VTT JULKAISUJA 823

Ylläpitotekniikoiden soveltaminen VHDL-kieleen perustuvassa suunnitteluprosessissa

Marko Palola

VTT Elektroniikka

TECHNICAL RESEARCH CENTRE OF FINLAND
ESPOO 1997

ISBN 951–38–4533–8 (nid.)
ISSN 1235–0613 (nid.)

ISBN 951–38–4534–6 (URL: http://www.inf.vtt.fi/pdf/)
ISSN 1455–0857 (URL: http://www.inf.vtt.fi/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

Valtion teknillinen tutkimuskeskus (VTT), Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, telekopio (09) 456 4374

Statens tekniska forskningscentral (VTT), Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, telefax (09) 456 4374

Technical Research Centre of Finland (VTT), Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
phone internat. + 358 0 4561, telefax + 358 0 456 4374

VTT Elektroniikka, Elektroniikan piirit ja järjestelmät, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde (08) 551 2111, faksi (08) 551 2320

VTT Elektronik, Elektroniska kretsar och system, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel (08) 551 2111, fax (08) 551 2320

VTT Electronics, Electronic Circuits and Systems,
Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland
phone internat. + 358 8 551 2111, fax + 358 8 551 2320

Toimitus Kerttu Tirronen

LIBELLA PAINOPALVELU OY, ESPOO 1997

Palola, Marko. Ylläpitotekniikoiden soveltaminen VHDL-kieleen perustuvassa suunnitteluprosessissa. [Adapting of maintenance techniques in VHDL based design process]. Espoo 1997, Valtion teknillinen tutkimuskeskus, VTT Julkaisuja – Publikationer 823. 74 s. + liitt. 18 s.

UDK 621.38:681.3.06:681.327.8
Avainsanat ASIC, design environment, programming languages, computer interfaces, workflow

TIIVISTELMÄ

Työssä suunniteltiin ja kehitettiin ASIC-piirien suunnitteluprosessi ja -ympäristö. Ympäristöstä suoritetaan suunnitteluprosessia, joka perustuu syntetisoitavan piirin toiminnan mallinnukseen SA/VHDL-menetelmällä. Työtehtäviinjaon jälkeen ympäristöön asennettiin olemassa olevia työkaluja.

Suunnitteluympäristön mallinnuksessa työprosessi kuvataan graafiseksi työvuoksi. Työvuo sisältää suunnittelun työvaiheet, työvaiheiden ohjeet ja työkalujen käynnistyskomennot. Työvuot ovat suoritettavia kuvauksia, joita käytettäessä prosessia voidaan mitata, parantaa, kontrolloida ja nopeuttaa.

Suunniteltavan prosessimallin vaatimuksiin kuuluivat ylläpito- ja uudelleenkäyttötyövaiheiden suunnitteleminen ja mallintaminen. Ylläpitomenetelmiä työssä olivat suunnitelmatiedostojen hallinta suunnittelussa ja uudelleenkäyttötilanteissa. Uudelleenkäytössä sovellettiin komponentointia ja dokumentointia tukevia menetelmiä.

Työn tuloksena saatiin graafinen suunnitteluympäristö, jonka avulla suunnittelu voidaan tehdä helpommin ja nopeammin. Ylläpitomenetelmien avulla parannettiin spesifikaatiomallien uudelleenkäytettävyyttä ja VHDL-kielisen suunnitelman hallintaa.

Palola, Marko. Ylläpitotekniikoiden soveltaminen VHDL-kieleen perustuvassa suunnitteluprosessissa [Adapting of maintenance techniques in VHDL based design process]. Espoo 1997, Technical Research Centre of Finland, VTT Julkaisuja – Publikationer 823. 74 p. + app. 18 p.

UDC 621.38:681.3.06:681.327.8
Keywords ASIC, design environment, programming languages, computer interfaces, workflow

ABSTRACT

In the work a design environment for ASIC design process has been designed and developed. The user of the design environment can carry out work tasks of a design process by advancing through a graphical interface. The work process is based on SA/VHDL specification, simulations and VHDL synthesis. Existing tools has been implemented into the environment after the design steps were found.

A workflow management is a modelling method of work processes. The graphical workflow contains work tasks and relationships between these tasks. Each task contains work instructions and commands for starting the tools. The workflow management offers methods for measuring, improving and controlling of the work process.

The modelled environment also includes the designing and the modelling of maintenance and reuse methods into the workflow. The focus has been to support the maintenance of design files in designing and in reusing work steps. Existing methods of componentation and documentation techniques have been used to implement the reuse tasks.

The result of the work is a design environment, which allows users to complete their designs faster and easier. Maintenance techniques improve the control and the reuse of the design objects.

ALKUSANAT

Tämä työ on tehty Elektroniikan suunnitteluautomaatioryhmässä VTT Elektroniikassa Oulussa.

Työn ohjaajina toimivat VTT Elektroniikassa Kari Tiensyrjä ja Juha-Pekka Soininen. Työn valvojana toimi apulaisprofessori Juha Röning Oulun Yliopistosta.

Esitän kiitokset työn ohjaajille ja valvojille, Hannu Heusalalle toisena tarkastajana toimimisesta muiden kiireiden ohella, Timo Juntuselle SA-kirjastointiohjelman opastuksesta, Kari Laitiselle luonnollisen nimeämisen kommentoinnista, Tuomo Huttuselle työkoneen lainaamisesta ja ryhmän muille diplomityöntekijöille Tero Ala-Poikelalle ja Ville Veijalaiselle yhteistyöstä.

Lisäksi kiitokset Matti Sipolalle Oulun yliopistoon DMMTABLE-ohjelman prototyypin käytöstä, Kari Koskiselle Intergraph:lle DMM kysymyksiin vastaamisesta, Olli Tyrkölle Synopsykselle synteesiopastuksesta sekä muille työni valmistumiseen vaikuttaneille.

