TIETO- JA SÄHKÖTEKNIIKAN TIEDEKUNTA

ELEKTRONIIKAN JA TIETOLIIKENNETEKNIIKAN TUTKINTO-OHJELMA

KANDIDAATINTYÖ

UVM järjestelmäpiirien verifioinnissa IP-lohkotasolla

Tekijä Touko Kinnunen

Ohjaaja Jukka Lahti

Helmikuu 2023

Kinnunen T. (2023) UVM järjestelmäpiirien verifioinnissa IP-lohkotasolla. Oulun yliopisto, tieto- ja sähkötekniikan tiedekunta, elektroniikan ja tietoliikennetekniikan tutkinto-ohjelma. Kandidaatintyö, 23 s.

TIIVISTELMÄ

Tässä kandityössä esitellään UVM ja pohditaan sen käyttöä järjestelmäpiirien verifioinnissa IP-lohkotasolla. Työ on katsaus UVM:n perusteisiin, RTL-tason verifiointiin, SystemVerilogiin ja olio-ohjelmointiin. Tarkastelu keskittyy syvemmin UVM:n käyttöön järjestelmäpiirin CPU:n väylän ympärille suunniteltavan logiikan verifioinnissa. UVM-komponenteista keskitytään tarkastelemaan pääosin agenttia.

Avainsanat: Universaali varmennusmenetelmä, SystemVerilog, System-on-Chip, verifiointi.

Kinnunen T. (2023) UVM in the verification of SoCs at IP block level. University of Oulu, Degree Programme in Electronics and Communications Engineering. Master's Thesis, 23 p.

ABSTRACT

This Bachelor's thesis is about UVM and the main focus is on its use in the verification of SoC at IP block level. The work is an overview of the basics of UVM, RTL level verification, SystemVerilog and object-oriented programming. The review focuses more deeply on the use of UVM in the verification of the logic designed around the SoC's CPU bus. Of the UVM components, the focus is mainly at the agent.

Key words: Universal Verification Methodology, SystemVerilog, System-on-Chip, verification.

SISÄLLYSLUETTELO

TIIVISTELMÄ
ABSTRACT
SISÄLLYSLUETTELO
ALKULAUSE
LYHENTEIDEN JA MERKKIEN SELITYKSET
1 JOHDANTO 8
2 RTL-TASON VERIFIOINTI 9
2.1 Tavoite 10
2.2 Spesifikaatiot 10
2.3 Testipenkki 10
3 SYSTEMVERILOG 11
3.1 Perusteet 11
3.2 Ominaisuudet 11
3.2.1 Olio-ohjelmointi 11
3.2.2 Liitynnät 12
4 UVM 13
4.1 Perusteet 13
4.2 UVM-luokkahierarkia 13
4.3 UVM-testipenkin rakenne 14
4.4 TLM-kommunikaatio 15
4.5 UVM-luokat 16
4.5.1 UVM-komponentit 16
4.5.1.1 UVM-testi 16
4.5.1.2 UVM-ympäristö 16
4.5.1.3 UVM-agentti 17
4.5.1.4 UVM-sekvensseri 17
4.5.1.5 UVM-ohjain 17
4.5.1.6 UVM-monitori 17
4.5.1.7 UVM-tilaaja 18
4.5.1.8 UVM-tulostaulu 18
4.5.2 Muut UVM-luokat 18
4.5.2.1 Konfiguraatiot 18
4.5.2.2 UVM-tehdas 18
4.6 UVM-simulointivaiheet 19
4.6.1 Rakennusvaihe 19
4.6.2 Kytkentävaihe 19
4.6.3 Suoritusvaihe 19
4.6.4 Raportointivaihe 19
4.7 UVM IP-lohkojen verifioinnissa 19
5 POHDINTA 20

6 YHTEENVETO 21
7 LÄHDELUETTELO 22

ALKULAUSE

Opiskelen elektroniikka ja tietoliikennetekniikan tutkinto-ohjelmassa Oulun Yliopistossa. Tarkoituksena on valmistua kolmannen opiskeluvuoteni keväällä ja aloitan samaan aikaan työt teknologia-alan yrityksessä digitaalisuunnittelu ja -verifiointi tiimissä. Kävin kolmannen opiskeluvuoteni syksyllä Digitaalitekniikka 2 -kurssin, jonka myötä kiinnostuin hakemaan töitä digitaalisuunnittelun ja -verifioinnin puolelta. Huomasin kandityön aihetta miettiessä, että monissa yrityksissä järjestelmäpiirien verifiointi toteutetaan pohjautuen universaaliin varmennusmenetelmään (Universal Verification Methodology, UVM). Kokemukseeni perustuen monet muut opiskelijat ovat samassa tilanteessa, että töihin mennessä heillä ei ole aiempaa kokemusta UVM:stä.

UVM on aiheena monimutkainen ja teoreettinen eikä tilannetta ainakaan parantanut se, ettei minulla ollut ollenkaan kokemusta UVM:stä ennen tämän työn tekemistä. Haluan kiittää Jukka Lahtea hyvästä ohjauksesta sekä vaimoani Edlaa tuesta kandia kirjoittaessa. Tämän työn tekeminen oli erityisesti opettavaista.

Oulussa 10.2.2023

Touko Kinnunen

LYHENTEIDEN JA MERKKIEN SELITYKSET

UVM	Universal Verification Methodology. Universaali varmennusmenetelmä.
SoC	System-on-Chip. Järjestelmäpiiri.
IP	Intellectual property. Valmislohko.
RTL	Register transfer level. Rekisterisiirtotaso.
I/O	Input/Output. Sisääntulo/Ulostulo.
USB	Universal Serial Bus. Universaali sarjaväylä.
DUT	Device under test. Testattava laite.
VHDL	Very High-speed integrated circuit hardware Description Language. Erittäin nopea integroitu laitteistonkuvauskieli.
OPP	Object-Oriented Programming. Olio-ohjelmointi.
OVM	Open Verification Methodology. Avoin varmennusmenetelmä.
TLM	Transaction level modeling. Transaktiotason mallinnus.

1 JOHDANTO

Tämän kandityön tarkoituksena on esitellä lukijalle universaali varmennusmenetelmä (Universal Verification Methodology, UVM) ja pohtia sen käyttöä järjestelmäpiiriin (System-on-Chip, SoC) suunniteltavien rekisterisiirtotason (Register Transfer Level, RTL) valmislohkojen (Intellectual property, IP) verifioinnissa. Työn kautta pyrin keräämään yhteen olennaiset tiedot lukijalle, joka ei entuudestaan tunne UVM:ää, mutta tietää tulevansa sen kanssa tekemisiin. Tavoitteena on helpottaa monimutkaisen ja vaikeasti ymmärrettävän UVM:n oppimista, sekä kertoa lukijalle olennaiset asiat UVM:n käytöstä järjestelmäpiirin IP-lohkojen verifioinnissa.

