Heikki Aho

FPGA- piirin testaus SystemVerilogilla

Opinnäytetyö

KESKI-POHJANMAAN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma

Lokakuu 2009

KESKI-POHJANMAAN AMMATTIKORKEAKOULU
MELLERSTA ÖSTERBOTTENS YRKESHÖGSKOLA

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Yksikkö	**Aika**	**Tekijä/tekijät**
Ylivieskan yksikkö	lokakuu 2009	Heikki Aho
Koulutusohjelma Tietotekniikka		
Työn nimi **FPGA-testaus SystemVerilogilla**		
Työn ohjaaja Joni Jämsä		**Sivumäärä** 31
Työelämäohjaaja Mika Pahkasalo		

Toimeksiantajana opinnäytetyössä oli CENTRIA, Keski-Pohjanmaan Ammattikorkeakoulun tutkimus- kehitys- ja täydennyskoulutusyksikkö. Työn aiheena oli FPGA- testaus SystemVerilogilla. Tavoitteena oli tutkia SystemVerilog-standardin ominaisuuksia ja esimerkkien avulla hahmottaa lukijalle FPGA-testauksen sekä OVM- testausjärjestelmän toimintaa. Lukijalle selvitettiin myös käytettyjen työkalujen toiminta. Työstä käy ilmi SystemVerilog- standardin tehokkuus mutta myös standardin ongelmat. Standardin ongelmat liittyvät käytännön toteutuksiin sen käyttöönotossa, erityisesti siihen liittyvän tiedon jakelussa.

Asiasanat

FPGA, SystemVerilog, testaus

CENTRAL OSTROBOTHNIA
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

CENTRAL OSTROBOTHNIA UNIVERSITY OF APPLIED SCIENCES	**Date** October 2009	**Author** Heikki aho

Degree programme Information Technology	
Name of thesis FPGA- verification using SystemVerilog language	
Instructor Joni Jämsä	**Pages** 31
Supervisor Mika Pahkasalo	

The mandator of this thesis was Centria, research and development unit of Central Ostrobothnia University of Applied Sciences. The subject of the thesis was study FPGA-verification using SystemVerilog- language. My goal was to observe features of the SystemVerilog standard and clarify to the reader the opreation of the FPGA verification and OVM verification system. The reader was also told the operation of the tools being used. Thesis shows effectiveness of SystemVerilog- standard, but also reveal some problem with standard. Problems are related to the practical implementation of its deployment, especially in the distribution of related information.

Key words

FPGA, SystemVerilog, verification

LYHENTEET

OVM	Open Verification Methodology
AVM	Advanced Verification Methodology
VHDL	Very high speed integrated circuit Hardware Description Language
FPGA	Field-Programmable Gate Array
HDL	Hardware Design Language
HDVL	Hardware Design and Verification Language
PSL	Process Specification Language
DPI	Direct Programming Interface
TLM	Transaction Level Modeling

1. JOHDANTO

Tämän opinnäytetyön tavoitteena on tutkia SystemVerilog- ohjelmointikielen käyttöä ohjelmoitavien logiikkapiirien testaukseen. Tutkimus tehdään CENTRIA:lle, Keski-Pohjanmaan Ammattikorkeakoulun tutkimus- kehitys- ja täydennyskoulutusyksikölle. Tarkoituksena on antaa lukijalle yleiskuva SystemVerilog- kielen ominaispiirteistä sekä kahden yksinkertaisen esimerkin avulla esitellä sen ominaisuuksia testauksessa. Koska OVM- testausympäristö kuuluu SystemVerilog- standardiin , työssä tutkitaan yleisluontoisesti myös OVM:n ominaisuuksia. Kyseinen aihepiiri on hyvin laaja, joten tässä opinnäytetyössä ei tarkastella SystemVerilog- kielen syntaksia yksityiskohtaisesti vaan lukija hahmottakoon sen oheisista esimerkeistä sekä erillisistä oppikirjoista. C/C++-kielen ja/tai Verilogia osaavalla lukijalla on tuskin suuria vaikeuksia lukea esimerkeissä käytettyä ohjelmakoodia niiden syntaksin samankaltaisuuden takia.

Esimerkkiohjelmien kääntämisessä ja simuloimisessa käytetään kahden SystemVerilogia kehittäneen valmistajan ohjelmaa. Nämä ovat Cadencen IUS 8.2 ja Mentor Graphicsin Modelsim PE 6.5a opiskelijaversiota. Modelsim tukee SystemVerilogia joiltain osin, mutta esimerkiksi OVM:n kääntäminen ei onnistu sillä. Rajoitukset johtunevat enimmäkseen lisensoinnista, ilmaisen opiskelijaversion ominaisuuksia on karsittu melkoisesti. Cadencen kääntäjä sen sijaan toimii OVM:nkin kanssa moitteettomasti. Jälkimmäinen esimerkki on käännetty Cadencen ohjelmalla. Ongelmana siinä onkin vain maksullinen lisenssi.

Lähteinä tätä tutkimusta tehdessä käytin Chris Spearin kirjaa SystemVerilog for Verification, internet- julkaisuja sekä työkalujen mukana tulevia käyttöohjeita. Mentor Graphics on tehnyt paljon yhteistyötä Douloksen kanssa standardin koulutuksesta. Douloksen sivut ovatkin erinomainen paikka tutustua SystemVerilog- standardin ominaisuuksiin. Intialainen insinööri Deepak Kumar Tala taas keskittyy ylläpitämällään sivustolla enemmän SystemVerilog- kielen syntaksiin. Sivuilla on paljon selkeitä esimerkkejä varsinaiseen ohjelmointivaiheeseen. OVM on kehittynyt AVM-testausohjelmistosta ja näin ollen sen toiminnan perusperiaatteita voi opiskella myös Mentor Graphicsin Advanced Verification Cookbook:ista

2. SYSTEMVERILOG

Logiikkapiirien ohjelmointiin suunniteltu Verilog- kieli kehitettiin 1990-luvulla, ja sitä on päivitetty useita kertoja vastaamaan logiikkapiirien kehitystä. Tuorein päivitys Verilog-kieleen tehtiin vuonna 2005, jolloin siitä erkani omaksi kielekseen SystemVerilog.

SystemVerilog on standadoitu IEEE- järjestön toimesta numerolla 1800-2005. Kieli on hyvin pitkälle Verilog-kielen syntaksia, mutta siihen on sisällytetty ominaisuuksia myös muista HDL – ja ohjelmointikielistä. SystemVerilogiin liitetään termi Hardware Design and Verification Language (HDVL), se on siis suuniteltu varsinaiseen logiikkapiiriohjelmointiin mutta myös ohjelmistojen ja piirien testaukseen.(Hamzani, 2008).

2.1 SystemVerilog -kielen ominaisuudet

SystemVerilog lainaa ominaisuuksia mm. Verilogin kaikista standardeista, SuperLogista, VHDL:stä PSL:stä, C/C++-kielistä sekä Vera-kielestä. VHDL- ja Java-kielten package-tyyppiset ominaisuudet sekä ennen kaikkea luokat mahdolistavat tehokkaan oliopohjaisen ohjelmoinnin. Monipuolisuutta taulukkorakenteisiin tuo VHDL:stä tutut moniulotteiset taulukot sekä dynaamisesti laajenevat taulukot ja assosiatiiviset taulukot. C++- kielen String -luokan tyyppiset oliot sekä rajattu satunnaislukujen generointi monipuolistavat ohjelmointia. Standardi mahdollistaa muuttujien ja ominaisuuksien tallentamisen erilliseen rakenteeseen, josta ohjelmiston eri osat voivat niitä kutsua. Tämä vähentää virheiden mahdollisuuksia kutsuttaessa suuria määriä signaaleja tai toiminnallisuuksia. (Spear, 2008)(Aynsley, 2009).

SystemVerilog- kieli on oliopohjainen kieli. Oliopohjaisuudesta hyödytään erityisesti suurissa ohjelmointiprojekteissa, jonka eri osa-alueita suorittavat erilliset ryhmät, kuten vaikkapa suunnitteluryhmä ja testausryhmä. Jokaisen ryhmän jäsenet ovat useinmiten oman ryhmänsä osa-alueen asiantuntijoita, mutta osaaminen toisen ryhmän työstä saattaa olla puutteellinen. Jos ohjelmisto kirjoitetaan oliopohjaisella kielellä, voidaan projekti helpommin pilkkoa osakokonaisuuksiin joita ryhmät voivat kehittää. Tällöin esimerkiksi

testausryhmän on helpompi työskennellä suunnitteluryhmän kanssa projektin eri vaiheissa.(Spear, 2008).

2.2 DPI (Direct Programming Interface)

C- kielellä ja SystemVerilogilla on kummallakin omat vahvuutensa ja käyttötarkoituksensa logiikkapiirien ohjelmoinnissa. Jotta nämä vahvuudet saataisiin monipuolisesti käyttöön, tarvitaan helppokäyttöinen rajapinta kielten välille. DPI on SystemVerilog-standardiin lisätty ohjelmointirajapinta, jonka tarkoituksena on yhdistää SystemVerilog-kieli ja muut HDL -kielet keskenään toimivaksi ohjelmistokokonaisuudeksi. Tällä hetkellä DPI-rajapinta on rakennettu toimivaksi C-kielellä. Rajapinnan luvataan tukeva nmyös muita ANSI-C- standardin kieliä.