Oulussa toukokuussa 1997

Marko Palola

SISÄLTÖ

LIITTEET

LYHENTEET

ASIC	Application Specific Integrated Circuit, asiakaskohtainen piiri
BC	Synopsys Behavioral Compiler, syntetisointityökalu IC-piirien suunnitteluun
CAN	Controller Area Network, väyläkontrollistandardi
db	BC:n synteesitietokanta
dc_shell	BC:n eräs käyttöliittymä
DLL	Dynamic Link Library
DMM	Design Methodology Management, työvuonhallintatyökalu
EDA	Electronic Design Automation, Elektroniikan suunnitteluautomaatio
Exceed	X-ikkunoinnin mahdollistava ohjelma
ESA	European Space Agency, Euroopan avaruushallinto
FSM	Finite State Machine, syntetisoidun suunnitelman tilakone
NT	New Technology, käyttöjärjestelmä
OSI	Open Systems Interconnection, standardi
PRDB	Product registration database, DMM-työkalun prosessitietokanta
Prosa	Insoft Oy:n työkaluohjelma
PVCS	Intersolv Inc:n versionhallintatyökalu
RTL	Register Transfer Level, eräs VHDL-kielen muoto
SA	Structured Analysis, rakenteinen analyysi
SCM	Software configuration management, ohjelmistotuotteen hallinta
UNIX	Käyttöjärjestelmä
vhdl	Synteesityökalusta saatava HDL-kielinen kuvaus
VHDL	Very High Speed Integrated Circuit Hardware Description Language, piirikuvauskielistandardi
VHDL2000	Simulointityökalu
vhdlan	VHDL-analysaattori
vhdldbx	Synopsys-simulointityökalu
VSS	Synopsys-simulointityökalu
X	X-ikkunointi, hajautettu, laite- ja käyttöjärjestelmäriippumaton ikkunointijärjestelmä

1 JOHDANTO

Suunnittelu on iteratiivista työskentelyä, jolle on ominaista työskentelytapojen toistuvuus, erilaisten työkalujen käyttäminen ja suunnittelun oikeellisuuden tarkistaminen suunnittelun edetessä. Nämä ominaisuudet ovat hyvin esillä VHDL-kielisessä suunnittelussa. Ajallisesti piirisuunnittelun kesto voi olla kuukausien työmäärän kokoinen, riippuen esimerkiksi suunniteltavasta piiristä ja työmenetelmistä. Piirisuunnittelussa käytettävät menetelmät ovat usein kompleksisia, jolloin suunnittelija tarvitsee tietoa menetelmien ja työkalujen käyttämisestä pystyäkseen tehokkaaseen työhön.

Prosessinhallinta on yksi EDA-alueen tutkimuskohteista, jossa suunnitteluprosessien hallitsemiseksi on kehitetty sopivia työkaluja. Tämä on tullut ajankohtaiseksi monestakin syystä. Vaativaa laitesuunnittelua pyritään automatisoimaan mahdollisimman paljon suunnitteluaikojen ja kulujen pienentämiseksi. Suunniteltavien laitteiden vaatimukset kasvavat, piirien pinta-ala kasvaa ja nykyaikaisilla suunnittelutyökaluilla on mahdollista tuottaa ja hallita yhä suurempia suunnittelukokonaisuuksia. Myös suunnittelijan käyttämä aika suunnitteluun verrattuna muuhun suunnittelun mahdollistamiseksi tehtävään päätetyöskentelyyn on yksi ajan säästökohteista.

VTT:n EDA-ryhmän kiinnostus piirisyntetisointiprosessin mallintamiseen johtuu seuraavista seikoista: Syntetisointiprosessi on uusi, vakinaista paikkaa syntetisoijalle ei ole kaavailtu ja korkean tason synteesin sopivuutta käytössä olevaan SA/VHDL-spesifiointiprosessiin halutaan kokeilla. Syntetisointiprosessi itsessään on monimutkainen, sen käyttöönotto ja käyttäminen ovat vaativia. Prosessin helpottamiseksi käytetään prosessin mallinnustyökalun tarjoamia mahdollisuuksia, jolloin ehkä suunnittelun aloittamista ja suorittamista voidaan nopeuttaa.

1.1 TYÖPROSESSIMALLIN MERKITYS

Työprosessimallilla eli työvuomallilla tarkoitetaan graafista kuvausta, johon on yhdistetty työvaiheet ja niiden suoritusjärjestys, työn seurantaominaisuuksia sekä työssä käytettäviä työkaluja. Tällaisten mallien tekemiseksi on kehitetty useita ohjelmistoja, joista yksi on tässä työssä käytettävä DMM-työkalu. Tällaisen työkalun avulla työprosessin suorittaja voi ajaa työprosessimalleja, jolloin työskentelystä tulee kontrolloitua. Työvuomallit voidaankin käsittää eräänlaisina käyttäjän liityntämekanismeina suunnitteluprosesseihin.

Työvuossa olevien työvaiheiden välille voidaan määrätä erilaisia riippuvuuksia. Työvaiheen käynnistämiselle voidaan asettaa aikaraja tai estää työvaiheeseen siirtyminen ennen jonkun toisen työvaiheen valmistumista. Työvaiheissa tehtävä työ voidaan tarkistaa ohjelmallisesti, jolloin voidaan estää työvaiheen päättyminen ennen varsinaisen työosan tekemistä. Työvuota

käytettäessä työvaiheiden suorituskerroista ja suoritusajoista voidaan tuottaa raportit.

Työvuot voivat olla usealle käyttäjälle yhteisiä, jolloin työn edistymisen seuranta voi haluttaessa tapahtua reaaliaikaisesti, sillä työvuossa tapahtuvat muutokset näkyvät heti kaikille osapuolille. Työvoissa voidaan kuvata myös erilaisia asioita, kuten työvaiheita tai dokumenttien liikkumista organisaation eri jaostoissa.

1.2 TYÖN LÄHTÖKOHDAT JA TAVOITTEET

Tässä työssä suunnitellaan ja mallinnetaan piirisuunnitteluprosessi. Työprosessiin liitetään ylläpitomenetelmiä SA-spesifikaatioiden uudelleenkäyttämisen ja suunnitelmatiedostojen hallinnan tueksi. Lisäksi tutkitaan luonnollisen nimeämisen käyttämistä synteesisuunnittelun ja uudelleenkäytön tukena. Työvuomalliin tallennetaan tarvittavat työvaiheet, niiden järjestys ja eri työvaiheisiin liittyvien työkalujen ajamisessa tarvittava tieto.

Mallinnettava työprosessi on tarkoitettu ASIC-piirien suunnittelemiseksi, jossa hyödynnetään korkean tason suunnittelumenetelmää vaatimusten määrittämisessä VHDL-kielen ja graafisen kuvauksen avulla. Prosessin työtehtävät valitaan esimerkkinä olevan synteesisuunnittelun kautta.

Korkeammalla tasolla suoritettava toiminnallinen mallinnus tehdään SA/VHDL-menetelmällä. Menetelmän avulla pyritään varmentamaan valmistettavan piirin toiminta ja suunnitteluratkaisujen toimivuus ennen piirin suunnittelemista tarkemmalla VHDL-kielen tasolla. Työprosessimalliin lisätään uudelleenkäyttötyövaiheita, joilla pyritään nopeuttamaan määrittelyvaiheen suunnittelua.

