OULUN SEUDUN
AMMATTIKORKEAKOULU

Tuula Mikkola

SYSTEEMIMODUULIN VAIHE-EROMITTAUKSEN TESTAUKSEN KEHITTÄMINEN ROBOT FRAMEWORKILLÄ

SYSTEEMIMODUULIN VAIHE-EROMITTAUKSEN TESTAUKSEN KEHITTÄMINEN ROBOT FRAMEWORKILLÄ

Tuula Mikkola
Opinnäytetyö
Kevät 2012
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistojen kehitys

Tekijä: Tuula Mikkola
Opinnäytetyön nimi: Systeemimoduulin vaihe-eromittauksen kehittäminen Robot Frameworkillä
Työn ohjaaja(t): Eero Nousiainen
Työn valmistumislukukausi ja -vuosi: Kevät 2012 Sivumäärä: 62 + 4 liitettä

Opinnäytetyön aiheena oli tehdä Robot Frameworkillä testi Nokia Siemens Networks Oy:n Flexi-tukiaseman systeemimoduulille, jossa DACin avulla säädetään OCXOn taajuutta ja tutkitaan sen vaikutusta vaihe-eromittauksiin.

Opinnäytetyön testi tehtiin Robot Frameworkillä, joka on kehitetty hyväksyntätestaukseen ja hyväksyntätestauksen ohjauksen suunnitteluun. Testin aikana perehdyttiin tarkemmin testin vaihe-eromittaukseen, testiympäristöön ja systeemimoduulin toimintaan.

Työn tuloksena syntyi toimiva vaihe-eromittauksen tutkimiseen tarkoitettu testi, joka laitettiin Platform SW:n hyväksyntätestaukseen. Itse teoriaosuutta vaihe-eromittauksesta, systeemimoduulista ja testiympäristöstä tullaan käyttämään Platform SW:ssä tiedonlähteenä.

Asiasanat: Robot Framework, testaus, tukiasemat, vaihe-ero, Python

ABSTRACT

Oulu University of Applied Sciences
Information Technology and Telecommunications, Software development

---

Author: Tuula Mikkola
Title of thesis: System Module phase-difference measurement development to the Robot Framework
Supervisor: Eero Nousiainen
Term and year when the thesis was submitted: Pages: 62 + 4 appendices
Spring 2012

Topic of the thesis was to create test for Robot Framework for Nokia Siemens Networks' Flexi System Module. Test purpose was to control frequency of OCXO with DAC and to check effect to the phase-difference measurements.

Test case for the thesis was made with Robot Framework, which is developed for acceptance testing and designing of control of acceptance testing. During test development, the phase-difference measurements, test environment and behavior of System Module was studied more closely.

New test case for checking phase-difference measurements was completed as a result from the thesis work. Test case was taken into use to Platform SW acceptance testing. Theory part of phase-difference measurements, System Module and test environment will be used as information source in Platform SW.

Keywords: Robot Framework, base stations, testing, phase-difference, Python

ALKULAUSE

Haluan kiittää tämän opinnäytetyön loppuun saamisesta R&D Manager Teemu Uusimaata ja Senior Specialist Teemu Aholaa Nokia Siemens Networksiltä. Ilman heidän antamaansa mahdollisuutta ja Teemu Aholan kärsivällisyyttä ja neuvoja ei tämä työ olisi valmis. Kiitän myös ohjaavaa opettajaa lehtori Eero Nousiaista ja kielenohjaaja lehtori Tuula Hopeavuorta suuresta avusta. Haluan myös kiittää aviomiestäni aiheen opastuksesta ja kärsivällisyydestä.

Oulussa 20.4.2012

Tuula Mikkola

SISÄLLYS

LYHENTEET

API	Application Programming Interface, ohjelmointirajapinta, jolla voidaan ohjelmien välillä käsitellä pyyntöjä ja eri toimintoja
ASIC	Application Specific Integrated Circuit, sovelluskohtainen mikropiiri
BCN	BTS Clock Number, tukiaseman kellonumero
BTS	Base Transceiver Station, tukiasema
CCSSA	Clock, Control, System, Synchronization and Application, ohjelmalohko systeemimoduulissa
DAC	Digital-to-Analogue Converter, konvertteri digitaalisesta signaalista analogiseen signaaliin
DSP	Digital Signal Processing, digitaalisen signaalinprosessointiyksikkö
EAC	External Alarms and Controls interfaces, ulkoinen hälytys- ja ohjausliitäntä
ETH	Ethernet
FCLK	GSM Specific Frame Clock, GSM:n kehyksen kello
FCT	Flexi Control Multiplexing and Ethernet Transport, tukiaseman ohjausyksikkö
FRM	Flexi Radio Module, Flexi-radiomoduuli
FSM	Flexi System Module, Flexi-systeemimoduuli
FSP	Flexi Signal Processing, Flexi-signaaliprosessointiyksikkö, sisältää DSP-prosessorit
GPS	Global Positioning System, globaali paikannusjärjestelmä

HTML	HyperText Markup Language, kuvauskieli jolla voidaan kuvata hyperlinkkejä sisältävää tekstiä eli hypertekstiä. Käytetään HTML-sivujen tekemisessä.
IQ -Data	In-phase Quadrature Data, lähetettävä signaali jaetaan modulaattorissa I (In phase)- ja Q (Quadrature) -komponentteihin
LTE	Long Term Evolution, matkapuhelinverkkotekniikka
MCU	Microcontroller Unit, mikroprosessori
MSC	Message Sequence Chart, sanomasekvenssikaavio. Kaavion avulla voidaan kertoa sanomien väliset yhteydet.
MURKKU2	Base Band -väylän lohko, joka tarjoaa summaus- ja multipleksaustoiminnallisuudet (ASIC RP3-01)
OBSAI	Open Base Station Architecture Interface, tukiaseman avoin arkkitehtuurirajapinta
OCXO	Oven Controlled Crystal Oscillator, uuniohjattu kristallioskillaattori
PPS	Pulse per Second, pulsseja sekunnissa
reST	RestructuredText, tieto on jäsennelty yhtäsuuruusmerkkien ja välilyöntien avulla
RIDE	Robot Interactive Development Environment, Robot Frameworkille suunnattu testien editointiohjelma
RP3-01	OBSAI Reference Points 3 (reference interface), rajapinta yksiköiden välillä
SIC	System Internal Communication, systeemin sisäinen yhteydenpito
SICAP	SIC Application Protocol, ohjelmistoprotokolla SIC
SM	System Module, Flexi-tukiaseman systeemimoduuli

SRIO	Serial Rapid IO, nopea sarjaväyläinen liitäntä. Mahdollistetaan optiset liitännät FSM-moduuleiden välillä
TSV	Tab Separated Values, tiedot on eroteltu tiedostossa tabulaattorilla
UMTS	Universal Mobile Telecommunications System, kolmannen sukupolven matkapuhelinteknologia
WCDMA	Wideband Code Division Multiple Access, UMTS-verkoissa käytettävä radiorajapinta

1 JOHDANTO

Opinnäytetyössä lähdettiin kehittämään testiä Robot Frameworkillä Nokia Siemens Networks Oy:n LTE (Long Term Evolution) -Flexi-tukiasemassa olevalle systeemimoduulin MCU:lle (Micro Controller Unit). Opinnäytetyön idea tuli harjoitteluprojekteista, jotka suoritin Nokia Siemens Networksillä.

Nokia Siemens Networks on perustettu vuonna 2007, jolloin Nokia Networks ja Siemensin COM-divisioona (ilman Enterprise Business -yksikköä) yhdistyivät. Yritys on yksi maailman johtavista kommunikaatiopalvelujen rakentajista. Nokia Siemens Networks tarjoaa kiinteitä ja langattomia verkkoratkaisuja ja palveluita. (Nokia Siemens Networks.)

Oulun verkkoliiketoiminnan tuotekehitysyksikkö on ollut toiminnassa jo vuodesta 1975 lähtien, ja tällä hetkellä siellä työskentelee yli 2000 henkilöä. Oulussa tutkitaan ja kehitetään tukiasema-, Media Gateway- ja radioverkkojen tuotteita sekä teknologioita. (Nokia Siemens Networks.)

Harjoitteluprojektien aikana pääsin tutustumaan Robot Frameworkiin ja testien tekemiseen, jolloin itse opinnäytetyö oli luontevinta tehdä samalle alueelle. Syksyllä 2011 aloitin opinnäytetyön testin rakentamisen Robot Frameworkillä. Testin aiheena oli vaihe-eromittaus Flexi-tukiaseman systeemimoduulista. Testissä säädettiin OCXOn (Oven Controlled Crystal Oscillator) taajuutta DACin (Digital-to-Analogue Converter) avulla ja tutkittiin sen vaikutusta vaihe-eromittauksiin. Valmis testi laitettiin Platform SW:n hyväksyntätestaukseen.

Testin lisäksi opinnäytetyössä perehdyttiin tarkemmin vaihe-eromittaukseen ja itse testausympäristöön, missä testiä ajettiin. Robot Frameworkin käytöstä ja teoriasta kerrotaan opinnäytetyön alussa. Työn loppuosassa esitellään testin rakentaminen Robot Frameworkillä. Systeemimoduuliin ja sen osiin perehdytään teorian merkeissä. Vaihe-eromittauksista kerrotaan testin rakentamisen aikana tarkemmin, samoin vaihe-eron näkymisestä testissä.

2 FLEXI-TUKIASEMA

Flexi-tukiasema (kuva 1) koostuu useammasta moduulista (mm. Flexi-radiomoduuli ja Flexi-systeemimoduuli), joista saadaan rakennettua halutunlainen kokonaisuus. Flexi-tukiasema on ohjelmistopohjainen tukiasema, joka tukee 3GPP (3rd Generation Partnership Project) -teknologioita (GSM, WCDMA ja LTE). Flexi-radiomoduulista (FRM) on lisätietoa alla ja Flexi-systeemimoduuliin (FSM) perehdytään tarkemmin luvussa 6.1.

KUVA 1. Flexi-tukiasema (Nokia Siemens Networks. 2012)

FRM (Flexi Radio Module) on oma osansa Flexi-tukiaseman kokonaisuudessa. Se toimii lähetin-vastaanotinmoduulina, johon on integroitu antennisuodatin. Flexi-radiomoduuli pystyy tukemaan yhdestä kolmeen Flexi WCDMA (Wideband Code Division Multiple Access) -tukiaseman sektoria. FRM ohjaa radiotaajuutta ja myös antaa virran Flexin antennitulolle. FRM:ssä suoritetaan osa radioliikenteeseen liittyvien signaalien prosessoinnista. Varsinainen signaalin käsittely tehdään FSP:ssä (Flexi-signaaliprosessointiyksikkö). (Flexi WCDMA BTS RF Module and Remote Radio Head description. 2009, 8.)

Seuraavissa luvuissa kerrotaan yleisesti tukiasemasta ja matkapuhelinverkkotekniikoista, joita Flexi-tukiasema tukee.

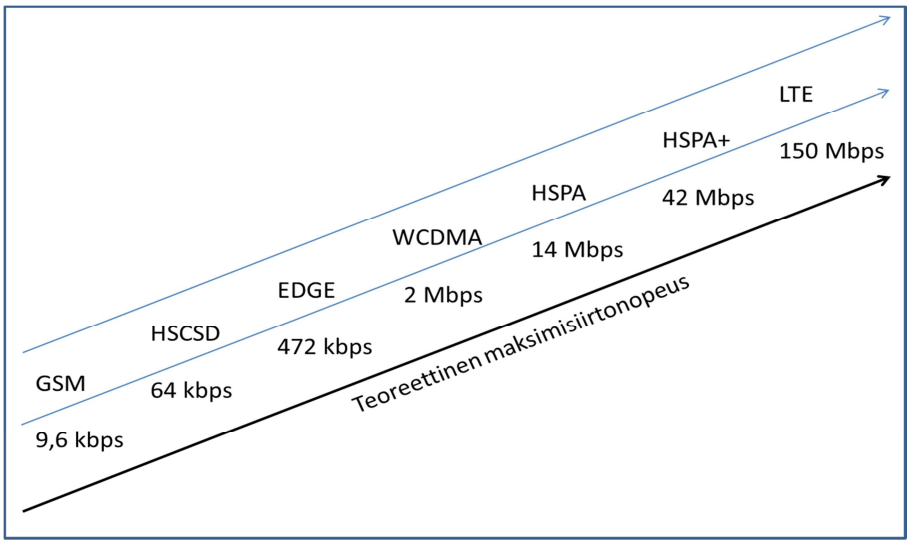
2.1 Tukiasema

Tukiasema on matkapuhelinverkon solmukohta. Tukiasema vastaanottaa matkapuhelimelta tulevat radioaallot, jotka se lähettää eteenpäin matkapuhelinkeskukseen. Matkapuhelinkeskus välittää puhelun lähimpään tukiasemaan tai kiinteän puhelinverkon vastaanottajalle. (Tukiasemat ovat matkapuhelinverkon solmukohtia. 2012.)

Tukiasemien sijoittaminen perustuu radioverkkosuunnitteluun. Tukiasemalla on tietyn kokoinen peittoalue ja vain rajattu määrä kapasiteettia vastaanottamiseen ja lähettämiseen. Hyödyntämällä radiotaajuusalue mahdollisimman tehokkaasti pystytään lähettämisessä ja vastaanottamisessa käyttämään alhaisia tehotasoja. Kun käytetään alhaista tehotasoa, myös häiriötasot radioverkossa vähenevät. Tukiasemien määrä maastossa määräytyy teknologian ja matkapuhelimia käyttävien ihmisten lukumäärän mukaan. (Tukiasemat ovat matkapuhelinverkon solmukohtia. 2012.)

2.2 Matkapuhelinverkkotekniikat

Flexi-tukiasema tukee GSM (Global System for Mobile Communications, globaali matkapuhelinjärjestelmä)-, WCDMA- ja LTE (Long Term Evolution) -matkapuhelinverkkotekniikoita eli tunnetuimmin 2G, 3G ja 4G. Kuvasta 2 nähdään siirtonopeuksien kehitys GSM:stä LTE:hen.



KUVA 2. GSM:stä LTE:hen siirtonopeuksien kehitys (Holma - Toskala 2009, 8)

GSM on suunniteltu toisen sukupolven matkapuhelintekniikaksi (2G). GSM käyttää digitaalista TDMA:ta (Time Division Multiple Access, aikajakokanavointi) ja FDMA:ta (Frequency Division Multiple Access, taajuusjakokanavointi). Käytettäessä TDMA-tekniikkaa voidaan enemmän käyttäjiä sisällyttää vapaana olevaan kaistanleveyteen. GSM-tekniikka oli ensimmäinen, jolla saatiin tukiaseman ja matkapuhelimen välinen ilmarajapinta salaiseksi, jolloin toisten puheluja ei voitu enää kuunnella. (Poole a.)

GSM käyttää kaksisuuntaista tekniikkaa eli FDD:tä (Frequency Division Duplex, taajuusjakokanavointi). Lähetystaajuusalue on 933–960 MHz, yleensä pelkästään 900 MHz. Tiedon lataamisen lähetystaajuus on 890–915 MHz, yleensä pelkästään 900 MHz. Kanavan väli on 200 kHz ja kehyksen (frame) kesto on 4,615 ms. (Poole a.)

WCDMA on UMTS-verkoissa (Universal Mobile Telecommunications System) käytettävä radioteknologia. WCDMA tukee sekä FDD- että TDD-tekniikkaa (Time Division Duplex, aikajakokanavointi).

FDD-tekniikassa varataan omat taajuusalueet lähetykseen ja vastaanottoon. Ylävirralle eli paluusuunnalle 1920–1980 MHz taajuudet ja alavirralle eli tulosuunnalle 2110–2170 MHz. Tämän myötä on samanaikaisesti käytössä noin 250 puhekanavaa. (Granlund 2001, 213.)

TDD-tekniikassa sekä ylä- että alavirtaan on käytössä sama taajuusalue, jossa toimitaan vuorosuuntaisesti. Taajuusalueeksi on varattu 1900–1920 MHz sekä 2020–2025 MHz. TDD mahdollistaa 120 yhteyttä samanaikaisesti. (Granlund 2001, 215.)

LTE on 4G-tekniikkaa. LTE käyttää kahta teknologiaa, MIMOa (Multiple Input Multiple Output) ja OFDM:ää (Orthologonal Frequency Division Multiple Access).

MIMO on radioteknologia, joka käyttää datan kuljettamiseen tukiasemasta päätelaitteeseen useita radioteitä. MIMO-tekniikassa signaali lähetetään kahdesta tai useammasta eri antennista erilaisten datavirtojen avulla. Vastaanottimessa on yksi tai useampi antenni. Tullut signaali käsitellään MIMO-dekoodauksella. Tämän avulla saadaan kasvatettua kapasiteettia kanavalla. (Holma - Toskala 2009, 80.)

OFDM on tiedon lähettämistä tukiasemasta päätelaitteeseen, jossa tiedon siirrossa käytetään taajuuskanavia yhtä aikaa. Taajuuskanavat eivät häiritse toisiaan, koska ne ovat ortogonaalisesti toisiinsa nähden. Ortogonaalisuudella tarkoitetaan sitä, että yhden kantoaallon ollessa keskikohdassa ovat muut kantoaallot nollassa. Päätelaitteesta tukiasemalle käytetään SC-FDMA (Single Carrier - Frequency Division Multiple Access) -tekniikkaa. (Poole c.)

3 OHJELMISTOJEN TESTAUS

Testaus on ohjelmistoprojektin tärkeimpiä osia, johon voidaan saada kulumaan yli puolet projektin resursseista. Hyvällä testauksella pystytään virheet minimoimaan, mutta sillä ei voida kuitenkaan todeta ohjelmaa virheettömäksi. (Haikala − Märijärvi 2006, 284-286.)

Testauksessa esille tulevat virheet ovat poikkeamia spesifikaatiosta. Ilman spesifikaatiota ei voida olla varmoja, miten testin pitäisi oikein toimia. Yleensä spesifikaatioiksi voidaan luokitella toiminnallinen määrittely ja tekninen määrittely. (Haikala − Märijärvi 2006, 287.)