DPI:n toimintaperiaate on yksinkertainen. C-kielen standardin mukaisesti nimettyjä SystemVerilogin task-lohkoja tai funktioita ja C-kielen funktioita voidaan kutsua rajapinnan kautta riippumatta siitä missä ne on esitelty tai mistä niitä kutsutaan. Rajapinta osaa linkittää task-lohkojen ja funktioiden nimet ja sijainnit keskenään. Siirrettävien tiedostotyyppien täytyy tietenkin olla yhteensopivia näissä funktioissa. (Edelman, 2009)(Project VeriPage, 2009/2).

SystemVerilog- tietotyypit	C- tietotyypit
Byte	Byte
Int	Int
Longint	Long long
Shortint	Short int
Real	Double
Shortreal	Float

TAULUKKO 1. SystemVerilogin ja C- kielen yhteensopivat tietotyypit (Doulos, 2009).

2.3 SystemVerilogin vertailut

Kuten Verilog-kielessä, SystemVerilog- kielessä ohjelmoija voi käyttää tavallisen SystemVerilog- syntaksin ohella vertailuja. Näillä lyhyillä ohjelmilla koodin sisällä voidaan suorittaa vertailuja muuttujien välillä, tehdä laskutoimituksia ja tulostaa käyttäjälle viestejä esimerkiksi ohjelman suorituksen tilasta tai tietoa jonkin muuttujan sisällöstä. Vertailuja voivat käyttää suunnittelijat osana ohjelman rakennetta sekä testaajat osana testiympäristön rakennetta.

SystemVerilog- standardi tukee kolmea erityyppistä vertailurakennetta. Yksittäisiä vertailuja voidaan suorittaa simuloinnin edetessä ohjelmalohkon kyseisessä vaiheessa. Vertailu sisältää ehdon, jonka täytyy olla totta jotta virheilmoitusta ei näytetä simuloinnin aikana.

Esimerkissä vertaillaan onko muuttujan1 sisältö sama kuin muuttujan2 sisältö. Jos muuttuja1 eroaa muuttuja2:sta, tulostetaan virheilmoitus:

```
assert (muuttuja1==muuttuja2);
```

Käyttäjä voi tehdä virheen tulostuksen myös käsin:

```
assert (muuttuja1==muuttuja2); $display(”muuttuja1 on yhtä kuin muuttuja2”);
```

Vertailua voidaan täydentää else- rakenteella, jolloin voidaan tulostaa jo hyvin monipuolisia virheilmoituksia jolloin suunnittelija voi paikantaa virheen tarkemmin:

```
assert (muuttuja1==muuttuja2); $display(”muuttuja1 on yhtä kuin muuttuja2”);
    else $error(”muuttuja1 ei ole yhtä kuin muuttuja2”);
```

Vertailujen suoritushetken voi määrätä myös jokin ulkoinen signaali, esimerkiksi kellosignaali.Tällöin vertailu suoritetaan vaikkapa jokaisella kellon nousevalla reunalla.

```
assert property (@(posedge Clock) muuttuja1 == muuttuja2);
    else $error(”muuttuja1 ei ole yhtä kuin muuttuja2);
```

Näiden lisäksi voidaan kirjoittaa vertailuja, jotka suoriutuvat peräkkäin niin, että suoritusten välillä lasketaan tietty määrä kellosignaalin nousevia reunoja. Tällä estetään tietyissä tapauksissa viiveen aiheuttamat ongelmat käytännön toteutuksessa.(Project VeriPage, 2009/1) (Doulos, 2009)(Deepak, 2009)

SystemVerilog- standardi määrittelee sequence- lohkon, jolla tapahtumia ja toisia sequence-lohkoja voidaan ketjuttaa keskenään. SystemVerilogin ##- operaattorin avulla voidaan näiden lohkojen välille luoda viivettä. Esimerkiksi käsky ##3 luo tapahtumien välille 3 kellosignaalin nousevan reunan verran viivettä. Näin vertailuja voidaan ketjuttaa suoritettavaksi peräkkäin niin, ettei niiden toimintaa tarvitse sitoa varsinaiseen kellosignaaliin vaan ne suoriutuvat ohjelman toiminnan edetessä.

```
Sequence jakso1;
 ##3 jakso2;
ensequence

sequence jakso3:
##5 jakso1;
endsequence
```

##- operaattoria voidaan käyttää hyvin monella eri tavoin. Yllä olevassa esimerkissä jakso1 suorittaa jakso2:n kolmen kellosignaalin nousevan reunan jälkeen. Jakso3 taas suorittaa jakso1:n 5 nousevan kellosignaalin jälkeen, joten jos jakso 3 suoritetaan ensin, suoritetaan jakso2 8 nousevan kellosignaalin jälkeen. Viiveitä voidaan myös ketjuttaa samaan lohkoon seuraavalla tavalla:

```
jakso3  ##5  jakso1  ##3  jakso2;
```

Viiveen voi asettaa myös käyttämällä muuttujaa kiinteän kokonaisluvun sijasta:

```
jakso3  ##viive1  jakso1  ##viive2  jakso2;
```

Tällöin voidaan ohjelman sisällä esimerkiksi tutkia jonkin osa-alueen viivettä ja korjata ohjelman toimintaa kesken suorituksen muuttamalla toimintojen välisiä viiveitä. Viive voidaan antaa myös viiveen alueina.

Jakso3 ##[1:3] jakso1 ##[viive1:viive2] jakso2;

Tai avoimena alueena, jolloin jakso suoritetaan kun tietyt kellosignaalin nousevat reunat on laskettu

jakso3 ## [1:$] jakso1 ## [viive3:$] jakso2;

Jaksot voidaan määrätä toistumaan *- operaattorilla seuraavalla tavalla yhden nousevan kellosignaalin välein.

jakso3 ##5 jakso1 [*3] ##3 jakso2; tarkoittaa samaa kuin

jakso3 ##5 (jakso1 ##1 jakso1 ##1 jakso1) ##3 jakso2;

Myös *-operaattorille voidaan määritellä toistorajat, jolloin esimerkiksi jakso3 [*1:5] voi tarkoittaa joko jakso3 [*1] tai jakso3[*5]. Tässä tapauksessa voidaan käyttää ylärajana $-merkintää, jolloin toistoja on yksi tai useampia.

Goto- operaattorilla [→] voidaan tarkastaa, kuinka monta kertaa jonkun jakson vertailu on ollut totta:

muuttuja1 [→ 2] muuttuja2; siis toteuttaa muuttuja2:n, kun muuttuja1 on ollut kaksi kertaa totta.

muuttuja1 [→ 2:4] muuttuja2; toteuttaa muuttuja2:n kun muuttuja1 on ollut totta 2, 3 tai 4 kertaa.

SystemVerilogin vertailut voidaan sijoittaa osaksi ohjelmakoodin rakennetta, jolloin ne

suoritetaan normaalisti koodin ajon yhteydessä. Ne voidaan myös kirjoittaa erilliseksi ohjelmalohkoksi jolloin ne suoritetaan rinnan varsinaisen ohjelmakoodin kanssa. Vertailut voidaan lisäksi kirjoittaa erilliseen tiedostoon josta niitä voi kutsua varsinaisesta ohjelmakoodista. (Cohen, Venkataramanan & Kumari, s. 48-50, 2005).

3 FPGA-OHJELMISTOJEN TESTAUS

3.1 Yleistä FPGA- ohjelmistojen testauksesta

Fpga-ohjelmistojen testaus eroaa suunnittelusta melkoisesti, ja usein ohjelmien testaamiseen onkin koottu kokonaan erilliset työryhmät. Yksinkertaisin testaamisen muoto on suora toiminnallisuuden testaaminen. Piirille syötetään käyttäjän toimintaa matkivia sekvenssejä ja ulostuloista tarkistetaan kuinka se käytännössä toimii.(Spear, 2008).

Tällainen testaustapa on tehokasta ja nopeaa yksinkertaisilla ohjelmistoilla, mutta ohjelman vaatimusten kasvaessa myös vaatimukset testauksen monipuolisuudesta kasvavat. Jossain vaiheessa saavutetaan raja, jolloin testausinsinööri ei enää kykene ottamaan huomioon kaikkien ominaisuuksien tarvitsemia testisekvenssejä. Tällöin on turvauduttava satunnaislukujen avulla testaamiseen.

Testaussuunnitelmassa määritetään testauksen kattavuus ja satunnaisluvuille raja-arvot joiden sisällä testaus suoritetaan. Testipenkki hoitaa tämän jälkeen satunnaislukujen syöttämisen testattavaan ohjelmaan, testaustulosten näyttämisen sekä pitää huolen testin kattavuudesta. Tässä testaustyylissä suurin haaste testaajalle on määritellä sopivat raja-arvot ja kattavuus testille. Liian suuri skaala vie resursseja ja aikaa sekä itse testausvaiheessa että testitulosten tarkastelussa. Liian kapea testauksen skaala taas heikentää testin luotettavuutta.

Usein myös käytetään näiden kahden mainitun testaustavan yhdistelmää. Ensin ohjelman toiminta varmistetaan suoralla sisääntulojen testauksella jolloin itse ohjelman toimintaan liittyvä suunnitteluvirheet saadaan kitkettyä pois. Tämän jälkeen aloitetaan testit

satunnaisluvuilla, joita ajetaan kunnes testaussuunnitelmassa määritetty toimivan ohjelman kriteerit saavutetaan.

