.

$ source environment-setup-cortexa7hf-neon-vfpv4-poky-linux-gnueabi

$ runqemu qemuarm /home/rauno/poky_sdk/build/tmp/deploy/images/qemuarm

3.4.2 Käyttöjärjestelmä Raspberry Pi -piirille

Raspberry Pi -piiriä varten luotiin omat järjestelmätiedostonsa samalla tavoin kuin Qemulle kappaleessa 3.4.1. Muuttujat local.conf -tiedostossa määriteltiin uudelleen seuraavalla tavalla:

MACHINE = "raspberrypi3"
IMAGE_FSTYPES = "ext4 rpi-sdimg"

MACHINE-muuttuja määrittelee kohdelaitteen ja IMAGE_FSTYPES (=Image File System Types) puolestaan tiedostojärjestelmätyypin ja millaisen paketin ohjelma tuottaa. Tässä ext4 on Linux-järjestelmissä yleisimmin käytetty tiedostojärjestelmä ja rpi-sdimg tarkoittaa, että tuotetaan Raspberry Pi -piiriä varten SD-kortille asennettavaksi sopiva paketti. Tuossa paketissa on mukana ydin, tiedostojärjestelmä ja osioiden asettelutiedot (engl. partition layout). Paketti purettiin ja asennettiin SD-kortille Etcher-ohjelmalla, joka on saatavissa myös Windows käyttöjärjestelmälle. Asennuksen jälkeen järjestelmän toiminta testattiin vielä käynnistämällä Raspberry Pi.

3.5 Eclipse IDE:n ja Yocto-lisäosan asennus ja testaus

Eclipse IDE on avoimen lähdekoodin ohjelmointityökalu sovellusten kehittämiseen hyvin monenlaisilla ohjelmointikielillä (mm. Java, C, C++, PHP). Eclipse IDE on myös saatavilla erilaisille käyttöjärjestelmille (Windows, Linux, Mac OS), ja siihen saa asennettua hyvin monenlaisia lisäosia (engl. plug-in) muita järjestelmiä kuten tässä työssä käytettyä Yocto-projektia varten. Tässä luvussa esitetty Eclipse IDE:n konfigurointi noudattaa pääpiirteissään lähteenä käytetyn Yocto-projektin manuaalin [23] ohjeita, eikä joka kohdassa ole siksi erikseen merkitty viitenumeroa.

Eclipse IDE on toteutettu pääasiassa Java-ohjelmointikielellä ja tarvitsee toimiakseen Java-ympäristöä. Tässä työssä asenettiin versio openjdk-8.

$ sudo apt-get update
$ sudo apt install openjdk-8-jdk

3.5.1 Eclipse IDE:n asennus

Eclipse IDE asennetaan aina omana versionaan eri ohjelmointikieliä varten (tässä C++). Asennuksessa käytettiin Ubuntu Make -ohjelmaa, joka on kehitetty eri ohjelmien helppoa asennusta varten [25]. Yhdellä komennolla se lataa lähdekoodipaketin ja kaikki riippuvuudet, kääntää lähdekoodin sekä lisää työpöydän käynnistyskuvakkeen. Ubuntu Make ja Eclipse IDE:n uusin versio 2019-03 (4.11.0) asennettiin seuraavasti:

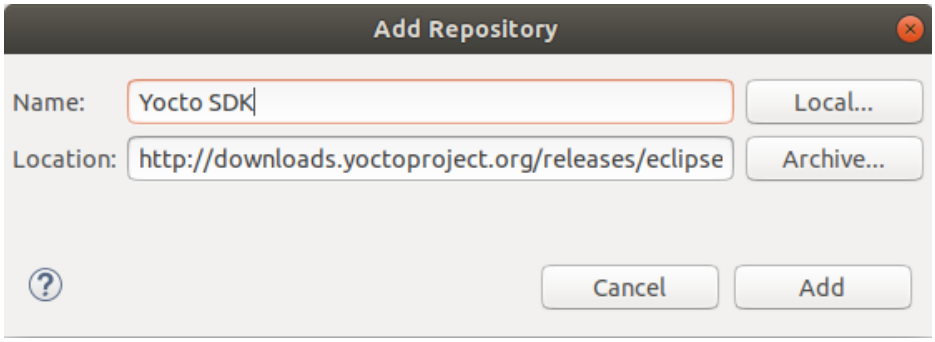
$ sudo apt-get update

$ snap install ubuntu-make -classic

$ ubuntu-make.umake ide eclipse-cpp

3.5.2 Yocto-lisäosan asennus

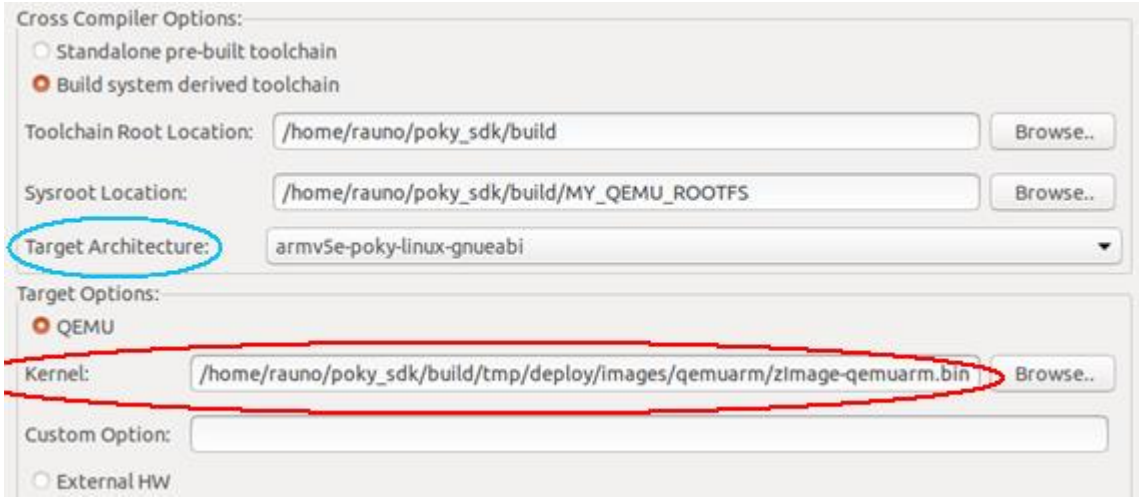
Eclipse IDE:n integroimiseksi Yocto SDK -järjestelmään asennettiin siihen Yocto -lisäosa. Valikosta ”Help > Install New Software > Add” avautuvan ikkunan location -ruutuun asetettiin uusimman lisäosan URL-osoite (kuva 5). Tämän jälkeen hyväksyttiin kaikki asennetavat paketit sekä käynnistettiin Eclipse IDE uudelleen.



Kuva 5. Eclipse IDE kuvakaappaus. Yocto-lisäosa ladataan osoitteesta http://downloads.yoctoproject.org/releases/eclipse-plugin/2.6.1/oxygen.

3.5.3 Eclipse IDE:n ja Yocto-lisäosan konfigurointi

Yocto-lisäosa konfiguroitiin avaamalla valikko ”Window > Preferences” ja valitsemalla vasemmasta sivupalkista juuri asennettu Yocto-lisäosa. Avautuvassa ikkunassa (kuva 6) määritellään Linux-ytimen ja kohdelaitteen tiedostojärjestelmän sijainnit sekä muut perusasetukset, joista ensimmäinen ja merkittävin on työkalukokonaisuuden valinta. Tarjolla on kaksi vaihtoehtoa: "Standalone pre-built toolchain” ja “Build System Derived Toolchain”, joista jälkimmäinen on oletuksena, ja sitä myös tässä työssä käytettiin. Ensimmäistä vaihtoehtoa voi käyttää, jos tarkoitus ei ole tehdä kohdelaitteen käyttöjärjestelmää ja työkaluvalikoima on ladattu erikseen. Jälkimmäistä käytetään, kun työkaluvalikoima on tehty osana käännösprosessia, kuten tässä työssä.



Kuva 6. Eclipse IDE kuvakaappaus.