Maailmassa erilaisten järjestelmäpiirien määrä kasvaa jatkuvasti ja piirien suunnittelusta tulee yhä monimutkaisempaa. Tämä lisää myös monimutkaisuutta jo entuudestaan työlääseen verifiointiin. Verifioinnin osuus koko piirin suunnittelusta on yli 70 prosenttia [1]. Aikaa vievää RTL-suunnittelua ja -verifiointia voidaan vähentää suunnittelemalla ja verifioimalla järjestelmäpiirejä RTL:ää korkeammilla kuvauskielillä ja käyttämällä suunnitelmissa uudelleen käytettäviä IP-lohkoja [2]. Jonkun täytyy suunnitella ja varmentaa myös nämä uudelleen käytettävät osalohkot (IP-lohkot). Kaikki verifioinnissa säästetty aika säästää myös resursseja. UVM kehitettiin verifiointiympäristön uudelleen käyttämiseksi [3]. Pienillä koodimuutoksilla verifiointiympäristöä voidaan hyödyntää uudelleen uudessa suunnitelmassa [3].

Pyrin vastaamaan tässä työssä seuraaviin tutkimuskysymyksiin:

- Mikä on UVM?
- Mihin UVM:ää käytetään ja miten?
- Miten UVM-testipenkki rakentuu?
- Miltä tutkinto-ohjelman opiskelijana perehtyminen UVM:ään tuntuu?

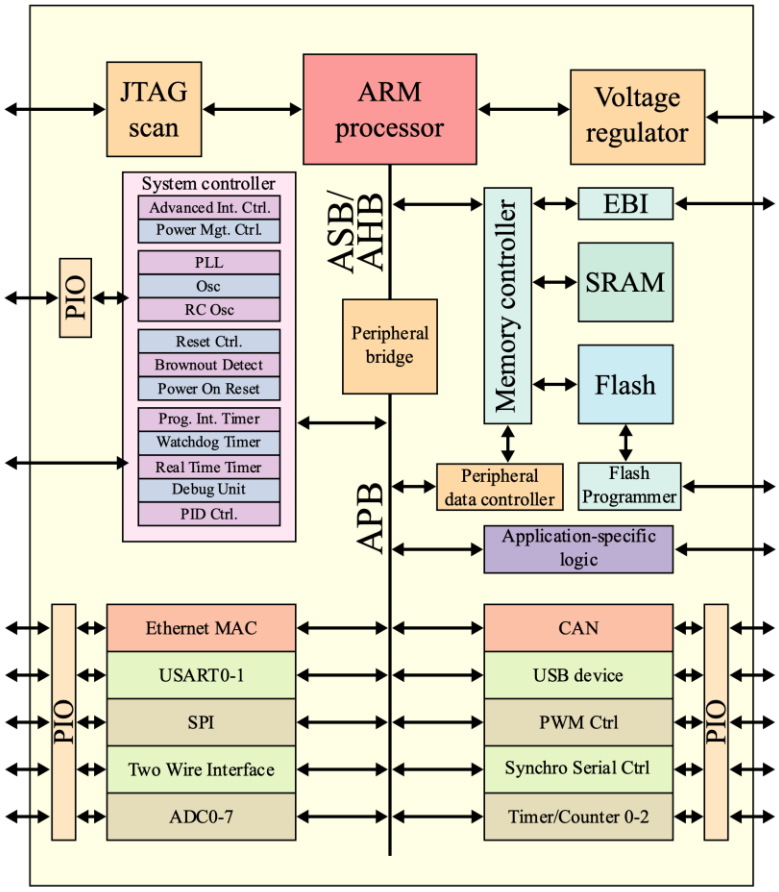
Tämä kandityö toteutetaan kirjallisuustutkimuksena. Tutkimuksen perustana on kasvattaa omaa tietämystä aiheesta, koota tutkimani tiedot kandidaatintyöksi ja välittää lukijalle kootut tiedot tutkimastani aiheesta. Tässä työssä ei ole teknistä osuutta, esimerkiksi UVM-testipenkin koodaamista havainnollistamassa UVM:n käyttöä. Olen kuitenkin aiheeseen perehtyessäni tutustunut UVM:n käyttöön käytännössä.

2 RTL-TASON VERIFIOINTI

RTL-suunnittelu sisältää monta vaihetta. Ensin luodaan spesifikaatiot ja arkkitehtuuri suunniteltavalle piirille. Tämän jälkeen SoC tai sen sisältämät osalohkot suunnitellaan jollakin laitteistokuvauskielellä esim. SystemVerilogilla. RTL-tason suunnittelu tarkoittaa synkronisen digitaalilogiikan suunnittelua signaali tasolla (I/O) rekisterien välille [4]. RTL-tason suunnittelussa työläin vaihe on valmiin laitteistonkuvauksen varmennus eli RTL-tason verifiointi.

Tässä luvussa RTL-tason verifiointi esitellään yleisesti. Kerrotaan, miksi se on merkityksellistä, mitä suunnitelmasta tulee tietää ennen verifiointia ja miten verifiointi toteutetaan. Tässä työssä RTL-tason verifioinnissa keskitytään etenkin SoC verifiointiin IP-lohkotasolla. Kuvasta 1 näet lohkokaavion kautta käytännön esimerkin, kuinka SoC tyypillisesti koostuu suorittimesta (engl. processor) sekä sen väylän ympärille suunniteltavista IP-lohkoista.

IP-lohkot ovat lisensioituja tiettyyn käyttötarkoitukseen suunniteltuja funktionaalisia valmislohkoja, joita voi ostaa ja käyttää SoC suunnitelmissa ja toteutuksissa [5, s. 8]. Nykypäiväisissä järjestelmäpiireissä IP-lohkoja voi olla satoja [5, s. 8]. Esimerkkinä voisi käyttää muistilohkoa, USB-liitäntä moduulia tai radiolohkoa. UVM on käytännöllinen tällaisten lohkojen verifioinnissa.



Kuva 1. Esimerkki SoC:n lohkokaaviosta [6]

2.1 Tavoite

Laitteistonkuvaus on valmis koodi eli mallinnus, joka kuvaa miten suunniteltu piiri, suoritin tai lohko toimii. Laitteistonkuvauksen pohjalta voidaan tehdä logiikkasynteesi, joka kuvaa porttitasolla suunnitellun piirin logiikan. Logiikkasynteesin tuloksia voidaan käyttää syötteenä layout-suunnittelulle, jonka perusteella voidaan valmistaa fyysinen tuote. Ensin on kuitenkin erittäin tärkeää varmistaa, että laitteistonkuvaus on suunniteltu oikein. Fyysiseen tuotteeseen päätyneitä virheitä ei voi enää muuttaa.