Mahdollisuus testiohjelmien uudelleenkäyttöön on tärkeää, kun testausryhmä työskentelee usein monimutkaisten ohjelmistojen kimpussa. Usein ratkaistavat ongelmat ovat samankaltaisia kuin jossain aikaisemmassa projektissa, jolloin arkistoista löytyvät oikein dokumentoidut ja uudelleenkäyttöä varten suunnitellut testipenkit helpottava testausta huomattavasti. Testaukseen varta vasten kehitetyt yleisimmät assertionpohjaiset testausohjelmistot kuten OVM ja AVM ovat suunniteltu uudelleenkäytettävyyttä silmällä pitäen. Uudelleenkäytettävyyteen vaikuttaa selkeä modulaarinen rakenne. Kaikki testausohjelmiston asetusarvot jotka vaikuttavat kyseisen ohjelmiston toimintaan, on sijoitettu niille tarkoitettuihin ja erikseen määriteltyihin moduuleihin, joita muokkaamalla testipenkki saadaan toimintakuntoon. (Doulos, 2008)(Mentor Graphics, 2008).

3.2 Testausohjelmistot

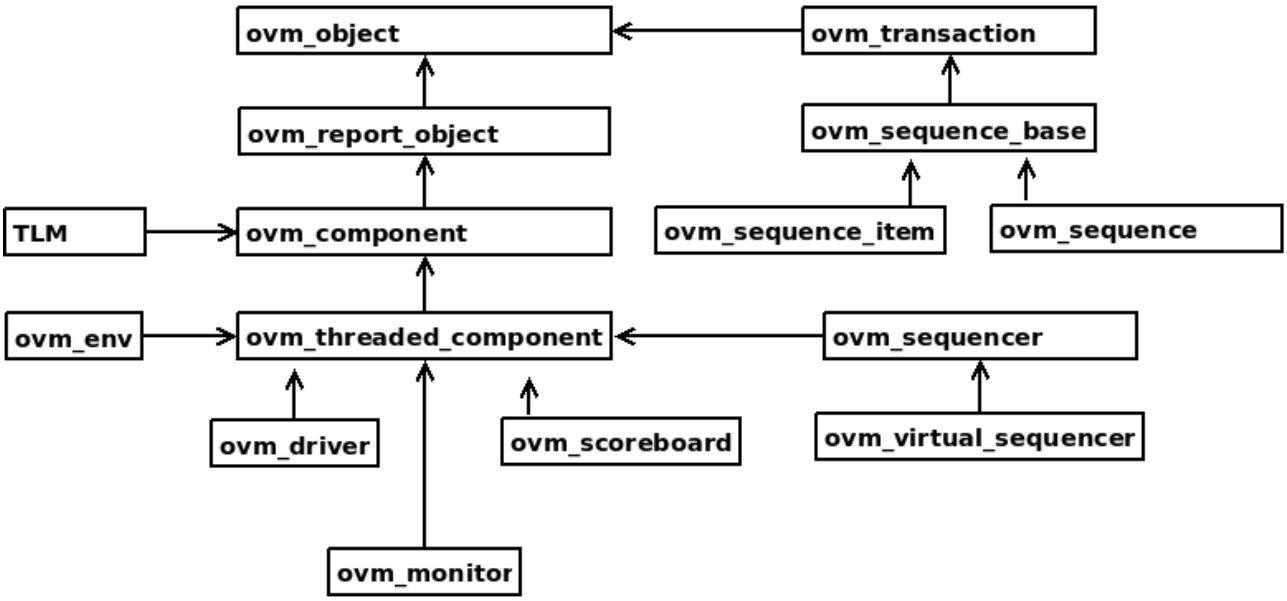
SystemVerilogin kehityksessä on otettu huomioon kasvava tarve kehitettävien ohjelmien tarkkaan testaukseen. Ohjelmien vaatimustason kasvaessa piireillä suoritetaan yhä monimutkaisempia toimintoja, jolloin niiden ohjelmointi myös monimutkaistuu. Tämä johtaa tarpeeseen sisällyttää FPGA-ohjelmointikieliin testausominaisuuksia, joilla tuotettua koodia on helpompi testata ja tutkia testituloksia. Kulut ohjelmointivirheiden etsimisessä ja korjauksessa kasvavat eksponentiaalisesti ohjelmointiprojektin edetessä. Ohjelmiston yhden ominaisuuden tai osa-alueen suora testaaminen sille varta vasten suunnitellulla testausohjelmistolla on tehokas tapa etsiä virheitä projektin silloisessa vaiheessa, mutta ohjelma pitää myös kokonaisuudessaan testata jotta eri osa-alueet toimivat keskenään vaaditulla tavalla. (Spear, 2008).

3.2.1 OVM- testausjärjestelmä

OVM on mentorin ja cadencen yhdessä kehittelemä ja IEEE-1800- standardiin lisätty testausmenetelmä, joka määrittelee SystemVerilog-kääntäjille perusvaatimukset OVM:n

suorittamiseksi. Tämän standardoinnin tarkoituksena on siis varmistaa, että kaikki SystemVerilogia tukevat kääntäjät tukevat myös OVM;ää: ja näin ollen suunnittelijoiden ja yritysten väliset ohjelmistojen vaihdot ja testaukset helpottuisivat.(Doulos, 2009).

OVM -järjestelmä on luokkakirjasto, joka sisältää testaukseen tarvittavia komponentteja sekä niiden välillä tietoa siirtäviä TLM -rajapintoja. Useimmat luokat periytyvät suoraan tai epäsuorasti ovm_void- luokalta. OVM- luokkahierarkia on tarkkaan kirjattu OVM-paketin mukana tulevaan OVM Reference Book- asiakirjaan.(ovmworld.org, 2008/1) (Khailtash, 2008).



KUVIO 1. OVM luokkahierarkia (ovmworld.org, 2008/3).

Hahmottamisen ja testausympäristön suunnittelun helpottamiseksi OVM voidaan jaotella suuremmiksi, teoreettisemmiksi kokonaisuuksiksi (Open Verification Components). Driver on näistä komponenteista lähimpänä testattavaa sovellusta emuloiden sen sisään- ja ulostuloja. Sequencer- niminen komponentti luo satunnaisluvut driverin syötettäväksi sovellukseen sekä komponentin monitor vertailtavaksi. Nämä komponentit sitoo kokonaisuudeksi agent-niminen ympäristö. Lopullinen kokonaisuus muodostuu environment-komponentista joka voi sisältää useampia Agentteja. Master-agentit toimivat testattavalle ohjelmalle lähettävänä osapuolena ja slave-agentit vastaanottavat siltä tietoa. (ovmworld.org , s. 10,11, 16-20, 2008/1).

3.2.2 Testauksen kattavuus

Ajonaikainen testin kattavuuden seuranta sekä testausjärjestelmän itseanalyysi on OVM:n tapaisten järjestelmien vahvuus. Mitä monimutkaisempi ohjelmisto on kyseessä, sitä enemmän sen testaamiseen joudutaan käyttämään aikaa. Jos testauksen aikana löydetyt virheet voidaan paikallistaa testausohjelmiston avulla välittömästi testin jälkeen, voidaan niiden vaikutukset ohjelmiston muihin osa-alueisiin ja siten koko projektin etenemiseen arvioida heti. Tämä vaatii testausohjelmistolta mahdollisuutta analysoida omaa toimintaansa ja testin kattavuutta. OVM:n Tiedonsiirtorajapinta TLM mahdollistaa tämän. (Michelson, 2009).

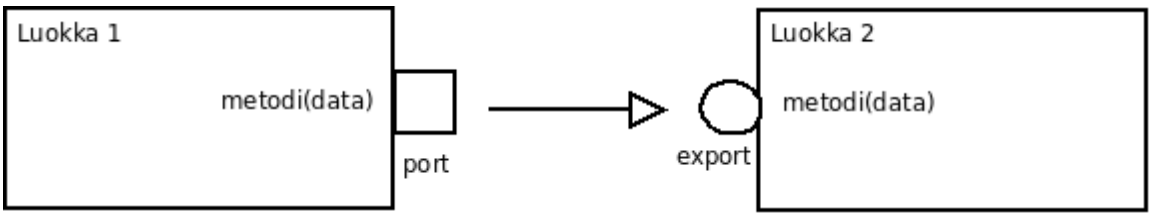
3.2.3 Transaction Level Modeling (TLM)

Vain perinteisillä HDL-kielillä ohjelmoinut suunnittelija joutuu OVM- testausmenetelmään tutustuessaan opiskelemaan myös uuden tavan siirtää tietoa ohjelman eri osa-alueiden välillä. Yksinkertaisin tapa luoda testausympäristö on ohjelmoida se testaamaan jotain tiettyä ohjelmistoa. Kuitenkin jos yksinkertaisempaa testausympäristöä halutaankin kasvattaa testaamaan suurempaa kokonaisuutta, koko ympäristö joudutaan suunnittelemaan aina alusta koska yksinkertaisen testausympäristön komponentit yleensä huolehtivat tiedonsiirrosta toisiin komponentteihin. Tämän takia erillisiä testausympäristöjä kehittävät yhteisöt käyttävätkin SystemVerilogin luokkia.

Luokkia käytettäessä yksittäisen ohjelman osa-alueen ei tarvitse keskittyä välittämään tietoa. Tiedonsiirrosta huolehtii erillinen rajapinta, eivätkä luokat ole sinänsä tietoisia toisistaan. Tällöin luokkia ei tarvitse kirjoittaa joka kerta uudestaan kun testiympäristö muuttuu. Luokassa on ainoastaan sen toiminnan vaatimat portit. Kun luokan sisällä tehty tarvittava toiminta, se kutsuu porttiaan ja lähettää sille tietoa. Tiedonsiirrosta huolehtiva rajapinta hoitaa datan siirron haluttuun luokkaan. Tällainen ohjelmarakenne helpottaa myös usean ohjelmointikielen käyttämisen testiympäristön osana.