SA/VHDL-spesifikaation avulla suunnitellaan syntetisoituva käyttäytymistason VHDL-kuvaus, josta voidaan aloittaa piirin syntetisointi työkalulla. SA/VHDL-kuvauksen osia, herätteitä ja hierarkiajakoa käytetään apuna syntetisoituvaa mallia suunnitellessa ja toteutettaessa. Syntetisoituva kuvaus varmennetaan simuloimalla samaan tapaan kuin sen spesifikaatiokin.

Syntetisointityökalun avulla käyttäytymistason kuvaus käännetään rekisterieli RTL-tasolle käyttäytymistason synteesissä, josta logiikkasynteesin kautta syntesisoidaan piirikuvaus ASIC-valmistajan tarvitsemalle porttitasolle. Tässä tapauksessa syntetisoituva kuvaus koostuu useista erillisistä pienemmistä prosesseista SA-spesifikaation mukaan. Prosessit viedään synteesin läpi yksitellen ja synteesityökalun tuottama kuvaus yhdistetään logiikkasynteesin jälkeen.

Syntetisointi suoritetaan antamalla komentoja synteesityökalulle komentotiedoston (script) kautta. Työvuossa käytettävät komentotiedostot muoka-

taan sellaisiksi, että niiden käyttö työvuosta käsin on mahdollisimman helppoa.

Ylläpidon, uudelleenkäytön ja luonnollisen nimeämisen ajatellaan tässä työssä kuuluvan toisiinsa siten, että ylläpidolla ymmärretään SA/VHDL- ja syntetisoituvien VHDL-mallien hallittua muuttamista ylläpitomenetelmien avulla, jolloin mallien uudelleenkäyttäminen ajatellaan tässä työssä eräänlaiseksi ylläpidoksi. Luonnollisista nimeämisestä etsitään apua uudelleenkäytettävyyden parantamiseksi.

Varsinaisten ylläpitomenetelmien apuvälineitä etsitään ohjelmistotekniikan puolelle kehitetyistä apuvälineistä, jotka soveltuvat riittävän hyvin VHDL-kielisen suunnitelman hallintaan.

Luonnollisten nimien käytöllä voidaan saavuttaa etua SA/VHDL-spesifikaatiossa ja synteesisuunnittelussa. Lisäksi SA-kuvien luettavuuteen voidaan vaikuttaa erilaisilla piirtämistyyleillä ja tavoilla. Luettavuuden paraneminen ja luonnollisten nimien käyttäminen signaalien, tilojen ja tietomuunnosten niminä voi nopeuttaa suunnitelman ymmärtämistä, mikä on eräs perusedellytys suunnitelman uudelleenkäytettävyydelle. Uudelleenkäytön tunnettuja ongelmia pyritään välttämään etsimällä menetelmiä uudelleenkäytettävyyden parantamiseksi muualta ja käyttämällä jo olemassa olevia uudelleenkäyttömenetelmiä.

Tässä työssä uudelleenkäytöllä on pienempi merkitys kuin ylläpitomenetelmillä ja työvuomallin rakentamisella. Uudelleenkäyttö mahdollistetaan työvuossa valmiin ohjelman avulla. Sen sijaan SA/VHDL-mallien uudelleenkäytettävyyden parantamiseksi etsitään joitakin ratkaisuja. Valmistettavan työvuomallin tärkeimpiin ominaisuuksiin kuuluvat myös helppo ja sujuva käytettävyys.

2 SUUNNITTELUPROSESSIN MALLINNUS- JA TYÖMENETELMÄT

Seuraavissa kappaleissa esitellään työprosessimallinnuksen työmenetelmät ja työssä mallinnettavassa suunnitteluprosessissa käytettäviä työskentelymenetelmiä. Lisäksi selvitetään taustaa SA-mallin tasolla tapahtuvalle uudelleenkäytölle, ohjelmistotekniikassa käytettäville ylläpitomenetelmille ja luonnolliselle nimeämiselle.

2.1 PROSESSIN MALLINNUSMENETELMÄT

Suunnitteluprosessin mallintamisella pyritään kuvaamaan suunnittelussa käytettävä työprosessi. Saman työprosessimallin avulla työ voidaan suorittaa tai työmenetelmät voidaan esimerkiksi esitellä muille. Työprosessista tehdyn mallin avulla työskenneltäessä voidaan havaita epäkohtia työssä ja työprosessia voidaan kehittää.

DMM (Design Methodology Management) on ohjelma, jolla työprosessi voidaan mallintaa graafiseksi kuvaukseksi. Prosessi kuvataan siten, että työvuon käyttäjällä on näkyvissään työskentelyssä käytettävät työvaiheet ja niiden suoritusjärjestys. Työvaiheisiin voidaan liittää työkaluohjelman suorituskomentoja erilaisilla menetelmillä ja työohjeita työvaiheen suorittamiseksi. Työvaihetta työvuokaaviossa voidaan kutsua prosessiksi.

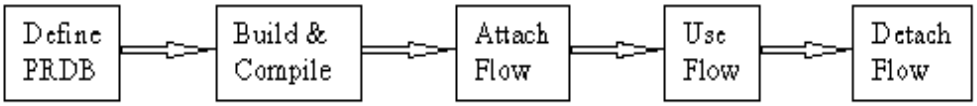
DMM-työvuosta on runsaasti apua työn edistymisen seurannassa, työtehtävien nopeutumisessa, työvaiheiden kontrolloimisessa ja työprosessin analysoinnissa. Työn edistymisen seurannan mahdollistaa työvuomallien havainnollinen ulkoasu, josta tilan hahmottaminen on helppoa. Työtehtävät nopeutuvat, koska työvaiheisiin voidaan liittää ohje ja oikeat työkalut käynnistyvät oikeaan aikaan.

Työvuomallissa työskentely on kontrolloitua: mallin käyttäjä ei voi vahingossa jättää tärkeitä työvaiheita tekemättä, jos niiden suoritus on ehtona työvuossa etenemiselle. Työprosessin eri vaiheissa käytettyä aikaa voidaan analysoida ja löytää mallista aikaavievät kohdat, joissa esimerkiksi tarvitaan enemmän ohjeita tai joissa työvaiheet ovat liian suuria. Ohjelmasta saadaan kattavat raportit esimerkiksi työvaiheeseen kulutetusta ajasta ja suorituskerroista.