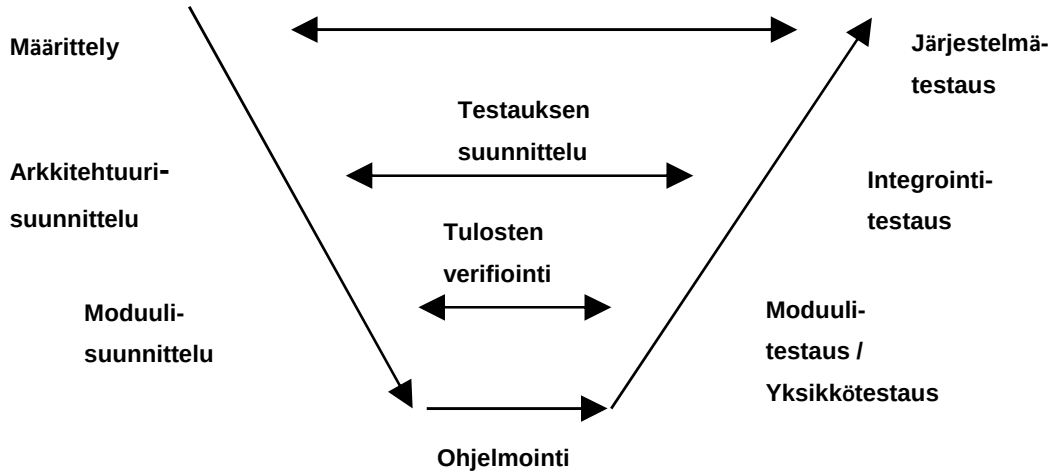
Perinteisesti testauksen kulku menee esimerkiksi V-mallin tai vesiputousmallin mukaisesti. Luvussa 3.1.1 kerrotaan tarkemmin V-mallista. Nykyään ohjelmistoprojektien testaukset yritetään saada menemään ketterien menetelmien mukaisesti (Agile) (luku 3.2.1). Ketterien menetelmien avulla testausta tehdään jokaisen pienen muutoksen jälkeen, jolloin saadaan toimivaa ohjelmistokoodia nopeampaa tahtia.

3.1 Testauksen vaiheet

Alla tullaan kuvaamaan testausvaiheet (moduulitestaus, integraatiotestaus, järjestelmätestaus ja hyväksyntätestaus) ja testauksen V-malli.

3.1.1 Testauksen V-malli

Testaus suoritetaan yleensä V-mallin mukaisesti (kuva 3). V-malliin kuuluu moduulitestaus (module testing), yksikkötestaus (unit testing), integrointitestaus (integration testing) ja järjestelmätestaus (system testing). Järjestelmätestausta voi seurata myös erillinen hyväksyntätestaus (acceptance testing). Nämä testausvaiheet tehdään perinteisesti V-mallin oikeassa haarassa. (Haikala − Märijärvi 2006, 288.)



KUVA 3. Testauksen V-malli (Haikala - Märijärvi 2006)

Havaittujen virheiden korjaukset tulevat sitä kalliimmaksi, mitä myöhemmin ja ylempänä V-mallissa ne havaitaan. Olisikin hyvä, että uudelleentestaus eli regressiotestaus olisi automatisoitu. (Haikala − Märijärvi 2006, 290.)

Testauksen määrää on vaikea arvioida. Järjestelmätestaustakin voidaan jatkaa ”kunnes aika ja/tai rahat loppuvat”. Olisikin hyvä asettaa testauksen lopettamiselle hyväksymiskriteerit, jotka määriteltäisiin testaussuunnitelmassa. (Haikala − Märijärvi 2006, 293.)

3.1.2 Moduulitestaus

Moduulitestauksessa testataan pienempiä ohjelmayksiköitä, esimerkiksi funktioita ja niistä muodostuvia yksittäisiä moduuleita. Testauksen suorittaa moduulin toteuttaja ja testin tarkoituksena on löytää selvät virheet ja korjata ne. Moduulitestauksessa verrataan moduulin toimintaa tekniseen määrittelydokumenttiin. Testattavien moduulien määrittelyjen ja toiminnan välillä yritetään löytää moduulitestin avulla ristiriidat. (Software Business Competence.)

Moduulitestauksen toimivuutta voidaan testata erillisellä testipedillä (test bed). Testipeti toteutetaan itse ja siihen voi kuulua ohjelman ympäristöä simuloivia

osia, testiajureita ja tynkämoduuleita (test stubs). Testiajureilla saadaan moduulit toteutettua, palveluita kutsuttua ja tuloksia tarkasteltua. Tynkämoduuleilla korvataan testattavan moduulin tarvitsemat muut moduulit, jos niitä ei ole olemassa. (Haikala − Märijärvi 2006, 289.)

3.1.3 Integrointitestaus

Integrointitestauksessa on tarkoituksena yhdistää moduuleita tai moduuliryhmiä (osajärjestelmiä), jolloin saadaan tutkittua moduulien välisten rajapintojen toimivuutta (Haikala − Märijärvi 2006, 290).

Integrointitestausta on tarpeetonta suorittaa erillään moduulitestistä ja yleensä se eteneekin rinnan moduulitestauksen kanssa. Testi etenee kokoavasti (bottom up) alimman tason moduuleista ylöspäin. Jäsentävässä eli osittaisessa (top down) testauksessa etenemissuunta on päinvastainen. (Haikala − Märijärvi 2006, 290.)

3.1.4 Järjestelmätestaus

Järjestelmätestausta voidaan kutsua myös systeemitestaukseksi. Testauksen kohteena on koko järjestelmä ja tuloksia verrataan määrittelyvaiheen dokumenttiin. (Software Business Competence.) Osaksi järjestelmätestausta voi liittyä myös kenttätestaus (field testing) tai hyväksyntätestaus (acceptance testing) (Haikala − Märijärvi 2006, 290).

Järjestelmätestaukseen kuuluu myös järjestelmän ei-toiminnallisten ominaisuuksien testaaminen. Silloin käytössä ovat esimerkiksi kuormitustestit, luotettavuustestit, asennustestit, käytettävyystestit jne. (Haikala − Märijärvi 2006, 290.)

3.1.5 Hyväksyntätestaus

Hyväksyntätestaus on mustalaatikkotestausta, joka suorittaa testit ohjelmistolle annettuja vaatimuksia vasten. Vaatimukset ovat ne, mitä asiakas on tilannut. Hyväksyntätestaus tehdään kaikkein viimeisimpänä testin vaiheena.

Mustalaatikkotestaus tarkoittaa sitä, että testaaja käyttää määriteltyjä syötearvoja ja vertaa, että ulostulosta tulevat vastaukset ovat ne mitä haluttiin. (Wilborn 2012.)

Hyväksyntätestauksen testitapaukset on yksilöllisesti dokumentoitu testisuunnitelmiin. Jokainen testitapahtuma sisältää lähetettävän tiedon ja ennalta määrätyn lopputuloksen. Jos testin ulostulo on odotettu, on testi tällöin onnistunut. Testien tulokset ovat joko ”pass” tai ”fail” eli onnistunut tai epäonnistunut. Jotkut testit luokitellaan todella tärkeiksi ja osa taas ei niin tärkeiksi. (Wilborn 2012.)

3.2 Testausmenetelmät

Seuraavissa alaluvuissa tullaan kertomaan ketterästä menetelmästä ja TDD-tekniikasta (Test-driven development). Ketterissä menetelmissä järjestelmän koon mukaan voidaan soveltaa ketteriä menetelmiä kaikkiin vaiheisiin, esimerkiksi järjestelmä- ja hyväksyntätesteissä. TDD-tekniikkaa voidaan käyttää heti alusta asti, vaikka kaikki kokonaisuudet eivät olisi valmiita. Sopivasti tehty toiminnallinen runko antaa jo varhaisessa vaiheessa käsityksen, onko alkupään määrittelyt tehty oikein.

3.2.1 Ketterät menetelmät

Ketterät menetelmät (agile software development) ovat uusi tapa pitää yllä ohjelmistoprojekteja. Ketterissä menetelmissä ei yritetä heti saada kaikkia ohjelman vaatimuksia kasaan. Yleensä vaatimukset tarkentuvat ketterissä menetelmissä projektin kuluessa ja vaatimusten saamiseksi suositaan käyttäjätarinoita. Käyttäjätarinat kertovat, miksi järjestelmä on tärkeä, mutta eivät sitä, miten halutaan järjestelmän toimivan. (Tolvanen 2011b.)

Yleensä testaus on kaikkein viimeisin osa ohjelmistoprojekteissa ja tämä asia on korjattu ketterissä menetelmissä. Ketterissä menetelmissä projekti jaetaan mahdollisimman pieniin osiin ja testataan aina osa kerrallaan. Projektin paloitteleminen pieniin osiin mahdollistaa sen, että aina on jotain valmista

koodia testattuna, koska tällöin on käyty projektin tarvittavat vaiheet monta kertaa läpi. (Tolvanen 2011b.)

Yleisimpiä ketteriä käytänteitä ovat lyhyet päiväpalaverit, joissa jokainen ryhmän jäsen kertoo tilannekatsauksen omista töistään. Myös jatkuva integraatio on yksi menetelmä, jossa ajetaan kehitystä, että ohjelmisto on julkaistavissa nopeasti. Jatkuvassa integraatiossa projekti etenee reaaliaikaisesti eli pieniin muutoksiin reagoidaan nopeasti. Yleensä integrointi on tehty mahdollisimman automaattiseksi. (Tolvanen 2011a.)

3.2.2 Test-driven development

TDD on testivetoista kehitystä. Tämä tarkoittaa sitä, että ohjelmiston tekeminen etenee testien mukaan. Ensin tehdään testit ja sen jälkeen siihen sopiva ohjelmakoodi. Yleensä nämä testit ovat yksikkötestauksia ja niin tarkoin määriteltyjä, että ohjelman toiminta saadaan niistä selville. (Tolvanen 2011a.)

4 ROBOT FRAMEWORK -TESTIAUTOMAATIOTYÖKALU

Robot Framework on avainsanapohjainen testiautomaatiotyökalu hyväksyntätason testaukseen. Sen kehitti Nokia Siemens Networksin Pekka Klärck vuonna 2005. Robot Framework on Apache-lisenssin alainen ja vuodesta 2008 lähtien lähdekoodi on ollut avointa kaikille. (Larman − Vodde 2010, 4.)

Robot Framework on rakennettu Python-ohjelmointikielellä ja ominaisuuksia voidaan laajentaa käyttäjän itse tekemillä Python- tai Java-testikirjastoilla. Käytettäessä Java-testikirjastoja ajetaan Robot Framework Jythonilla (Python Java-ympäristölle). Aito Python-koodi toimii sekä Pythonilla että Jythonilla, kunhan kaikki koodissa käytetyt moduulit ja metodit ovat saatavilla myös Jythonilla. (Robot Framework User Guide version 2.6.2. 2011.)

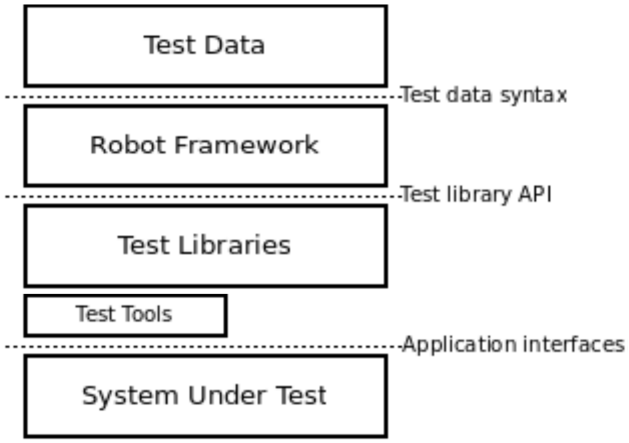
Testit käynnistetään joko pybot- tai jybot-komennolla. Nämä ovat toiminnaltaan samanlaisia, ainoa ero on, että jybot ajetaan kun kyseessä on Java-testikirjastot. Python on nopeampi kuin Jython, jolloin on parempi käyttää pybotia, jos vain mahdollista. (Robot Framework User Guide version 2.6.2. 2011.)

4.1 Testien rakenne Robot Frameworkissä

Testit rakennetaan taulukkomaisesti, yleensä HTML (HyperText Markup Language)- ja tekstimuodossa. Muita tuettuja muotoja ovat TSV (Tab Separated Values)- ja reST-formaatit (reStructuredText). HTML-muodossa Robot Framework käyttää vain taulukoita eikä välitä ylimääräisistä teksteistä, joita voidaan käyttää dokumentoinnissa. (Larman − Vodde 2010, 6.)

Robot Frameworkissä käytetään korkean tason arkkitehtuuria (kuva 4). Ylimmällä tasolla (Test Data) on itse kehitetty testi, esimerkiksi HTML-muodossa tehtynä. Robot Framework käsittelee tehdyn testin ja suorittaa siihen toteutetut testitapahtumat. Suorituksen jälkeen Robot Framework tekee lokit ja raportit testituloksista. Robot Frameworkin ja testattavan järjestelmän (System

Under Test) välissä oleva testikirjasto (Test Libraries) hoitaa kommunikoinnin testattavan järjestelmän ja testitapahtumien välillä. Testattavaa järjestelmää voidaan käyttää suoraan testikirjastojen kautta tai alemman tason testiajureiden (Test Tools) kanssa. (Robot Framework User Guide version 2.6.2. 2011.)



KUVA 4. Robot Frameworkin arkkitehtuuri (Robot Framework User Guide version 2.6.2. 2011)

Robot Frameworkillä tehdyt testit sisältävät asetukset (settings), muuttujat (variables) ja avainsanat (keywords). Asetuksissa määritellään testin paikalliset asetukset, testikirjastot, vakiomuuttujat yms. Muuttujiin saadaan tallennettua vakiotietoa, jota käytetään useasti testin aikana. Käytettäessä valmista muuttujaa voidaan sitä käyttää testissä missä tahansa ja se on helposti muutettavissa missä tahansa testin vaiheessa. Avainsanat määrittelevät muut testitapaukset, jotka suoritetaan jossain vaiheessa testitapahtuman aikana. Testin avainsanat voivat olla määriteltyinä joko itse testitapahtumassa tai testikirjastoissa. (Adzic 2009.)

Taulukosta 1 nähdään Robot Frameworkin asetuksista esimerkki. *Documentation*-kenttään voidaan laittaa tietoa testistä ja *Metadata*-kenttään laitetaan esimerkiksi versiotiedot. *Metadata*-kentässä pystytään kertomaan myös dokumentin nimi, jossa testitapaus on kuvattu, ja muitakin tietoja testistä, joita halutaan tallentaa testin mukana.

TAULUKKO 1. Asetuksien periaate

Settings	Value	Value
Documentation	Tässä testissä verrataan muuttujia	
Library	laskutoimitukset.py	
Variables	muuttujiennimet.py	
Metadata	version	0.1
Resource	resource/resourceFile.html	

Avainsanan (*keyword*) sisältöä nähdään taulukossa 2. *Keyword*-solun alla nähdään avainsanan nimi, jolla sitä testissä kutsutaan. *Action*-solusta nähdään toiminto, mitä avainsanassa tullaan tekemään. Esimerkkikuvassa käytetään actionina Robot Frameworkin omaa avainsanaa *Should Be Equal*. Kyseinen *Should Be Equal* -avainsana tarkistaa, onko VALUE-muuttujan arvo 1234.

TAULUKKO 2. Avainsanan (keyword) sisältöä

Keyword	Action	Argument	Argument
Testing parameters	Should Be Equal	${VALUE}	1234

Taulukosta 3 nähdään arvon asettamisen muuttujalle. Sille voidaan antaa numero- tai tekstiarvo. Muuttujille ei tarvitse erikseen kertoa, mitä tyyppiä ne ovat. Alustettaessa muuttujat tietyllä arvolla osaa Robot Framework tunnistaa, mitä tyyppiä muuttujat ovat. Testin aikana muuttujien arvoja voidaan muuttaa, jos on tarvetta.

TAULUKKO 3. Skalaarimuuttujaesimerkki

Variable	Value	Value
${VALUE}	1234	
${PATH}	../k/t	

Robot Frameworkissä muuttujat voivat olla joko paikallisia tai globaaleja. Paikallinen tai globaali muuttuja saadaan tehtyä erillisten avainsanojen avulla. Esimerkiksi täysin globaali muuttuja, jota voidaan käyttää kaikissa testeissä ja

Test Suiteissa, saadaan aikaiseksi Set Global Variable -avainsanalla. Tätä globaalia muuttujaa kannattaa kuitenkin käyttää harkiten. Parempi vaihtoehto globaalille muuttujalle on paikallinen muuttuja, jota voidaan käyttää suorituksessa olevassa Test Suitessa. Tämä muuttuja luodaan *Set Suite Variable* -avainsanalla tai *Set Test Variable* -avainsanalla. *Set Variable* -avainsanalla saadaan luotua paikallinen muuttuja, jota ei voida käyttää missään muussa testin vaiheessa. (Robot Framework User Guide version 2.6.2. 2011.)

Taulukossa 4 nähdään, mistä Test Case -osio koostuu. *Test Case* -sarakkeen alla nähdään testitapausten nimet, jotka ajetaan esimerkissä. *Actionin* alla nähdään toiminnot, jotka tehdään testitapauksissa. *Taking parameters* -kohta käyttää taulukon 2 *Testing parameters* -avainsanaa. *Taking parameters* -kohta ei tee mitään muuta kuin ajaa *Testing parameters* -avainsanan. *Add Parameters* -kohdassa nähdään *Actionissa* TimeOut, joka määrää, että Test Case täytyy suorittaa kymmenessä sekunnissa tai muuten testi epäonnistuu. *Set Suite Variable* -avainsana on Robot Frameworkin oma ja sillä luodaan muuttuja tai laitetaan uusi arvo jo olemassa olevaan muuttujaan. Tämä mahdollistaa muuttujan käyttämisen missä tahansa Test Suiten aikana. Arvon muuttamisen jälkeen ajetaan jälleen itse tehty *Testing parameters* -avainsana.