Yksinkertaisimmassa TLM- tiedosiirtomuodossa kaksi luokkaa voivat siirtää tietoa yhteen

suuntaan. TLM sisältää tiedonsiirrosta vastaavia rajapintoja jotka taas sisältävät tiedon kutsuttavista metodeista ja niiden tarvitsemista tietotyypeistä. Tiedonsiirtoa tarvitsevassa luokassa port määrittää minkä tyyppistä, tiedonsiirtorajapinnan tuntemaa metodia se haluaa kutsua. Exportin sisältävä luokka taas kertoo tiedonsiirtojapapinnalle minkä tyyppisen metodin se tarjoaa kutsuttavaksi. Kun luokka, jossa port on, kutsuu metodia jota export tarjoaa, tieto siirtyy. Exportin sisältävän luokan prosessin suoritus alkaa vasta kun sen tarjoamaa metodia kutsutaan portin kautta. (Bromley, 2009)(ovmworld.org, 2008/1).



KUVIO 2. Tiedonsiirto luokkien välillä käyttäen TLM- rajapintaa.

Jos tietoa halutaan siirtää prosessien välillä häiritsemättä prosessien toimintaa, voidaan käyttää OVM:n tlm_fifo-kanavaa. ovm_fifo on käytännössä puskuri, johon lähettävä osapuoli tarjoaa lähetettävän tiedon ja josta vastaanottava osapuoli vastaavasti tietoa hakee. Prosessien suoritus keskeytyy vasta siinä tapauksessa että puskuri on täysi kun sille tarjotaan tietoa tai se on tyhjä kun siltä kysytään tietoa.

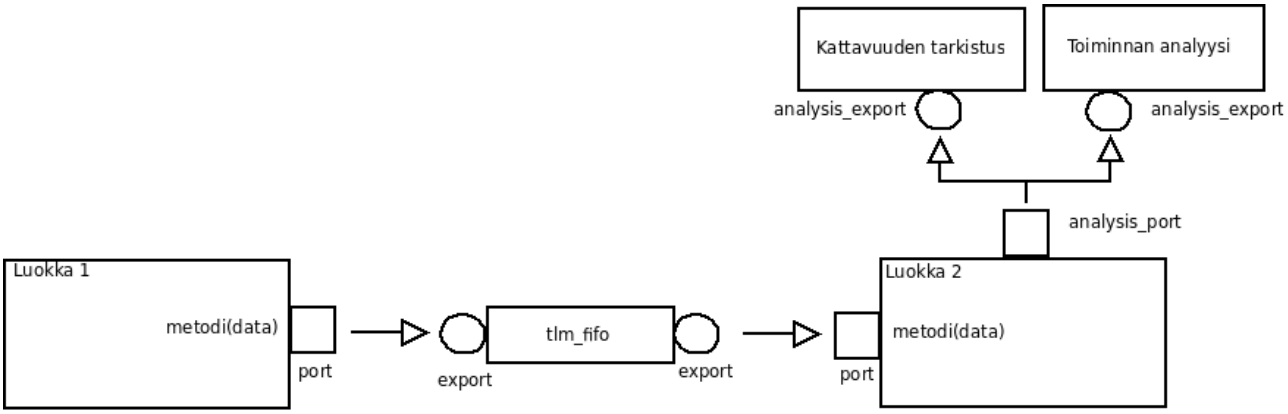
KUVIO 3. Tiedonsiirto luokkien välillä käyttäen TLM:n tlm_fifo -puskuria.

Nämä kaksi tiedonsiirtotapaa keskeyttävät prosessit jos tiedonsiirto ei ole mahdollista. TLM tarjoaa samantyyppiset tiedonsiirtotavat myös niin että prosessit eivät keskeydy vaan jatkavat toimintaansa.

3.2.4 OVM analysis- portit

Testauksen tulosta sekä kattavuutta tutkiville OVM -ohjelmiston osille on tehty oma siirtokanava, joka sallii näiden testauksen kannalta kriittisten komponenttien vastaanottaa

testin aikaista tiedonsiirtoa ohjelmiston eri osista. Portti analysis_port sisältää listan analysis_export- liityntöjä, jotka se tarvittaessa kutsuu ja hakee tarjotut tiedot. Analysis_port kutsuu write()- metodia jolloin se kutsuu jokaisen siihen liitetyn exportin write()- metodia.(Bromley, 2009)(ovmworld.org s. 25 – 35, 2008/1).



KUVIO 4. analysis_port toimintaperiaate.

4. ESIMERKIT JA TESTAUSTULOKSET

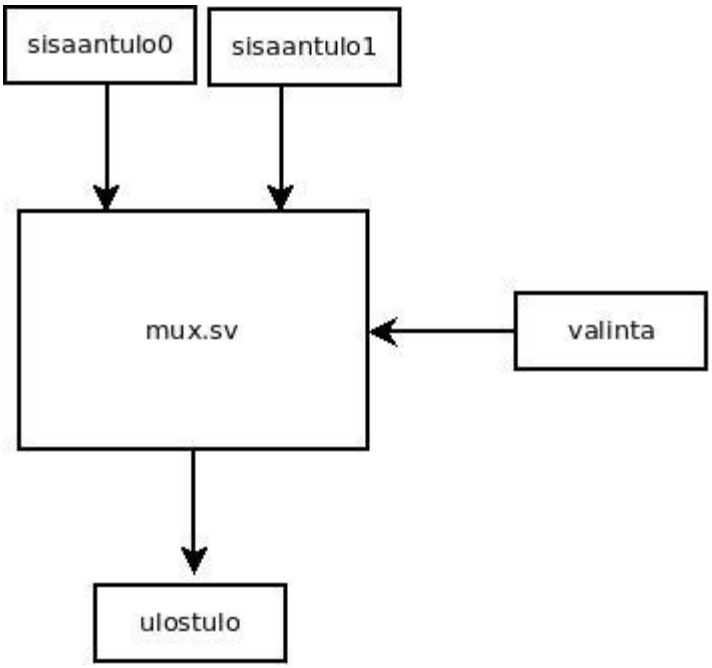
Seuraavassa esitän kaksi FPGA-ohjelman testauksen pääperiaatteita kuvaavaa esimerkkiä. Esimerkeistä käy ilmi miten testattavan ohjelmiston laajuus vaikuttaa käytettyyn testaustapaan. Ensimmäisen esimerkin tarkoituksena on selventää, mitä aikaisemmin tekstissä mainittu suoraan testaaminen tarkoittaa. Myöhemmin esittelen testaustavan, jossa käytetään kehittyneempiä testausmenetelmiä kuten satunnaislukugenerointia ja tehokkaampia tulosten selvittämiseen tarkoitettua työkaluja.

4.1 Esimerkki SystemVerilog- ohjelman testauksesta

Tässä esimerkissä käytän SystemVerilog- kielellä ohjelmoitua esimerkkiohjelmaa sekä samalla kielellä ohjelmoitua testausympäristöä.

4.1.1 Ohjelman rakenne

Esimerkissä testattavana ohjelmana toimii yksinkertainen multiplexeri. Ohjelmassa valitaan kahdesta sisääntulosta toinen ja ohjataan se ohjausmuuttujalla ulostuloksi.



KUVIO 5. mux.sv:n toimintaperiaate.

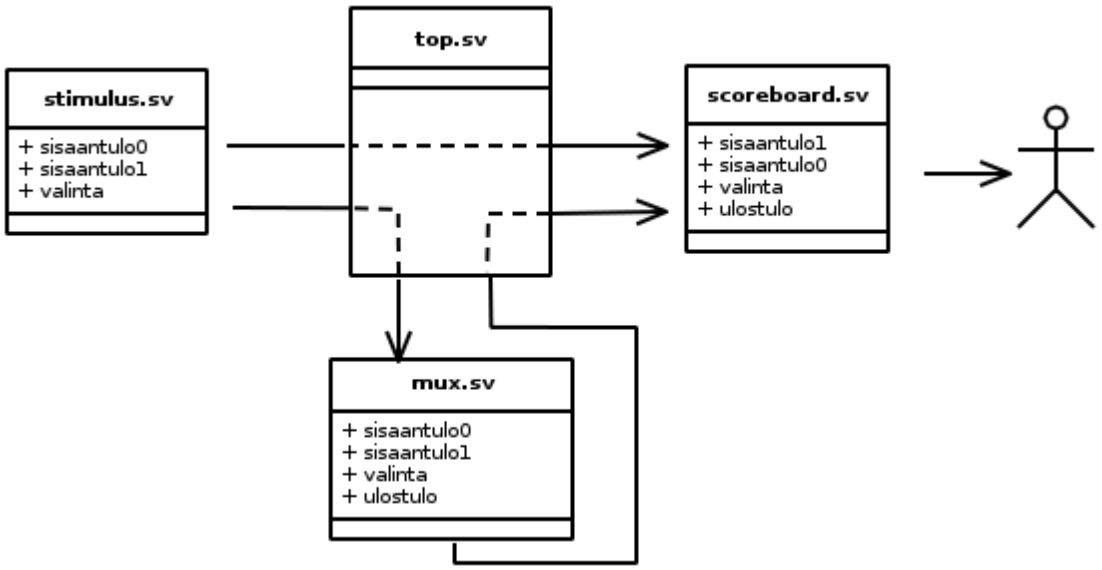
```
// Multiplekseri. valinta-sisääntulon perusteella valitaan
// kahdesta sisääntulosta toinen ja ohjataan se ulostuloksi
// Alkuperäinen koodi löytyy osoitteesta
// http://www.asic-world.com/examples/systemverilog/mux.html
// ja sen on kehittänyt Deepak Kumar Tala.
//------------------------------------------------------

//muuttujat
module  mux(
input  wire  sisaantulo0      , // Ensimmäinen sisääntulo
input  wire  sisaantulo1     , // Toinen sisääntulo
input  wire  valinta         , // Sisääntulojen valinta
output reg   ulostulo          // Ulostulo
);

//mux-koodi
always @ (*)
begin
     case (valinta)
        1'b0 : ulostulo = sisaantulo0;
        1'b1 : ulostulo = sisaantulo1;
 endcase
end
endmodule
```

Top

top.sv on moduuli, joka hallitsee kaikkia muita ohjelman osia. Se käynnistää ohjelmiston eri osa-alueet, huolehtii eri testausmoduulien ja testattavan ohjelman välisestä tiedonsiirrosta sekä käynnistää ja pysäyttää testauksen. Tämän ohjelmalohkon ansiosta testausohjelmiston laajennettavuus paranee huomattavasti. Muut ohjelmiston osat eivät ole suoraan kytköksissä toisiinsa joten uusia moduuleja voidaan kytkeä ohjelmistoon rajaton määrä.