Suunnitteluprosessin mallia rakennettaessa on syytä ottaa huomioon, että malli tehdään juuri tietynlaiselle työasema- ja ohjelmistokonfiguraatiolle, koska DMM toimii asiakas-palvelinmenetelmällä, jossa työvuota suoritettaessa vuossa määrätyt ohjelmat käynnistetään asiakaskoneessa ja työvuota käytetään palvelinkoneesta käsin. Työvuot valmistetaan ja ne säilytetään palvelimessa. Yhtä työvuota voi käyttää useampi henkilö ja yksittäisille työvaiheille voidaan asettaa käyttöoikeuksia. [1,2].

2.1.1 Työvuon mallinnusmenetelmät

DMM:llä työvuota tehtäessä käytetään erillisiä työvaiheita (kuva 1). Jokaisessa työvaiheessa käytetään eri työkalua. Asiakaskoneessa valmiiden työvoiden ajamiseksi tarvitaan liittämistä, käyttämistä ja poistamista tukevat työkalut. Palvelinkoneessa käytetään kaikkia työkaluja.



Kuva 1. Työvuota mallinnettaessa tarvittavat työvaiheet.

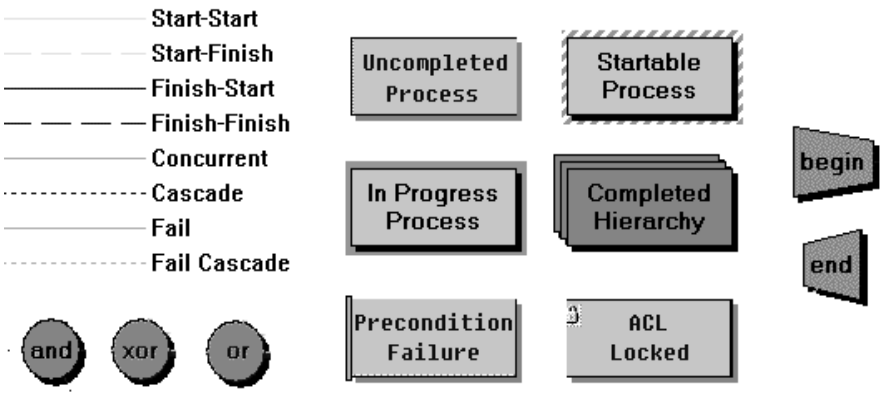
DMM-työkaluilla työvuon mallintaminen ja käyttäminen tapahtuvat seuraavien pääaskelien mukaisesti:

- Määritellään tarvittavat prosessit ja niiden parametrit PRDB-työkalulla tietokantaan.
- Työvuo rakennetaan tietokantaan määritellyistä prosesseista ja ne yhdistetään erilaisilla poluilla DMM Builder-työkalulla.
- Työvuo käännetään käyttöä varten.
- Vuo nimetään ja liitetään työhakemistoon DMM Attacher-työkalulla.
- Työvuota käytetään ja testataan DMM Displayer-työkalulla.
- Työvuo poistetaan käytöstä DMM Detacher-työkalulla.

Vuota käytettäessä työskentelyssä syntyvät tiedostot sijaitsevat vuohon liitetyssä työhakemistossa. PRDB (Product registration database) on tietokanta, jonne määritellään työvuossa tarvittavat työprosessit samannimisellä työkalulla. [1].

2.1.2 Työvuon mallinnusobjektit

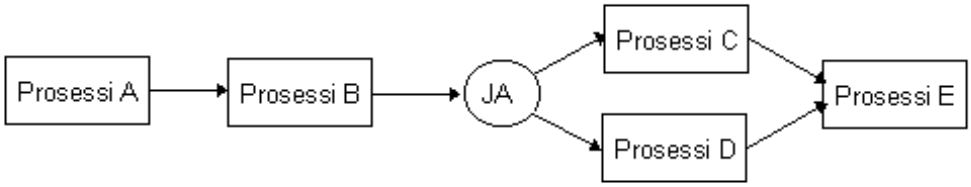
Työvuossa olevat prosessit kuvataan laatikoilla ja niiden väliset polut piirretään erilaisilla nuolilla. Kuva 2 sisältää DMM:n avulla tehtävässä mallinnuksessa käytettäviä komponentteja. Erilaisten yhteystyyppien ja prosessien tarjoamien mahdollisuuksien avulla työvuon mallinnuksessa on runsaasti vaihtoehtoja.



Kuva 2. Työvuon perusosat.

Hierarkkisten prosessien avulla malli voidaan koota erillisistä työvoista. Työvuon haaroittamisessa käytetään seuraavia operaattoreita: JA, POIS SULKEVA TAI ja TAI. Terminaaleilla alku ja loppu kuvataan työvuon alku- ja loppupisteitä. Prosessien väliset kytkennät määritellään kahdeksalla erilaisella yhteystyypillä.

Työvuota käytettäessä prosesseilla on 4 päätilaa. Käynnistettävä-tilassa oleva prosessi voidaan käynnistää suorituksessa-tilaan, jolloin prosessiin määritelty työkalu käynnistyy. Työkalun sulkeuduttua ja prosessin onnistuttua prosessi siirtyy suoritettu-tilaan. Prosessin alkutila on aina suorittamaton. Muita tiloja ilmoittavat kuvan 2 alimman rivin prosessit. Ehtorivin virheestä ilmoittaa prosessi, jonka vieressä on punainen pystypalkki. Käyttölukitulla (Access Control List) prosessilla kuvataan prosessia, jonka käyttöoikeus on toisella käyttäjällä. [1,2].



Kuva 3. Prosessien väliset kytkennät.

Kuva 3 havainnollistaa operaattorin merkitystä työvuon haaroittamisessa. JA-operaatiolla haaroitettu työpolku voi siirtyä prosessiin E, kun molemmat prosessit C ja D on suoritettu onnistuneesti. Tällöin prosessi E siirtyy käynnistettävään tilaan.

Haaroitettujen polkujen tulee onnistua, vaikka toisen prosessin yhteystyyppi antaisikin prosessille E luvan käynnistyä ennen haaran toisen prosessin onnistumista. TAI-operaatiossa käyttäjä voi valita 1 tai useamman haaroista. POIS SULKEVASSA TAI operaatiossa käyttäjä voi valita vain yhden haa-

roista. Haarojen ei ole pakko johtaa minnekään, vaan ne voivat myös päättyä.

Työvuossa operaattoriin saa mennä ainoastaan yksi polku, mutta operaattorista voi lähteä tarpeellinen määrä polkuja. Lähtevien polkujen tyyppi on sama kuin operaattoriin tulevan polun.