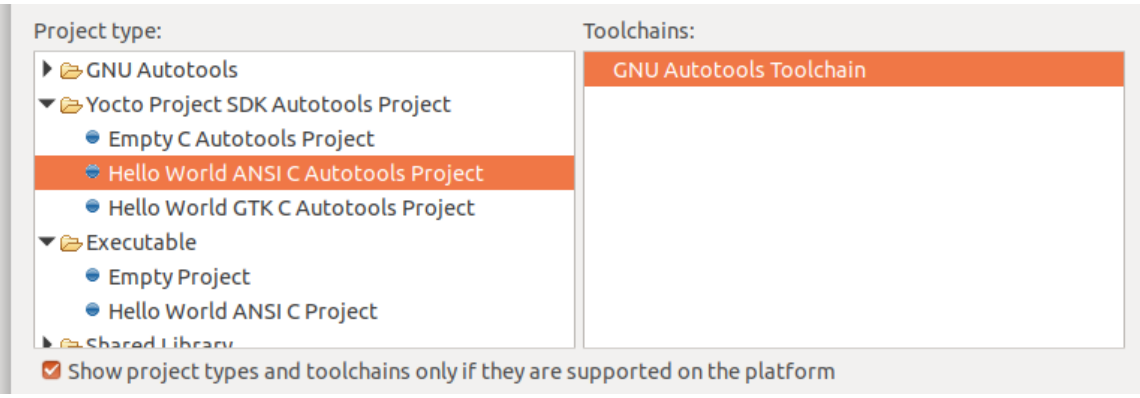
Toolchain Root Location -ruudussa määritellään polku bitbake-komennon suoritushakemistoon (~poky_sdk/build).

Sysroot Location tarkoittaa polkua sysroot-hakemistoon ~build/MY_QEMU_ROOTFS, jonne Qemua varten luotu järjestelmätiedosto aiemmin purettiin (luku 3.4.1).

Myös kohde (Target Architecture, kuvassa 6 sininen kehys) määritellään sen mukaan, mitä laitearkkitehtuuria aiotaan emuloida. Alasvetovalikosta tulee esiin tarjolla olevat vaihtoehdot, joita tässä tapauksessa oli vain yksi eli ”armv5e-poky-linux-gnueabi”. Lopuksi valitaan vielä, käytetäänkö kohdejärjestelmänä Qemua vai ulkoista fyysistä laitetta. Tässä työssä käytettiin Qemua ohjelmoinnin yhteydessä testiympäristönä ja asetuksiin täytyi siksi määritellä polku Qemua varten tehtyyn Linux-ytimeen (kuvassa 6 punainen kehys). Custom option -asetusta tarvitaan, jos Qemulle halutaan antaa jotain lisämääritteitä, mutta tässä tapauksessa se jätettiin tyhjäksi.

3.5.4 Hello World -testiohjelman käyttö

Eclipse IDE:n ja Yocto eSDK:n asennus ja yhteistoiminnan käyttöönotto on monivaiheinen prosessi, jossa on huomioitava paljon yksityiskohtia. Tätä yhteistoimintaa testattiin aluksi Eclipsestä löytyvällä valmiilla Hello World -malliprojektilla valitsemalla "File > New > C/C++ Project > Next > C Managed Build > Next” ja laajentamalla valikko "Yocto Project SDK Autotools Project". Sitten valittiin "Hello World ANSI C Autotools Projects" (kuva 7), joka on C-kielellä toteutettu malli ja jolla pääsee helposti ohjelmoinnin alkuun. Projektille annettiin nimi HelloTest, ja toiminto viimeisteltiin painamalla Finish.



Kuva 7. Eclipse IDE kuvakaappaus.

Qemun toiminta yhdessä Eclipse IDE:n kanssa onnistuu debug-tilassa, jolloin Qemu käynnistetään Eclipse IDE:stä suoraan, ja käännetty ohjelma asennetaan samalla Qemuun. Tämän toteuttamiseksi täytyi tehdä vielä useita asetuksia niin Eclipse IDE:ssä kuin isäntäkoneessakin. Ensin tarkistettiin onko isäntäjärjestelmään asennettu rpcbind. Rpcbind-apuohjelma on palvelu, joka muuntaa RPC-ohjelmien numerot universaaleiksi osoitteiksi. Sen on oltava käynnissä isäntälaitteessa, jotta se voi tehdä RPC-yhteydenottoja. Ohjelman toiminta tarkistettiin ja todettiin, ettei sitä oltu asennettu. Tarkistus ja asennus suoritettiin seuraavilla komennoilla:

$ sudo service rpcbind status

$ sudo apt-get install rpcbind

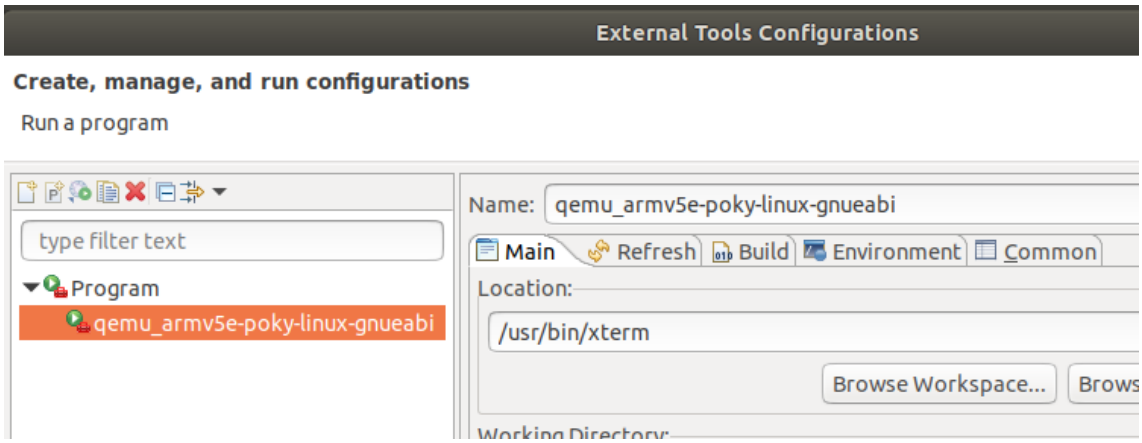
Asentamisen jälkeen täytyi vielä muokata tiedostoa /etc/init.d/rpcbind siten, että siellä on seuraava rivi:

OPTIONS="-i -w"

Rivi oli kyllä jo olemassa, mutta siitä puuttui i-kirjain. Muokkauksen jälkeen palvelu täytyi käynnistää uudelleen komennolla:

$ sudo service portmap restart

Kun rpcbind-palvelu oli näin kunnossa, avattiin Eclipse IDE:stä valikko ”Run > External Tools > External Tools Configurations“ (kuva 8) ja valittiin vasemmalta valikkopalkista kohdearkkitehtuurin työkaluohjelmisto.