RTL-tason verifiointi varmistaa suunnitellun laitteistonkuvauksen virheettömyyden, ennen kuin sitä käytetään suuremmissa kokonaisuuksissa tai siitä valmistetaan tuote. Virheiden paikallistaminen suuressa kokonaisuudessa voi olla hyvin haasteellista. Tämän takia yksittäiset RTL-tason komponentit ja lohkot on tärkeää verifioida huolellisesti. Virheiden ilmaantuessa ne voidaan korjata ja estää niiden vaikutus korkeamman hierarkian lohkoissa. [7, s. 9]

2.2 Spesifikaatiot

Piirin laitteistonkuvauksesta tehdään spesifikaatiot, jotka kertovat miten piiri toimii, mitä siitä testataan ja miten testataan. Suunnitteludokumentti kertoo, miten piiri on suunniteltu, minkälaisia toimintoja ja rekistereitä piirissä on. Spesifikaatioiden ja suunnitteludokumentin avulla verifiointi-insinööri osaa poimia asiat, jotka piiristä tulisi testata. [5, s. 9–10]

2.3 Testipenkki

Testipenkki on arkkitehtuuri, joka antaa testeille rakenteen, luo mahdollisuuden kommunikointiin testattavan laitteen (Device Under Test, DUT) kanssa ja tarkkailee, miten testi käyttäytyy DUT:n kanssa [8, s. 4]. Testipenkin avulla DUT:lle syötetään tietoa, tarkkaillaan sen käyttäytymistä ja varmennetaan, toimiiko DUT suunnitellulla tavalla [8, s. 4]. Testipenkeistä voidaan luoda erilaisia toteutuksia useilla eri kielillä, mutta päämäärä kaikissa testipenkeissä on sama [7, s. 10].

3 SYSTEMVERILOG

Tässä luvussa esitellään SystemVerilog, joka on laitteiston suunnittelu- määrittely- ja verifiointikieli [9]. Tarkoituksena on esitellä SystemVerilogista olennaiset asiat verifiointiin liittyen, jotta lukija ymmärtäisi paremmin UVM:stä kertovat luvut.

3.1 Perusteet

SystemVerilog pohjautuu Verilog kieleen, josta se on kehittynyt uusissa standardeissa lisättyjen laajennuksien myötä säilyttäen kuitenkin Verilog kielen toiminnallisuudet. SystemVerilog ei ole siis itsessään uusi kieli, vaan laajennus Verilog-kielestä. Standardointiensa ansiosta SystemVerilog saavuttanut nykyisen tilansa, jossa se sopii sekä suunnitteluun että verifiointiin. SystemVerilog perustuu Verilog-, VHDL- ja C/C++ -rakenteisiin, joita insinöörit ovat käyttäneet vuosikymmeniä [10]. SystemVerilogin uusin standardi on julkaistu vuonna 2017. [9], [10]

Verifioijien ja suunnittelijoiden toimet on nähty aikojen saatossa pohjimmiltaan erilaisina. Tämä on johtanut kielten kohdennettuun ja suppeaan kehitykseen, joka on ajanut tieteenalojen välisen kommunikaation ahtaalle. SystemVerilogin mukana tuomat laajennukset ovat ratkaisseet kommunikaatio ongelmat; kenenkään ei tarvitse luopua tarpeellisista ominaisuuksista sekä yhteinen syntaksi ja semantiikka parantaa viestintää osapuolien välillä. [10]

SystemVerilog ja VHDL (Very High-speed integrated circuit hardware Description Language) ovat tunnetuimpia laitteistonkuvauskieliä, joiden avulla piiri voidaan kuvata abstraktin tason esityksenä järjestelmästä. Tämä mahdollistaa simulaation ja testauksen ennen fyysisten komponenttien valmistamista. SystemVerilog on käytetyin kieli verifioinnissa etenkin sen monipuolisten toiminnallisuuksien ja UVM yhteensopivuuden ansiosta [11], [12]. Varmennetun laitteistonkuvauksen pohjalta piiristä muodostetaan logiikkaportti-tason piirustus fyysisen komponentin valmistusta varten.

3.2 Ominaisuudet

SystemVerilogilla on paljon hyödyllisiä ominaisuuksia suurten ja kompleksisten kokonaisuuksien verifiointiin etenkin sen laajennusten ansiosta. UVM:n ymmärtämisen kannalta on tärkeää ymmärtää tiettyjen SystemVerilogin ominaisuuksien perusteet, kuten olio-ohjelmointi.

3.2.1 Olio-ohjelmointi

Olio-ohjelmoinnin (Object-Oriented Programming, OOP) ideana on kapseloida tiedot ja niitä käsittelevät toiminnallisuudet malleiksi, joita kutsutaan luokiksi (engl. class). Jokainen luokka omaa tiettyjen tietojen ja toiminnallisuuksien kautta tietyn tehtävän. Luokasta muodostettava olio (engl. object) toimii kahvana (engl. handle), jotta luokan ominaisuuksiin ja metodeihin päästään käsiksi [13]. Olio-ohjelmointi on tapa kuvata järjestelmien toimintaa ja rakennetta [14]. [10, ss. 131–132]

Luin hyvän esimerkin, jossa olio-ohjelmoinnin mahdollistamaa rakennetta verrattiin nykypäivän autoon. Hyvin alkukantaisessa autossa piti itse säätää, tarkkailla ja olla tietoinen auton toiminnoista. Piti säätää ryyppyä, moottorin kierroksia ja olla tietoinen renkaiden pidosta eri keliolosuhteissa. Nykyään vuorovaikutus auton kanssa on korkeatasoista, vaikka toiminnallisuudet ja tiedot ovat paljon kompleksisimpia. Kuljettaja suorittaa vain korkean tason toimintoja, kuten kaasuttaa, laittaa vilkun päälle, kääntää rattia tai pysähtyy ja autossa eri tasoiset luokat hoitavat omia tehtäviään. Liukkaalla kelillä ABS-jarrut toimivat ilman eri käskyä. Kuljettajan ei tarvitse ajatella matalan tason yksityiskohtia. [10, s. 132]

Samalla tavalla olio-ohjelmointi mahdollistaa testipenkin rakenteellisen luokituksen, jossa luokat pystyvät kommunikoimaan toistensa kanssa. Olio-ohjelmoinnilla luotua testipenkkiä on helppo hyödyntää uudelleen, kun osaa sen perusteet. Vertauksena esimerkkiin; autolla ajaminenkin tarvitsee opetella eikä autoa tarvitse kehittää aina uudelleen, vaikka sen malli ja osa toiminnoista muuttuisi. Eli auto on luokka ja erilaiset autot ovat olioita [14]. [10, s. 132]

Taulukko 1 listaa olio-ohjelmoinnin tärkeimmät termit.

Taulukko 1. Olio-ohjelmoinnin terminologia [10, ss. 134–135], [14]

Nimi	**Selitys**
Luokka (engl. Class)	Olio-ohjelmoinnin peruspalikka, joka määrittelee tietyn kokonaisuuden. Luokka sisältää ominaisuuksia eli jäsenmuuttujia (property) ja toiminnallisuuksia eli jäsenfunktioita (method).
Olio (engl. Object)	Luokan esiintymä, joka omaa luokan ominaisuudet ja toiminnallisuudet.
Kahva (engl. Handle)	Osoittaa olioon. Se on kuin olion osoite.
Jäsenmuuttuja (engl. property)	Jäsenmuuttuja määrittelee luokalle tietyn ominaisuuden.
Jäsenfunktio (engl. Method)	Jäsenfunktio määrittelee luokalle tietyn toiminallisuuden. Usein jäsenfunktio käyttää tehtävän suorittamiseen jäsenmuuttujien arvoja.