2.1.3 Prosessien väliset yhteystyypit ja niiden merkitys

Työvuon käyttäytyminen määräytyy prosessien välisten yhteystyyppien ja prosessien tilojen yhteisvaikutuksesta. Yhteystyypin avulla on esimerkiksi mahdollista estää seuraavan prosessin käynnistyminen, mikäli edellinen prosessi on vielä suorittamatta. Kaksi peräkkäistä prosessia, esimerkiksi kuvan 3 prosessit A ja B, voidaan yhdistää taulukon 1 yhteystyypeillä [1,2].

Taulukko 1. Prosessien erilaiset liityntätavat työvuossa

Yhteystyyppi	Selite
Start-Start	B ei voi käynnistyä ennen kuin A on käynnistynyt. A voidaan jättää käyntiin samalla, kun työprosessi etenee B suorituksen loputtua.
Start-Finish	B ei voi lopettaa ennen kuin A on käynnistynyt, mutta B voi käynnistyä ennen A:ta
Finish-Start	Prosessi B voi käynnistyä vasta, kun A on suoritettu kokonaan. Tiukasti peräkkäinen suoritus.
Finish-Finish	B ei voi lopettaa ennenkuin A on lopettanut onnistuneesti. Aloitusjärjestystä ei ole rajattu.
Concurrent	B ei voi aloittaa ennenkuin A on aloittanut ja B ei voi lopettaa ennenkuin A on lopettanut onnistuneesti.
Cascade	Kun A prosessi on onnistuneesti lopettanut, automaattinen B prosessi käynnistetään
Fail	Ainoa vaihtoehto takaisinkytkennälle, kaikki väliin jäävät prosessit palautetaan alkutiloihin.
Fail Cascade	Sama kuin Fail-yhteys, mutta automaattisille prosesseille.

Liityntätapojen runsaus mahdollistaa hyvinkin erilaisten työvoiden rakentamisen samalle työprosessille. Työvuon käytettävyys tai toiminta voi olla ratkaiseva kriteeri voita piirrettäessä.

2.1.4 Prosesseihin määriteltävät parametrit

Jokaiselle kaavion prosessille määritetään nimi, lyhyt kuvaus, lopetusviestit, lopetusehdot, käyttöoikeuksia ja työkalun suorituskomento. Lisäksi komennolle voidaan määrittää 2 suoritusehtoa ennen ja jälkeen työkalun suorituksen ja työkalun suoritustapa.

Prosessin nimi toimii tunnisteena palvelimelle ja saa esiintyä vain kerran työvuokaaviossa. Prosessin kuvaukseen voidaan esimerkiksi kirjoittaa ohjeita työvaiheen suorittamiseksi tai kuvaus prosessissa tehtävästä työstä.

Prosessin tekstimuotoiset lopetusviestit kytketään prosessista lähteviin polkuihin. Lopetusviestit voidaan liittää suoraan työkaluohjelman palauttamaan arvoon tai vaihtoehtoisesti käyttäjältä kysytään, mikä viesti valitaan, kun prosessiin määritelty työkaluohjelma on sulkeutunut.

Prosessin työkalukomennon suoritus- tai lopetusehdoksi voidaan tehdä ohjelmia, jotka testaavat työvaiheessa tarvittavaa asiaa, esimerkiksi päiväystä tai tietyn tiedoston löytymistä. Komentorivillä ja ehtoriveillä suoritettavien ohjelmien pitää palauttaa palvelimelle virheettömän suorituksen arvo, jotta DMM hyväksyy prosessin onnistuneeksi.

Käyttöoikeuksilla työvuon eri osia määrätään eri henkilöille, jolloin esimerkiksi yhteiskäytössä toinen käyttäjä voi jatkaa työskentelyä, kun käyttäjän prosessi siirtyy käynnistettävä-tilaan [1,2].

2.1.5 Työkalujen liittäminen prosesseihin

Prosessin komentorivillä vuosta ajettavalle ohjelmalle voidaan asettaa erilaisia suoritustapoja. Eri tapoja ovat kääritty (wrapper) komento, pelkkä komentorivi, kuittauskomento (signoff), kirjastoitu ohjelma (DLL), hierarkkinen komento ja dmmexec-komento. Lisäksi prosessin komento voidaan valita automaattisesti tai tilasta riippumatta suoritettavaksi.

Kääritylla ja dmmexec-komennoilla vuohon liitetään ohjelmia, jotka eivät aseta palautusarvoa. Tälläisen ohjelman päätyttyä DMM kysyy käyttäjältä, mikä työvuon poluista valitaan.

Pelkkää komentoriviä voidaan käyttää, jos ohjelma palauttaa suorituksestaan ainakin onnistui- ja epäonnistui-arvot, joiden perusteella DMM osaa päätellä prosessin onnistumisen.

Kuittaus-komento tuottaa näytölle ikkunan, jonne työvuon käyttäjä voi raportoida esimerkiksi suuremman kokonaisuuden päättymisestä. DLL-kutsuilla voidaan suorittaa kirjastoituja ohjelmia.

Hierarkkisessa prosessissa määritellään alemman tason työvuon nimi. Hierarkiatasoa alemman työvuon alkupisteestä täytyy lähteä yhtä monta pol-

kua kuin hierarkkiseen prosessiin on tulopolkuja. Työvuon loppupisteeseen menevät polut voidaan liittää hierarkkisesta prosessista lähteviin polkuihin vapaasti.

Dmmexec-komento mahdollistaa erilaisten kyselykaavioiden laatimisen esimerkiksi työkaluohjelmien käynnistysparametrien asettamiseksi. Ohjelma tuottaa käyttöliittymän avulla kyselykaavakkeen, jonka sisältö on määrätty sille tehdystä tekstimuotoisesta valintatiedostosta (options). Tiedostoon kirjoitetulla kielellä voidaan tuottaa vuon käyttäjälle kyllä- tai ei-vaihtoehtoja (question), alasvedettäviä valikkoja (choose one) tai kysyä käyttäjältä numero tai kirjaintietoa esimerkiksi tiedoston nimeä varten (parameter). Kaikki valinnat vaikuttavat valintatiedostossa asetettujen lippujen (flag) arvoihin. Samassa valintatiedostossa myöhemmissä valinnoissa voidaan estää tai pakottaa kysymyksiin vastaaminen käyttämällä lippua testaavaa komentoa (onlyifflag) [1,2].

Valintatiedoston kautta pystytään työvaiheessa valitsemaan esimerkiksi eri työkaluohjelmista tai työtiedostoista. Valintatiedostossa eri työkalujen parametrit ja komennot voidaan asettaa tarvittaviksi lippujen avulla.