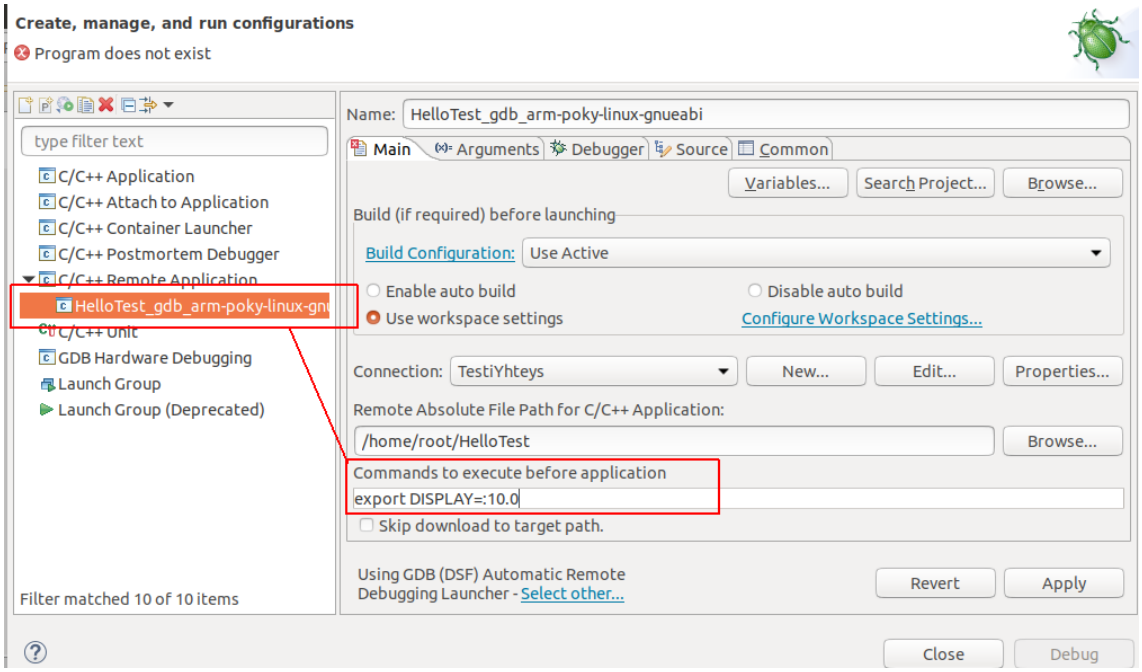
Kuva 8. Eclipse IDE kuvakaappaus.

Qemu käynnistyy komennolla Run (paneelin alalaidassa) ja ensin avautuu xterm -terminaaliemulaattori, johon täytyy antaa oman Linux-järjestelmän salasana. Vasta sen jälkeen Qemu käynnistyy tty1-terminaalissa, ja käyttäjätunnuksella root pääsee sisään. Qemun IP-osoitteen voi tarkistaa komennolla ifconfig, mutta se näkyy myös xterm-terminaaliemulaattorissa. Se on yleensä 192.168.7.2, ellei mitään poikkeavaa ole tapahtunut.

Kun Qemu on käynnistynyt, siihen otetaan isäntäkoneen terminaalista ssh-yhteys:

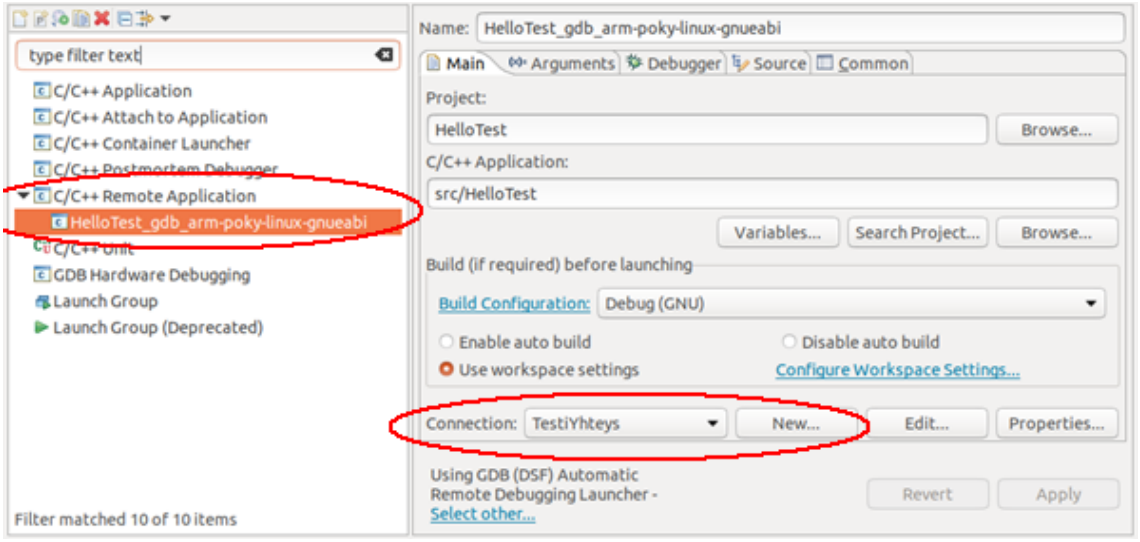
$ ssh -XY root@192.168.7.2

Seuraavaksi asetetaan komento export DISPLAY=:10.0 suoritettavaksi ennen sovelluksen käynnistymistä (kuva 9).



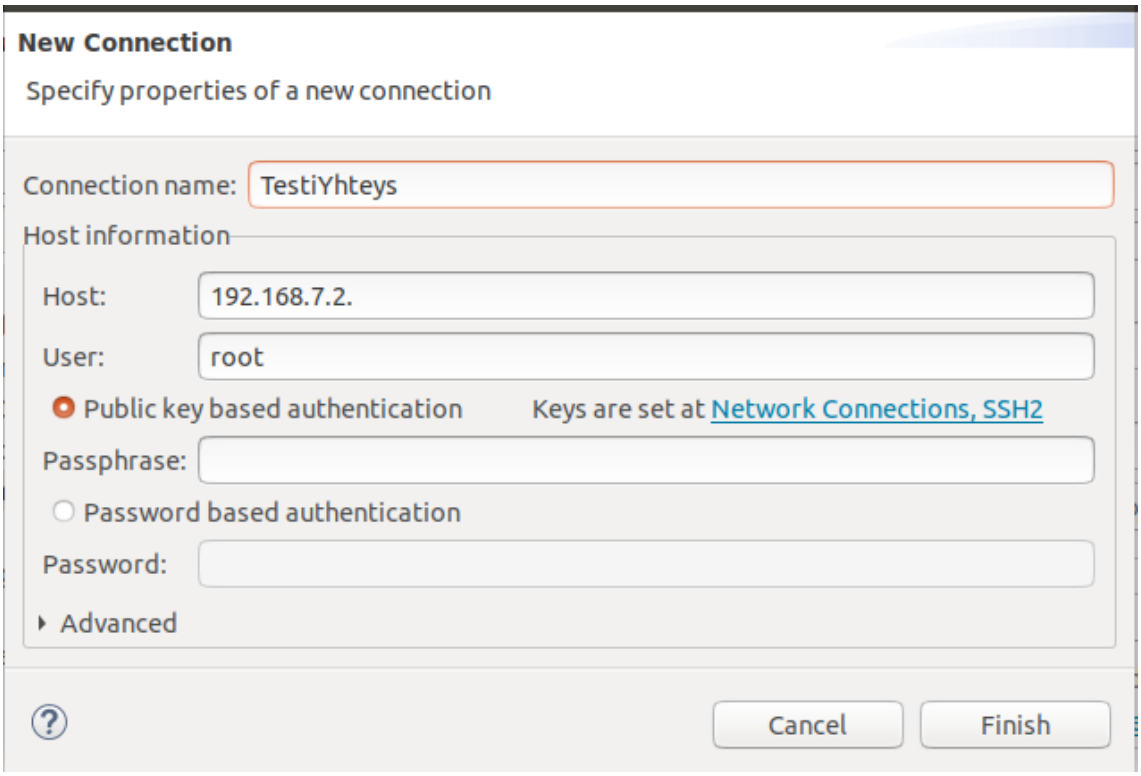
Kuva 9. Eclipse IDE kuvakaappaus.

Ssh-yhteys täytyi pitää toiminnassa tämän jälkeen koko prosessin ajan. Seuraavaksi avattiin valikko ”Run > Debug Configuration” ja vasemmalta valikkopalkista laajennettiin "C/C++Remote Application" ja valittiin oma projekti (kuva 10). Projektin asetusnäkymän Debugger-välilehdeltä tarkistettiin käytössä oleva virheenjäljitin (engl. debugger), ja Main-välilehdellä luotiin uusi yhteys valitsemalla New.



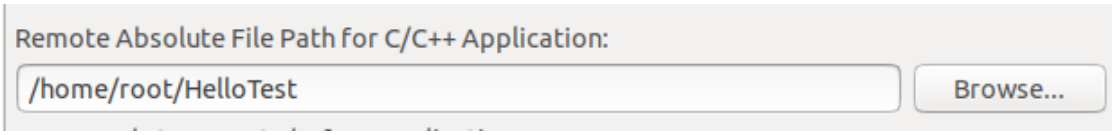
Kuva 10. Eclipse IDE kuvakaappaus.