TAULUKKO 4. Test Casen periaate

Test Case	Action	Argument	Argument
Taking parameters	[Documentation]	Taking parameter and testing it.	
	Testing parameters		
Add Parameters	[Documentation]	Add new parameter and test it.	
	[TimeOut]	10 seconds	
	Set Suite Variable	/${VALUE}	3456
	Testing parameters		

4.2 Robot Frameworkin API-testikirjastot

Robot Frameworkissä on kolme erilaista API-testikirjastoa: Static (staattinen), Dynamic (dynaaminen) ja Hybrid. API (Application programming interface) on

ohjelmointirajapinta, jolla voidaan ohjelmien välillä käsitellä pyyntöjä ja eri toimintoja. (Robot Framework User Guide version 2.6.2. 2011.)

Staattinen kirjasto on Python-moduuli tai -luokka; luokka voi olla myös Javalla tehty. Metodien nimet ovat samat kuin avainsanojen nimet. Avainsana ottaa samat parametrit, jotka toteutuksiin on laitettu. Avainsanat raportoivat virheistä poikkeuksilla ja kirjoittavat virheet Robot Frameworkin lokiin sekä konsoliin. Avainsanat voivat myös palauttaa kutsujalle paluuarvon. (Robot Framework User Guide version 2.6.2. 2011.)

Dynaamiset kirjastot ovat melkein samanlaisia kuin staattiset kirjastot. Ainoa ero staattisen ja dynaamisen kirjaston välillä on se, miten Robot Framework havaitsee kirjastojen tarjoamat avainsanat, parametrit ja dokumentaation sekä avainsanojen suorittamisen todellisuudessa. Staattisessa kirjastossa kaikki avainsanat ovat yhdessä luokassa, kun taas dynaamisessa kirjastossa voidaan avainsanat rakentaa erikseen luokkaan. Avainsanan pystyy havaitsemaan dynaamisesti ajon aikana. Raportointi, lokin teko ja parametrien arvojen palautukset on tehty kuten staattisessa kirjastossa. (Robot Framework User Guide version 2.6.2. 2011.)

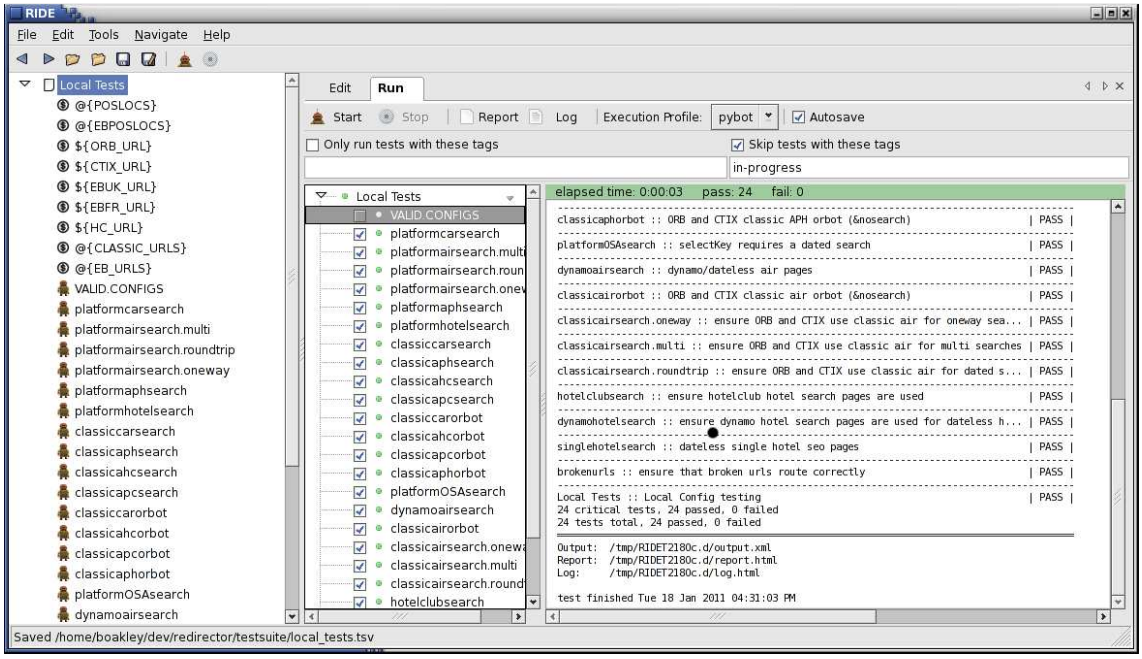
Hybrid toimii staattisten ja dynaamisten kirjastojen välissä. Kirjastot ovat luokkia, joissa kerrotaan, mitä avainsanoja on toteutettu. Hybrid-kirjaston avainsanojen täytyy olla suoraan käytettävissä. Hybrid toimii kuten staattinen kirjasto, mutta poikkeaa avainsanojen havaitsemisessa. (Robot Framework User Guide version 2.6.2. 2011.)

4.3 RIDE-editointiohjelma Robot Frameworkille

RIDE (Robot Interactive Development Environment) on Robot Frameworkille suunnattu testien editointiohjelma, joka on täysin ohjelmoitu Python-ohjelmointikieltä käyttäen. Kehittäjänä toimivat samat henkilöt kuin Robot Frameworkissä. RIDE on Apache-lisenssin alainen ja lähdekoodi on kaikille avointa. (Härkönen 2011.)

RIDE tukee HTML-, TSV- ja tekstimuotoisia formaatteja. Sillä saadaan testit helposti tehtyä Robot Frameworkille ymmärrettävään muotoon.

Käyttämällä Test Runner -pluginia voidaan RIDEllä myös ajaa Robot Frameworkin testit. Test Runner -pluginia käytettäessä ei tarvitse konsolin kautta erikseen käynnistää Robot Frameworkin testin suoritusta. Tämä plugini edellyttää, että Robot Framework on myös asennettu koneelle. Kuvasta 5 nähdään, minkälainen ajotilanne tulee RIDEn kautta *Run*-toimintaa käyttämällä. Testit pystytään ajamaan joko pybotilla tai jythonilla. (Härkönen 2011.)



KUVA 5. Näkymä RIDEn Runin käytöstä (Härkönen 2011)

4.4 Havaitut ongelmat ja puutteet Robot Frameworkissä

Projektiharjoittelujeni ja opinnäytetyöni aikana huomasin Robot Frameworkissä muutamia puutteita tai ongelmia. Joissakin testeissä tarvitaan for-silmukkaa pyörittämään useasti samaa toimintoa, ja muutaman kerran tarvitsin kaksi sisäkkäin olevaa silmukkaa. Kahden sisäkkäisen silmukan tekeminen ei Robot Frameworkissä onnistu, vaan toinen silmukka pitää asettaa avainsanan (keyword) sisään ja sitä kutsutaan ensimmäisen silmukan ajon aikana (taulukko 5).

TAULUKKO 5. Kaksi for-silmukkaa sisäkkäin

Test Case	**Action**	**Argument**	**Argument**	**Argument**
For-loop example	:FOR	${index}	10	
		next loop ${index}		
Keyword	**Action**	**Argument**	**Argument**	**Argument**
next loop ${index}	:FOR	${index2}	2	
		Append To File	test.txt	${index}

Jos halusi testin aikana hyödyntää jo aikaisemmin tehtyä Test Casea, ei tämä ollut mahdollista. Testin aikana Test Caset kutsutaan järjestyksessä ja Test Casesta ei voi kutsua toista Test Casea. Tähän ongelmaan oli tietysti ratkaisuna, että Test Casen toiminto pistetään avainsanan sisään ja avainsanaa kutsutaan tarvittaessa. Juuri tähän kyseinen toiminto on rakennettu. Tämä samainen ongelma tulee myös vastaan, jos halutaan kutsua Test Suitesta toista Test Suitea. Tämäkään ei testiä ajaessa onnistu. Testi, jota halutaan kutsua toisesta testistä, pitää rakentaa täysin avainsanoilla, jotta niitä voitaisiin kutsua ja käyttää.

Kaipasin Robot Frameworkissä ominaisuutta, jossa olisi voinut olla kaksi samannimistä Test Casea. Monesti sama toiminta tehtiin useammassa eri Test Casessa, jolloin Test Casen nimi saattoi olla samanlainen. Tämän ominaisuuden pystyi ainoastaan korvaamaan niin, että rakensi yhden Test Casen josta kutsuttiin avainsanaa. Tämä kutsuttu avainsana sisälsi toiminnan,

mitä piti tehdä. Avainsanan käyttäminen lyhentää testiä, koska samaa toimintoa voidaan kutsua vain yhdellä rivillä. Jos kutsutaan yhdestä Test Casesta vain avainsanoja, testiä ajaessa tulostuu konsoliin vain Test Casen nimi, jolloin ei ole nähtävissä, missä vaiheessa testi on menossa. Ratkaisin ongelman lisäämällä Test Casen loppuun lyhyitä apusanoja. Tällä tavalla saatiin konsoliin tulostettua jokaisen Test Casen nimi, jolloin nähtiin testin suorituskohta.

5 SYSTEEMIMODUULIN TESTAUSYMPÄRISTÖ

Tämän luvun tarkoituksena on kertoa systeemimoduulilla käytössä olevasta testausympäristöstä (liite 1). Jokainen käytössä oleva osa ja niiden asema testausympäristössä on kerrottuna alla omina alalukuinaan. Alaluvussa 5.5 on kerrottu opinnäytetyössä käytetystä testausympäristöstä.

5.1 Systeemimoduuli (SM/FSM)

Systeemimoduuli (SM/FSM, Flexi-systeemimoduuli) sisältää laitteen ohjelmiston (kuva liitteessä 2). Systeemimoduuli on osa Flexi-tukiasemaa. Tämän luvun tarkoituksena on kertoa tarkemmin opinnäytetyön kannalta tärkeimmistä osista systeemimoduulissa.

Systeemimoduuli saa sanomakyselyt Robot Frameworkiltä ja näiden rajapintojen väliset yhteydet hoidetaan Python-kirjastojen avulla. PPS-kellotaajuuden (Pulse per Second, pulsseja sekunnissa) saamiseksi liitettiin systeemimoduulin SYNC-IN-tuloon GPS-vastaanotin (Global Positioning System, globaali paikannussysteemi). Systeemimoduulin kellojärjestelmä pohjautuu 30,72 MHz:n systeemikelloon, joka saadaan OCXOlta (30,72 MHz Oven controlled oscillator) tai MURKKU2:lta. Opinnäytetyössä systeemikellona toimi aina OCXO.

Systeemimoduulin FCT:hen (Flexi Control Multiplexing and Ethernet Transport) MURKKU2:lle tulee valokuitua pitkin DSP:ltä (digitaalisen signaalinprosessointiyksikkö) IQ-dataa (In-phase Quadrature), joka lähetetään RP3-01 (OBSAI Reference Points 3) -rajapintaa pitkin FRM:lle (Flexi Radio Module). FRM:lle tullut data lähtee lähettimen kautta radioteitse puhelimiin. Lisätietoa Flexi-radiomoduulista löytyy luvusta 2.

Systeemimoduuli sisältää FSP- ja FCT-yksiköitä (SM:n sisällä), ulkopuolisia rajapintoja, BTS:n (Base transceiver station, tukiasema) kellon sekä synkronisoinnin ulkoisista kelloista ja välityspalvelut. Ulkoiset rajapinnat tarkoittavat ulkoisia hälytyksen rajapintoja, BTS:n ulkoista synkronisointia sekä

kellon I/O:ta ja LMU-rajapintaa (Location Measurement Unit, paikanmittausyksikkö). (Scheppach 2009, 6.)

Systeemimoduulit voivat olla joko indoor-asennettavia eli ne asennetaan 19” standardi telineeseen tai outdoor-asennettavia malleja, jotka ovat erillisiä moduuleita. Suurin ero toisiinsa on näillä se, että outdoorissa on kolme FSP:tä kun taas indoorissa niitä on vain yksi.

FCT (Flexi Control Multiplexing and Ethernet Transport) on FSM:n ohjausyksikkö. FCT pitää huolen systeemimoduulin O&M (Operation & Maintenance, hallinta ja ylläpito) -ominaisuuksien ohjaamisesta. Se sisältää älykkään lämpötilaohjauksen, kellon jakamisen tukiasemalle, kantataajuus ja RF-synkronisoinnin (Radio Frequency, radiotaajuus), käyttäjätason resurssien hallintaa, Ethernetin siirtorajapinnan, optisen rajapinnan toisiin moduuleihin, BTS ja systeemimoduulin O&M funktiot sekä kantataajuuden kokoaminen summaukselle ja kanavoinnille. (Scheppach 2009, 34.)

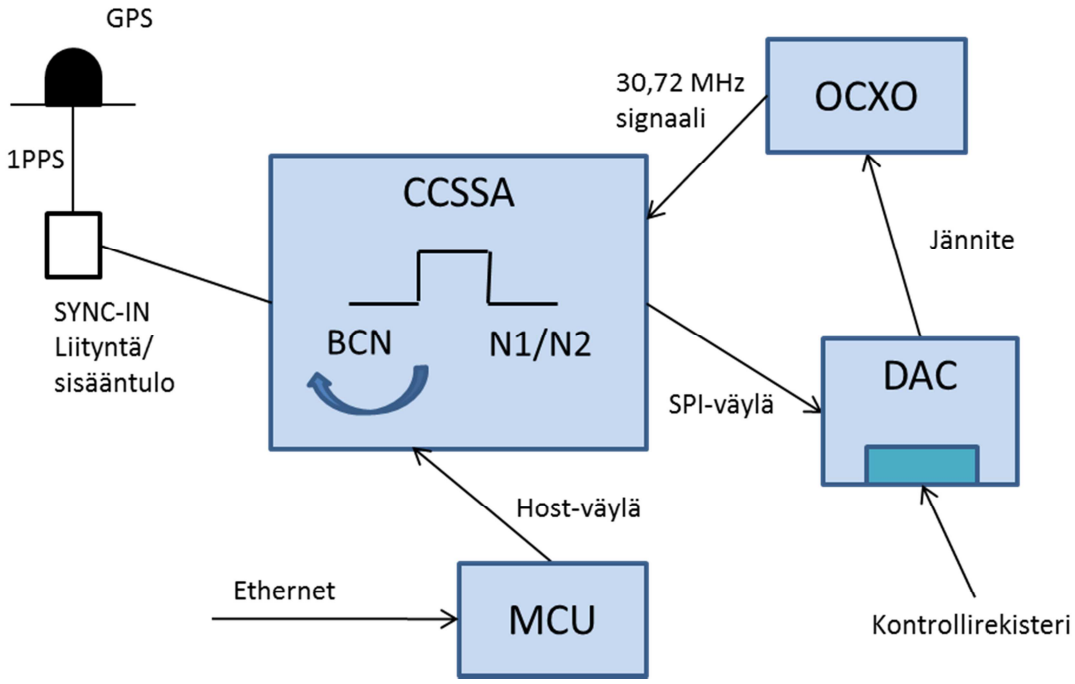
FCT:n tärkeimpiin ominaisuuksiin kuuluu esimerkiksi 30,72 MHz:n signaalin generointi MURKKU2-rajapinnalle (lisätietoa luvussa 5.1.4). FCT:n ominaisuuksiin kuuluu myös systeemin kehyskellon (SFCLK) ja kehysnumeron (SFN) generointi FSP:lle sekä taajuuden ja vaiheen ylläpito. (Scheppach 2009, 34.) FCT:ssä suoritetaan ohjaustehtäviä, joita tarvitaan multiradiosysteemimoduulissa. Multiradiosysteemimoduuli tukee GSM-, WCDMA- sekä LTE-tekniikoita. (Scheppach 2009, 6.)

MCU (MicroControl Unit) on prosessori, joka sijaitsee kontrollimoduulin ohjausyksikössä (FCT). MCU:n kautta menee tietoa DSP-prosessorille (digitaalisen signaalinprosessointiyksikkö) SRIO (Serial Rapid IO, nopea sarjaväyläinen liitäntä) -väylää pitkin. MCU:n ohjelmistolle tein opinnäytetyön aiheena olevan testin. (Scheppach 2009, 35.)

ETH (Ethernet) kautta Flexi-tukiasema kytketään testi-PC:hen tai Robot Frameworkiin Ethernet-verkkoliitännän kautta.

5.1.1 CCSSA (Clock, Control, System, Synchronization and Application)

CCSSA (kuva 6) on keskeinen osa FCT-yksikön käynnistyssekvenssiä ja alkuarvoihin asettamista. Sen tärkeimpiin tehtäviin kuuluu systeemikellon synkronisointi referenssikellojen avulla. CCSSA sisältää ohjauksen OCXOlta DACille. CCSSA:n rekisterit kirjoitetaan ja luetaan Host-väyläliittymän kautta MCU:lta. Funktionaalisuutta ohjataan Host-rajapinnan kautta. (Rathenow 2011, 8.)



KUVA 6. CCSSA:n liittymät (oleellisemmat opinnäytetyön kannalta)

CCSSA saa PPS-kellopulssin GPS:ltä tukiaseman SYNC-IN-sisääntulon kautta. CCSSA sisältää tukiaseman keskus-BCN (Basestation Clock Number, tukiaseman laskuri) -laskurit (N2 = 24 bit ja N1 = 40 bit). Laskurit saadaan synkronoitua sisään tulevalla referenssikellolla eli tässä tapauksessa PPS-kellopulssilla. BCN-laskurit näytteistetään säätöohjelmistoa varten, joka on käynnistetty sisään tulevalla referenssisignaalilla. Nämä BCN-laskurit

käynnistävät WCDMA:n laskurin. BCN- ja WCDMA-laskureita käytetään käynnistämään kellopulssit MURKKU2:lle ja FSP-korteille. (Rathenow 2011, 8.)

5.1.2 DAC (Dicital-to-Analogue Converter)

DAC on nimensä mukaisesti konvertteri digitaalisen signaalin muuttamisesta analogiseksi. DACin tarkoituksena on säädellä OCXOn taajuutta analogisella jännitteellä.