2.1.6 Työvoiden käyttäminen

Työvuossa suoritettava työprosessi käsittelee tietoa työhakemistossa työkalujen kautta. Valmistettu työvuo on käännöksen jälkeen pohja, josta käyttäjä kopioi työskentelyssä käytettävän työvuon itselleen ja liittää siihen työtiedoston tai työhakemiston [1,2].

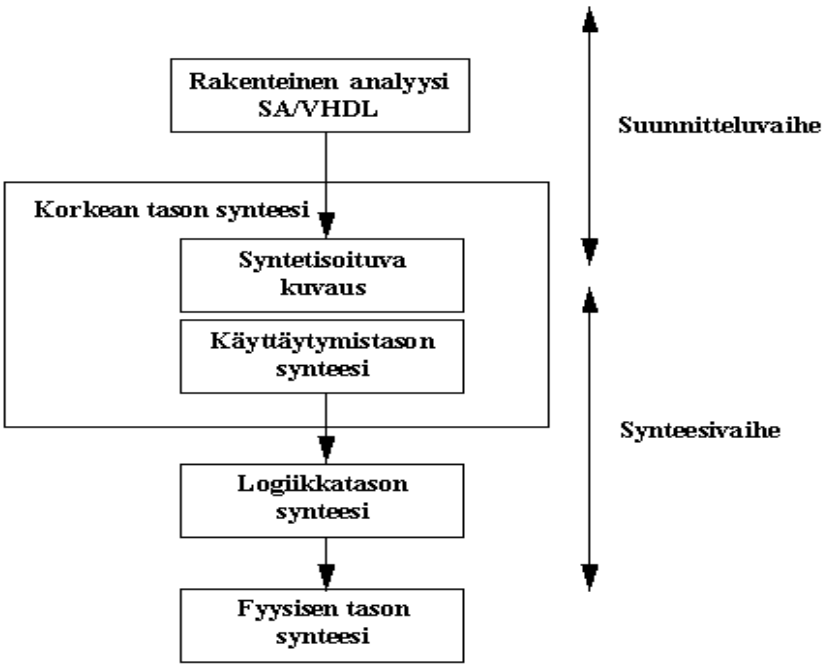
Työvoiden käyttäminen on nopeaa ja helppoa. Tavalliset prosessit työvuon käyttäjä käynnistää hiiren avulla. Osa prosesseista voidaan määritellä automaattisiksi, jolloin työkalukomento käynnistyy itsestään prosessin siirtyessä käynnistettävä-tilaan.

2.2 KORKEAN TASON SYNTEESIN SUUNNITTELUMENETELMÄT

Korkean tason synteesillä ymmärretään piirien syntetisoimista käyttäen lähtökohtana korkean tason HDL-kieltä. Esimerkiksi VHDL-kielellä piirin toiminta voidaan kuvata tietovuotasolla, rekisteritasolla ja porttitasolla. Korkean tason synteesissä käytettävä kieli on tietovuotason ja rekisteritason välissä. Siinä prosessien toiminta on kuvattu synkronisena, kellolla ja alustussignaalilla ohjattavana mallina työkalun vaatimukset huomioon ottaen. Tällä tasolla kirjoitetun kuvauksen tuottaminen on huomattavasti nopeampaa kuin rekisteritason kuvauksen kirjoittaminen, jota käytetään laajasti ASIC-suunnittelussa.

Kuva 4 havainnollistaa työssä mallinnettavan synteesiprosessin kulkua. Suunnittelija voi yksin suorittaa yksinkertaisten ja digitaalisten tuotteiden synteesin nykyaikaisten työkalujen avulla.

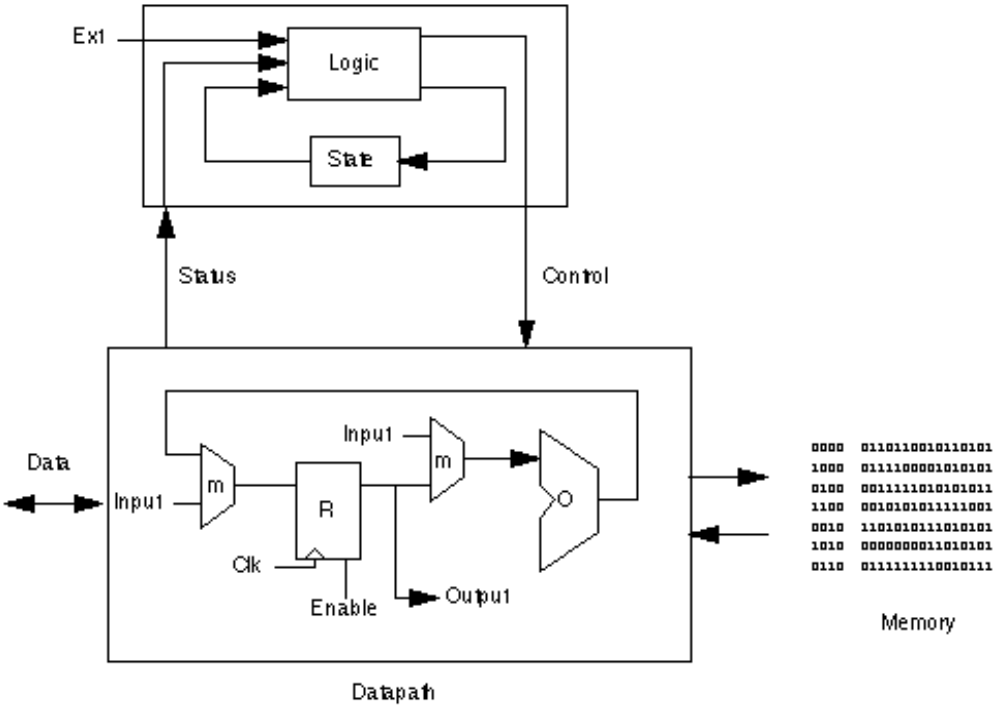
Syntetisoitavan piirin suunnittelu aloitetaan SA/VHDL-mallin rakentamisella. SA-kuvaus on toteutusriippumaton kuvaus, jolla suunnitelman toiminta kuvataan käyttäytymistasolla. Tämän avulla suunnitellaan syntetisoituva kuvaus, jota käytetään käyttäytymistason synteesissä. SA-mallin osia voidaan hyödyntää syntetisoituvaa mallia kirjoitettaessa.



Kuva 4. Korkean tason synteesin sijoittuminen suunnitteluprosessissa.