Yhteydeksi asetettiin ssh, nimeksi TestiYhteys, Host-ruutuun Qemun IP-osoite ja käyttäjäksi root (kuva 11). Salasana-ruutu jätettiin tyhjäksi ja lopuksi valittiin Finish sekä varmistettiin vielä main-näkymän alasvetovalikosta (Connection), että haluttu yhteys on valittuna (kuva 10).



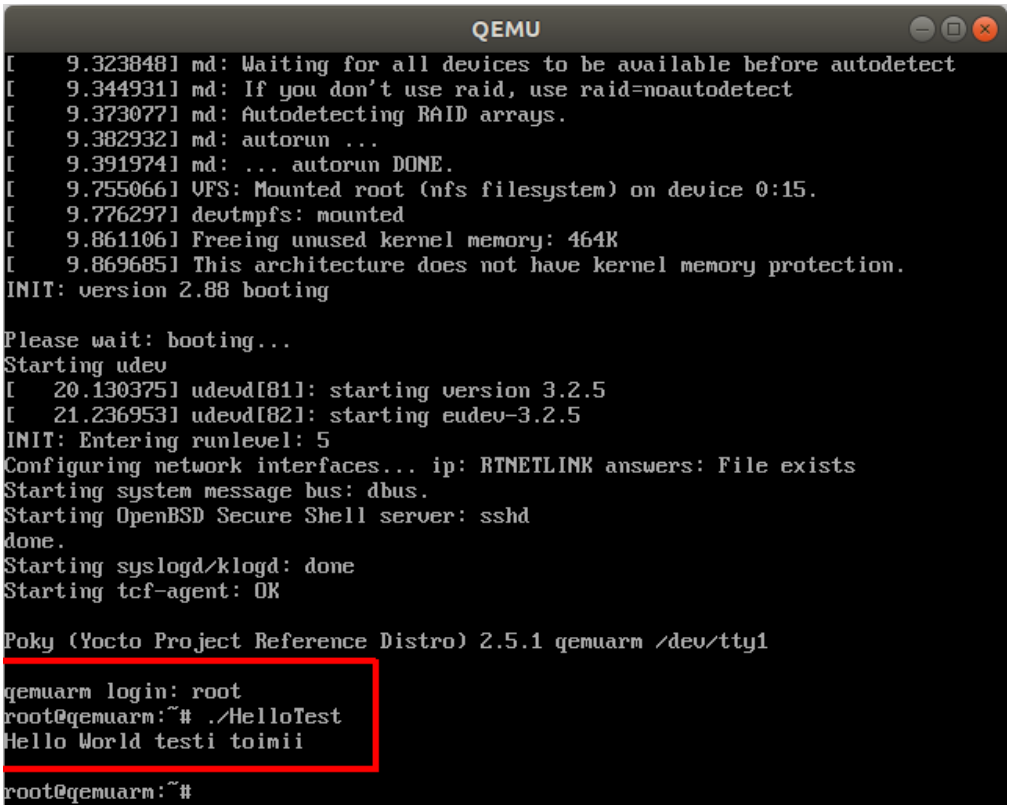
Kuva 11. Eclipse IDE kuvakaappaus.

Kohdejärjestelmästä (Qemu) valitaan haluttu sijainti asennettavalle ohjelmalle joko Browse-komennolla selaamalla tai kirjoittamalla suoraan halutun kansion polku ja ohjelman nimi (kuva 12). Kansion olemassaolo ja polun oikeellisuus on syytä varmistaa. Tässä ohjelman asennuskansioksi valittiin /home/root ja nimeksi HelloTest.



Kuva 12. Eclipse IDE kuvakaappaus.

Eclipse IDE:ssä täytyy asettaa Debug-perspektiivi, kun ohjelma kysyy siihen siirtymistä. Mikäli se ei sitä automaattisesti kysy, niin tuo muutos tehdään itse avaamalla valikko ”Window > Perspective > Open Perspective > Other” ja valitsemalla avautuvasta näkymästä Debug ja Open. Sen jälkeen valikosta ”Run > Debug Configurations” valitaan Debug, jolloin ohjelma käännetään ja asennetaan Qemuun ssh-yhteyttä käyttäen. Sen jälkeen voi testiohjelmaa kokeilla joko isäntäjärjestelmän terminaalissa tai Qemussa suoraan. Tässä työssä ohjelmaa testattiin Qemun terminaalissa (kuva 13).



Kuva 13. Kuvakaappaus Qemu-terminaalista. HelloTest-ohjelma tulostaa ruudulle lauseen ”Hello World testi toimii”.

Edellä kuvattu ohjelmakoodin testaus suoraan Eclipsen Debug-komennolla vaatii monia asetuksia Eclipse IDE:ssä, mutta on toisaalta kätevä tapa, jos koodin toimivuutta halutaan nopeasti testata, eikä fyysistä laitetta ole saatavilla. Toinen tapa toteuttaa testaus olisi luoda projektista devtool-työkalulla (luvussa 4 tarkemmin) esim. hello-niminen resepti, joka käännetään ja asennetaan Qemuun seuraavilla komennoilla.

$ devtool add hello /home/rauno/eclipse-workspace/HelloWorld

$ devtool build hello

$ devtool deploy-target -c -s hello root@192.168.7.2

Tämä testaustapa ei ole yhtä nopea, sillä Eclipse IDE:stä käsin voi ohjelmaa testata helposti moneen kertaan ohjelmoinnin aikana valikosta ”Run > Run Configurations” komennolla Run, jolloin ohjelman tuotos tulostuu Eclipse IDE:n konsoliin. Näiden eri järjestelmien (Eclipse IDE ja Yocto SDK) yhteistoiminnassa tosin ajoittain esiintyy ongelmia, jotka mahdollisesti aiheutuvat Yocto-projektin nopeasta kehitysvauhdista.

4 OHJELMISTOTUOTANTO YOCTO SDK -KEHITYSYMPÄRISTÖSSÄ

4.1 Tarvittavat ohjelmat ja laitteet

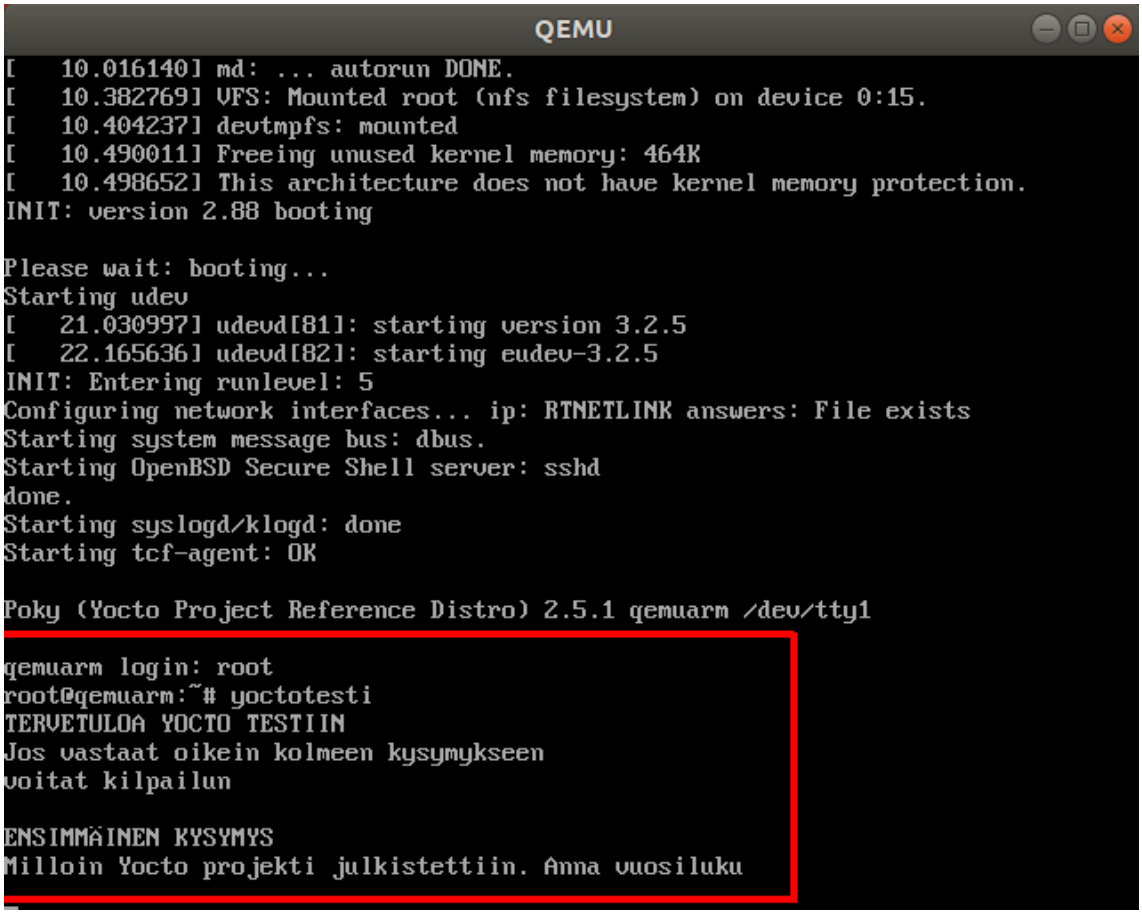
Tässä opinnäytetyössä käytettiin seuraavia ohjelma- tai järjestelmäversioita ja laitteita:

- PC-tietokone x86_64 prosessoriarkkitehtuurilla
- Oracle VirtualBox 5.2.28
- Linux Ubuntu 18.04.2
- Yocto Project (2.5.1, sumo)
- Eclipse IDE 2019-03 (4.11.0)
- Eclipse IDE:n Yocto-lisäosa (engl. plug-in) 2.6.1/oxygen
- Raspberry Pi 3 B+

Yocto-projekti ja Yocto SDK asennettiin samalle PC-tietokoneelle, mutta eri virtuaalilaitteille, joissa molemmissa käytettiin Ubuntu 18.04.2 käyttöjärjestelmää. Raspberry Pi -piirille asennettiin Linux-käyttöjärjestelmä luvussa 3.4.2 kuvatulla tavalla.

4.2 Esimerkkiohjelman koodaus ja toiminnan testaaminen

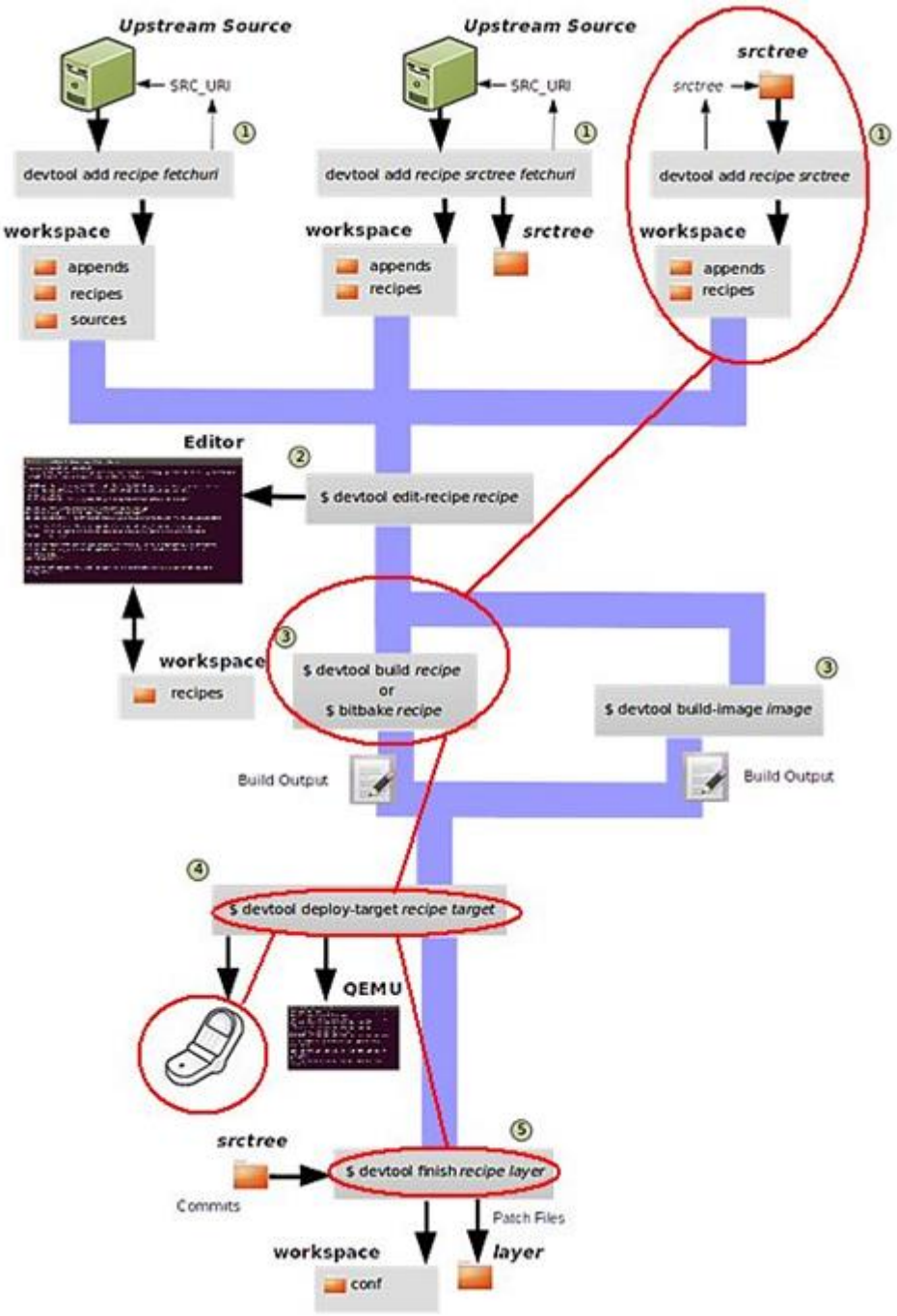
Järjestelmän toiminnan testaamiseksi koodattiin pieni ohjelma nimeltä Yoctotesti. Ohjelma on erittäin yksinkertainen tietovisa, joka tulostaa ruudulle kolme kysymystä tämän opinnäytetyön aiheista (kuva 14) ja laskee saadut pisteet. Käyttäen C++ -ohjelmointikieltä ja Eclipse IDE -työkalua luotiin ensin uusi projekti valikosta ”File > New > C/C++ Project > C++ Managed Build > Yocto Project SDK Autotools Project > Hello World C++ Autotools Projects". Tästä Hello World -malliprojektista muokattiin tietovisa, ja sitä testattiin Qemussa Eclipsen Debug-komennolla, samalla tavalla kuin Hello World -ohjelmaa kappaleessa 3.5.4. Ohjelma asennettiin Qemun kansioon /usr/bin, jolloin ohjelman voi suorittaa home-kansiosta ilman polkumäärittelyä. Eclipse IDE:n konsolille tulostui asennuksen etenemisen vaiheet ja lopuksi ilmoitus asennuksen onnistumisesta.



Kuva 14. Kuvakaappaus Qemun tty1-terminaalin näytöstä Yoctotesti-pelin testausvaiheessa. Tietovisan ensimmäinen kysymys on ruudulla.

4.3 Sovelluskehityksen työnkulku ja reseptin luominen

Kuvassa 15 on Yocto SDK -kehitysympäristön työnkulku, eli se miten ohjelmien kehitys etenee ja mitä vaiheita siinä on. Aluksi ohjelmakoodi tuodaan järjestelmään devtool add -komennolla, joka automaattisesti luo koodista valmiin reseptin. Resepti sisältää konfigurointeja, joiden pohjalta bitbake-ohjelma osaa koostaa määritellyn ohjelmapaketin (ks. luku 2.1.3). Kuvan yläosassa on esitetty kolme erilaista tapaa tuoda ohjelmakoodi. Ensimmäisessä mallissa (vasemmalla) ohjelma tuodaan verkosta (parametrinä vain URL-osoite) devtool-ohjelman oletustyökansioon (workspace/sources). Keskimmäisessä mallissa ohjelma ladataan verkosta erilliseen lähdekansioon (srctree), josta se noudetaan ja siitä luodaan resepti. Tässä työssä edettiin kolmannen vaihtoehdon mukaan, missä koodi tuodaan paikalliselta koneelta antamalla parametriksi polku (srctree) projektikansioon (tässä Eclipse IDE workspace).