KUVIO 6. Ohjelman mux toiminta.

```
// Top. Kokoaa testipenkin eri moduulit yhteen ja huolehtii muuttujien välisestä tiedonsiirrosta
// Lähteinä AVM Cookbook esimerkki sivuilta 11, 12 ja 13.
//-------------------------------------------------------------------------------------------------

    module top;

        //käynnistetään testipenkin osat ja esitellään niiden välillä tietoa siirtävät muuttujat

            stimulus    s(sisaantulo0, sisaantulo1, valinta);
            mux         m(sisaantulo0, sisaantulo1, valinta, ulostulo);
            scoreboard sb(ulostulo, sisaantulo0, sisaantulo1, valinta);

    initial
```

```
        //ohjelmaa ajetaan 10 000 kellon nousevan reunan ajan kunnes lopetetaan
        #10000 $finish(2);
endmodule
```

Stimulus generator

stimulus.sv luo testaukseen tarvittavat signaalit; kaksi ohjattavaa signaalia sekä niitä ohjaavan signaalin valinta. Se lähettää ne testattavalle ohjelmalle sekä testauksen tulokset keräävälle ja analysoivalle scoreboard -moduulille.

```
//------------------------------------------------------------------------------
// Stimulus. Syöttää testattavaan ohjelmaan kaikki mahdolliset testattavat kombinaatiot.
// Lähteenä AVM Cookbook esimerkki sivuilta 11, 12 ja 13.
//
//  sisaantulo0    sisaantulo1    valinta    ulostulo
//  0              0              0          0
//  0              1              0          0
//  1              0              0          1
//  1              1              0          1
//
//  0              0              1          0
//  0              1              1          1
//  1              0              1          0
//  1              1              1          1
//------------------------------------------------------------------------------------------


module stimulus(output bit sisaantulo0, sisaantulo1, valinta);

  always begin

//Luodaan testiä varten mux -modulille sisääntulot 100 kellosignaalin nousevan reunan välein

 #100 sisaantulo0 = 0; sisaantulo1 = 0; valinta=0;

  #200 sisaantulo0 = 0; sisaantulo1 = 1; valinta=0;
```

```
        #300 sisaantulo0 = 1; sisaantulo1 = 0; valinta=0;

        #400 sisaantulo0 = 1; sisaantulo1 = 1; valinta=0;

        #500 sisaantulo0 = 0; sisaantulo1 = 0; valinta=1;

        #600 sisaantulo0 = 0; sisaantulo1 = 1; valinta=1;

        #700 sisaantulo0 = 1; sisaantulo1 = 0; valinta=1;

        #800 sisaantulo0 = 1; sisaantulo1 = 1; valinta=1;

    end

endmodule
```

Testitulosten käsittely, scoreboard

scoreboard.sv ottaa vastaan testattavalta ohjelmalta ulostulon sekä stimulus.sv -moduulilta sisääntulon ja vertaa niitä keskenään. Vertailun tulokset tulostetaan ajon jälkeen käyttäjälle.

```
// Scoreboard. Vastaanottaa mux:in ulostulon ja tarkastaa sen oikeellisuuden. Vertailu tehdään
// stimulus-moduulin ulostulo- ja valinta-muuttujien avulla
// Lähteinä AVM Cookbook esimerkki sivuilta 11, 12 ja 13.
//
//  sisaantulo0    sisaantulo1    valinta    ulostulo
//  0              0              0          0
//  0              1              0          0
//  1              0              0          1
//  1              1              0          1
//
//  0              0              1          0
//  0              1              1          1
//  1              0              1          0
//  1              1              1          1
//-------------------------------------------------------------------------------------------------

module scoreboard(input bit ulostulo, sisaantulo0, sisaantulo1, valinta);

  //sisääntulot
 reg apu;
```

```
reg taulu[2][2][2];
      initial begin
         //Syötetään taulu-muuttujaan vertailtavien muuttujien arvot kuten ne luotiin stimulus-moduulissa
    taulu[0][0][0] =  0;
         taulu[0][1][0] =  0;
         taulu[1][0][0] =  1;
         taulu[1][1][0] =  1;

         taulu[0][0][1] =  0;
         taulu[0][1][1] =  1;
         taulu[1][0][1] =  0;
         taulu[1][1][1] =  1;
 end
      //suoritetaan vertailu mux -ohjelman sisääntulojen (sisaantulo0, sisaantulo1, valinta) ja ulostulon ulostulo välillä
      always @(*) begin
          //syötetään apu-nimiseen muuttujaan taulukosta sisääntulojen perusteella senhetkinen oikea ulostulo.
        #20  apu = taulu[sisaantulo0][sisaantulo1][valinta];

        #40   $display("@%4t - %b%b %b : apu=%b, ulostulo=%b (%0s)",$time, sisaantulo0, sisaantulo1, valinta, apu, ulostulo,((apu == ulostulo) ?   "Match" : "Mis-match"));
end
endmodule
```

scoreboard- moduulin rivi

#40 $display("@%4t - %b%b %b : apu=%b, ulostulo=%b (%0s)",$time, sisaantulo0, sisaantulo1, valinta, apu, ulostulo,((apu == ulostulo) ? "Match" : "Mis-match"));

tulostaa siis käyttäjälle 40 kellopulssin nousevan reunan pituisen viiveen jälkeen kaikkien testien suoritusajan, vertailussa käytetyt muuttujat sekä testin tuloksen kirjallisena:

```
# @  60 - 00 0 : apu=0, ulostulo=0 (Match)
# @ 360 - 01 0 : apu=0, ulostulo=0 (Match)
# @ 660 - 10 0 : apu=1, ulostulo=1 (Match)
# @1060 - 11 0 : apu=1, ulostulo=1 (Match)
# @1560 - 00 1 : apu=0, ulostulo=0 (Match)
```

@2160 - 01 1 : apu=1, ulostulo=1 (Match)
@2860 - 10 1 : apu=0, ulostulo=0 (Match)
@3660 - 11 1 : apu=1, ulostulo=1 (Match)

4.1.2 Ohjelman kääntäminen ja simulointi ModelSim PE Student Edition 6.5a -ohjelmalla

SystemVerilog-kieli on vielä neljä vuotta julkaisunsa jälkeenkin melko huonosti tuettu monissa yleisissä HDL-kääntäjissä. Kokeiltuani useaa eri ohjelmaa päädyin Mentor Graphicsin ModelSim-ohjelman opiskelijalisenssillä ilmaiseksi saatavaan versioon sen helppokäyttöisyyden takia.

Kuten useimpia HDL-käyttöliittymiä, ModelSim:iä voidaan käyttää konsolikäskyillä. Konsoli sijaitsee ohjelman alareunassa, ja se tukee Unix -ympäristöistä tuttua TAB-täydennystä. Käskyhistoriaa voi selata nuolinäppäimillä.

Aluksi täytyy luoda uusi projekti. Tämän voi tehdä esimerkiksi valitsemalla ohjelman File-alasvetovalikosta New -> Project. Kun projetin ominaisuudet on täytetty, voidaan sille luoda samaisesta valikosta Source- tiedostot. Näihin tiedostoihin voidaan siis ohjelmakoodi kirjoittaa.

Koodauksen jälkeen ohjelma on käännettävä. Aluksi on luotava ModelSim:in kääntämiseen tarvitsema kirjasto. Tämän voi tehdä kirjoittamalla konsoliin *vlib <kirjaston nimi>*.

Kirjaston nimenä voisi tässä tapauksessa olla vaikkapa mux:

vlib mux

Kirjaston luomisen jälkeen voidaan kääntää koodit kirjoittamalla konsoliin *vlog <source1.sv> <source2.sv>...* Esimerkkiohjelma kääntyy siis käskyllä:

vlog mux.sv top.sv scoreboard.sv stimulus.sv

Tässä vaiheessa konsoli ilmoittaa mahdollisista virheistä. Jos virheitä ei ole, tulostus on seuraavanlainen:

Model Technology ModelSim PE Student Edition vlog 6.5a Compiler 2009.03 Mar 28 2009

-- Compiling module mux

-- Compiling module top

-- Compiling module scoreboard

-- Compiling module stimulus

#

Top level modules:

top

Kuten tulosteesta näemme, top.sv valittiin automaattisesti ohjelmiston top- tasoksi. Tätä tietoa tarvitaan seuraavassa vaiheessa, simuloinnissa. Simulointi tapahtuu käskyllä *vsim <kirjasto> <top-taso>:*

vsim mux top

Jos simuloinnissa ei tapahdu virheitä, konsolin tulostuksen pitäisi olla:

vsim mux top

Loading sv_std.std

Loading work.mux

Loading work.top

Loading work.stimulus

Loading work.scoreboard

Onnistuneen simuloinnin jälkeen ohjelma voidaan ajaa käskyllä:

run –all

4.2 OVM hello_world esimerkki

OVM-paketin mukana tulee Apache2-lisensoituja esimerkkiohjelmia joista käytän OVM hello_world- ohjelmaa. Esimerkistä ilmenee kuinka valmis ohjelma testipenkkeineen käännetään ja kuinka tulokset analysoidaan. Tulosten analysoinnin ohessa käydään läpi OVM:n tärkeimpiä osia sekä niiden kytköstä esimerkkiohjelmaan.