Käyttäytymistason synteesissä ensimmäinen vaihe on syntetisoituvan koodin analysointi (analyze), jossa suunnitelman koodin syntaksi tarkastetaan ja malli siirretään synteesitietokantaan (db). Seuraavaksi suoritetaan allokointi (elaborate), jossa suunnitelmasta etsitään kaikki operaatiot [3]. Synteesityökalun etsimät operaatiot ovat VHDL-kielessä esiintyvät laskutoimitukset, muuttujien ja signaalien kirjoitus- ja lukuoperaatiot [4].

Syntetisoitavan kuvauksen perusteella synteesityökalu valitsee tarvittavat rekisterit ja operaatioiden toteuttamisratkaisut työkaluun asetettujen suunnitelmarajoitusten puitteissa. Suunnitelmassa olevien operaatioiden ja rekisterien loogiset komponentit on valittu teknologiakirjastosta ja niiden käyttöaste on pyritty maksimoimaan skeduloimalla eli jakamalla samankaltaisia operaatioita samoille komponenteille.



Kuva 5. Käyttäytymistason synteesin tulos.

Käyttäytymistason synteesin tuloksena (kuva 5) saadaan suunnitelmasta prosessin tilakone- (FSM) ja tietopolkukuvaukset (datapath). Suunnitelmassa voi olla lisäksi piirin ulkoista tai sisäistä muistia.

Suunnitelman tilakone ohjaa suunnitelman tietopolkua multiplekserien ja rekisterien avulla [5]. Tilakonekuvaus mahdollistaa suunnitelmassa olevien komponenttien käyttämisen eri aikoina samankaltaisiin operaatioihin, jolloin suunnitelmasta saadaan ehkä pienempi. Tilakonekuvaus syntetisoituu automaattisesti skeduloitaessa.

Logiikkasynteesissä käyttäytymistason synteesin tulos optimoidaan ja komponenteiksi valitaan oikeita teknologiakirjaston komponentteja. Synteesin jälkeen on käytettävissa tarkat kuvaukset mallin komponenttien aiheuttamista viiveistä, tehon kulutuksesta ja mallin koosta. Mallista saadaan komponenttien ohella myös kuvaus langoituksista (netlist). [4].

Synteesin eri vaiheissa suunnitelmaa voidaan optimoida nopeuden, tehon ja koon suhteen antamalla synteesityökalulle ohjeita suunnittelurajoituksien (constraints) muodossa [4].

Fyysisen tason synteesillä tarkoitetaan piirin toteutusta: Komponenttien ja johdinten sijoittelua piille ja valmistamista. Nämä voi tehdä piirivalmistaja.

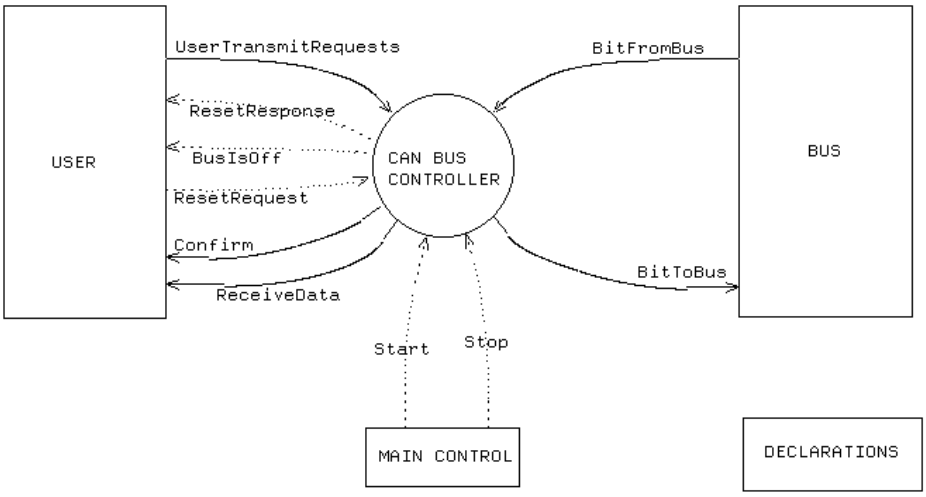
2.2.1 Korkean tason synteesin vaiheet ja eteneminen

Korkean tason synteesiin perustuvassa suunnitteluprosessissa on havaittavissa kolme selkeää osaa: SA/VHDL-mallilla tehtävä toiminnallinen suunnittelu, eri tasoisten mallien simulointi ja mallien syntetisointi. SA/VHDL-mallin toiminnan varmistamiseksi mallia simuloidaan VHDL-simulointityökalulla. Kun SA-malli on valmis, voidaan suunnitella syntetisoituva kuvaus SA-mallin pohjalta. Syntetisoituva malli simuloidaan perusteellisesti sen toiminnan varmistamiseksi ennen sen viemistä synteesiprosessiin.

2.2.2 SA/VHDL-menetelmä

SA eli rakenteinen analyysi on spesifiointi- ja varmennusmenetelmä, jossa mallinnetaan tiedon kulkemista suunniteltavassa ympäristössä. Mallinnukselle on ominaista sen korkea abstraktiotaso ja mallilla tähdätään toiminnan ja käyttäytymisen suunnitteluun. Malli kuvataan tietovuokaavioilla ja tilakaavioilla. SA-kuvissa ongelma jaetaan hierarkkisesti pienempiin osiin, kunnes alimmalla tasolla tietomuunnokseen jää riittävän yksinkertainen tehtävä, jolle voidaan kirjoittaa kuvaus sen toiminnasta eli minispesifikaatio. Kukin minispesifikaatio on VHDL-prosessi, joka koostuu peräkkäin suoritettavista VHDL-lauseista. Näitä tietomuunnoksia ohjataan graafisesta tilakaaviosta signaalien avulla [6]. Mallin toiminta voidaan varmentaa simuloimalla mallia työkalujen avulla.

VTT:llä SA/VHDL-mallia käytetään suunnitelman dokumentaationa ja spesifikaationa ohjelmisto- ja piirisuunnittelussa.



Kuva 6. SA/VHDL-mallin ympäristökaavio.

SA/VHDL-mallin suunnittelu aloitetaan rakentamalla ympäristömalli (context). Ympäristömallissa on piirretty suunnitelman liityntä ulkomaailmaan (kuva 6). Ulkopuoliset osat merkitään terminaattoreilla. Niistä suunnitelmaan tulevat ja suunnitelmasta lähtevät tietovuot piirretään ja

niiden tietotyypit määritetään. Suunnitelmassa käytettävät omat tietotyypit määritetään ”DECLARATIONS”-osan viittaamassa tiedostossa. [7,8,9].