Luokkien periytyminen on tärkeä osa olio-ohjelmointia. Uusia luokkia voidaan muodostaa niin, että olemassa olevan luokan ominaisuudet ja toiminnallisuudet periytyvät uudelle luokalle sisällyttäen mukaan myös uusia ominaisuuksia ja toiminnallisuuksia. Luokista voidaan muodostaa niiden alle ns. erikoistapauksia. Alkuperäistä luokkaa kutsutaan *kantaluokaksi* ja uutta luokkaa *aliluokaksi*. [15, s. 118]

3.2.2 Liitynnät

Liitynnät (engl. interface) tarjoavat lohkojen tiedonsiirron välille rajapinnan. Liityntä muodostaa oman rakenteen, jossa se käsittelee signaaleja. Niiden toiminta muistuttaa jokseenkin luokkien toimintaa. Lohkot voivat muodostaa yhteyksiä toisensa välille kutsumalla liityntöjen toimintoja. Liitynnät ovat siis helposti uudelleen käytettävissä tai korvattavissa jos esimerkiksi kommunikaation abstraktio tasoa täytyy muuttaa. [9, s. 49]

Liitynnät voivat sisältää prosesseja tai jatkuvia tehtäviä, jotka ovat hyödyllisiä testipenkkien rakentamisessa. Liityntä voi sisältää esimerkiksi oman protokollan tarkistusohjelman, joka tarkastaa siihen liitettyjen moduulien protokollien oikeanmukaisuuden. [9, s. 49]

4 UVM

Tässä luvussa esitellään UVM ja sen perusteita.

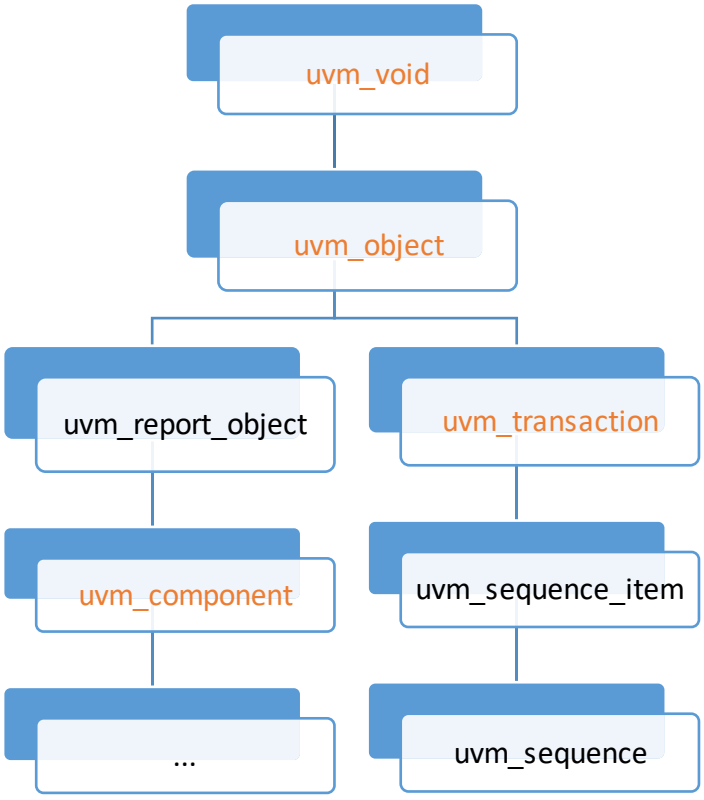
4.1 Perusteet

UVM standardi kehitettiin yhtenäistämään, nopeuttamaan ja mukauttamaan verifiointi ympäristöjä ja testipenkkien tekemistä. UVM:n avulla testipenkeistä saadaan joustavia, uudelleen käytettäviä ja yhteentoimivia. UVM:ää oli kehittämässä maailmanlaajuisesti suurimmat teknologia-alan yritykset. UVM:n pohjana toimii Open Verification Methodology (OVM). [8, s. 1], [16]

UVM:n käyttö edellyttää SystemVerilogin ja olio-ohjelmoinnin tuntemista. UVM perustuu testipenkin rakenteelliseen luokittamiseen olio-ohjelmointiin perustuen sekä SystemVerilogin toimintoja hyödyntäen.

4.2 UVM-luokkahierarkia

UVM-hierarkiassa on neljä pääluokkaa, joista muut luokat periytyvät (kuva 2). [12, Luku 5], [8, s. 6]