Säätöohjelma voi kirjoittaa 16-bittisen luvun DACin kontrollirekisteriin. Tämän 16-bittisen luvun DAC konvertoi analogiseksi jännitteeksi OCXOn säätöjännitetuloa varten. OCXOn uunikvartsi virittää taajuutensa sisääntulevan jännitteen perusteella. Tällä tavalla ohjelmisto pystyy säätämään taajuutta omiin tarpeisiin sopivaksi. DACin alkuarvo systeemimoduulissa on heksalukuna 0x8000. (Rathenow 2011, 78.)

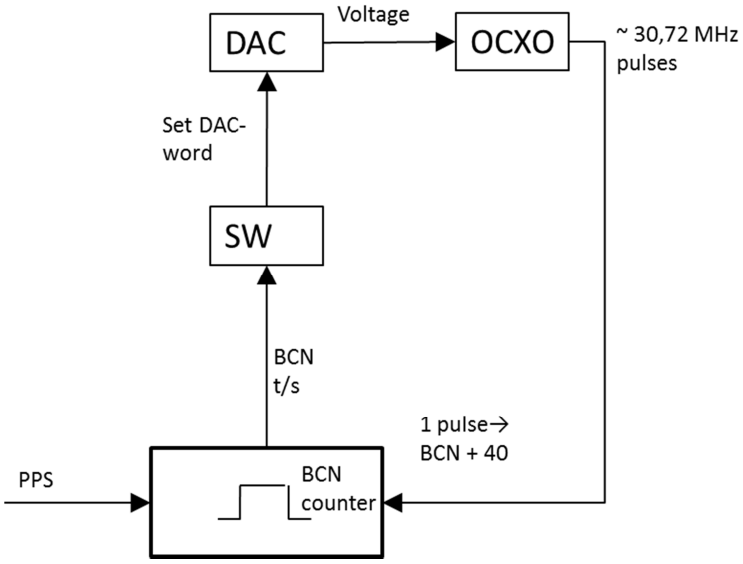
5.1.3 OCXO (Oven Controlled Oscillator)

OCXOa käytetään generoimaan tasalaatuista referenssisignaalia. OCXO sisältää kristallioskillaattorin, lämmittimen, kontrollipiirilevyn ja lämpöeristyksen kristallioskillaattorin ympärillä. Kvartsikristallien värähtelyominaisuudet muuttuvat lämpötilavaihteluiden vuoksi, mutta OCXOn sisältämällä lämmittimellä pidetään lämpötilavaihtelut minimissä. Kristallin valmistuksessa käytetty leikkaustapa vaikuttaa ominaisuuksien muutosherkkyyteen. OCXOt ovat fyysisesti paljon suurempia kuin tavalliset kristallioskillaattorit. (Poole b.)

OCXO generoi 30,72 MHz:n systeemikellon tukiasemalle ja sitä kontrolloidaan ohjelmiston CCSSA:lla ja DACilla. Referenssilähde OCXOn kontrollille voi olla GPS_PPS, Ethernet tai ulkoiset rajapinnat. (Scheppach 2009, 38.)

Kun indoor-Flexejä on käytössä kaksi, on masterilla käytössä OCXO, joka toimii 30,72 MHz:n referenssikellona. Slavella on käytössä MURKKU2, mutta OCXO antaa kellon masterin kautta myös MURKKU2:lle käyttöön. OCXOlle tulee PPS-kellopulssi GPS-vastaanottimen kautta.

Kuvasta 7 voidaan nähdä OCXOn säätäminen DACin avulla. Ohjelman (SW) avulla asetetaan uusi DAC-sana, jonka avulla säädetään OCXOlle menevää jännitettä. Jännitteellä säädetään OCXOn taajuutta, jonka noin 30,72 MHz:n pulssit lähetetään BCN-laskurille. Jokaisen tulevan pulssin aikana BCN-laskuri kasvaa 40:llä. Luku 40 saadaan aikaiseksi, kun BCN:n 1,2288 GHz:n määritelty taajuus jaetaan OCXOn 30,72 MHz:n taajuudella. OCXOn taajuutta säätämällä korjataan vaihe-eroa ja taajuusvirhettä, jotta saadaan jokaisella PPS-kellopulssilla otettua uusi arvo mahdollisimman läheltä vaihe-eron nousevaa reunaa.



KUVA 7. OCXOn säätäminen DACin avulla

5.1.4 MURKKU2

MURKKU2 on yhteyspiiri, joka toimii Flexi-systeemimoduulissa. Piiri pitää huolen antennien kantoaalloista ja tiedon reitityksestä DSP-piirien ja FRM:n välillä. IQ-data kulkee RP3-01-rajapinnan kautta FRM:lle. (Wäljas 2011, 24.)

MURKKU2:n aika perustuu BCN-laskuriin. MURKKU2 käyttää N2-laskurista 22 bittiä 24 bitin sijaan ja kaksi viimeistä LSB:tä (vähiten merkitsevä bitti) ovat aina nollia. (Wäljas 2011, 26.)

MURKKU2 generoi 30,72 MHz:n ulostulokellon RP3-01-virrasta. RP3 on OBSAI (Open Base Station Architecture Interface) -ohjelmiston rajapinta. Kun indoor-Flexejä on käytössä kaksi (slave, master), on tällöin slavessa käytössä MURKKU2-kello. Itse 30,72 MHz:n referenssikello saadaan kuitenkin RP3-01-virran kautta masterin OCXOlta.

5.1.5 Signaalin prosessointiyksiköt sekä SRIO

FSP (Flexi-signaaliprosessointiyksikkö) -yksiköitä on sisäisessä systeemimoduulissa yksi kappale, kun taas ulkoisessa systeemimoduulissa FSP-yksiköitä voi olla jopa kolme kappaletta. FSP:ssä tapahtuu signaalin prosessointi.

DSP (Digitaalisen signaalin prosessointiyksikkö) on digitaaliseen signaalinkäsittelyyn rakennettu suoritin. Systeemimoduulin FSP sisältää näitä DSP-suorittimia useamman kappaleen. DSP tuottaa ja vastaanottaa IQ-dataa FRM:lle ja FRM:ltä.

SRIO (Serial Rapid IO) on väylä, joka yhdistää MCU:n ja DSP:n toisiinsa. Serial RapidIO on standardisoitu kytkinpohjainen vertaisverkko.

SRIO:ssa jokaisella paketin siirrolla on takuu, että mikään paketti ei häviä ruuhkassa. Jokaisen siirron voi jäljittää jäljitys-ID:llä ja jokainen vastauksena tullut paketti on myös jäljitettävissä omalla jäljitys-ID:llä. Standardin määrittelyssä on myös ”ovikello”-paketti (”doorbell”). Nämä paketit on keskeytetty loppupisteestä Serial RapidIO -systeemiin. Baseband-yksiköiden DSP:t käyttävät näitä keskeytyksiä ilmaistakseen, että kokonainen IQ-datalohko on vastaanotettu SRIOn kautta ja prosessointi voi alkaa. (Hiat 2007, 1.)

5.2 Robot Framework

Robot Frameworkillä suoritetaan systeemimoduulille testien ajot. Robotilta lähetetään SICAP-sanomia (SIC Application Protocol) systeemimoduulille, jonka jälkeen saadaan vastaukseksi SICAP-sanomia Robot Frameworkille. Robot Frameworkillä tarkistetaan tulleen sanoman parametrien oikeellisuus. Robot Frameworkistä löytyy lisätietoa luvusta 4.

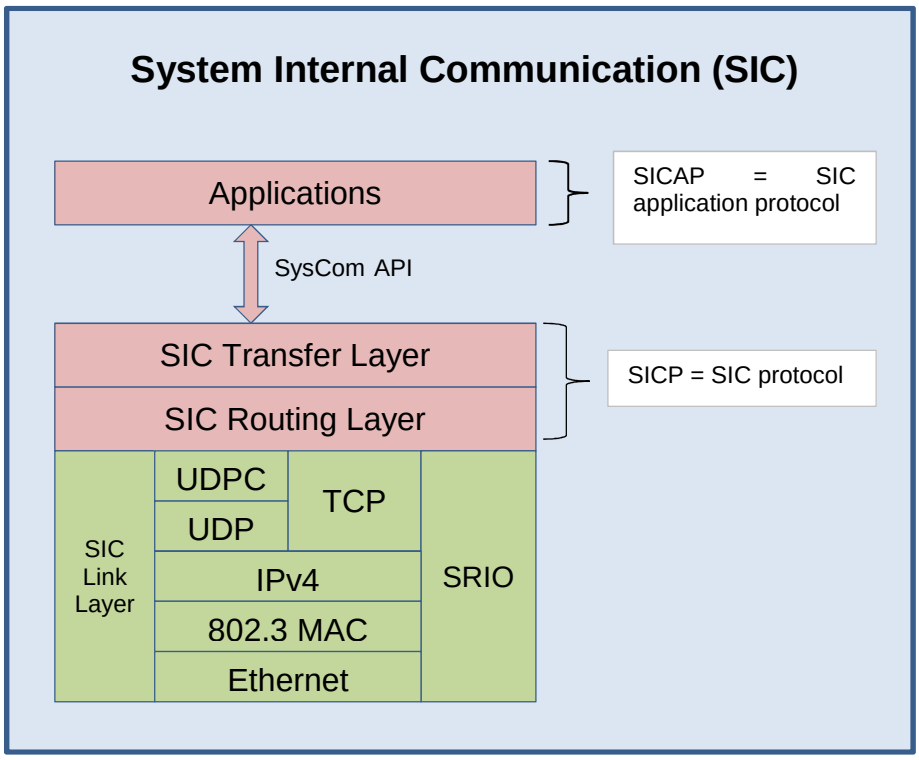
5.3 Sanomakonsepti

Sanomat kulkevat Robot Frameworkin ja systeemimoduulin välillä SICAP-sanomina. Sanomat kulkevat SysCom-ohjelmistorajapinnan kautta.

SysCom on yksi CC&S:n (Common Computer & Support) palvelu. CC&S:llä on yleisiä palveluita tukiaseman kaikille ohjelmalohkoille eri prosessoriympäristöissä. Jokaisella palvelulla on omat ohjelmointirajapintansa ja sanomansa.

SysComilla on yksinkertaiset metodit SICAP-sanomien lähettämiseen ja vastaanottamiseen. SysComissa on keino muille ohjelmistoille kommunikoida sisäisesti ja ulkoisesti käyttämällä samaa ohjelmointirajapintaa. (Cundekovic 2011, 1.)

Ohjelmien ei tarvitse tietää, mitä protokollia käytetään tai mitä reittejä pitkin sanomat lähetetään vastaanottajalle (kuva 8). Lähettäjä merkitsee pelkästään vastaanottajan SIC-osoitteen ja SysCom välittää tiedon oikeaan paikkaan käyttäen aikaisemmin konfiguroituja reittejä.



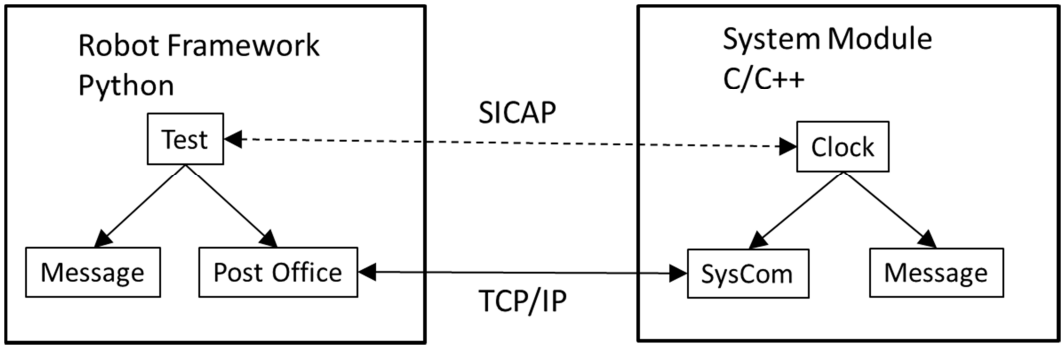
KUVA 8. Sisäisen kommunikoinnin protokollakerrokset (Lell 2011, 27)

SysComilla on oma sisäinen yhteydenpito-osoitteensa SICAD (System Internal Communication Address). SICAD koostuu yhteydenpitopaikkatunnistimesta CPID (Communication Point Identifier) ja solmutunnistimesta NID (Node Identifier). NID on yksilöllinen laitteiston ja prosessoreiden osoite. CPID on ohjelmiston oma tunniste, joka on paikallisesti yksilöllinen. CPID:tä voi olla yksi tai useampi kappale jokaisessa kommunikoivassa ohjelmakokonaisuudessa. (Cundekovic 2011, 1.)

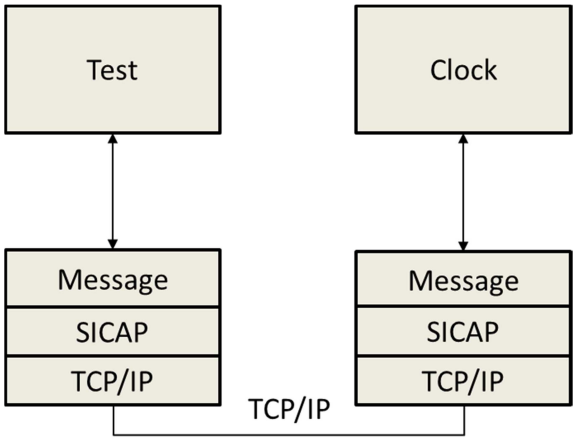
Testissä sanomien lähetys tapahtuu kuvan 9 tapaisesti. Robot Frameworkissä suorituksessa oleva testi luo sanoman, joka lähetetään Post Officen (SysComin python toteutus) avulla TCP/IP-yhteyden yli systeemimoduulin SysComille.

Opinnäytetyön sanomat kuuluvat ApiClockService-rajapintaan, joka määrittelee SICAP-viestin sisällön. Rajapinta pitää olla toteutettuna aina sekä C:llä että Pythonilla. ApiClockService on muodostettu alkuperäisen C-structin sisällöstä

(osa koodista liitteessä 3). Liitteessä 3 on myös yhden sanoman header-tiedosto, josta voidaan nähdä sanoman rakenne eli mitä parametreja siihen kuuluu. Kuvasta 10 voidaan nähdä protokollapinona sanoman kulku Testin ja Clockin välillä.



KUVA 9. Sanomien kulku Robot Frameworkin ja Systeemimoduulin välillä



KUVA 10. Protokollapino testin ja clockin välillä

5.4 Jenkins CI Server

Jenkins CI (Continuous Integration, jatkuvaa integrointia) Server on automaattisten testien ajamiseen tarkoitettu ohjelma. Jenkinsin yleisimmät tehtävät ovat ohjelmistojen kääntämiset sekä testien suorittamiset. Jokaiselle tehtävälle voidaan määritellä erikseen, missä vaiheessa ne käynnistetään. Yleensä osa testeistä käynnistyy automaattisesti käännöstehtävän päätyttyä. Esimerkiksi tämän opinnäytetyön käytössä olevassa testiympäristössä Jenkins

antaa käskyn Robot Frameworkille, että se voi aloittaa testien suorittamisen. (Mikkola 2011.)

Jenkinsin tehtäviä voidaan suorittaa myös erilaisten pluginien avulla. Esimerkiksi Robot Frameworkin ohjaus ja tulosten esittämisen tuki tarjotaan pluginin avulla. Plugineja Jenkinsille voi olla useita kymmeniä mm. käännösten tekemiseen, testauksiin, raportointiin sekä graafisten asioiden piirtämiseen. Jenkinsillä voidaan suorittaa tehtäviä ilman pluginiakin. Tehtävät suoritetaan komentoriviskriptejä kutsuen tai suoraan eri ohjelmia käynnistäen. (Mikkola 2011.)

Komentoriviltä käynnistettävien ohjelmien käytössä pitää muistaa se, että vaikka niistä avautuisi graafisia käyttöliittymiä, niihin ei voida Jenkins-komentoriviltä syöttää enää mitään. Pahimmassa tapauksessa suorituksessa oleva tehtävä jää jumiin Jenkinsille sekä käynnistetty ohjelma jää käyntiin renkikoneelle. (Mikkola 2011.)

Jenkinsille voidaan luoda ajoitettuja tehtäviä. Esimerkiksi Jenkinsin asetuksiin on laitettu tieto, että kello seitsemän aikaan aamulla ajetaan skripti, joka ohjaa virtakytkintä. Tämän myötä valaisin, tuuletin ja kahvinkeitin kytkeytyvät päälle kello seitsemän aamulla, jolloin aamutoimet voisi aloittaa kahvin tuoksuun, tuulen vireeseen ja ”auringon paisteeseen”. (Mikkola 2011.)

Jenkins Serverillä (master) voi olla renkeinä (slave) Linux-, Windows-, Mac- sekä Unix-koneita. Koneita voi olla useita ja näille voidaan määritellä yksi tai useampi työjono. Työjonojen määrä riippuu koneen muistista ja prosessorista. Jos koneella on useampi ydin prosessorissa, voidaan tälle laittaa useampi työjono. (Mikkola 2011.)

5.5 Opinnäytetyössä käytössä oleva testausympäristö

Opinnäytetyössä oli käytössä liitteen 1 mukainen testausympäristö, mutta käytössä oli yksi systeemimoduuli (System Module 1). Jenkins reagoi versionhallintaan tehtyihin tiedostojen muutoksiin, jotka saavat aikaiseksi

ohjelmien uudelleen kääntämisen. Käännöstöiden jälkeen Jenkins rupeaa suorittamaan hyväksyntätestausta, jonka ensimmäisenä tehtävänä on siirtää (SW load) onnistuneesti käännetyt ohjelmat systeemimoduulille (opinnäytetyössä System Module 1). Ohjelmien siirtämisen jälkeen Jenkins antaa käskyn Robot Frameworkille, että se aloittaa (Start SCT) testitapausten (Test Case) suorittamisen. Test Casen aikana Robot Framework hoitaa SICAP-sanomien lähetyksen ja vastaanottamisen systeemimoduulin kanssa.

Koska opinnäytetyössä on vain käytössä yksi systeemimoduuli, ei tällöin diodi /rele-kytkintä tarvita.