Hello world-esimerkissä käytetään OVM-kirjastoja TLM-rajapinnan tiedonsiirron testaamiseen. Kahden luokan välillä siirretään tietoa käyttäen port-export- sekä tlm fifo-yhteyttä. Näitä yhteyksiä varten producer -luokkaan luodaan kaksi erillistä producer -toimintoa jotka yhdistetään consumer-luokan exportiin. Yhteyden toiminnasta tulostetaan raportti käyttäjälle. Luokka packet on OVM-järjestelmän ovm_transaction. Se sisältää TLM- rajapinnan tiedonsiirtoon tarvitsemat tiedot. hello_world- moduuli on järjestelmän top-moduuli jota kutsutaan simuloinnin yhteydessä ja joka liittää luokat järjestelmän simulointiin. Hello_world- moduulissa on myös järjestelmän muutettavat parametrit. Top-luokka hallitsee hello_world-moduulin liittämiä luokkia ja niiden välistä tiedonsiirtoa. Esimerkin rakenteen takia erillistä komponenttia tulosten analysointiin ei käytetä, vaan simulointitulokset tulostetaan komponenttien suorituksen aikana.

4.2.1 Esimerkin tärkeimmät kytkennät OVM- järjestelmään

ovm_component

ovm_component on OVM -järjestelmän juuritason luokka joka tarjoaa järjestelmän käyttöön tärkeimpiä rajapintoja.

connect

ovm_component- luokan connect- metodilla käynnistetään tiedonsiirto komponenttien välillä.

ovm_default_table_printer

ovm_printer- luokan metodi jonka avulla OVM- järjestelmästä voidaan tulostaa tietoa taulukkomuodossa.

set_config_int

ovm_pkg- tilaan voidaan esitellä ohjelmakoodissa set_config_int tai set_config_string- metodilla globaaleja muuttujia. Näitä muuttujia voidaan tämän jälkeen kutsua muissa OVM:n osissa tai ohjelmakoodissa käskyllä import ovm_pkg. Myös funktioiden esittely on mahdollista metodilla set_config_object.

enable_print_topology

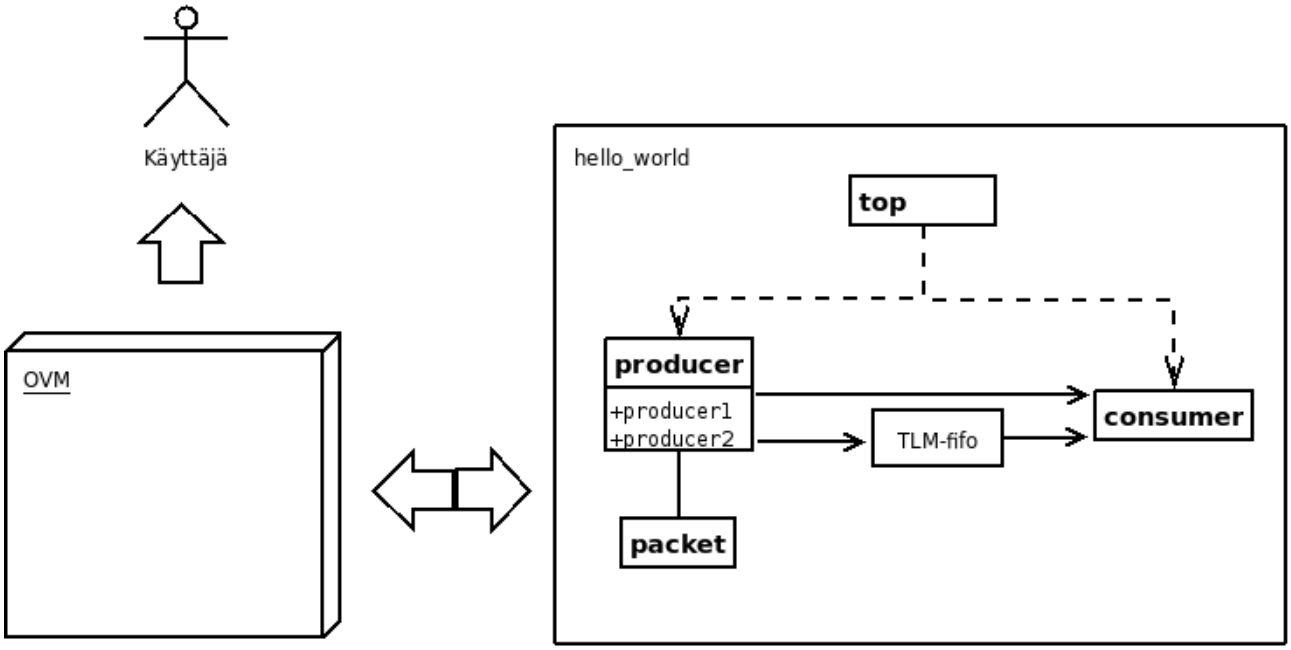
ovm_root- luokan metodi, joka tulostaa järjestelmän topologian.

`ovm_component_utils_begin

ovm_component_utils kuuluu OVM:n utility- makroihin. Näitä makroja tarvitaan muun muassa objektien kopioimiseen ja debuggaukseen.

`ovm_field_object

ovm_*_utils_begin ja ovm_*_utils_end -lohkojen välillä esitellyt ovm_field- makroilla esitellään OVM:n sisäisiä tapahtumia kuten copy, compare, pack, unpack, record, print ja sprint. (ovmworld.org, 2009/1)(ovmworld.org, 2008/3)



KUVIO 7. hello_world -esimerkin toiminta.

4.2.2 Hello world-esimerkin komponentit

hello_world.sv

hello_world top-moduuli. Sisältää simuloinnin parametrit ja liittää komponentit simulaatioon. Moduulissa määritetään simulaation aikamääritykset, OVM -tulostimen asetukset, lähetettävien pakettien määrä sekä simulaation kesto. hello_world käynnistää myös simulaation. (Liite 2).

top.sv

Luokka top. Hallitsee komponenttien välistä tiedonsiirtoa. Luokassa esitellään tiedonsiirrossa käytettävät komponentit, liitetään ne toisiinsa ja luodaan OVM:n järjestelmän TLM- tiedonsiirtorajapinta. (Liite 1).

```
// esitellään tiedonsiirtoa käyttävät komponentit
    p1 = new("producer1",this);
    p2 = new("producer2",this);
    f  = new("fifo",this);
    c  = new("consumer",this);

// luodaan tiedonsiirtoväylät
    p1.out.connect( c.in );
    p2.out.connect( f.blocking_put_export );
    c.out.connect( f.get_export );
```

pakcet.sv

Packet- luokka sisältää TLM- rajapinnan tiedonsiirtoon tarvitsemat asetukset. (Liite 3).

producer.sv

producer- luokka. Luo lähetettävän paketin ja lähettää sen consumer- luokan vastaanottavaan export- porttiin suoraan sekä tlm_fifo- puskurin kautta. (Liite 4).

```
//luodaan yhteys johon kytketään paketti T
  ovm_blocking_put_port #(T) out;

  task run();
    T p;

// paketin lähettäminen consumerille
      out.put(p);
```

consumer.sv

consumer- luokka vastaanottaa producer- luokan lähettämät paketit ja tulostaa paketin tiedot. (Liite 5).

```
  ovm_blocking_put_imp #(T,consumer #(T)) in;
```

```
task run ();
   T p;
   while(out.size()) begin
     out.get(p);
     put(p);
   end
 endtask

 task put (T p);
…
end task
```

4.2.3 Kääntäminen ja simulointi käytännössä

Kääntäjänä tässä esimerkissä on Cadencen IUS 8.2- versio. Modelsimin opiskelijalisenssillä ei onnistu esimerkiksi satunnaislukujen generointi. Ohjelmisto simuloidaan hello world-esimerkin mukana tulevalla, IUS-ohjelmistolle tarkoitetulla tiedostolla tai kääntämällä jokainen tiedosto erikseen. Tiedostoa apuna käyttäen IUS osaa kääntää tarvittavat tiedostot oikeassa järjestyksessä ja oikeilla parametreillä sekä käynnistää simuloinnin. Esimerkin ajo käynnistetään IUS:n kosolissa. Ensin konsolissa siirrytään kansioon, jossa ajettava ohjelmisto sijaitsee. Konsolikäskyinä toimivat UNIX-tyyppiset käskyt kuten cd ja ls. Tässä konsolissa TAB-täydennykset valitettavasti puuttuvat.