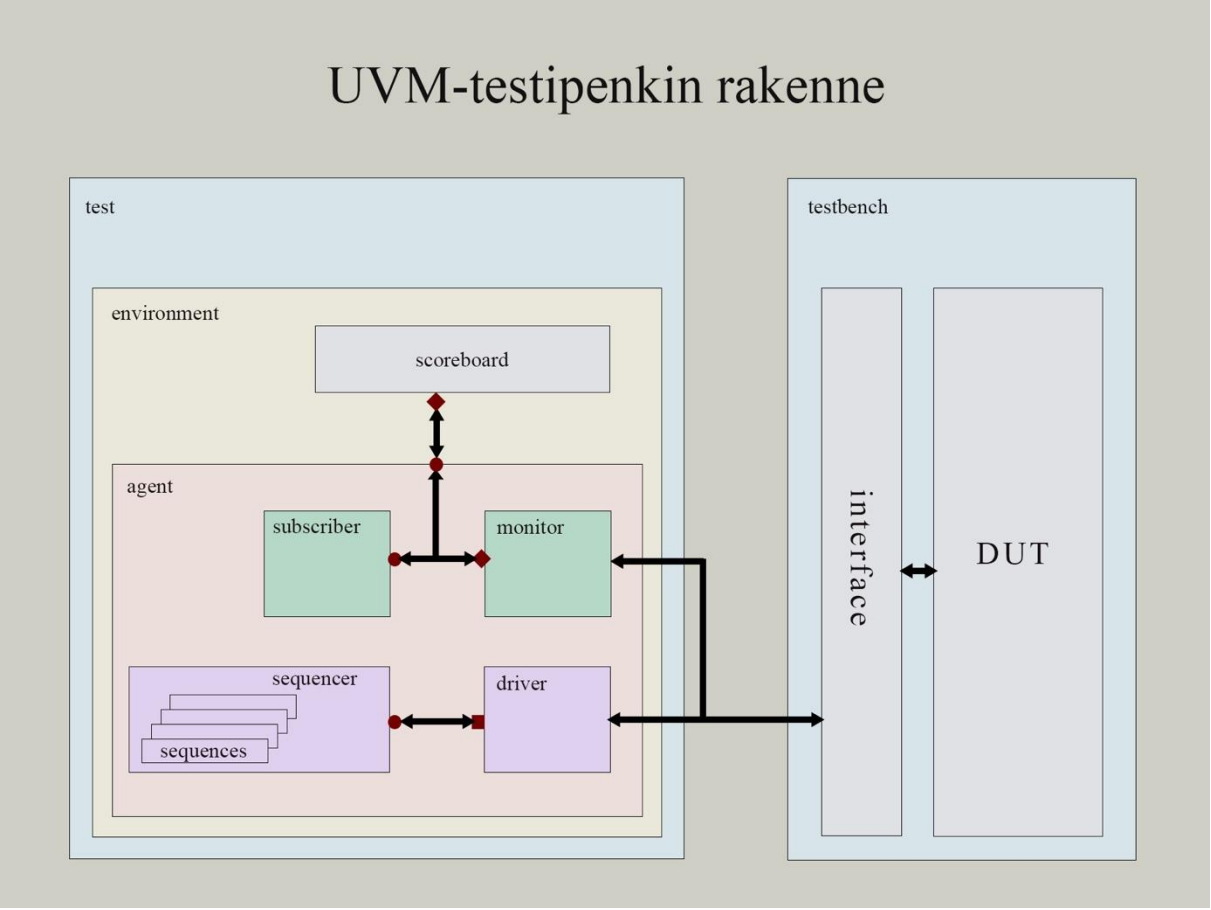


Kuva 2. UVM-pääluokkien hierarkia.

uvm_void luokka on kaikille yhteinen abstrakti kantaluokka, joka ei sisällä jäsenmuuttujia tai -funktioita. *uvm_object* luokka määrittää joukon perusominaisuuksia ja -toiminnallisuuksia, jotka periytyvät sen aliluokille. Tärkeitä *uvm_object* luokan funktioita ovat mm. create, copy, compare, print, sprint ja record. *uvm_component* luokka on testipenkkiin tarvittavien komponenttien kantaluokka. uvm_report_object lisää uvm_object luokkaan raportointiominaisuudet. uvm_transaction luokka on kantaluokka transaktioille, jossa on uvm_object luokan metodien lisäksi ajanhallintaan ja tallennukseen työkaluja. [8, s. 6], [12, Luku 5]

4.3 UVM-testipenkin rakenne

UVM-testipenkki muodostuu *uvm_component* luokan aliluokista. UVM-komponentit ja niiden toiminta esitellään myöhemmin. Alla (kuva 3) näet UVM-testipenkin rakenteen eli *uvm_component* luokan aliluokkien hierarkian. Testipenkin rakenne ja etenkin agentin toiminnot ovat olennaisia verifioinnissa IP-lohkotasolla UVM:ää hyödynnettäessä. [8, ss. 4–5]



Kuva 3. UVM-testipenkin rakenne

Jokaisella komponentilla on hyvin suunniteltu oma tehtävänsä. UVM-testipenkki mahdollistaa verifioijan ajattelun transaktiotasolla keskittyen siihen mitkä toiminnallisuudet pitää verifioida. Samaan aikaan UVM-testipenkki huolehtii siitä, miten testi käyttäytyy DUT:n

kanssa. Transaktiotason viestitys tapahtuu *test* luokan sisällä (nuolien päädyissä pallot ja neliöt) ja signaalitason viestitys monitorin (engl. monitor), ohjaimen (engl. driver), SystemVerilogin virtuaalisen *interface*-käyttöliittymän ja DUT:n välillä. [6, s. 4–5, 25–28, 74–78]

Verifioija voi lisätä tai poistaa UVM-komponentteja haluamansa ja tarpeidensa mukaan, mutta tässä tapauksessa (kuva 3) on esitelty ainoastaan käytetyimmät UVM-komponentit. Usein esimerkiksi agentteja saattaa olla useampi ja joissakin tapauksissa agentista voidaan tehdä passiivinen, jolloin käytössä on pelkkä monitori [17].

4.4 TLM-kommunikaatio

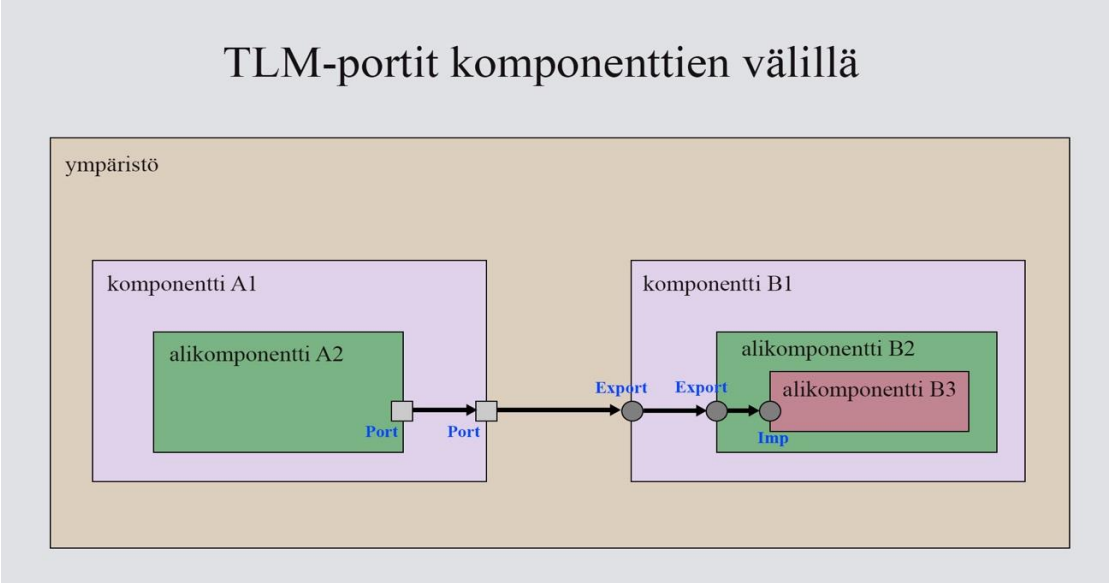
UVM-komponenttien väliseen tiedonsiirtoon käytetään tapahtumatason mallinnusta (Transaction Level Modeling, TLM). Viestintä perustuu datapakettien lähettämiseen rajapintojen kautta. Viestintää ei siis kuvata signaalitasossa. Jokaisella luokalla on oma rajapinta, jonka kautta viesti kulkee; esimerkiksi monitori (engl. monitor) ei voi viestittää suoraan tulostaululle (engl. scoreboard) vaan viesti kulkee agentin (engl. agent) rajapinnan kautta (ks. kuva 3). [2], [8], [12, s. 120]

Rajapinnoilla oleva porttityyppi vaihtelee kommunikaation luonteen perusteella [8]. TLM porttityypit ovat esitelty taulukossa 2.