6 TESTIN SUUNNITTELU

Tämän luvun tarkoituksena on kertoa alaluvuissa testin suunnittelun kulku. Kerrotaan käytetyt sanomat ja niiden käyttötarkoitukset. Esitetään kaavio käytettävistä sanomista ja tiedosta, jota ne kuljettavat. Kerrotaan, miten saadaan mittausarvot tukiasemalta ja miten arvoja muutetaan sanomia apuna käyttäen.

6.1 Testauksen kohde ja tavoitteet

Testauksen kohteena oli Nokia Siemens Networksin kehittämä Flexi-systeemimoduuli (FSM). Tarkoituksena oli saada vaihe-eromittausten tuloksien muutos vaihtumaan päinvastaiseen suuntaan muutettaessa DAC-sanan asetussanoman parametrin arvoa. Mittausten muuttuminen näkyy niin, että jos aluksi mittaustulokset tulevat kasvavana, niin DAC-sanan muuttamisen jälkeen mittaustulokset tulevat laskevana. Tällä testillä varmistettiin, että DAC-sanoman yhtä parametrin arvoa muuttamalla ohjelmisto kykenee ohjaamaan OCXOn taajuutta. Tarkemmin DAC-sanan vaikutuksesta OCXOn taajuuteen kuvataan luvussa 6.3.

Testistä rajattiin pois paljon muita sanomien tutkimisia, koska ne tulevat testatuksi muissa aikaisemmin rakennetuissa testeissä. Testissä käytettiin vain testin kannalta oleellisimpia sanomia. Testin sanomat käsitellään luvussa 6.5 Sanomasekvenssikaavio.

6.2 Testauksen organisointi ja raportointi

Testiä testattiin jokaisen testiin vaikuttavan muutoksen jälkeen. Tärkeintä oli, ettei uusia toimintoja tehty testiin, ennen kuin edellinen muutos toimi oikein. Testitapahtumissa käsiteltäviin sanomien parametreihin testattiin monenlaisia arvoja, jos se oli mahdollista, ja katsottiin, miten ne vaikuttivat testin toimivuuteen. Testi täytyi rakentaa sanoma kerrallaan ja katsoa, mitä sanomia testi tarvitsee ja mitä ei.

Testin jokaisen suorittamisen jälkeen Robot Framework loi lokitiedoston ja myös raportin HTML-muodossa. Raportista nähtiin suoraan onnistuneet ja epäonnistuneet kohdat. Lokitiedostosta pystyttiin seuraamaan testin eteneminen ja havaitsemaan, mihin kohtaan virheet tulivat. Samaisesta lokitiedostosta pystyttiin myös seuraamaan systeemimoduulin omaa lokitiedostoa, josta nähtiin tulostukset jokaisesta vaiheesta systeemimoduulin käynnistyksestä lähtien. Systeemimoduulin lokitiedostoon tallentui myös suurin osa ajetun testin sanomista. Systeemimoduulin uudelleenkäynnistämisen jälkeen uudet tiedot tallentuvat vanhan lokitiedoston päälle. Yleensä tärkeimmät lokitiedostot nimettiin uudelleen, jolloin pystyttiin seuraamaan muutokset edelliseen ajokertaan nähden.

6.3 Testattavat toiminnot ja niiden hyväksymiskriteerit

Testattaviin toimintoihin ja hyväksymiskriteereihin kuuluu vaihe-eromittaustuloksien saaminen systeemimoduulilta ja DAC-arvolla OCXOn taajuuden säätäminen ja sen myötä mittaustuloksien muuttaminen.

DAC-arvo voi olla 0–65 536 välillä eli heksadesimaalilukuna arvo on 0–FFFF. Kun arvo on noin puolessa välissä maksimiarvosta tai yli sen, rupeavat mittaustuloksien arvot pienenemään. Testattiin ja kokeiltiin tarkemmin, miten eri DAC-arvot saavat vaihe-eron muuttumaan testin aikana. Mitä lähemmäs DAC-arvon puoltaväliä mentiin eli noin 31 500–32 500, rupesivat mittausarvot vaihtelemaan sekä nousevasti että laskevasti tai sitten mittausarvo saattoi myös pysyä samana. Laitettaessa DAC-arvo nollaksi oli mittausarvon muutos edelliseen mittausarvoon noin 2000. Kun taas lähestyttiin puoltaväliä, oli mittausarvoissa enää eroa 0–50. DAC-arvon laittaminen maksimiarvoon toi mittausarvoihin heittoa 2000:n luokkaa.

Mittausarvo on BCN-luku, jonka teoreettinen resoluutio on määritelty 1,2288 GHz:n taajuudelle. BCN-luvun muutos yhdellä vastaa noin 0,81 ns:a, joka saadaan laskettua seuraavalla tavalla: $\frac{1\mathrm{s}}{1{,}2288\ \mathrm{GHz}} \approx 0{,}81\ \mathrm{ns}$.

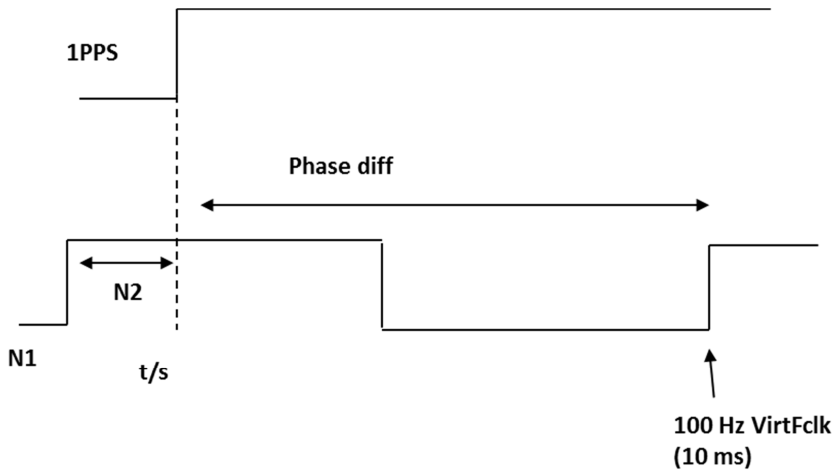
Käytettäessä GPS:n PPS:ää referenssikellona tulevat mittausarvot sekunnin välein. Esimerkiksi 2000:n muutos peräkkäisissä mittausarvoissa tarkoittaa, että 1PPS:n ja 10 ms:n virtual frame clockin nousevien reunojen ero muuttuu 1,62 µs sekunnissa. Tämä ero saadaan laskettua seuraavalla tavalla: $2000 * 0{,}81\ \text{ns} = 1{,}62\ \mu\text{s}$.

Edellisen arvon perusteella voidaan laskea, kuinka kauan kuluu aikaa, että vaihe-ero muuttuu 0 ms:sta 10 ms:iin maksimi-DAC-sanalla. Esimerkiksi jos mittausarvo muuttuu 2000:lla joka sekunti, menee tällöin maksimi-DAC-sanalla 6173 sekuntia, jotta vaihe-ero muuttuu 0 ms:sta 10 ms:iin. Tämä aika voidaan laskea seuraavalla tavalla: $\frac{10\ \text{ms}}{1{,}62\ \mu\text{s}} \approx 6173\ \text{s}$.

6.4 Vaihe-eromittaukset

Kuvassa 11 on kuvattu, miten vaihe-eromittaus tapahtuu. Kuvasta nähdään opinnäytetyön testin kannalta tarpeellisimmat laskurit (N1 ja N2) ja kellopulssi (1PPS). t/s kuvassa tarkoittaa sekä N1:tä ja N2:ta.

BCN-laskuri on 64-bittinen luku ja se on pilkottu 40-bittiseen N1-laskuriin ja 24-bittiseen N2-laskuriin. BCN-laskurista on kerrottu myös luvussa 5.1.1



KUVA 11. Vaihe-eron mittaus

Systeemimoduulilta tulee kuvan 12 ja kuvan 13 kaltaiset tulostukset lokitiedostoon, kun vaihe-eromittaus onnistuu. Laskevan vaihe-eron mittausarvon saa laittamalla DAC-arvoksi alle 31 500 ja nousevan vaihe-eromittausarvon saa laittamalla yli 32 000 olevan DAC-arvon.

N1/N2-sarakkeessa on BCN:ään kuuluvien N1 ja N2 arvot. High-sarakkeessa on High Accuracyn arvo eli tästä arvosta nähdään, kuinka lähellä 30,72 MHz:n nousevaa reunaa mittausarvo on. Koska 30,72 MHz:n kello ei ole tarpeeksi tarkka, saadaan High-arvolla parannettua 30,72 MHz:n tarkkuutta. N1, N2 ja High saadaan CCSSA:lta S&H-lohkosta (Sample & Holding). VirtFclk-sarakkeessa on ns. aikaleima siitä kohdasta, mistä alkaa uusi nouseva reuna Virtual frame clock:lla (kuva 11). Vaihe-eron mittaustulos, joka raportoidaan testille, löytyy Phase diff -sarakkeesta. Delta curr-prev -sarakkeesta nähdään. kuinka paljon mittausarvo (phase diff) on muuttunut edelliseen mittausarvoon nähden. Ensimmäiselle mittaukselle Delta curr-prev on aina nolla.

t/s (N1 N2):	High:	VirtFclk (L):	Phase diff (L):	Delta curr-prev (L):
843 933 11399858	0x1ff	10370260992000	888142	0
844 033 11398428	0x1f	10371489792000	889572	1430
844 133 11397050	0xfff	10372718592000	890950	1378
844 233 11395630	0xf	10373947392000	892370	1420
844 333 11394205	0x3fff	10375176192000	893795	1425
844 433 11392783	0x7f	10376404992000	895217	1422
844 533 11391355	0x3	10377633792000	896645	1428
844 633 11389933	0x7ff	10378862592000	898067	1422
844 733 11388508	0x1f	10380091392000	899492	1425
844 833 11387088	0x1fff	10381320192000	900912	1420

KUVA 12. Vaihe-eromittauksen arvot systeemimoduulista (nouseva)

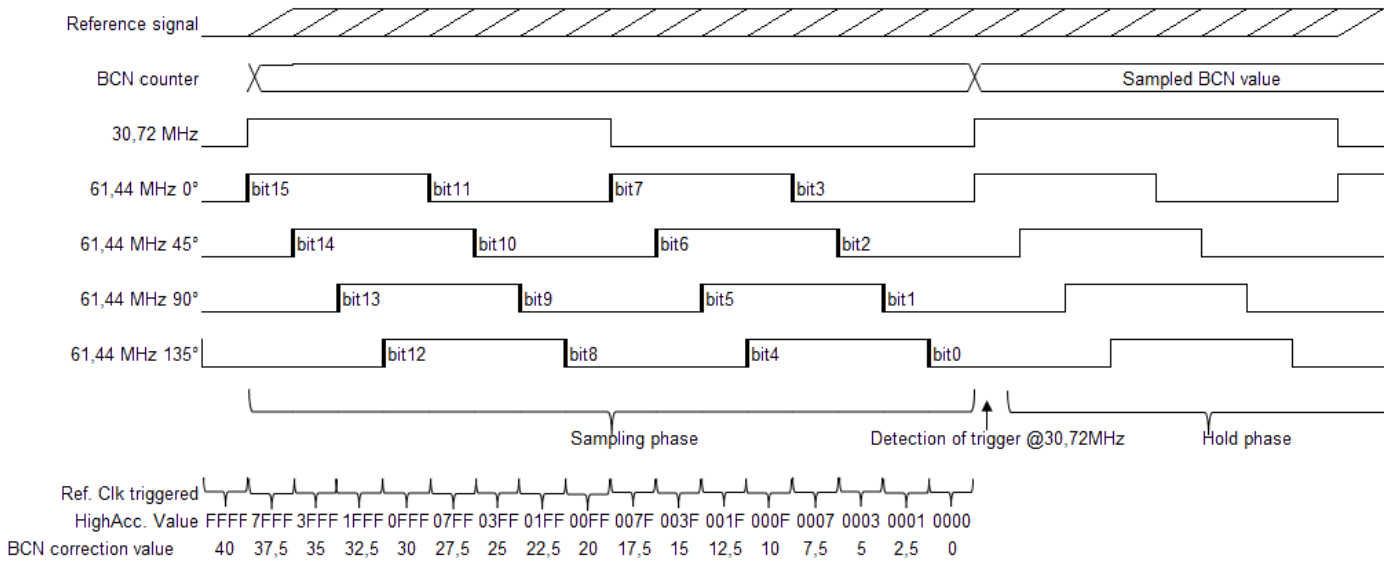
t/s (N1 N2):	High:	VirtFclk (L):	Phase diff (L):	Delta curr-prev (L):
34 721 892 1739805	0x3fff	426662621184000	10548195	0
34 721 992 1739878	0x1	426663849984000	10548122	-73
34 722 092 1739948	0x1f	426665078784000	10548052	-70
34 722 192 1740065	0x3f	426666307584000	10547935	-117
34 722 292 1740138	0x1ff	426667536384000	10547862	-73
34 722 392 1740208	0x1fff	426668765184000	10547792	-70
34 722 492 1740278	0x1	426669993984000	10547722	-70
34 722 592 1740403	0x7fff	426671222784000	10547597	-125
34 722 692 1740473	0x7	426672451584000	10547527	-70
34 722 792 1740545	0x3f	426673680384000	10547455	-72

KUVA 13. Vaihe-eromittauksen arvot systeemimoduulista (laskeva)

N1/N2 ovat BCN:ään kuuluvia laskureita ja ovat N-formaatissa. N1-laskuri ei pyörähdä ympäri tukiaseman elinaikana, koska se on 40-bittinen luku eli 2^{40} = 1 099 511 627 776. N1 kasvattaa arvoaan 10 ms:n välein ja sen ympäripyörähtämiseen menisi aikaa 348 vuotta, 238 päivää, 6 tuntia, 57 minuuttia ja 57 sekuntia. PPS-referenssikellon syke tulee sekunnin välein, jolloin N1 ehtii kasvattaa arvoaan sadalla. N2-laskurin arvot pyörähtävät 10 ms:n aikana kokonaan ympäri ja sen arvot ovat välillä 0–12 287 999.

High accuracy (kuva 14) on 16-bittinen luku ja mahdollisia arvoja voi olla 16 kappaletta välillä 0000–FFFF. High'n tarkkuusarvo tuotetaan käyttämällä FlipFlop-kiikkuja 61,44 MHz:n taajuudella. Käytössä on kahdeksan rinnakkaista kiikkua. Jokaisella kiikulla on oma astelukunsa, jonka aikana se vaihtaa vaihetta. (Rathenow 2011, 88.)

Asteluvut High'lla ovat 0˚, 45˚, 90˚ ja 135˚. High Accuracyn -arvoa käytetään määrittämään luku, jolla BCN-lukua korjataan. High Accuracy -arvoa pidetään rekisterissä samaan aikaan, kun 30,72 MHz:n systeemi huomaa, että liipaisu on tapahtunut, jonka jälkeen se tallentaa N1/N2-arvot BCN-laskuriin. Esimerkiksi jos High'n arvo on FFFF, tällöin BCN-lukua korjataan arvolla 40. (Rathenow 2011, 88.)



KUVA 14. BCN-arvon korjaus High accuracy -arvon avulla (Rathenow 2011, 88)

VirtFclk on BCN-luku L-formaatissa eli lineaarisessa muodossa. Se on aikaleima seuraavasta 10 ms:n virtual frame clockin alkavasta reunasta, johon PPS:n nousevan reunan aikaleimaa verrataan (kuva 11, VirtFclk). BCN N-formaatin luvun lineaariseksi muuttaminen saadaan laskettua kaavalla 1.

lineaarinen = N1 * 12288000 + N2 *KAAVA 1*

Kuvassa 11 näkyvä VirtFclk timestamp (t/s) saadaan laskettua kaavan 2 mukaan. t/s on BCN-luku, joka on 1PPS kellopulssin nousevan reunan kohdalta CCSSA:n S&H näytteistämä arvo (katkoviiva kuvassa 11). 12288000 on virtual frame clockin (kuva 11, VirtFclk) jakson pituus eli 10 ms. Kaavan 2

floor($\frac{t/s}{12288000}$) antaa arvoksi, kuinka monta kokonaista virtual frame clockin (kuva 11, VirtFclk) jaksoa on kulunut.

$$\left[floor\left(\frac{t/s}{12288000}\right)+1\right]*12288000 \qquad \textit{KAAVA 2}$$

6.5 Sanomasekvenssikaavio

MSC-kaaviolla (Message Sequence Chart, sanomasekvenssikaavio) kuvataan testissä käytössä olevien sanomien järjestys kaavion muodossa. Opinnäytetyön testin sanomasekvenssikaavio nähdään kuvassa 15. Sanoman nimen lopussa olevat *Req* on lähetettävä sanoma, jolla laitetaan parametrit kuntoon, ja *Resp* on vastauksena tuleva sanoma. *Resp*-sanomasta tarkistetaan lähetettyjen parametrien oikeellisuus.