Ennen simulointia on syytä tarkistaa että simuloitavan ohjelman kansion käyttöoikeudet ovat käyttöjärjestelmän puolelta kunnossa. Kääntäjä kirjoittaa logitiedostonsa kansioon, ja jos kirjoitus estetään käyttöjärjestelmän tiedostojärjestelmän toimesta, käännös epäonnistuu. IUS:n konsoli tarvitsee myös jossain tapauksessa ohjelman sisäiset käyttöoikeudet lukea, kirjoittaa ja yhdistää ohjelman eri osa-alueita toisiinsa. Oikeuksia voidaan muokata konsolissa parametrilla -access. Oikeudet ovat luku-, kirjoitus-, ja yhdistämisoikeudet; r, w, ja c.

Simulointia ennen tiedostot on myös käännettävä. Valmis, irun-komennolle tarkoitettu

tiedosto huolehtii kääntämisestä, mutta jos käyttäjä jostain syystä haluaa tehdä tämän käsin, se tehdään ncvlog -sv komennolla. Ncvlog-komento on tarkoitettu Verilog-tiedostojen kääntämiseen ja -sv-parametri lisää SysteVerilog- kirjastot mukaan. Komennon jälkeen syötetään käännettävien tiedostojen tiedostonimet sekä kääntämisessä käytettävät parametrit.

Esimerkin mukana tulevat kääntäjälle valmiit tiedostot Questa ja IUS- ohjelmille. Kääntäminen ja simulointi käynnistetään komennolla irun -f compile_ius.f +incdir+[ovm src-kansio]. Tarvittaessa lisätään -access- parametri. Simuloinnin tarkastelua helpottaa graafinen simulointiympäristö, joka voidaan käynnistää lisäämällä loppuun parametri -gui.

4.2.4 Simulointitulokset

Simulointituloksiin tulostuu ensin ohjelmiston rakenne ja komponenttien parametrit:

```
OVM-2.0.1
(C) 2007-2008 Mentor Graphics Corporation
(C) 2007-2008 Cadence Design Systems, Inc.
----------------------------------------------------------------
OVM_INFO @ 0 ns: reporter [RNTST] Running test ...
OVM_INFO @ 0 ns: reporter [OVMTOP] OVM testbench topology:
---------------------------------------------------------------- ----------------------------------------
------------
Name                                Type                    Size        Value
----------------------------------------------------------------------------------------------------------
------------
top                                 top                     -               -
  consumer                          consumer #(T)             -               -
    in                              ovm_blocking_put_imp      -               -
    recording_detail                ovm_verbosity            32          OVM_FULL
    out                             ovm_get_port            -               -
    recording_detail                ovm_verbosity           32          OVM_FULL
    count                           integral                32              'd0
    recording_detatail              ovm_verbosity           32          OVM_FULL
  fifo                              tlm_fifo #(T)           -               -
    get_ap                          ovm_analysis_port       -               -
    recording_detail                ovm_verbosity           32          OVM_FULL
    get_peek_export                 ovm_get_peek_imp        -               -
```

```
  recording_detail          ovm_verbosity          32        OVM_FULL
  put_ap                    ovm_analysis_port      -         -
  recording_detail          ovm_verbosity          32        OVM_FULL
  put_export                ovm_put_imp            -         -
  recording_detail          ovm_verbosity          32        OVM_FULL
  recording_detail          ovm_verbosity          32        OVM_FULL
producer1                   producer #(T)          -         -
  out                       ovm_blocking_put_port  -         -
  recording_detail          ovm_verbosity          32        OVM_FULL
  proto                     packet                 -         -
  num_packets               integral               32        'd2
  count                     integral               32        'd0
  recording_detail          ovm_verbosity          32        OVM_FULL
producer2                   producer #(T)          -         -
  out                       ovm_blocking_put_port  -         -
  recording_detail          ovm_verbosity          32        OVM_FULL
  proto                     packet                 -         -
  num_packets               integral               32        'd4
  count                     integral               32        'd0
  recording_detail          ovm_verbosity          32        OVM_FULL
recording_detail            ovm_verbosity          32        OVM_FULL
```

Tämän jälkeen tulostuu ohjelmiston toiminta eli pakettien liikkuminen komponenttien välillä. Tulosteessa näkyy tapahtuman ajanhetki, kyseinen komponentti, kuvaus tapahtumasta sekä laskuri vastaanotetuille paketeille. Tuloksista nähdään että tlm_fifo -puskuria käyttävä producer2:n lähettämä paketti odottaa puskurissa kunnes vastaanottava komponentti on vapaa, joten sen lähettämät paketit vastaanotetaan viimeisenä. Producer1 lopettaa toimintansa vasta kun kaikki sen lähettämät paketit on vastaanotettu kun taas producer2 voi lopettaa toimintansa heti kun paketit on lähetetty:

```
[0 ns] hier=top.producer2: Starting.
[0 ns] hier=top.producer2: Sending producer2-0
[0 ns] hier=top.producer1: Starting.
[0 ns] hier=top.producer1: Sending producer1-0
[10 ns] hier=top.producer2: Sending producer2-1
[20 ns] hier=top.producer2: Sending producer2-2
[40 ns] hier=top.consumer: Received producer1-0 local_count=1
[50 ns] hier=top.producer1: Sending producer1-1
[80 ns] hier=top.consumer: Received producer2-0 local_count=2
[90 ns] hier=top.producer2: Sending producer2-3
[120 ns] hier=top.consumer: Received producer1-1 local_count=3
[130 ns] hier=top.producer1: Exiting.
[160 ns] hier=top.consumer: Received producer2-1 local_count=4
[170 ns] hier=top.producer2: Exiting.
[200 ns] hier=top.consumer: Received producer2-2 local_count=5
[240 ns] hier=top.consumer: Received producer2-3 local_count=6
```

5 TULOKSET JA POHDINTA

SystemVerilog- kieli on tehokas työkalu rakennettaessa testiympäristöjä. OVM:n tapaiset testausohjelmistot tuovat moderniin FPGA- ohjelmointiin tarkasti rajattavaa ja hallittua testausta. Jos jo ohjelmistoa suunnitellessa suunnittelija ottaa huomioon testivaiheen, on tällaisen järjestelmän liittäminen testattavaan ohjelmaan suhteellisen helppoa. Järjestelmien oliopohjaisuus ja modulaarisuus helpottavat testauksen muokattavuutta ja uudelleenkäytettävyyttä. Standardointi auttaa eri suunnitteluryhmiä ymmärtämään helpommin ohjelmiston rakennetta ja näin projektin eri vaiheisiin voidaan tuoda tarvittavaa osaamista ryhmän ulkopuoleltakin.

Mielestäni SystemVerilog ei ole ohjelmointikielenä VHDL:ää tai Verilogia niin paljon tehokkaampi, että esimerkiksi yksittäisen kokeneen VHDL- suunnittelijan kannattaisi pelkän syntaksin helppouden takia opetella varta vasten uutta kieltä. SystemVerilogin tehokkuus tulee standardoinnista ja sen mukaisista ajattelutavoista ohjelmointiprosessin aikana. Jos yrityksen työntekijöiden ja työryhmien työskentelytavat muutetaan vastaamaan uutta ajattelutapaa, standardin käyttämisestä on hyötyä ohjelmistoprojektin kaikilla osa-alueilla.

Aihetta tutkiessani eniten ongelmia aiheutti sopivien ja toimivien kääntötyökalujen sekä selkeän informaation puute. Vaikka SystemVerilog- ja OVM- standardit ovat avoimia standardeja, varsinkin OVM:ää kunnolla tukevia kääntäjiä on harvassa ja niiden lisenssit ovat kalliita. Suurin ongelma kuitenkin on kääntäjistä ja standardin tuesta julkaistujen artikkeleiden epätarkkuus. Ohjelmistojen hankkimista suunnittelevan tai asiaa tutkivan on vaikea löytää itselleen sopiva kääntäjä koska kääntäjien ominaisuuksia ja SystemVerilog-standardin tuen kattavuutta liioitellaan. Täydellisesti standardia tukevia ohjelmistoja ovat kirjoitushetkellä Mentor Graphicsin Questa ja Cadencen IUS, eli standardia suunnittelevien yhtiöiden tuotteet.

LÄHTEET

Painetut lähteet:

Spear, Chris: SystemVerilog for Verification, Second Edition, A Guide to Learning the Testbench Language Features, Springer, New York, 2008.

Elektroniset lähteet:

Aynsley, John, Doulos: SystemVerilog as the New Verilog, http://www.youtube.com/watch?v=b9mtVeVaYwU katsottu 6.6.2009

Bromley Jonathan, Doulos: TLM in OVM http://www.youtube.com/watch?v=3b5xGfnM9XY katsottu 3.6.2009

Bromley Jonathan, Doulos: Observation in VMM and OVM http://www.youtube.com/watch?v=d0RZCZ4phWw katsottu 12.6.2009

Cohen Ben, Venkataramanan Srinivasan, Kumari Ajeetha: SystemVerilog Assertions Handbook –for Formal and Dynamic Verification http://books.google.fi/books?id=m6CMyKhWX9YC&lpg=PA47&ots=l3TKXnMFqb&dq=systemverilog%20sequence&pg=PA48 , luettu 3.7.2009

Doulos: SystemVerilog DPI Tutorial http://www.doulos.com/knowhow/sysverilog/tutorial/dpi/ 23.6.2009

Doulos: SystemVerilog Assertions Tutorial http://www.doulos.com/knowhow/sysverilog/tutorial/assertions/ luettu 4.7.2009

Deepak Kumar Tala: SystemVerilog Assertions Part-II http://www.asic-world.com/systemverilog/assertions2.html luettu 5.7.2009

Doulos: Getting started with OVM, http://www.doulos.com/knowhow/sysverilog/ovm/tutorial_0/ luettu 19.8.2008