Taulukko 2. TLM porttityypit [8], [18]

Yleisimmät TLM porttityypit	
port	TLM-liitin, joka käynnistää tiedonsiirron ja välittää paketteja korkeammalle hierarkia tasolle.
export	TLM-liitin, joka hyväksyy ja toteuttaa tiedonsiirron ylemmiltä hierarkia tasoilta määränpäähän.
imp	TLM-liitin, jota käytetään määrittämään sijoitusmenetelmä komponenteissa, joissa on metodit määrittelyä varten.
analysis	TLM-liitin, joka toimii analyysiporttina. Jakaa tietoa useammalle kohteelle, jotka tilaavat *analysis* transaktioita.

Porttityypeistä *analysis* on niin sanottu erikoistapaus ja muut ovat tavallisia porttityyppejä. Porttityypeillä on myös erilaisia kommunikointitapoja: *blocking*, *non*-blocking ja *combination*. Alla oleva kuva 4 havainnollistaa tavallisten porttityyppien käyttöä.



Kuva 4. TLM-portit komponenttien välillä. [18]

4.5 UVM-luokat

Tässä osiossa esitellään UVM-luokat. Tarkemmin UVM-komponentit luokka ja muita UVM-luokkia, joita ei ole vielä aiemmin esitelty.

4.5.1 UVM-komponentit

UVM-komponentit on UVM-luokka, jonka avulla UVM-testipenkki rakennetaan. UVM-agentti on oleellinen komponentti. Sen toiminta ja merkitys on hyvä ymmärtää UVM:ää käytettäessä IP-lohkojen verifiointiin. UVM-komponenttien periytymisen ja hierarkian näet kuvasta 2. Sitä kannattaa käyttää hyödyksi, kun seuraavaksi esitellään tavallisimmat ja käytetyimmät UVM-komponentit.

4.5.1.1 UVM-testi

UVM-testi (engl. test) on testipenkin ylimmän tason virtuaalinen UVM-komponentti, joka rakentaa ja konfiguroi UVM-ympäristön (engl. environment) sekä suorittaa testin. UVM-testi sisällyttää kaikki testiin tarvittavat UVM-komponentit. Testi käynnistetään initial-proseduurissa kutsumalla funktiota *run_test()*. [8, s. 5], [12, s. 172]

4.5.1.2 UVM-ympäristö

Tyypillisesti UVM-ympäristö (engl. environment, env) sisältää UVM-agentin, konfiguraatio osan ja muita komponentteja, mutta tämä on aina tapauskohtaista. Koska UVM-ympäristö on itsessään komponentti, sitä on mahdollista uudelleen käyttää osana suurempaa kokonaisuutta. [8, s. 5], [12, s. 173]

4.5.1.3 UVM-agentti

UVM-agentti (engl. agent) on UVM hierarkiassa UVM-ympäristön aliluokka. Agentti sisältää tärkeitä ja keskeisiä komponentteja IP-lohkojen verifiointiin liittyen. Tyypillisimpiä niistä ovat ohjain, joka hoitaa signaalitasoisen viestinnän DUT:n kanssa, monitori, joka analysoi DUT:lta tulevia signaaleja sekä sekvensseri, joka hoitaa transaktioiden syöttöä. Agentti myös hoitaa kaiken protokollakohtaisen viestinnän DUT:n kanssa. [8]

Agentti tyyppejä on kaksi; aktiivinen tai passiivinen [12, ss. 173–174], [17], [19]. Nämä agentti tyypit ovat esitelty tarkemmin taulukossa 3.

Taulukko 3. Agentti tyypit [12, ss. 173–174], [17], [19]

aktiivinen agentti	- sisältää kolme komponenttia: ohjain, monitori, sekvensseri - mahdollista data siirron DUT:lle ja luvun DUT:lta
passiivinen agentti	- sisältää ainoastaan monitorin - tarkoitettu DUT:lta tulevan datan tarkkailuun ja raportointiin

4.5.1.4 UVM-sekvensseri

UVM-sekvensseri (engl. sequencer) on agentin tärkeä osa, joka on vastuussa sen sekvenssien transaktiovirtojen suorittamisesta ja ohjaamisesta ohjaimen ja sekvenssin välillä. Sekvenssi on siis sekvensserin aliluokka, joka muodostaa transaktioita ja sekvensseri ohjaa niitä (ks. kuva 3). [8], [12]

Sekvensserityyppejä on kaksi; ohjain aloittaa uusien transaktioiden pyynnöt (uvm_sequencer #(REQ,RSP)) tai sekvensseri puskee transaktioita ohjaimelle (uvm_push_sequencer #(REQ,RSP)). Oletusarvoisesti sekvensserityyppi on REQ. [8], [12, ss. 175–176, 194–195]

4.5.1.5 UVM-ohjain

UVM-ohjain (engl. driver) vastaa transaktiotasolla TLM viestinnästä sekvensserin kanssa ja muuttaa TLM pohjaisen viestinnän signaalitason ajoitetuksi viestinnäksi DUT:lle virtuaalisen interface rajapinnan avulla. Ohjain siis käsittelee TLM transaktiot ja viestittää ne protokollan mukaisesti signaaleina DUT:lle. Ohjain pystyy erikoistapauksissa toimimaan myös molemmin suuntaisena viestittäjänä. Ohjain sisältää suunnitelmakohtaiset signaalitason protokollatiedot. Ohjain on agentin tärkeä aliluokka. [8, s. 15]

4.5.1.6 UVM-monitori

UVM-monitori (engl. monitor) on agentin tärkeä osa, jolla on samanlainen tehtävä kuin ohjaimella; kääntää signaalitason viestitystä transaktiotason viestitykseksi. Suurin ero ohjaimen ja monitorin välillä on, että ne on tarkoitettu päinvastaisiin tehtäviin. Monitori käsittelee DUT:lta virtuaalisen interface rajapinnan kautta tulevaa signaali tason viestintää ja muuttaa sen

protokollan mukaiseksi transaktiotason viestinnäksi. Monitori on passiivinen komponentti, joka tarkoittaa, että se toimii jatkuvasti ja välittää *analysis* TLM-porttityypin kautta tietoa kaikille tilaajille. Samoin kuin ohjain, monitori sisältää signaalitason protokollatiedot viestinnästä DUT:n kanssa. [8, ss. 16–17]

4.5.1.7 UVM-tilaaja

UVM-tilaaja (engl. subscriber) analysoi saamaansa dataa. Tilaaja omaa *export* TLM-liittimen *analysis* transaktioiden vastaanottamiseen, joka tarkoittaa, että se tilaa itselleen siihen liitetyn *analysis* liittimen transaktiopaketteja eli kuuntelee *analysis* porttia. Tilaaja tyypillisesti analysoi onko monitorin toiminta protokollan mukaista. [12, s. 176], [20]

4.5.1.8 UVM-tulostaulu

UVM-tulostaulu (engl. scoreboard) kertoo toimiiko DUT halutulla tavalla vai ei. Se omaa samalla tavalla *export* liittimen kuin tilaajakin, jolloin se kuuntelee siihen liitettyä monitorin *analysis* liitintä. Tulostaulu on tyypillisesti sijoitettu ympäristön alle aliluokaksi. [8, ss. 124–130], [12, s. 174], [21]

Tulostaulu on usein vaikea toteuttaa. Sen toteutuksen voi jakaa kuitenkin kahteen vaiheeseen. Ensimmäisessä vaiheessa määritellään oikeat toiminnallisuudet ja toisessa vaiheessa tarkastellaan DUT:n käyttäytymistä ja verrataan vastaako se määriteltyjä toiminnallisuuksia. Tulostaulussa kannattaa erottaa omiin aliluokkiinsa toiminnallisuuksien määrittely ja tulosten arviointi, jolloin niitä on helppo uudelleen käyttää tai korvata. [8, s. 124]

4.5.2 Muut UVM-luokat

UVM:n pääluokkien lisäksi UVM:ssä on muita toiminnallisia luokkia, jotka esitellään tässä lyhyesti.