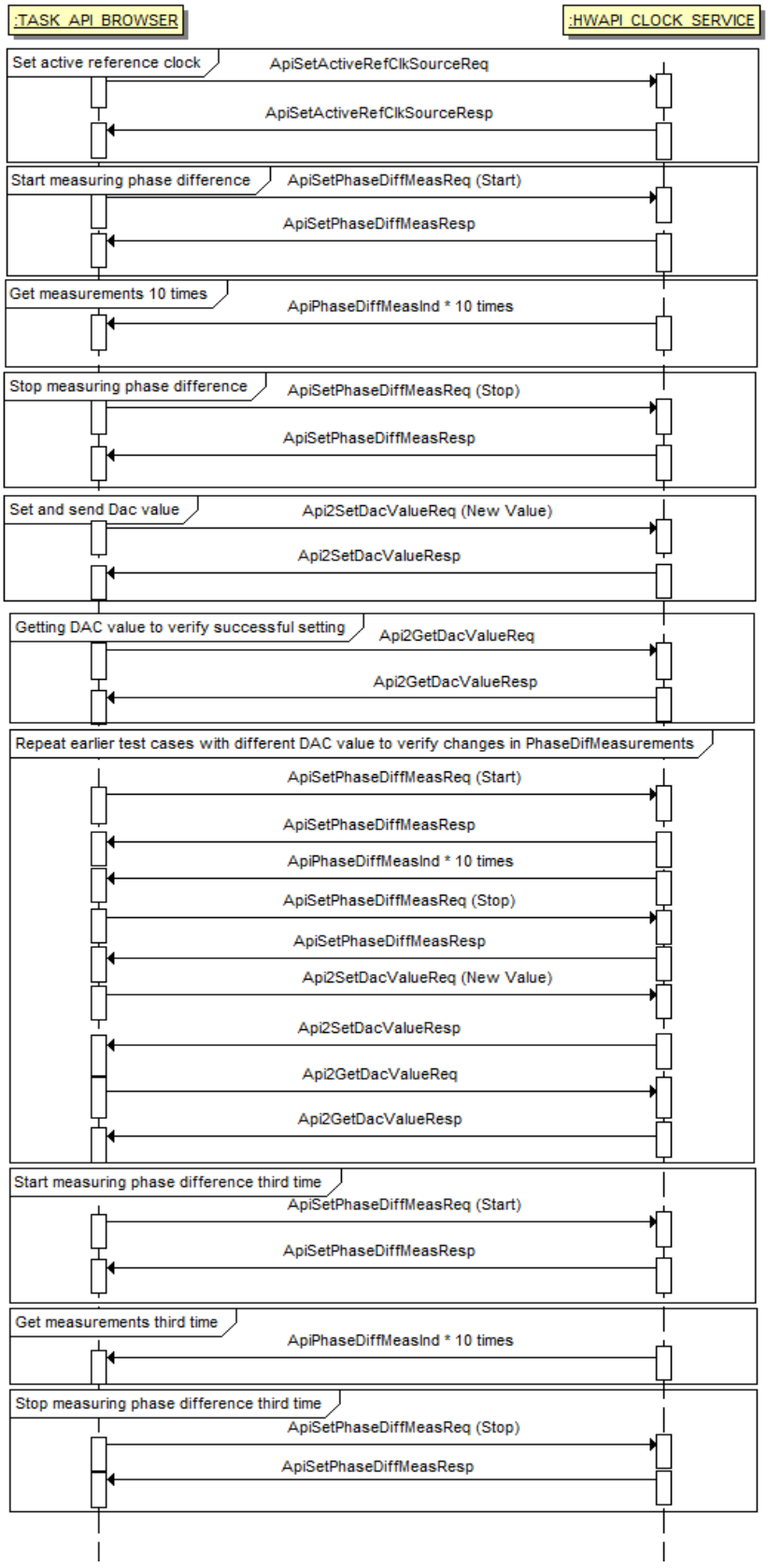
ApiSetActiveRefClkSource-sanomalla valitaan mitä FCT:n ohjelmistoa on käytettävä. Tässä testissä referenssikellona käytettiin PPS-kellopulssia.

Ensimmäisellä *ApiSetPhaseDiffMeas*-sanomalla käynnistetään vaihe-eromittaukset *Start*-parametrilla. Lähetettäessä *ApiSetPhaseDiffMeas*-sanoma parametrilla *Stop* vaihe-eromittaukset lopetetaan.

ApiPhaseDiffMeasInd-sanoman avulla FCT:n ohjelmisto raportoi mitatun vaihe-eron.

Api2SetDacValue-sanomalla asetetaan uusi DAC-arvo, jolla vaikutetaan OCXOn taajuuteen, joka vaikuttaa vaihe-eromittauksen tulokseen. *Api2GetDacValue*-sanomalla saadaan tarkistettua asetettu DAC-arvo. Tätä sanomaa ei kannata kutsua ilman, että on asettanut ensin itse DAC-arvon. *Api2GetDacValue*-sanoman kutsumisessa tarvitaan tieto aikaisemmin asetetusta DAC-arvosta.

Testiä ajettaessa nähdään, saadaanko aikaiseksi toivottu tulos.



KUVA 15. Testin sanomasekvenssikaavio

7 TESTIN RAKENNE JA TOTEUTUS

Testi rakennettiin Robot Frameworkille RIDEllä HTML-muodossa. Testi ajettiin Robot Frameworkillä Linux-ympäristössä ja käynnistettiin pybotilla komentotulkista, johon myös tulostuivat näkyville testin tulokset.

Testistä tuli kolme eri versiota, joista suppeampi versio otettiin hyväksyntätestaukseen mukaan ja laajemmasta tein opinnäytetyön kirjallisen osan. Opinnäytetyön kirjalliseen osaan otetun version koodit ovat liitteessä 4.

7.1 Testin rakenne

Opinnäytetyön testi rakentui sanomien lähettämisestä ja vastaanottamisesta. Näitä sanomia tarvitaan eri yksiköiden väliseen kommunikointiin. Opinnäytetyössä sanomien tarkoituksena oli kommunikoida eri ohjelmalohkojen välillä. Näistä sanomista kerrotaan enemmän luvuissa 6.5 ja 7.2.

Opinnäytetyön Test Suite koostuu neljästä eri osiosta, *Setting*, *Variable*, *Test Case* ja *Keyword*. Alla esitellään osa osioissa olevista asioista.

Setting-osiossa (taulukko 6) laitetaan testiä varten kirjastojen ja muiden tarpeellisten tietojen polut oikein. Tähän kohtaan voidaan myös kertoa testin versionumero ja muita tietoja, esimerkiksi mistä testiin on saatu vaatimukset.

Taulukosta 6 nähdään, miten tietoa voidaan tallentaa *Setting*-osioon. *Suite Setupissa* asetetaan testin alkuasetukset. Tässä testissä *Suite Setupissa* ajetaan *Initialize suite* -avainsana. *Suite Teardown* -avainsana määrää, mitä tehdään, kun testi on ajettu läpi, ja tämä ajetaan aina riippumatta siitä, onko testi onnistunut. *Suite Teardownissa* ajetaan *Uninitialize suite* -avainsana. Default Tagsista nähdään, minkä tyyppinen testi on kyseessä. Tässä tapauksessa kyseessä on regression-testi. *Metadataan* on laitettu versionumero tälle testille. *Libraryssä* nähdään Python-kirjasto, jota käytetään testin aikana. *Variables*-kohdassa nähdään Python-tiedostot, joissa on vakiomuuttujille

määrätyt arvot. Näitä vakiomuuttujia yleensä käytetään sanomien muuttujien arvoja laitettaessa.

Metadata on osio, johon voidaan tallentaa tietoa testistä, esimerkiksi mistä testi on saatu ja mistä saa lisätietoa. *Metadata* tulee näkyviin myös Robot Frameworkin raporttiin ja lokitiedostoon. Tagit ovat mekanismeja, joilla saadaan lajiteltua Test Caset. Ne ovat vapaita tekstikenttiä ja niitä voidaan käyttää esimerkiksi tutkimalla raporteista ja lokeista, miten tietyillä tageilla olevat testit ovat läpäisseet testit. Tageilla voidaan nähdä onnistumiset yhtenä tilastona (pass, failed ja kaikki Test Caset yhdessä). Tagien avulla voidaan määritellä, millä nimellä olevat testit ajetaan ja mitkä jätetään huomioimatta. (Robot Framework User Guide version 2.6.2. 2011.)

TAULUKKO 6. Osa testin setting-asetuksista

Setting	Value			
Documentation	Changing of DAC value changes phase difference measurements to ascending or descending.			
Suite Setup	Initialize suite	# Initializations done before any testcase is executed See Helper Keywords for Test Cases chapter		
Suite Teardown	Uninitialize suite	# Uninitialization done after all testcases are executed See Helper Keywords for Test Cases chapter		
Default Tags	regression-test			
Metadata	version	0.1		
Library	robot/MessageHelpers.py			
Variables	env/bts/api_def.py			
Variables	env/bts/EStatus.py			

Variable-osiossa (taulukko 7) kerrotaan muuttujat ja arvot, joilla ne alustetaan. Opinnäytetyönä tehdyssä testissä oli käytössä neljä eri muuttujaa. Kun muuttujat esitellään *Variable*-osiossa, voidaan näitä käyttää missä testin vaiheessa tahansa. Tarkemmin muuttujista löytyy luvussa 4.1.

Muuttuja ${TEMP_MEAS_FILE} on alustettu *measurements.csv*-tekstillä. Se on tiedoston nimi, johon tallennetaan vaihe-eromittausten tulokset. Tarvittaessa

tämän tiedoston sisältöä pystytään katsomaan Robot Frameworkin lokitiedostosta.

Testin muuttuja @{measurement} on lista, jota ei ole alustettu millään arvoilla eli se on aluksi tyhjä lista. Muuttuja ${DAC_value} on alustettu arvolla 8192, johon tallennetaan testin aikana käytettävä DAC-arvo. Tätä DAC-arvoa käytetään *Api2SetDacValue*-sanomassa.

Toinen DAC-arvoa varten oleva muuttuja on ${First_DAC_value}. Tähän muuttujaan tallennetaan testin alussa DAC-arvo, jolla saadaan testin lopussa samanlainen vaikutus vaihe-eromittaukseen kuin testin alkaessa eli mittausarvot tulevat joko nousevana tai laskevana. Muuttujaan ${First_DAC_value} alkuperäisenä ideana oli tallentaa testin käynnistysvaiheessa systeemimoduulista senhetkinen DAC-arvo. Tutkittuani ja kyseltyäni asiasta selvisi, ettei DAC-arvoa pystytä saamaan ennen *Api2SetDacValue*-sanoman lähetystä. Ohjelmistossa on rajoitus, että DAC-sana pitää kirjoittaa vähintään kerran, ennen kuin se voidaan lukea *Api2GetDacValue*-sanomalla. Käytännössä tämä toimii siten, että ensin asetetaan DAC-arvo *Api2SetDacValue*-sanomalla ja sitten tarkistetaan se *Api2GetDacValue-sanomalla*.

TAULUKKO 7. Variable-osioon laitetut muuttujat

Variable	Value			
${TEMP_MEAS_FILE}	measurements.csv			
@{measurements}				
${DAC_value}	8192			
${First_DAC_value}	${EMPTY}			

Test Case -osiossa (taulukko 8) esitellään testitapaus ja sen sisältö. Test Caset ajetaan järjestyksessä ylhäältä alaspäin. Test Casea ei voida kutsua toisesta Test Casesta eivätkä niiden nimet voi olla samanlaisia. Taulukosta nähdään, miten testitapauksesta kutsutaan kahta avainsanaa, joissa lähetetään tieto vaihe-eromittauksen aloittamisesta.

TAULUKKO 8. Yksi testitapahtuma avainsanoineen

Test Case	Action	Arguments		
Start measuring phase difference				
	${EServiceState_Start} measuring			
	Wait for ${EServ-iceState_Start} measuring resp message			

Keyword-osiossa (taulukko 9) on itse tehdyt avainsanat testille. Näitä avainsanoja voidaan käyttää missä tahansa vaiheessa testiä ja niitä voidaan kutsua niin monta kertaa, kuin on tarvetta. Taulukosta 9 nähdään yksi esimerkki, miten avainsana rakennetaan. Avainsanalle annetaan parametrina DAC-arvo, jota käytetään *Api2SetDacValueResp*-sanoman value-parametrin tarkistuksessa. Parametri voidaan antaa joko argumenttina avainsanalle tai, kuten taulukossa 9 on tehty, parametri annetaan avainsanan nimessä. Tämän avainsanan tarkoituksena on hakea *Api2SetDacValueResp*-sanoma. Sanoman saavuttua tarkistetaan, että tulleet parametrit ovat oikein.

TAULUKKO 9. Keyword-osiosta sanoman vastaanoton rakenne

Keyword	Action	Arguments		
Set ${DAC} resp message	${Message}=	Receive first SICAP Message	Target	${API2_SET_DAC_VALUE_RESP_MSG}
	...	# 0x2C2A		
	${Response}=	Create SICAP ResponseMessage	Api2SetDacValueResp	${Message}
	...	Target		
	Should Be Equal As Integers	${Response.transactionId}	10	
	Should Be Equal As Integers	${Response.response.status}	${EStatus_NoError}	
	Should Be Equal	${Response.response.info}	not defined	
	Should Be Equal As Integers	${Response.value}	${DAC}	
	Should Be Equal As Integers	${Response.destination}	1	
	Should Be Equal As Numbers	${Response.frequency}	-1498789	
	Log Many	${Response.transactionId}	${Response.value}	${Response.destination}
	...	${Response.frequency}		

7.2 Testin toteutus

Testi aloitettiin tekemällä testipohja, jolla näin asetuksien oikeellisuuden ja sanoman lähetyksen ja sanoman vastaanoton onnistumisen. Todettuani testipohjan toimivuuden lisäsin aina yhden sanoman kerrallaan ja testasin jälleen toimivuuden. Testini käytti PPS-kellopulssia, joka tulee GPS-vastaanottimen kautta. Jos systeemimoduulissa ei ollut GPS-vastaanotin paikallaan, ei testi saanut PPS-kellopulssia ja tällöin testin *ApiSetActiveRefClkSource*-sanoma epäonnistui.

PPS-kellopulssin käyttöönottamiseksi vaihemittauksessa laitettiin *ApiSetActiveRefClkSource*-sanoman parametrina tieto PPS-kellopulssista. PPS-kellopulssin lisäämisen jälkeen otettiin tukiasemalta vaihe-eron mittausarvoja. Mittausarvot tulivat tukiasemalta PPS-kellopulssin tahdissa eli yhden sekunnin välein. Mittaustuloksien saaminen ja oikein toimiminen edellytti, että ensin lähetetään *ApiSetPhaseDiffMeas*-sanoma, jossa kerrotaan, että

mittaus aloitetaan. Aloittamisen jälkeen luetaan vaihe-eromittausten arvot ja sen perään lähetetään *ApiSetPhaseDiffMeas*-sanoma, jossa kerrotaan, että mittaus lopetetaan.

Mittauksen lopettamiseen tarkoitetun sanoman lähettämistä ei kannata unohtaa. Muuten systeemimoduulille tarkoitettu *ApiPhaseDiffMeasInd*-sanoma tulee väärällä hetkellä ja testin odottama sanomasekvenssi sekoaa vääristä sanomista. Tässä testissä mittaustuloksia vastaanotettiin jokaisella kerralla kymmenen kappaletta otettaessa mittaustuloksia systeemimoduulilta.

Mittaustulokset saatuani lisäsin testiin *Api2SetDacValue*-sanoman, jonka yhdellä parametrilla (DAC-value) sain muutettua systeemimoduulilta tulevia mittaustuloksia. *Api2SetDacValue*-sanomasta löytyy frequency-parametri, mutta sen muuttaminen ei vaikuttanut tämän testin toimivuuteen. Suurin virheeni tätä tehdessäni oli, että sanomaa rakentaessani olin laittanut kohteeksi flash-muistin enkä itse DACia, jolloin mittaustuloksia ei saanut muuttumaan, vaikka kuinka yritti.

Sanomilla oli monenlaisia parametreja, jotka kulkivat niiden mukana, jolloin oli tärkeää, että perehtyi jokaiseen sanomaan tarkasti. Yleensä parametreilla oli jokin tarkoitus sanomassa ja jos laittoi jonkin arvon väärin, ei testi toiminut enää halutulla tavalla.

Jokaisen testauksen jälkeen tutkin testin lokitiedoston ja Robot Frameworkin luoman raportin. Näistä tiedostoista pystyi näkemään testissä epäonnistuneet (FAIL) asiat ja pystyi myös seuraamaan, miten testi eteni ja tuliko hyväksytty (PASS) tulos oikealla toiminnalla. Lokitiedostoon tallentui myös vaihe-eromittausten tulokset, jotka esiteltiin luvussa 6.4.

7.3 Vaihe-eromittauslistojen käsittely

Taulukosta 10 nähdään, miten listalle tehtiin tallennettujen mittausarvojen tarkistus. Mittaustulosten saavuttua tallennettiin erilliseen listaan mittausarvot. Tämän jälkeen kutsuttiin *Check list and set new DAC word* -avainsanaa

(keyword). Kyseisessä avainsanassa tein listan tarkistuksen, jossa katsottiin, ovatko tulleet mittausarvot nousevia vai laskevia. Tässä vaiheessa listaan on tallennettu ensimmäiset kymmenen vaihe-eromittausarvoa. Listalta otettiin (*Get From List*) toinen ja kolmas arvo. Sen jälkeen verrattiin niitä toisiinsa if-ehtolauseessa (*Run keyword If*). Jos listan toinen arvo on suurempi kuin kolmas arvo, asetetaan *set DAC value* -avainsanassa DAC-arvoksi 8192, jolloin saadaan mittausarvot seuraavalla kerralla nousevaksi. Toisessa if-lauseessa tehdään vertailu toisin päin, jolloin DAC-arvoksi asetetaan 34 000 ja saadaan seuraavat mittausarvot laskeviksi.

TAULUKKO 10. Listalle tallennettujen mittausarvojen tarkistaminen ja DAC-arvon asettaminen muuttujaan

Check list and set new DAC word	[Documentation]	Cheking second and third value from the list.		
	${value}=	Get From List	${measurements}	2
	${value2}=	Get From List	${measurements}	3
	Run keyword If	${value} > ${value2}	Set DAC value to variable	8192
	...	34000		
	Run Keyword If	${value} < ${value2}	Set DAC value to variable	34000
	...	8192		

Taulukosta 11 nähdään, miten kahteen eri muuttujaan saatiin tallennettua DAC-arvot, joita myöhemmin testissä käytettiin. Tallennettava arvo tulee parametrina (*Arguments*) *set DAC value to variable* -avainsanalle. *Set Suite Variable* -avainsanan avulla laitettiin tullut arvo ${DAC_value}-muuttujaan ja ${First_DAC_value}-muuttujaan arvo, jolla saadaan vaihe-eromittausten suunta vaihtumaan päinvastaiseksi.