Kuva 15. Yocto SDK -järjestelmän sovelluskehitysprosessin etenemismalli eli ”workflow”, jossa on monia eri toteutusvaihtoehtoja. Punainen viiva osoittaa miten tässä työssä edettiin. Ohjelma asennettiin Raspberry Pi -laitteelle (kohta 4) eikä Qemua enää tässä vaiheessa käytetty. Qemua käytettiin koodin työstämisvaiheessa testaukseen. (ks. luku 3.5.4) [16].

Devtool-työkalu on erityisen kätevä ja säästää paljon konfigurointiin kuluvaa aikaa. Kun Eclipse IDE:n projektikansiosta voidaan suoraan luoda resepti, se vähentää myös inhimillisen virheen mahdollisuuksia. Kun resepti on valmis, se käännetään kohdejär-

jestelmää varten toimivaksi ohjelmaksi. Tässä työssä resepti luotiin ja käännettiin seuraavilla komennoilla:

$ source environment-setup-cortexa7hf-neon-vfpv4-poky-linux-gnueabi

$ devtool add yoctotesti /home/rauno/eclipse-workspace/Yoctotesti

$ devtool build yoctotesti

4.4 Ohjelman asennus Raspberry Pi -laitteelle

Reseptin kääntämisen jälkeen ohjelma voidaan asentaa ja testata joko fyysisessä laitteessa tai myös uudelleen Qemussa (kuva 15 kohta 4). Tässä työssä ohjelma asennettiin Raspberry Pi -laitteelle.

$ devtool deploy-target -c -s yoctotesti root@192.168.111.101

Yllä oleva komento asentaa ohjelman kohdelaitteelle kansioon /usr/bin, joka on Linux -järjestelmien oletuskansio niille suoritettaville binääritiedostoille, jotka eivät ole keskeisiä järjestelmän toiminnan kannalta. Tällöin ohjelman voi suorittaa suoraan käyttäjän home-kansiosta ilman polkumäärittelyä [26]. Asennuksen aikana Raspberry Pi oli liitetty suoraan kannettavaan tietokoneeseen Ethernet-kaapelilla ja se sai IP-osoitteensa (*192.168.111.101)* tietokoneelle asennetulta DHCP-palvelimelta. Komennon parametri -c tarkoittaa, että ssh-yhteyden isäntäkoneen avaimen tarkistus poistetaan käytöstä, ja -s puolestaan tulee sanoista show-status, jolloin ohjelma tulostaa terminaaliin etenemisensä eri vaiheet. Kun ohjelma oli asennettu, sitä testattiin ssh-yhteyden kautta root-käyttäjän kotikansiossa komennolla Yoctotesti.

$ ssh root@192.168.111.101

$ Yoctotesti

Jos toiminnassa olisi jotain vialla, niin ohjelman reseptiä voidaan muokata komennolla devtool edit-recipe. Muuttujalla EXTRA_OECONF voidaan lisätä tai poistaa joitain ominaisuuksia, jonka jälkeen resepti käännetään ja testataan, poistuiko ongelma.

$ devtool edit-recipe yoctotesti

Ohjelman asennus voidaan poistaa kohdelaitteelta seuraavalla komennolla:

$ devtool undeploy-target yoctotesti

Jos myös koko resepti halutaan poistaa, on se syytä tehdä devtool reset -komennolla, eikä esim. poistaa sitä käsin workspace/recipes -kansiosta.

$ devtool reset yoctotesti

Kuvan 15 kaaviossa on esitetty myös vaihtoehtoinen tapa asennukselle. Mikäli luodaan kokonainen käyttöjärjestelmä kohdelaitteelle komennolla devtool build-image <image>, voidaan käyttöjärjestelmän mukana asentaa myös kaikki lisäohjelmat.

4.5 Ohjelmistotuotannon viimeistelyvaihe

Kun kehitetty sovellus on valmis asennettavaksi eri ohjelmistokokonaisuuksiin, on viimeisenä vaiheena julkaista se luomalla siitä Yocto-resepti devtool finish -komennolla [27]. Se on periaatteessa samanlainen kuin devtool add -komennolla luotu resepti (luku 4.3), mutta devtool finish -komennolla resepti liitetään Yocto-kerrokseen (engl. layer), joko valmiiseen tai samassa yhteydessä luotavaan uuteen kerrokseen. Kerrokset ovat ohjelmistotuotannon lopullisia varastointipaikkoja, mistä niitä voi myöhemmin liittää uusiin käyttöjärjestelmiin tai asentaa valmiisiin järjestelmiin. Kerroksia liitetään bitbake -layers add-layer -komennolla aina kulloinkin työn alla olevaan projektiin, ja lista aktiivisen projektin kerroksista muodostuu bblayers.conf -tiedostoon.

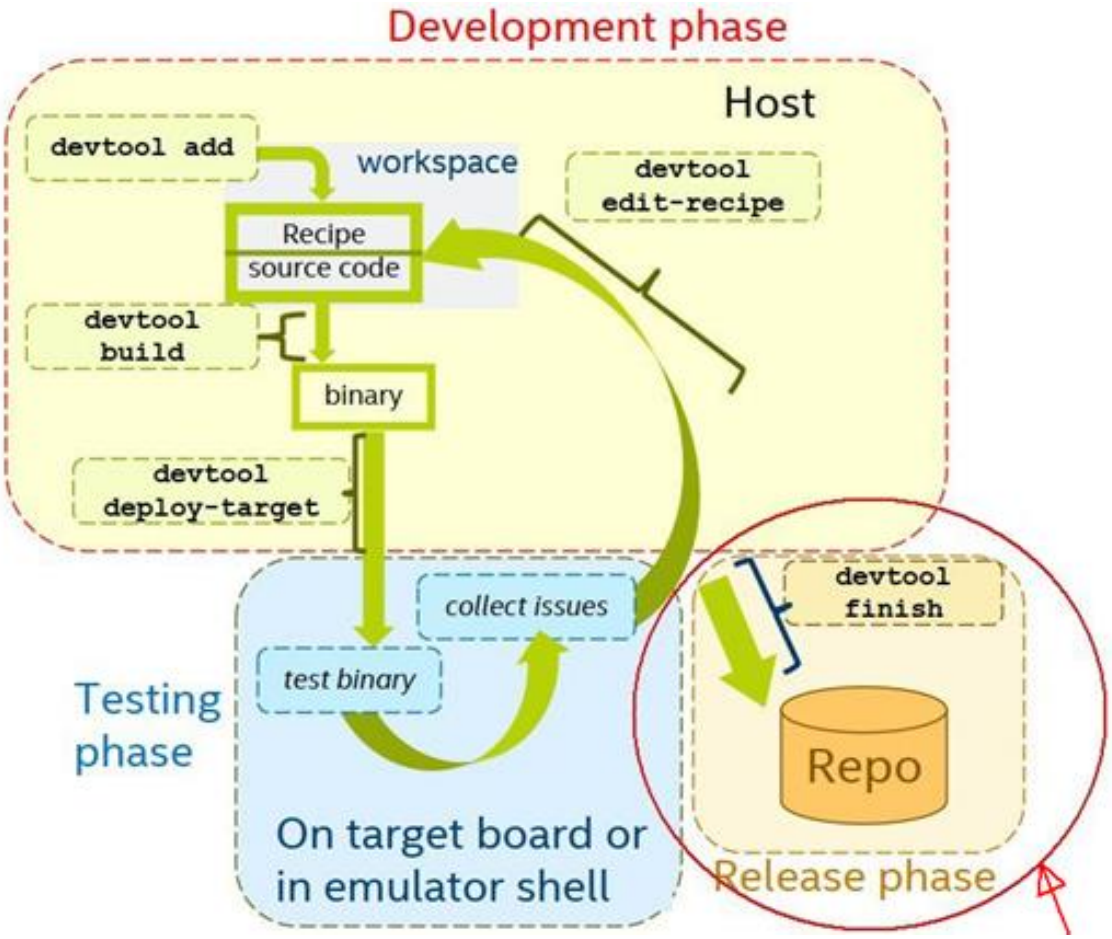
Ensin luodaan siis Yocto-kerros komennolla yocto-layer create <polku kerrokseen>, jonne resepti siirretään devtool finish -komennolla. Tässä työssä uusi kerros luotiin nimellä meta-yoctotesti, sillä vakiintuneen käytännön mukaisesti kerrokset nimetään meta-alkuisiksi [12]. Seuraavilla komennoilla luodaan kerros nimeltään meta-yoctotesti, lisätään sinne yoctotesti-resepti ja lopuksi kerros lisätään aktiiviseen projektiin.