4.5.2.1 Konfiguraatiot

Konfiguraatiotietokanta tarjoaa pääsyn kehitettyyn tietokantaan, jonne voi tallettaa ja sieltä voi hakea komponentin tyyppikohtaisia tietoja, kuten tietoja liitännöistä ja hierarkiasta. Tietokannan rajapinta on yhteensopiva komponenttikirjaston kanssa, joka mahdollistaa komponenttien tiedonhaun milloin tahansa simulaation aikana. [12, s. 392]

4.5.2.2 UVM-tehdas

UVM-tehtaan tarkoituksena on mahdollistaa jonkin olion korvaaminen toisella oliolla tarvitsematta muuttaa koko testipenkki rakennetta. Tämä toiminto on erittäin hyödyllinen muun muassa sekvenssin toiminnallisuuden muuttamisessa tai sen korvaamisessa toisella. Korvattavien komponenttien on oltava yhteensopivia TLM protokollan mukaisesti. [8, s. 8]

UVM-tehdas hyödyntää SystemVerilogin *create* funktiota ja makro ominaisuutta [10]. Tämän takia tehtaan koodaamisessa on noudatettava tiettyjä sääntöjä [8, s. 8].

4.6 UVM-simulointivaiheet

Jotta UVM-testipenkin simuloinnin suorittaminen olisi johdonmukaista, UVM käyttää simulointivaiheita [8, s. 11]. Seuraavaksi esitellään lyhyesti simulointivaiheet suoritusjärjestyksessä.

4.6.1 Rakennusvaihe

Rakennusvaiheessa testipenkin rakenne, hierarkia ja konfiguraatiot muodostetaan. Hierarkiajärjestyksessä ylhäältä alaspäin luokat keräävät konfiguraatiotietonsa ja kaikki komponentit tulevat alustetuiksi. Rakennusvaiheessa hyödynnetään UVM-tehdasta ja konfiguraatiotietoja. [8], [22]

4.6.2 Kytkentävaihe

Kytkentävaiheessa luodaan hierarkiatasossa alhaalta ylöspäin TLM-yhteydet komponenttien välille ja luodaan kahvat testipenkin resursseja varten. [8, s. 12]

4.6.3 Suoritusvaihe

Suoritusvaiheessa simulointi käynnistetään ja komponentit alkavat suorittaa tehtäviään. Sekvenssit muodostavat transaktioita, jotka ohjaimen kautta kulkevat herätteinä DUT:lle. Muut simulointivaiheet lisättiin UVM:ään, jotta suoritusvaiheessa komponenttien toiminnasta saataisiin selkeämpää. [8, s. 12]

4.6.4 Raportointivaihe

Raportointivaiheessa simuloinnin tulokset raportoidaan tyypillisesti erilliseen tiedostoon. Silloin selviää, menikö testi läpi vai ei. Raportointivaihe päättää testin. Raportoinnin suorittavat analyysikomponentit. [8, s. 14], [22]

4.7 UVM IP-lohkojen verifioinnissa

UVM: standardointi on parantanut yhteen toimivuutta suunnittelun ja verifioinnin välillä. Se on myös lisännyt IP-lohkojen uudelleen käyttöä SoC suunnitelmissa sekä UVM testipenkkien uudelleen käyttöä. UVM-agentin ominaisuudet sopivat hyvin IP-lohkotason verifioimiseen. [5, s. 7], [10]

5 POHDINTA

Tämän työn tavoitteena oli tutustua UVM:ään ja sen käyttöön järjestelmäpiirien verifioinnissa IP-lohkotasolla. Tavoitteessa onnistuttiin ja UVM tuli tutuksi. Työssä käytiin ensin läpi tarvittavat taustatiedot, jotka olisi hyvä tietää ennen UVM:n opettelua. Tämän jälkeen UVM:n periaatteet, toiminnot ja rakenne esiteltiin.

Pyrin käymään työssäni läpi tarvittavat asiat, jotta vaikeasti ymmärrettävä UVM avautuisi perusperiaatteiltaan ja tarvittaessa sen kanssa olisi helpompi aloittaa työskentely. Teoriaosuudet yritin pitää suppeana ja ymmärrettävänä, mutta varmasti osan asioista olisi voinut selittää laajemmin tai paremmin. Uskon, että tämän kandidaatintyön kautta lukija ymmärtää perusperiaatteet UVM:n käytöstä IP-lohkotason verifioinnissa.

Elektroniikka ja tietoliikennetekniikan tutkinto-ohjelman opiskelijana oli hyödyllistä opetella itse UVM:n perusteet, koska niitä ei vielä kandidaatin tutkinto-ohjelmassa opeteta. Monilla toisen tai kolmannen vuoden opiskelijoilla on kuitenkin sama tilanne, että he lähtevät töihin digitaalisuunnittelun ja verifioinnin puolelle eikä heillä kokemusta UVM:stä. Ehkä jokin kevytmuotoinen UVM:n perusteet (verkko)kurssi voisi olla paikallaan, vaikka valinnaisena. En kuitenkaan koe, että digitaalitekniikka 2 kurssiin kannattaisi nykyisen työmäärän lisäksi opettaa edes UVM:n perusteita.

Koen, että tämän kandidaatintyön tekeminen on antanut minulle paljon eväitä tulevaan työhöni digitaalisuunnittelun ja verifioinnin puolella. Sehän oli tämän kandidaatintyön idean alku. Odotan mielenkiinnolla, muuttuuko käsitykseni tai ajatukseni UVM:stä työelämään mennessä vai koenko, että olen opetellut oikeita asioita ja käsittänyt niiden merkityksen oikein.

UVM on niin suuri ja monimutkainen ymmärrettäväksi, että siinä riittäisi aihetta jatkotutkimukselle monelta kannalta. Hyviä aineksia jatkotutkimukselle olisi teorian syvemmässä ymmärtämisessä ja testipenkin koodaamisessa suunnitellulle lohkolle.

6 YHTEENVETO

Työssä käydään läpi UVM:n perusteet sekä sen käyttö SoC:ien verifioinnissa IP-lohkotasolla. Työ on kirjallisuuskatsaus aiheeseen ja siinä esitellään myös muut olennaiset asiat UVM:n perusteiden ymmärtämiseksi.