TAULUKKO 11. DAC-arvon asettaminen muuttujiin

Set DAC value to variable	[Arguments]	${value}	${value2}	
	Set Suite Variable	\${DAC_value}	${value}	
	Set Suite Variable	\${First_DAC_value}	${value2}	

8 TESTIN SUORITUS JA LOPPUTULOKSET

8.1 Testin suorittaminen

Testi suoritettiin Linuxin kautta komentotulkin avulla. Aluksi siirrettiin ohjelmat (.exe:t) systeemimoduulille, jonka jälkeen komentotulkin kautta käynnistettiin Robot Framework pybotilla, jolle annettiin parametrina kyseisen systeemimoduulin IP-osoite. IP-osoitteen avulla pystyttiin kontrolloimaan, että testi ajettiin oikeassa systeemimoduulissa.

Onnistunut testin suorittaminen näyttää kuvan 16 kaltaiselta.

```
==============================================================================
Phase difference measurement PPS from GPS :: Changing of DAC value changes...
==============================================================================
Active reference clock                                                | PASS |
------------------------------------------------------------------------------
Start measuring phase difference                                      | PASS |
------------------------------------------------------------------------------
Get measurements :: Taking 10 different measurements                  | PASS |
------------------------------------------------------------------------------
Stop measuring phase difference                                       | PASS |
------------------------------------------------------------------------------
Set and send Dac value                                                | PASS |
------------------------------------------------------------------------------
Getting DAC value                                                     | PASS |
------------------------------------------------------------------------------
Start measuring phase difference second time                          | PASS |
------------------------------------------------------------------------------
Get measurements second time                                          | PASS |
------------------------------------------------------------------------------
Stop measuring phase difference second time                           | PASS |
------------------------------------------------------------------------------
Check that phase diffenrence has changed :: Compare two parameters... | PASS |
------------------------------------------------------------------------------
Set and send Dac value second time                                    | PASS |
------------------------------------------------------------------------------
Getting DAC value second time                                         | PASS |
------------------------------------------------------------------------------
Start measuring phase difference third time                           | PASS |
------------------------------------------------------------------------------
Get measurements third time :: Taking 10 different measurements       | PASS |
------------------------------------------------------------------------------
Stop measuring phase difference third time                            | PASS |
------------------------------------------------------------------------------
Check that phase difference has changed second time:: Compare two ... | PASS |
------------------------------------------------------------------------------
Phase difference measurement PPS from GPS :: Changing of DAC value... | PASS |
16 critical tests, 16 passed, 0 failed
16 tests total, 16 passed, 0 failed
==============================================================================
Output:  /build/home/main2/trunk/output.xml
Report:  /build/home/main2/trunk/report.html
Log:     /build/home/main2/trunk/log.html
```

KUVA 16. Hyväksytty testi

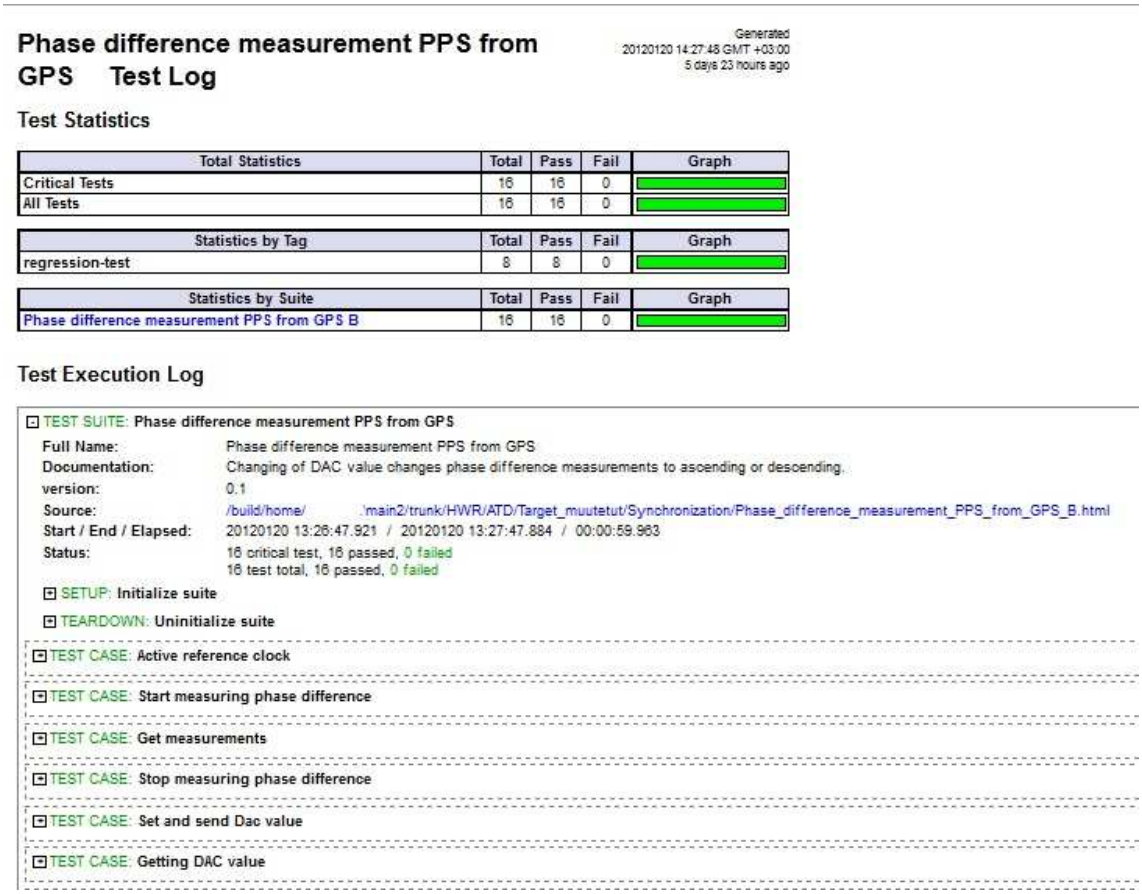
Kuvasta 17 nähdään, mitä tapahtuu, kun viimeisen vaihe-eromittauksen tarkistuksen aikana tulee virhe. Tarkistuksen aikana katsotaan, että tulleet mittausarvot ovat kääntyneet. Vertailussa 56 467 > 58 007 olisi ensimmäisen luvun pitänyt olla suurempi kuin jälkimmäisen. Tässä tapauksessa mittaustulosten olisi pitänyt vähentyä, mutta ohjelmassa on tapahtunut virhe ja mittaustulokset ovatkin tulleet kasvavana.

```
==============================================================================
Phase difference measurement PPS from GPS :: Changing of DAC value changes...
==============================================================================
Active reference clock                                                | PASS |
------------------------------------------------------------------------------
Start measuring phase difference                                      | PASS |
------------------------------------------------------------------------------
Get measurements :: Taking 10 different measurements                  | PASS |
------------------------------------------------------------------------------
Stop measuring phase difference                                       | PASS |
------------------------------------------------------------------------------
Set and send Dac value                                                | PASS |
------------------------------------------------------------------------------
Getting DAC value                                                     | PASS |
------------------------------------------------------------------------------
Start measuring phase difference second time                          | PASS |
------------------------------------------------------------------------------
Get measurements second time                                          | PASS |
------------------------------------------------------------------------------
Stop measuring phase difference second time                           | PASS |
------------------------------------------------------------------------------
Check that phase difference has changed :: Compare two parameters...  | PASS |
------------------------------------------------------------------------------
Set and send Dac value second time                                    | PASS |
------------------------------------------------------------------------------
Getting DAC value second time                                         | PASS |
------------------------------------------------------------------------------
Start measuring phase difference third time                           | PASS |
------------------------------------------------------------------------------
Get measurements third time :: Taking 10 different measurements       | PASS |
------------------------------------------------------------------------------
Stop measuring phase difference third time                            | PASS |
------------------------------------------------------------------------------
Check that phase difference has changed second time:: Compare two ... | FAIL |
'56467 > 58007' should be true
------------------------------------------------------------------------------
Phase difference measurement PPS from GPS :: Changing of DAC value... | FAIL |
16 critical tests, 15 passed, 1 failed
16 tests total, 15 passed, 1 failed
==============================================================================
Output:  /build/home/main2/trunk/output.xml
Report:  /build/home/main2/trunk/report.html
Log:     /build/home/main2/trunk/log.html
```

KUVA 17. Epäonnistunut testi

8.2 Testin lopputulokset ja raportti

Testi tuli toimivaksi ja systeemimoduulilta saatiin mittausraportti (kuva 12 sivulla 43 ja kuva 13 sivulla 44). Robot Framework luo sekä onnistuneesta että epäonnistuneesta ajokerrasta oman lokitiedostonsa. Testin onnistuttua muodostaa Robot Framework testistä kuvan 18 kaltaisen lokitiedoston. Onnistumisen merkkinä on vihreä väri TEST CASEssa ja TEST SUITEssa. Jos testi epäonnistuu, Robot Framework muodostaa muuten samanlaisen lokitiedoston, mutta epäonnistuneet TEST CASEt tulostuvat punaisena ja TEST SUITE -kohta on punainen.



Phase difference measurement PPS from GPS Test Log

Generated
20120120 14:27:48 GMT +03:00
5 days 23 hours ago

Test Statistics

Total Statistics	Total	Pass	Fail	Graph
Critical Tests	16	16	0	
All Tests	16	16	0	

Statistics by Tag	Total	Pass	Fail	Graph
regression-test	8	8	0	

Statistics by Suite	Total	Pass	Fail	Graph
Phase difference measurement PPS from GPS B	16	16	0	

Test Execution Log

TEST SUITE: **Phase difference measurement PPS from GPS**

Full Name:	Phase difference measurement PPS from GPS
Documentation:	Changing of DAC value changes phase difference measurements to ascending or descending.
version:	0.1
Source:	/build/home/ .main2/trunk/HWR/ATD/Target_muutetut/Synchronization/Phase_difference_measurement_PPS_from_GPS_B.html
Start / End / Elapsed:	20120120 13:26:47.921 / 20120120 13:27:47.884 / 00:00:59.963
Status:	16 critical test, 16 passed, 0 failed 16 test total, 16 passed, 0 failed

SETUP: **Initialize suite**

TEARDOWN: **Uninitialize suite**

TEST CASE: **Active reference clock**

TEST CASE: **Start measuring phase difference**

TEST CASE: **Get measurements**

TEST CASE: **Stop measuring phase difference**

TEST CASE: **Set and send Dac value**

TEST CASE: **Getting DAC value**

KUVA 18. Robot Frameworkiltä onnistuneen testin lokitiedosto

9 YHTEENVETO

Opinnäytetyön tarkoituksena oli tehdä testi Platform SW:n hyväksyntätestaukseen. Testin tarkoituksena oli DAC-arvoa muuttamalla säätää OCXOn taajuutta ja tutkia sen vaikutus vaihe-eromittauksiin. Tämän testin lisäksi perehdyin testausympäristöön ja tarkemmin vaihe-eromittaukseen.

Alun perin tarkoituksena oli tehdä monta eri testiä, mutta kävikin niin, että ne korvautuivat yhdellä kattavalla testillä ja siihen perehtymisellä. Pääsin testin myötä Robot Frameworkiin enemmän sisälle ja testistä tuli toimiva, jolloin se pystyttiin ottamaan hyväksyntätestaukseen mukaan. Testausympäristöstä pääsin hyvin käsitykseen, miten jokaisen osan pitää toimia ympäristössä. Systeemimoduulista ja sen osista tuli kerättyä paljon tietoa dokumenteista ja Nokia Siemens Networksin ohjaajalta.

Työn haastavimpana puolena oli tietoliikenteeseen liittyvien asioiden ymmärtäminen. Suuntautumiseni oli ohjelmistojen kehitys ja tietoliikenteen opintoja olen suorittanut vain yhden opintojakson verran. Opintojen kautta saadut pohjatiedot olivat tärkeitä, mutta opinnäytetyö vaati paljon itsenäistä opiskelua ja tiedon hakemista. Samalla se oli mielenkiintoinen ja opettavainen monessa eri suhteessa. Opinnäytetyön kautta saadut tiedot ja taidot ovat hyvä lähtökohta työelämään.

Opinnäytetyön aikana pääsin työskentelemään lähellä asiantuntijatiimiä, jonka työskentelytavat opettivat ketteriä menetelmiä käytännössä. Pääsin tutustumaan sekä kokeilemaan erityyppisiä testausmetodeita ja sen myötä opin ymmärtämään, miten ohjelmistojen testaus tapahtuu todellisessa ympäristössä.

Pidin aihealueestani todella paljon ja haluaisinkin päästä työelämässä enemmän käsiksi asiaan. Kiinnostukseni Robot Frameworkiin kasvoi opinnäytetyöni aikana ja sen myötä otinkin sen käyttöön myös omissa projekteissani kotona.

LÄHTEET

Adzic, Gojko 2009. Robot Framework review. Saatavissa: http://gojko.net/2009/05/28/robot-framework-review/. Hakupäivä 13.10.2011.

Alspaugh, Thomas 2011. Message Sequence Charts and Sequence Diagrams. Saatavissa: http://www.thomasalspaugh.org/pub/fnd/msc.html. Hakupäivä 4.11.2011.

Cundekovic, Vedran 2011. Common computer & support SW. Saatavissa: Nokia Siemens Networks intranet, vaatii salasanan. Hakupäivä 8.2.2012.

Flexi WCDMA BTS RF Module and Remote Radio Head description. 2009. Saatavissa: Nokia Simenes Networks intranet, vaatii salasanan. Hakupäivä 7.2.2012.

Granlund, Kaj 2001. Langaton tiedonsiirto. Jyväskylä: Docendo.

Haikala, Ilkka - Märijärvi, Ilkka 2006. Ohjelmistotuotanto. 11., uudistettu painos. Jyväskylä: Talentum.

Hiat, Trevor 2007. Standard sRIO Feature Use in Baseband, Part 1 of 3. Saatavissa: http://www.wirelessdesignmag.com/PDFs/2007/1207/wd712f2ol.pdf. Hakupäivä 29.12.2011.

Holma, Harri - Toskala, Antti 2009. LTE for UMTS - OFDMA and SC-FDMA based radio access. United Kingdom: John Wiley & Sons Ltd.

Härkönen, Janne 2011. Test Runner plugin. Saatavissa: https://github.com/robotframework/RIDE/wiki/Test-Runner-Plugin. Hakupäivä 21.10.2011.

Larman , Craig - Vodde, Bas 2010. Acceptance Test-Driven Development with Robot Framework. Saatavissa:

http://wiki.robotframework.googlecode.com/hg/publications/ATDD_with_RobotFramework.pdf. Hakupäivä 6.10.2011.

Lell, Stephan 2011. LTE eNB SW Architecture Training. Saatavissa: Nokia Siemens Networks intranet, vaatii salasanan. Hakupäivä 16.2.2012.

Wilborn, C 2012. What is Acceptance Testing?. Saatavissa: http://www.wisegeek.com/what-is-acceptance-testing.htm. Hakupäivä 23.2.2012.

Mikkola, Jani 2011. Senior Testing Engineer, Tieto Oyj. Haastattelu 26.11.2011.

Nokia Siemens Networks. Saatavissa: http://www.pestipaivat.fi/db/nayta_yritys.php?yritysid=28. Hakupäivä 15.9.2011.

Nokia Siemens Networks. 2012. Saatavissa: http://ru.nokiasiemensnetworks.com/portfolio/products/mobile-broadband/single-ran-advanced/flexi-multiradio-10-base-station. Hakupäivä 28.3.2012.

Poole, Ian a. GSM basic tutorial and overview. Saatavissa: http://www.radio-electronics.com/info/cellulartelecomms/gsm_technical/gsm_introduction.php. Hakupäivä 18.2.2012.

Poole, Ian b. OCXO, Oven Controlled Crystal Oscillator. Saatavissa: http://www.radio-electronics.com/info/data/crystals/ocxo.php. Hakupäivä 6.2.2012.

Poole, Ian c. OFDM Basic Tutorial. Saatavissa: http://www.radio-electronics.com/info/rf-technology-design/ofdm/ofdm-basics-tutorial.php. Hakupäivä 18.2.2012.

Rathenow, Markus 2011. CCSSA FPGA Functional and Architecture Specification. Saatavissa: Nokia Siemens Networks intranet, vaatii salasanan. Hakupäivä 5.1.2012.

Robot Framework User Guide version 2.6.2. 2011. Saatavissa: http://robotframework.googlecode.com/hg/doc/userguide/RobotFrameworkUserGuide.html?r=2.6.2. Hakupäivä 6.10.2011.

Scheppach, Günther 2009. FSM3 HW Architecture Specification. Saatavissa: Nokia Siemens Networks intranet, vaatii salasanan. Hakupäivä 12.1.2012.