$ layers/poky/scripts/yocto-layer create layers/meta-yoctotesti

$ devtool finish yoctotesti layers/meta-yoctotesti

$ layers/poky/bitbake/bin/bitbake-layers add-layer layers/meta-yoctotesti

Kuvassa 16 on esitelty havainnollisesti ohjelmistotuotannon päävaiheet, joista viimeisenä on varastoon (engl. repository) siirtäminen devtool finish -komennolla.



Kuva 16. Yocto SDK -sovelluskehityksen päälinjat. Viimeisenä vaiheena on valmiiden tuotosten julkaiseminen eli tallentaminen varastoon (engl. repository, ”Repo”) devtool finish -komennolla [27].

Edellä mainittua Yocto-projektin ohjeista poimittua komentosarjaa [20] testattaessa todettiin kuitenkin, ettei tarvittavaa yocto-layer-skriptiä löydy scripts-kansiosta. Syynä saattaa olla se, ettei sitä ole mukana kaikissa Yocto-versioissa. Niinpä kerroksen luomiseen käytettiinkin bitbake-komentoa. Koska bitbake käyttää eri ympäristömuuttujia kuin devtool, avattiin sitä varten oma terminaali, jossa suoritettiin seuraavat komennot:

$ source layers/poky/oe-init-build-env

$ bitbake-layers create-layer /home/rauno/poky_sdk/layers/meta-yoctotesti

Koska devtool finish -komento käyttää git-varastoa (engl. repository) korjausten (engl. patch) luomiseen [28], täytyi aiemmin luotu Eclipsen projektikansio muuttaa git -varastoksi. Git-komennot täytyi suorittaa komentorivillä Eclipsen projektikansiossa ja

devtool-komento poky_sdk -kansiossa. Seuraavilla komennoilla saatiin aikaan toivottu lopputulos:

$ git init

$ git config --global user.email rauno@example.com

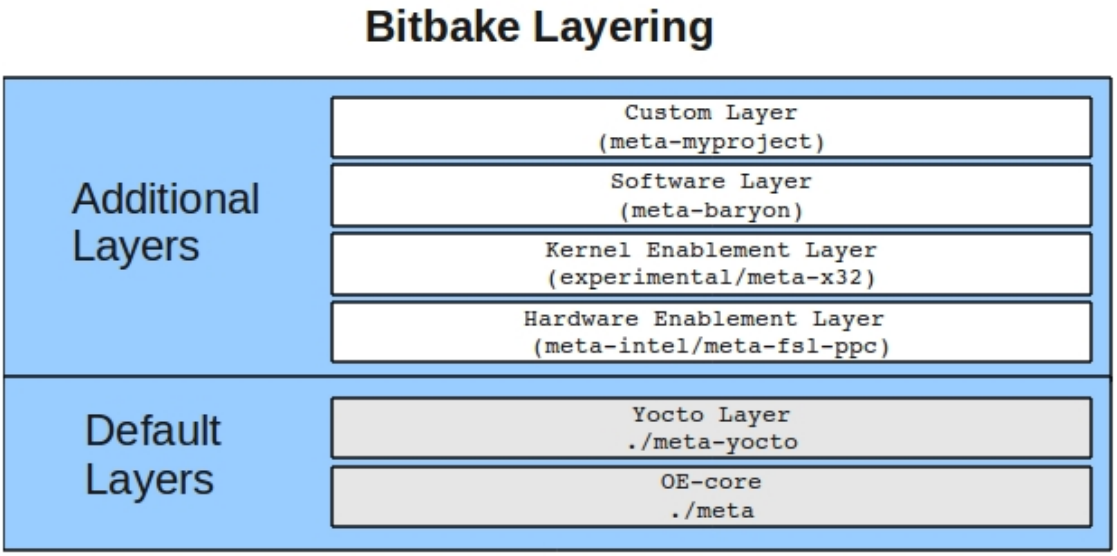
$ git config --global user.name rauno

$ git add .

$ git commit -m "yocto”

$ devtool finish yoctotesti layers/meta-yoctotesti

Kuvasta 17 voidaan nähdä millaisia kerroksia Yocto-projektissa on ja miten ne on jaoteltu. Alimpana on OE-core (OpenEmbedded-Core), joka sisältää laite- ja järjestelmäriippumatonta metatietoa ja on oletuksena mukana, kuten myös Yocto Layer.



Kuva 17. Yocto-projektin erilaisia kerroksia. Kaksi alinta ovat oletuksena mukana ja muodostavat perustan järjestelmälle. Niiden päällä on erikseen ladattavia laite- ja järjestelmäkohtaisia kerroksia ja ylimpänä käyttäjän itse luoma kerros, joita voi olla useitakin. Näistä valitaan haluttu kokoonpano kuhunkin projektiin lisäämällä niitä bitbake-layers add-layer -komennolla [29].

Seuraavana on laitekohtaisia kerroksia sekä ylimpänä käyttäjän itse luoma kerros. Yocto-kerroksia ja -reseptejä voidaan luoda tarvittava määrä ja jaotella ne parhaaksi katsotulla tavalla yhteensopiviksi kokonaisuuksiksi. Eri kerrokset ja niiden reseptit ovat siten käytettävissä moniin tarkoituksiin, joko erillisiin ohjelma-asennuksiin tai kokonaisten käyttöjärjestelmien luomiseen.

5 LOPPUPÄÄTELMÄ JÄRJESTELMÄN TOIMINNASTA

Tässä työssä oli tavoitteena perehtyä Yocto SDK -sovelluskehitysympäristöön, jonka avulla sulautettuihin järjestelmiin voidaan kehittää uusia sovelluksia tai täydellisiä Linux -käyttöjärjestelmiä ja jakeluversioita (engl. distro). Linux on vakiinnuttanut asemansa ja kasvattanut yhä suosiotaan paitsi tietotekniikassa laajemminkin, niin erityisesti sulautettujen järjestelmien alalla. Tässä työssä kohdelaitteena oli Raspberry Pi -piiri, joka monipuolisuutensa ja tehokkuutensa ansiosta on myös saavuttanut laajan suosion. Linux, Yocto SDK ja Raspberry Pi avaavat laajat mahdollisuudet luoda monia erilaisia toiminnallisia järjestelmiä erityisesti IoT-teknologiassa.

Yocto SDK:n asennus sujui täysin moitteettomasti, mutta syvällisempi perehtyminen eri työkaluihin ja niiden yhteistoimintaan vaatii ajoittain runsaasti aikaa ja konfigurointia. Tätä helpottavia työkaluja on kuitenkin kehitetty (mm. devtool), ja koko sulautettujen järjestelmien kehitysympäristö on muutenkin erittäin aktiivisen kehitystyön alaisena. Uusia ominaisuuksia luodaan säännöllisesti ja erilaisten laitearkkitehtuurien tuki on melko kattava.