UVM kehitettiin nopeuttamaan ja yhtenäistämään verifiointia. Se mahdollistaa testipenkkien uudelleen käytettävyyden pienillä koodimuutoksilla, jolloin verifioijien työ helpottuu ja nopeutuu sekä tuotteet saadaan nopeammin markkinoille. UVM standardi määrittelee säännöt ja ohjenuorat verifioinnille. UVM on erityisen käytännöllinen SystemVerilog kielellä verifioitaessa IP-lohkotasolla.

RTL-tason verifioinnissa tavoitteena on tarkastaa, että laitteistonkuvaukset ovat virheettöminä. Verifiointi on erittäin aikaa vievää ja siksi aikojen saatossa on kehitetty menetelmiä, jolla sitä voidaan nopeuttaa ja yksinkertaistaa. Verifioinnista on tullut entistä työläämpää, kun järjestelmäpiireihin integroidaan jo satoja IP-lohkoja.

SystemVerilog on erityisesti verifiointiin hyödyllinen kieli UVM:n ansiosta. Se on kehittynyt hyödylliseksi kieleksi myös siitä syystä, että sillä voidaan sekä suunnitella että verifioida piirejä. Tämä on helpottanut verifioijien ja suunnittelijoiden välistä kommunikaatiota. SystemVerilogin oli-ohjelmointi ominaisuus on erittäin käytännöllinen UVM:ää käytettäessä.

UVM:n standardoima tapa hyödyntää SystemVerilogia ja TLM:ää sekä sen omat toiminnallisuudet RTL verifioinnissa ovat nopeuttaneet verifiointia hurjasti. UVM:n sisäänrakennetut komponentit ja toiminnot mahdollistavat testipenkkien uudelleen käyttämisen pienemmällä vaivalla. Agentin ymmärtäminen on tärkeä osa UVM:ää, koska sen toiminnallisuudet ohjaavat kommunikointia DUT:n kanssa sekä se on hyödyllinen IP-lohkojen verifioinnissa.

Työ oli opettavainen ja koen sen olevan minulle hyödyksi aloittaessani oman alan työt. UVM on aiheena monimutkainen ja tämä työ tarjoaa katsauksen sen perusteisiin.

7 LÄHDELUETTELO

[1] K. Salah, ”A UVM-based smart functional verification platform: Concepts, pros, cons, and opportunities”, *Proceedings of 2014 9th International Design and Test Symposium, IDT 2014*, ss. 94–99, helmi 2015, doi: 10.1109/IDT.2014.7038594.

[2] Z. Bel Hadj Amor, L. Pierre, ja D. Borrione, ”System-on-chip verification: TLM-to-RTL assertions transformation”, ss. 1–4, elo 2014, doi: 10.1109/PRIME.2014.6872713.

[3] A. Hussien *ym.*, ”Development of a generic and a reconfigurable UVM-Based verification environment for SoC buses”, *Proceedings of the International Conference on Microelectronics, ICM*, vsk. 2019-December, ss. 195–198, joulu 2019, doi: 10.1109/ICM48031.2019.9021657.

[4] N. Bombieri, F. Fummi, ja G. Pravadelli, ”On the evaluation of transactor-based verification for reusing TLM assertions and testbenches at RTL”, *Proceedings -Design, Automation and Test in Europe, DATE*, vsk. 1, 2006, doi: 10.1109/DATE.2006.243898.

[5] V. S. Chakravarthi, ”A Practical Approach to VLSI System on Chip (SoC) Design”, *A Practical Approach to VLSI System on Chip (SoC) Design*, 2022, doi: 10.1007/978-3-031-18363-8.

[6] ”File:ARMSoCBlockDiagram.svg - Wikimedia Commons”. https://commons.wikimedia.org/wiki/File:ARMSoCBlockDiagram.svg (viitattu 10. helmikuuta 2023).

[7] Mirko Lindroth, ”UVM-testipenkin toteuttaminen IP-lohkotasolla”, Tampereen ammattikorkeakoulu, 2021. Viitattu: 7. helmikuuta 2023. [Verkossa]. Saatavissa: https://www.theseus.fi/bitstream/handle/10024/495864/Lindroth_Mirko.pdf?sequence=3&isAllowed=y

[8] F. Verification Methodology Team - Siemens EDA, ”Universal Verification Methodology (UVM) Cookbook | Verification Academy”, 2022. Viitattu: 6. helmikuuta 2023. [Verkossa]. Saatavissa: https://verificationacademy.com/cookbook/uvm

[9] *1800-2017 - IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language*. IEEE, 2018. Viitattu: 7. helmikuuta 2023. [Verkossa]. Saatavissa: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8299595

[10] C. Spear ja G. J. Tumbush, *SystemVerilog for Verification : a guide to learning the testbench language features*, 3rd ed. New York, NY: Springer, 2012.

[11] J. Chatterjee, A. Saxena, A. Mehra, G. Vyas, ja V. Mukesh, ”Verification and Debugging of LC-3 Test Bench Environment using System Verilog”, *Proceedings of the 2nd International Conference on Electronics, Communication and Aerospace Technology, ICECA 2018*, ss. 1253–1258, syys 2018, doi: 10.1109/ICECA.2018.8474724.

[12] ”IEEE Standard for Universal Verification Methodology Language Reference Manual”, *IEEE Std 1800.2-2020 (Revision of IEEE Std 1800.2-2017)*, 2020, Viitattu: 6. helmikuuta 2023. [Verkossa]. Saatavissa: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9195920

[13] ”SystemVerilog Classes”. https://www.chipverify.com/systemverilog/systemverilog-class (viitattu 8. helmikuuta 2023).

[14] J. Pohjolainen, ”Johdatus olio-ohjelmointiin”, kesäkuuta 2008. https://www.slideshare.net/pohjus/johdatus-olioohjelmointiin-presentation (viitattu 9. helmikuuta 2023).

[15] Matti. Rintala, Jyke. Jokinen, ja Gummerus), *Olioiden ohjelmointi C++:lla*, 4. uud. p. Talentum, 2005.
[16] ”UVM Tutorial for Beginners”. https://www.chipverify.com/uvm/uvm-tutorial (viitattu 9. helmikuuta 2023).
[17] ”UVM Agent - Verification Guide”. https://verificationguide.com/uvm/uvm-agent/ (viitattu 10. helmikuuta 2023).
[18] ”UVM TLM Port to Export to Imp”. https://www.chipverify.com/uvm/uvm-tlm-port-export-imp (viitattu 10. helmikuuta 2023).
[19] ”UVM Agent | uvm_agent”. https://www.chipverify.com/uvm/uvm-agent (viitattu 10. helmikuuta 2023).
[20] ”Subscriber [uvm_subscriber]”. https://www.chipverify.com/uvm/uvm-subscriber (viitattu 10. helmikuuta 2023).
[21] ”UVM Scoreboard”. https://www.chipverify.com/uvm/uvm-scoreboard (viitattu 10. helmikuuta 2023).
[22] ”UVM Phases”. https://www.chipverify.com/uvm/uvm-phases (viitattu 10. helmikuuta 2023).