Doulos: What is OVM? http://www.doulos.com/content/training/ovm.php luettu 15.6.2009

Edelman Rich, Mentor Graphics:A SystemVerilog DPI Framework for Reusable Transaction Level Testing, Debug and Analysis of SoC Designs http://www.design-reuse.com/articles/13403/a-systemverilog-dpi-framework-for-reusable-transaction-level-testing-debug-and-analysis-of-soc-designs.html luettu 23.6.2009

Khailtash Amal: SystemVerilog OOP Ovm Features Summary

http://www.slideshare.net/akhailtash/ovm-features-summary-presentation luettu 12.6.2009

Mentor Graphics: Advanced Verification Methodology Cookbook 2008

Michelson Akiva, EETimes: Indicators help magage coverage-driven verification http://www.eetimes.com/showArticle.jhtml?articleID=60402173 luettu 15.6.2009

Nir Hamzani, Sital Technology: What is SystemVerilog? http://www.sital.co.il/pdf/what_is_SystemVerilog.pdf , luettu 10.7.2008

ovmworld.org : Open Verification Methodology User Guide, 2008/1

ovmworld.org: OVM White Paper http://www.ovmworld.org/white_papers.php , luettu 18.8.2008/2

ovmworld.org: OVM Class Reference, 2008/3

ovmworld.org : foorumikeskustelu TLM-rajapinnasta http://www.ovmworld.org/forums/showthread.php?p=1193 luettu 11.6.2009/1

ovmworld.org: Utility and Field Macros, http://www.ovmworld.org/docs_2.0.2/html/files/macros/ovm_object_defines-svh.html, luettu 26.9.2009/2

Project VeriPage: SystemVerilog Assertion http://www.project-veripage.com/sva_1.php sivut 1-6, luettu 1-3.7.2009/1

Project VeriPage: SystemVerilog DPI Tutorial http://www.project-veripage.com/dpi_tutorial_2.php luettu 23.6.2009/2

LIITTEET

LIITE 1

```
// ovm-2.0.1/examples/hello_world/ovm/top.sv
//  http://www.apache.org/licenses/LICENSE-2.0
//luokka top. Hallitsee komponenttien välistä tiedonsiirtoa
class top extends ovm_component;

// esitellään tiedonsiirtoa käyttävät komponentit
// producer 1 liitetään suoraan consumerin exportiin
  producer #(packet) p1;
// producer 2 käyttää tlm_fifo-yhteyttä
  producer #(packet) p2;
// tlm_fifo-puskuri
  tlm_fifo #(packet) f;
// tiedonsiirtoa vastaanottava osapuoli consumer
  consumer #(packet) c;

// ovm:n top-makro
  `ovm_component_utils(top)
// liitetään komponentit ohjelmiston rakenteeseen
  function new (string name, ovm_component parent=null);
    super.new(name,parent);

   p1 = new("producer1",this);
    p2 = new("producer2",this);
    f  = new("fifo",this);
    c  = new("consumer",this);
// yhdistetään tiedonsiirtoväylät
    p1.out.connect( c.in );
    p2.out.connect( f.blocking_put_export );
    c.out.connect( f.get_export );
  endfunction

endclass
```

```
// ovm-2.0.1/examples/hello_world/ovm/hello_world.sv
//  http://www.apache.org/licenses/LICENSE-2.0
// hello_world top-moduuli. Sisältää simuloinnin parametrit ja liittää komponentit
simulaatioon

//asetetaan viiveille # skaala
`timescale 1ns / 1ns

module hello_world;

// sisällytetään komponentit simulaatioon
  `include "ovm.svh"
  `include "packet.sv"
  `include "producer.sv"
  `include "consumer.sv"
  `include "top.sv"

// luodaam top-taso
  top mytop;

  initial begin
        // aikaformaatin määritys
    $timeformat(-9,0," ns",5);
        // tulostuksen asettelu
    ovm_default_table_printer.knobs.name_width=20;
    ovm_default_table_printer.knobs.type_width=50;
    ovm_default_table_printer.knobs.size_width=10;
    ovm_default_table_printer.knobs.value_width=14;
        //pakettien määrä
    set_config_int("top.producer1","num_packets",2);
    set_config_int("top.producer2","num_packets",4);
        // ovm tallennuksen määritteeet
    set_config_int("*","recording_detail",OVM_LOW);
```

```
   ovm_top.enable_print_topology = 1;
          // tulostuksen määritteet
    //ovm_default_printer = ovm_default_tree_printer;
    ovm_default_printer.knobs.reference=0;
    mytop = new("top");
    ovm_default_table_printer.knobs.type_width=20;

          // käynnistetään testaus
    run_test();
  end

// testauksen kesto
  initial #1us ovm_top.stop_request();

endmodule
```

```
// ovm-2.0.1/examples/hello_world/ovm/packet.sv
//  http://www.apache.org/licenses/LICENSE-2.0
// TLM- rajapinnan tarvitsemat asetukset
class packet extends ovm_transaction;

`ifndef NO_RAND
  rand
`endif
// osoitteita varten integer
  int addr;
//ovm-makrojen asetukset
  `ovm_object_utils_begin(packet)
    `ovm_field_int(addr, OVM_ALL_ON)
  `ovm_object_utils_end

// consumerin osoitteiston rajaus
  constraint c { addr >= 0 && addr < 'h100; }

endclass
```

```
// ovm-2.0.1/examples/hello_world/ovm/producer.sv
//  http://www.apache.org/licenses/LICENSE-2.0
// producer- luokka. Luo lähetettävän paketin ja lähettää sen consumerin export- porttiin
// sekä suoraan että käyttämällä tlm_fifo- tiedonsiirtorajapintaa

class producer #(type T=packet) extends ovm_component;

//luodaan yhteys johon kytketään paketti T
  ovm_blocking_put_port #(T) out;

// luokkarakenne
  function new(string name, ovm_component parent=null);
    super.new(name,parent);

// liitetään out-yhteys luokkarakenteeseen
    out = new("out",this);

// haetaan hello_world- moduulista pakettien määrä
    void'(get_config_int("num_packets", this.num_packets));
  endfunction

// muuttujat
  protected T   proto       = new;
  protected int num_packets = 1;
  protected int count       = 0;

//ovm- parametrit edellä luotuihin muuttujiin
  `ovm_component_utils_begin(producer #(T))
    `ovm_field_object(proto, OVM_ALL_ON + OVM_REFERENCE)
    `ovm_field_int(num_packets, OVM_ALL_ON + OVM_DEC)
    `ovm_field_int(count, OVM_ALL_ON + OVM_DEC + OVM_READONLY)
  `ovm_component_utils_end
```

```
// luokan toiminnallisuus
//------------------------------------------------------------------------
  task run();
    T p;
    string image, num;

// tulostus ovm_infol- makrolla
    `ovm_info1(("Starting."))

//silmukka pyörii kunnes num_packets saavutetaan
   for (count =0; count < num_packets; count++) begin

// tyyppimuunnos p:lle
      $cast(p, proto.clone());

// tyyppimuunnos num-muutujaa varten integeristä stringiksi
      num.itoa(count);

// print-käskyä varten tallennetaan producerin nimi ja numero
      p.set_name({get_name(),"-",num});
      p.set_initiator(this);

// Jos ovm tallennus on käynnissä, aloitetaan pakettien generointi
      if (recording_detail!=OVM_NONE)
        p.enable_recording("packet_stream");

// satunnaislukujen generointi pakettien lähettämistä varten
      void'(p.randomize());
// tulostus ovm_infol- makrolla
      `ovm_info1(("Sending %s",p.get_name()))

      if (`ovm_msg_detail(OVM_HIGH))
```

```
// tulostetaan p
        p.print();

// paketin lähettäminen consumerille
      out.put(p);

      #10;

    end
// tulostus ovm_infol- makrolla
    `ovm_info1(("Exiting."))

  endtask

endclass
```

```
// ovm-2.0.1/examples/hello_world/ovm/consumer.sv
//  http://www.apache.org/licenses/LICENSE-2.0
// consumer- luokka. Vastaanottaa producer- luokan lähettämät paketit ja tulostaa paketin tiedot
class consumer #(type T=packet) extends ovm_component;

// Valmistellaan pakettien vastaanotto
  ovm_blocking_put_imp #(T,consumer #(T)) in;
  ovm_get_port #(T) out;

// luokkarakenne
  function new(string name, ovm_component parent=null);
    super.new(name,parent);
// exportin rakenne
    in=new("in",this);
    out=new("out",this,0);
  endfunction

// muuttujat
  protected int count=0;
  local semaphore lock = new(1);

// ovm- parametrit muuttujille
  `ovm_component_utils_begin(consumer #(T))
    `ovm_field_int(count,OVM_ALL_ON + OVM_READONLY + OVM_DEC)
  `ovm_component_utils_end

// luokan toiminnallisuus
//-------------------------------------------------------
```

```
  task run ();
```



```
  T p;
    while(out.size()) begin
      out.get(p);
      put(p);
    end
  endtask

  task put (T p);
    lock.get();
    count++;
//    void'(accept_tr(p));
// ovm:n tiedonsiirtoa (transcation) hallitsevat komennot
    accept_tr(p);
    #10;
    void'(begin_tr(p));
    #30;
    end_tr(p);

// tulostus ovm_infol- makrolla
    `ovm_info1(("Received %0s local_count=%0d",p.get_name(),count))
    if (`ovm_msg_detail(OVM_HIGH))
// tulostetaan p
      p.print();
// syötetään tedot luokan porttiin
    lock.put();

  endtask
endclass
```