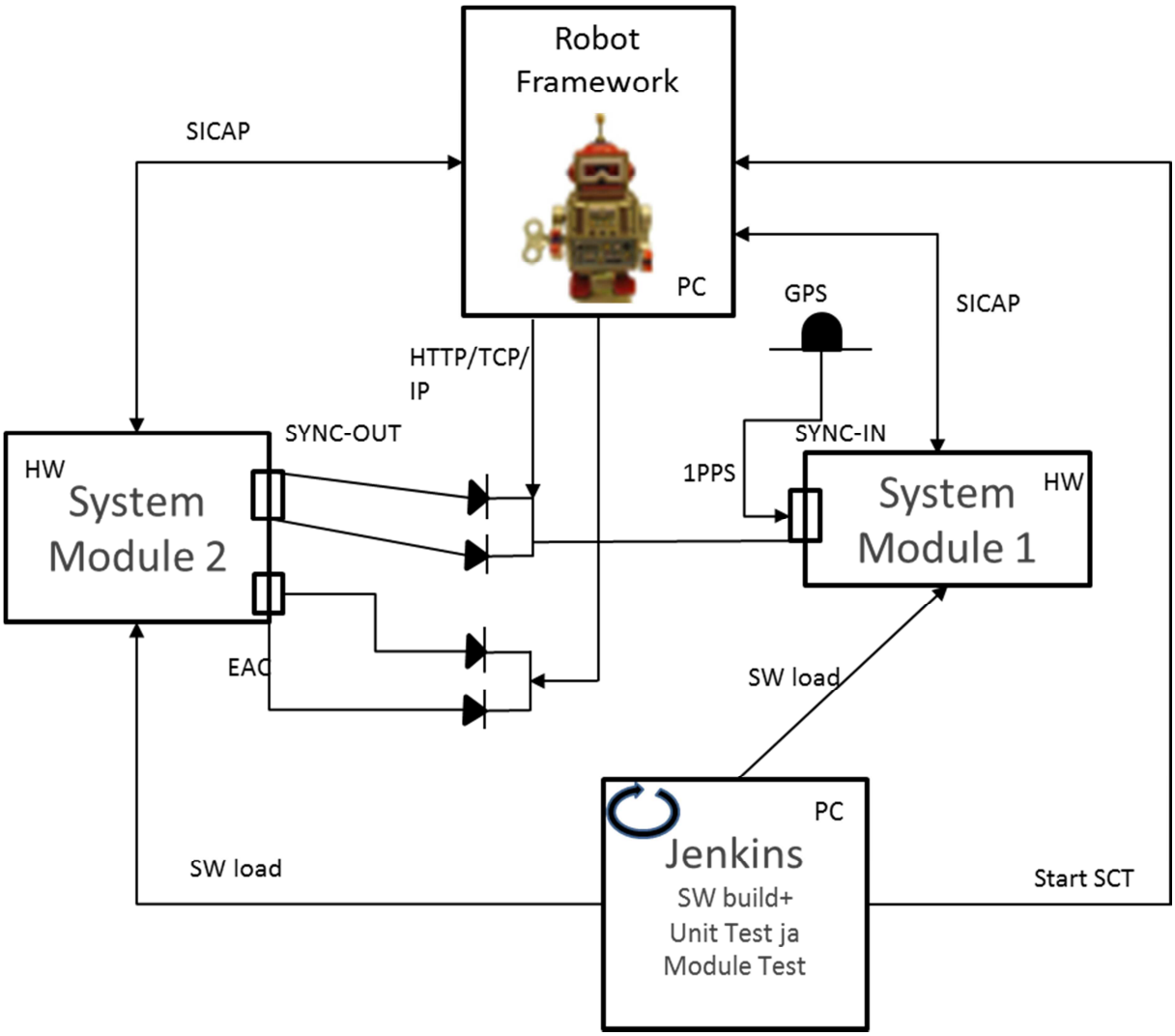
Software Business Competence. Saatavissa: http://www.oamk.fi/sbc/testaus/testaustasot.htm. Hakupäivä 26.1.2012.

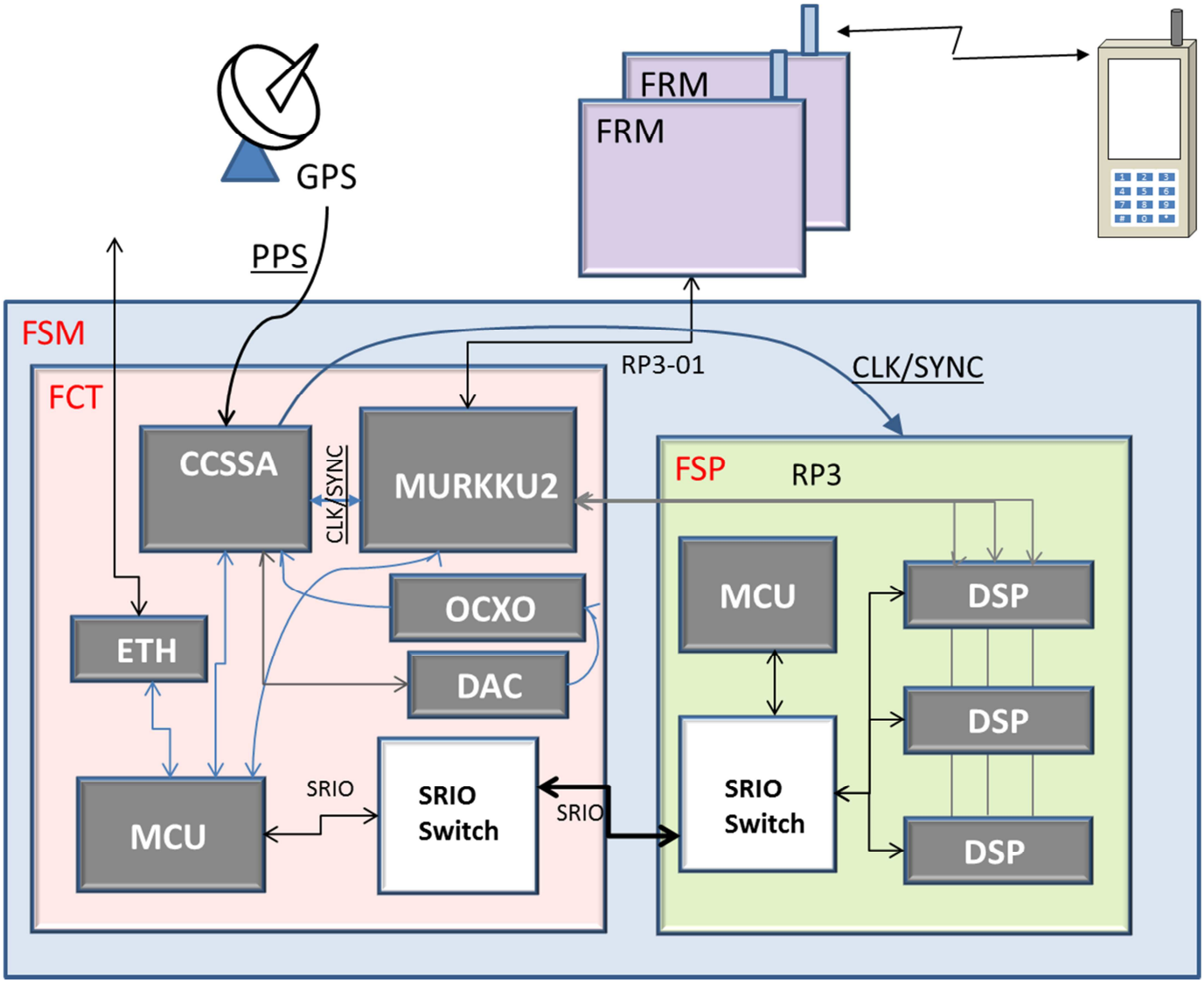
Tolvanen, Perttu 2011a. Ketteryys haltuun: Yleisimmät ketterät käytännöt. Saatavissa: http://www.meteoriitti.com/fi-FI/tiedotteet/ajankohtaista/ketteryys-haltuun-yleisimmat-ketterat-kaytannot. Hakupäivä 22.3.2012.

Tolvanen, Perttu 2011b. Ketteryys haltuun: Ketterän kehityksen yleiset periaatteet. Saatavissa: http://www.meteoriitti.com/fi-FI/tiedotteet/ajankohtaista/ketteryys-haltuun-ketteran-kehityksen-yleiset-periaatteet. Hakupäivä 22.3.2012.

Tukiasemat ovat matkapuhelinverkon solmukohtia 2012. Teknologiateollisuus. Saatavissa: http://www.teknologiateollisuus.fi/fi/ryhmat-ja-yhdistykset/faktat-tukiasemista.html. Hakupäivä 31.1.2012.

Wäljas, Ilmari 2011. FSM3.0 MURKKU2 ASIC user's guide. Saatavissa: Nokia Siemens Networks intranet, vaatii salasanan. Hakupäivä 23.3.2012.


Robot
Framework
PC
SICAP
SICAP
GPS
HTTP/TCP/
IP
SYNC-OUT
SYNC-IN
1PPS
HW
System
Module 2
System
Module 1
HW
EAC
SW load
PC
Jenkins
SW build+
Unit Test ja
Module Test
SW load
Start SCT


GPS
PPS
FRM
FRM
FSM
FCT
CCSSA
CLK/SYNC
MURKKU2
RP3-01
CLK/SYNC
FSP
RP3
ETH
OCXO
DAC
MCU
SRIO
SRIO Switch
SRIO
MCU
DSP
DSP
DSP
SRIO Switch

.h-tiedosto:

```
/**
 *  @brief Structure for signal API_SET_PHASE_DIFF_MEAS_REQ_MSG
 *
 *   Message is AaSysCom compatible. Earlier version in ClockService-
Messages.h was
 *  struct SApiSetPhaseDiffMeasReq.
 */

struct SApiSetPhaseDiffMeasReq
{
    TTransactionID       transactionId;       /**< Transaction Id for
this procedure. */
    EServiceState       state;                /**< Service state. */
    SMessageAddress       receiver;              /**< Receiver of the
API_PHASE_MEAS_IND_MSG. */
};
typedef struct SApiSetPhaseDiffMeasReq SApiSetPhaseDiffMeasReq;
```

Python-tiedosto:

```
class ApiSetPhaseDiffMeasReq(AaSysComMsg.AaSysComMsgPayload):
    '''
    struct SApiSetPhaseDiffMeasReq
    {
    TTransactionID       transactionId;       /**< Transaction Id for
this procedure. */
    EServiceState       state;              /**< Service state. */
    SMessageAddress       receiver;              /**< Receiver of the
API_PHASE_MEAS_IND_MSG. */
    };
    '''

    def __init__(self, msg = None):
        self.msgId = API_SET_PHASE_DIFF_MEAS_REQ_MSG
        self.transactionId = 0
        self.state = 0
        self.receiver  =  MessageAddress.SMessageAddress()  #  Create
placeholder for receiver information so that user does not need to do
it
        AaSysComMsg.AaSysComMsgPayload.__init__(self, msg)
        self._format = "=8" # We have non-32bit fields in the message
so use native-8bit format


    def pack(self):
        self.appendU32(self.transactionId)
        self.appendU32(self.state)
        self.appendMsg(self.receiver)
        return self._payload

    def unpack(self, msgPayload):
        self.transactionId = self.getU32(msgPayload, 0)
        self.state = self.getU32(msgPayload, 4)
        self.receiver = MessageAddress.SMessageAddress()
        self.receiver.doUnpack(msgPayload, 8, 4)
```

Phase difference measurement PPS from GPS

Setting	Value			
Documentation	Changing of DAC value changes phase difference measurements to ascending or descending.			
Suite Setup	Initialize suite	# Initializations done before any testcase is executed See Helper Keywords for Test Cases chapter		
Suite Teardown	Uninitialize suite	# Uninitialization done after all test-cases are executed See Helper Keywords for Test Cases chapter		
Default Tags	regression-test			
Metadata	version	0.1		
Library	robot/MessageHelpers.py			
Variables	env/bts/api_def.py			
Variables	env/bts/EStatus.py			
Variables	env/ps/mcu/EServiceState.py			
Variables	env/ps/mcu/EReferenceClock.py			
Resource	../common/Target_Connections.html	# Contains keywords for handling target		
Resource	../common/SysCom_Connections.html	# Contains keywords for common SysCom usage		
Library	robot/PostOffice.py	${TARGET_IP1}	${TESTPORT_PORT}	${CCS_DAEMON_ARGS_NODE_ID}
...	ApiClockServiceMsgs	WITH NAME	Target	

-
-
-

Variable	Value			
${TEMP_MEAS_FILE}	measurements.csv			
@{measurements}				
${DAC_value}	8192			
${First_DAC_value}	${EMPTY}			

Test Case	Action	Arguments		
Active reference clock	Select active reference clock to be GPS PPS			
	Run Keyword And Ignore Error	Wait for API_REFCLK_ACTIVITY_IND	${EReferenceClock_Pps}	${EStatus_Active}
	...	Target	# Arguments: refclk, Status, Target	

-
-
-

Get measurements	[Documentation]	Taking 10 different measurements		
	Receive 10 measurement raport			
	Check list and set new DAC word			
Stop measuring phase difference				
	${EServiceState_Stop} measuring			

	Wait for ${EServ-iceState_Stop} measuring resp message			
Set and send Dac value				
	Set ${DAC_value} req mes-sage			
	Set ${DAC_value} resp mes-sage			
Getting DAC value	Get DAC req message			
	Get ${DAC_value} resp message			

-
-
-

Check that phase differ-ence has changed	[Documentation]	Compare two parameters from first 10 measurements to same two paramenters from next 10 measurements (second list).		
	${value}=	Get From List	${measurements}	2
	${value2}=	Get From List	${measurements}	3
	Run keyword If	${value} > ${value2}	Check list	<
	...	12	13	
	Run Keyword If	${value} < ${value2}	Check list	>
	...	12	13	

-
-
-

Keyword	**Action**	**Arguments**		
Initialize suite	[Documentation]	Initializations done before any testcase is executed		
	Open PostOffice Connection	Target		
	Set AaConfig Rad Parameter	Target	${ERadHwapi_PrintClock Service}	1
	Log To Target With Timestamp	Target	>>>>> Running test suite '${SUITE NAME}' >>>>>	
Uninitialize suite	[Documentation]	Uninitialization done after all testcases are executed		
	Log To Target With Timestamp	Target	<<<<< Test suite '${SUITE NAME}' com-pleted <<<<<	
	Cleanup SICAP Message Receiving Queue	Target		
	Close PostOffice Connection	Target		
	Get Syslog From Target	${TARGET_IP1}	${OUTPUT_DIR}	
	Log File	${SYSLOG_FILE}		
	Log File	${TEMP_MEAS_FILE}		
	Log message trace			
Select active reference clock to be GPS PPS	${request}=	Create SICAP Message	Target	ApiSetActiveRefClkSourc eReq
	Assign Variable As Integer	${request}	transactionId	0x00110000
	Assign Variable As Integer	${request}	refClk	${EReferenceClock_Pps}
	Assign Variable As Integer	${request}	measInstance	0
	Log	${request}		
	Send SICAP Message	Target	${HWAPI_CLOCK_SERV ICE}	${request}

	${response}=	Receive SICAP Message	Target	ApiSetActiveRefClkSourceResp
	Log	${response}		
	Should Be Equal As Integers	${response.response.status}	${EStatus_NoError}	
Wait for API_REFCLK_ACTIVITY_IND	[Arguments]	${refclk}	${status}	${Office}
	[Documentation]	Waits for API_REFCLK_ACTIVITY_IND for given reference clock and ignores other reference clocks.		
	Wait Until Keyword Succeeds	10 secs	1 secs	Receive API_REFCLK_ACTIVITY_IND and check refclk and status
	...	${refclk}	${status}	${Office}
Receive API_REFCLK_ACTIVITY_IND and check refclk and status	[Arguments]	${refclk}	${status}	${Office}
	[Documentation]	CLOCK receives API_REFCLK_ACTIVITY_IND_MSG message from HWAPI		
	${Request}=	Receive First SICAP Message	${Office}	${API_REFCLK_ACTIVITY_IND_MSG}
	${ind}=	Create SICAP ResponseMessage	ApiRefclkActivityInd	${Request}
	...	${Office}		
	Should Be Equal As Integers	${ind.refClk}	${refclk}	
	Should Be Equal As Integers	${ind.status}	${status}	
	Log	${ind}		
	[Return]	${ind}		

-
-
-

Check list	[Arguments]	${merkki}	${index}	${index2}
	${value}=	Get From List	${measurements}	${index}
	${value2}=	Get From List	${measurements}	${index2}
	Should Be True	${value} ${merkki} ${value2}		
${serviceState} measuring	${request}=	Create SICAP Message	Target	ApiSetPhaseDiffMeasReq
	Assign Variable As Integer	${request}	transactionId	10
	Assign Variable As Integer	${request}	state	${serviceState}
	Assign Variable As Integer	${request}	receiver.board	0x10
	Assign Variable As Integer	${request}	receiver.cpu	0x11
	Assign Variable As Integer	${request}	receiver.task	${TASK_API_BROWSER}
	Send SICAP Message	Target	${HWAPI_CLOCK_SERVICE}	${request}
Wait for ${current_state} measuring resp message	${Message}	Receive first SICAP Message	Target	${API_SET_PHASE_DIFF_MEAS_RESP_MSG}
	...	# 0x2C25		
	${Response}=	Create SICAP ResponseMessage	ApiSetPhaseDiffMeasResp	${Message}
	...	Target		
	Should Be Equal As Integers	${Response.transactionId}	10	

	Should Be Equal As Integers	${Response.response.status}	${EStatus_NoError}	
	Should Be Equal As Integers	${Response.refClk}	${EReferenceClock_Pps}	
	Should Be Equal As Integers	${Response.currentState}	${current_state}	
Receive ${count} measurement raport	: FOR	${i}	IN RANGE	${count}
		${Message}=	Receive SICAP Message	Target
	...	ApiPhaseDiffMeasInd		
		Should Be Equal As Integers	${Message.refClk}	${EReferenceClock_Pps}
		Should Be Equal As Integers	${Message.numOfItems}	1
		Should Be Equal As Integers	${Message.response.status}	${EStatus_NoError}
		Log	${Message}	
		@{test}=	Set Variable	${Message.value}
		Append To List	${measurements}	@{test}
Set ${DAC} req message	${request}=	Create SICAP Message	Target	Api2SetDacValueReq
	Assign Variable As Integer	${request}	transactionId	10
	Assign Variable As Integer	${request}	value	${DAC}
	Assign Variable As Float	${request}	frequency	-1498789
	Assign Variable As Integer	${request}	destination	1
	Send SICAP Message	Target	${HWAPI_CLOCK_SERVICE}	${request}

-
-
-

Get DAC req message	${request}=	Create SICAP Message	Target	Api2GetDacValueReq
	Assign Variable As Integer	${Request}	transactionId	556456
	Assign Variable As Integer	${Request}	source	1
	Send SICAP Message	Target	${HWAPI_CLOCK_SERVICE}	${Request}
Get ${DAC} resp message	${Message}	Receive First SICAP Message	Target	${API2_GET_DAC_VALUE_RESP_MSG}
	...	# Message id 0x2C28		
	${Response}=	Create SICAP ResponseMessage	Api2GetDacValueResp	${Message}
	...	Target		
	Should Be Equal As Integers	${Response.transactionId}	556456	
	Should Be Equal As Integers	${Response.response.status}	${EStatus_NoError}	
	Should Be Equal	${Response.response.info}	not defined	
	Should Be Equal As Integers	${Response.value}	${DAC}	
	Should Be Equal As Integers	${Response.source}	1	