Kaiken kaikkiaan voitiin todeta, että Yocto-projekti ja siihen liittyvä SDK -kehitysympäristö kaikkine työkaluineen on erittäin kätevä ja toimiva väline sulautettujen järjestelmien kehitykseen. Suhteellisen vähällä vaivalla pääsee jo alkuun hyvien ohjekirjojen ja malliesimerkkien avulla. Pidemmälle edetessä erilaisia uusia mahdollisuuksia avautuu ja järjestelmän voima on siinä, että kaikki kootaan palasista, joita voi yhdistellä mielensä mukaan. Järjestelmä on siis hyvin joustava ja antaa tilaa uusille innovaatioille. Tämän opinnäytetyön puitteissa ei ollut mahdollista perehtyä laajemmin eri ominaisuuksiin, mutta jo tämä lyhyt katsaus antoi kuvan hyvin toimivasta systeemistä.

Pieni haittapuoli tässä kehitystyössä ovat pitkät, joskus tuntejakin kestävät käännös- ja koontiprosessit. Mutta kun ne ajoittaa suunnitelmallisesti sopiviin hetkiin, jolloin voi tehdä jotain muuta, ei sekään aiheuta suurempaa ongelmaa. Tätäkin ongelmaa on helpotettu välimuistimekanismin (sstate cache) avulla. Edistyneille kehittäjille ja ammattilaisille järjestelmästä löytyy valtava määrä työkaluja ja mahdollisuuksia luoda vaikka kokonainen Linux-jakelu omine sovellusohjelmineen ja työpöytäympäristöineen. Tässä opinnäytetyössä pienen esimerkkiohjelman luominen ja asennus Yocto SDK -järjestelmällä Raspberry Pi -piirille sujui erinomaisesti.

LÄHTEET

[1] Baccelli, E. ym. RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT. IEEE Internet of Things Journal, 2018. Vol 5:6. S. 4428 - 4440. [Viitattu 14.5.2019]. DOI: 10.1109/JIOT.2018.2815038.

[2] Sethi, P. & Sarangi, S. Internet of Things: Architectures, Protocols, and Applications. Journal of Electrical and Computer Engineering 2017. 25 s. [Viitattu 14.5.2019]. https://doi.org/10.1155/2017/9324035.

[3] Chen, D. ym. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware. The Network and Distributed System Security Symposium, San Diego, 21-24.2.2016. 16 s. [Viitattu 14.5.2019]. https://www.dcddcc.com/docs/2016_paper_firmadyne.pdf.

[4] Hallinan, C. Embedded Linux primer. A practical real-world approach. 2nd ed., Lontoo, UK: Pearson Education, Inc., 2011.

[5] US 9,100,248 B2. Method and system for extending the capabilities of embedded devices through network clients. Vorne Industries Inc, Itasca IL, USA. (Vorne R, Saks B, Tang K) 2/151,229, 5.5.2008. julk. 13.11.2008. 28 s. [Viitattu 14.5.2019]. https://patentimages.storage.googleapis.com/a1/05/34/b11c8ae187ecb7/WO2008137117A2.pdf

[6] Petrikowski, N. Getting to Know the Raspberry Pi. New York, USA: The Rosen Publishing Group, Inc., 2015.

[7] Raspberry Pi Learning Resources. Raspberry Pi Foundation. [Viitattu 14.5.2019]. https://www.raspberrypi.org.

[8] Sforzin, A. ym. RPiDS: Raspberry Pi IDS — A Fruitful Intrusion Detection System for IoT; 2016 The 13th IEEE International Conference on Ubiquitous Intelligence and Computing. Toulouse, Ranska. 18-21.7.2016. S. 440-448. ISBN: 978-1-5090-2772-9. [Viitattu 14.5.2019]. DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0080.

[9] Rifenbark, S. Yocto Project Overview and Concepts Manual. 2019. [Viitattu 14.5.2019]. https://www.yoctoproject.org/docs/latest/overview-manual/overview-manual.html#cross-development-toolchain-generation.

[10] Texier, P. & Mabèacker, P. Yocto for Raspberry Pi: create unique and amazing projects by using the powerful combination of Yocto and Raspberry Pi. Birmingham, UK: Packt Publishing, 2016.

[11] Salvador, O. ym. Embedded Linux development with Yocto project: develop fascinating Linux-based projects using the groundbreaking Yocto project tools. Birmingham, England: Packt Publishing, 2014.

[12] Rifenbark S. Yocto Project Mega-Manual. 2019. [Viitattu 14.5.2019]. https://www.yoctoproject.org/docs/2.6.2/mega-manual/mega-manual.html.

[13] OpenEmbedded. Welcome to OpenEmbedded. 2017. [Viitattu 14.5.2019]. https://www.openembedded.org/wiki/Main_Page.

[14] Purdie, R. ym. Poky HandBook. 2011. [Viitattu 14.5.2019]. https://www.yoctoproject.org/docs/1.0/poky-ref-manual/poky-ref-manual.html.

[15] Poky. Embedded Linux Wiki. 2012. [Viitattu 14.5.2019]. https://elinux.org/Poky.

[16] Rifenbark, S. *Yocto Project Application Development and the Extensible Software Development Kit (eSDK). Manual for the 2.7 release.* 2019. [Viitattu 14.5.2019]. https://www.yoctoproject.org/docs/2.7/sdk-manual/sdk-manual.html.

[17] Robinson, A. ym. Raspberry Pi projects. 3. painos. John Wiley & Sons, Ltd. United Kingdom. 2014. ISBN 978-1-118-55543-9 (paperback), ISBN 978-1-118-55556-9 (ePub), ISBN 978-1-118-55553-8 (ePDF).

[18] Osborn S. Makers at Work: Folks Reinventing the World One Object or Idea at a Time. 1st ed. Apress, 2013. 324 s.

[19] Bruce, H. Yocto Project Extensible SDK: Simplifying the Workflow for Application Developers, 2017. [Viitattu 14.5.2019]. https://www.youtube.com/watch?v=d3xanDJuXRA.

[20] Application Development with Extensible SDK. Yocto Project Wiki, 2017. [Viitattu 14.5.2019]. https://wiki.yoctoproject.org/wiki/Application_Development_with_Extensible_SDK.

[21] Subhendra, Singh. Yocto Software Development Kit(SDK) Guide, 2017.[Viitattu 14.5.2019]. https://wiki.rdkcentral.com/display/RDK/Yocto+Software+Development+Kit%28SDK%29+Guide#YoctoSoftwareDevelopmentKit(SDK)Guide-3.UsingtheSDK.

[22] Rifenbark S. Yocto Project Reference Manual, 2019. [Viitattu 14.5.2019]. https://www.yoctoproject.org/docs/2.6.1/ref-manual/ref-manual.html#required-packages-for-the-build-host.

[23] Rifenbark, S. *Yocto Project Application Development and the Extensible Software Development Kit (eSDK). Manual for the 2.6.1 release*, 2019. [Viitattu 14.5.2019]. https://www.yoctoproject.org/docs/2.6.1/sdk-manual/sdk-manual.html#application-development-workflow-using-eclipse.

[24] Cookbook guide to Making an Eclipse Debug Capable Image. Yocto Project Wiki, 2017. [Viitattu 14.5.2019]. https://wiki.yoctoproject.org/wiki/TipsAndTricks/RunningEclipseAgainstBuiltImage

[25] Roche, D. ym. Ubuntu Make 2019. [Viitattu 14.5.2019]. https://github.com/ubuntu/ubuntu-make.

[26] Kuutti, W & Rantala, A. Linux. 3 laitos, 1. painos. Porvoo. WSOYpro/Docendo-tuotteet 2007. 382 s.

[27] Orling, T. Using Devtool to Streamline Your Yocto Project Workflow, 2017. [Viitattu 14.5.2019]. https://www.youtube.com/watch?v=CiD7rB35CRE.

[28] Rifenbark, S. Yocto Project Development Manual, 2017. [Viitattu 14.5.2019]. https://www.yoctoproject.org/docs/2.3/dev-manual/dev-manual.html#using-devtool-in-your-workflow.

[29] Flanagan E. The Yocto Project. [Viitattu 14.5.2019]. https://www.aosabook.org/en/yocto.html.