VTT:ssä on kehitetty työkalu Velvet, jonka avulla tietovuo- ja tilakaavioista saadaan ”käännettyä” simuloitava VHDL-malli [10]. Työkalu tuottaa SA-mallista simuloinnissa tarvittavat VHDL-tiedostot. Suunnittelijan vastuulle jää prosessien aktiivisen toiminnan kuvaaminen VHDL-kielellä, tila- ja tietomuunnoskaavioiden piirtäminen sekä tiedostojen nimeäminen tiettyjä sääntöjä käyttäen [11].

2.2.3 Syntetisoituvan kuvauksen suunnittelu

Syntetisoitava malli suunnitellaan SA-spesifikaation perusteella. Toiminta SA-mallissa perustuu tapahtumien odottamiseen ja tapahtuman aikaansaaman muutoksen kuvaamiseen suunniteltavassa ympäristössä. SA-mallin avulla pyritään varmistamaan, että tehdään oikeita asioita oikealla tavalla jo korkeammalla tasolla ja suunnittelua helpotetaan jakamalla ongelma pienempiin osiin.

SA/VHDL-mallia käytetään myös dokumenttina mallinnetun systeemin toiminnasta. Valmistettavan suunnitelman toimintaa voidaan simuloida, jolloin suunnitelman toiminnalliset puutteet voivat tulla ilmi mahdollisimman aikaisessa vaiheessa.

Velvetiä käytettäessä SA/VHDL-minispesifikaation muoto on yleensä seuraavan kaltainen:

```
architecture behavior of sample is
        begin
        process
                variable ExecutionTime:Time:= 20 ns;
                variable EventTime:Time:= 1 ns;
                begin
                wait on Action'Transaction;
                        -- tulovuot muuttujiin
                        -- operaatiot muuttujille
                wait for ExecutionTime;
                -- Lähtövoiden asetus
        end process;
 end behavior;
```

Esimerkki 1. SA/VHDL-minispesifikaatio ennen muuttamista.

Kuten esimerkistä 1 nähdään minispesifikaatiossa on havaittavissa 3 osaa. Ensimmäisessä osassa on ”wait on .. transaction”-lauseke, jossa odotetaan herätesignaalia. Toisessa osassa sijoitetaan tulevat vuot paikallisiin muuttujiin ja suoritetaan niille operaatiot. Kolmannessa osassa tulokset asetetaan lähtövoihin simulointia varten asetetun suoritusviiveen jälkeen.

Syntetisoitava malli kuvaa suunnitelman käyttäytymisen lähempänä oikeaa ympäristöä. Mallista tehdään kellolla ohjattava synkroninen kuvaus, jossa määritellään kellojakson aikana tehtävät asiat eri prosesseissa VHDL- tai RTL-kielellä.

Esimerkissä 2 on kuvaus suunnitteluyksikön esittelystä, jossa kuvataan prosessin liityntä ulkomaailmaan. Velvet työkalu tuottaa graafisesta SA-mallista muitakin synteesisuunnittelussa käyttökelpoisia kuvauksia.

```
entity sample is
port(
        Action:in boolean;
        RESET: in boolean;
        CLOCK: in bit
);
end sample;
```

Esimerkki 2. Käyttäytymistason VHDL-prosessin esittely.

Tyypillinen syntetisoituvan prosessin rakenne voi olla esimerkin 3 mukainen: Ensin kuvataan prosessin käyttäytyminen alustuksessa. Seuraavaksi odotetaan herätesignaaleja silmukassa ja niiden jälkeen kuvataan SA-mallissa suunniteltu tehtävä käyttäen työkalulle ominaisia VHDL-rakenteita.

```
architecture behavior of sample is
begin
-- rinnakkaiset sijoitukset
behavior_sample : process
begin
        reset_loop : loop
        -- muuttujien ja voiden alustaminen
        main_loop : loop
                wait until CLOCK'event and CLOCK = '1';
                exit reset_loop when RESET = false;
                        waiting_loop : loop
                                wait until CLOCK'event and CLOCK = '1';
                                exit reset_loop when RESET = false;
                                exit waiting_loop when Action = true;
                                end loop waiting_loop
                wait until CLOCK'event and CLOCK = '1';
                exit reset_loop when RESET = false;
                -- toiminta osa, kättelyt, lähtövuot
                end loop main_loop;
        end loop reset_loop;
end process;
end behavior;
```

Esimerkki 3. Käyttäytymistason VHDL-prosessin esimerkkimalli.

Syntetisoituvan suunnitelman on oltava synkroninen. Tämä saavutetaan, kun jokaisen kellon nousevan reunan muutosta odottavan lauseen perään lisätään alustussignaalia testaava lause.

Prosessissa olevat kellon reunan odotuslauseet määräävät kellojakson rajat synteesissä. Sen vuoksi esimerkiksi uloslähtevään signaaliin kahdesti kirjoittaminen ei ole viisasta samassa kellojaksossa, koska signaalien arvot vaihtuvat vasta kellon reunan noustessa. Työkalu pyrkii asettamaan kellojakson ajalle niin monta operaatiota kuin mahdollista ja se voidaan asettaa tilaan, jossa tarpeen vaatiessa ylimääräiset operaatiot voivat siirtyä eri kellojaksoille.

Tiedon siirtämisessä prosessista toiseen on varmistettava, että vastaanottaja on valmis ottamaan tietoa vastaan ja että lähettäjä on varma siitä, että tieto on vastaanotettu. Prosessien välillä tarvitaan enemmän viestintää eikä dataa voida siirtää niin ideaalisesti kuin SA-mallissa. Herätteiden ja tiedon vastaanottamisessa ja lähettämisessä on huomattava, että nyt prosesseilla kuluu suoritukseen aikaa vähintään niin monta kellojaksoa kuin siellä on kellon reunan odotuslauseita.

Tilakoneiden muuttamisessa voidaan käyttää Mealy- tai Moore-tyyppistä tilakoneratkaisua [12]. Velvetin avulla saadaan SA-mallin tilakoneesta pohja, jonka avulla muokkaus nopeutuu. Tarkempia ohjeita koodin kirjoittamiseen ja käyttäytymistason synteesiin on viitteessä [13].

Synopsyksen BC:llä syntetisoitavien prosessien tulisi olla kooltaan 250:sta muutamaan tuhanteen riviä pitkiä ja ne saisivat sisältää korkeintaan 150 operaatiota. Suurempien prosessien syntetisointi voi viedä liikaa koneresursseja tai aikaa. Käyttäytymistason synteesissä resurssien jako toimii sitä paremmin mitä enemmän operaatioita voidaan suorittaa samoilla komponenteilla. Tämä vaatii, että peräkkäiset samankaltaiset operaatiot lasketaan eri kellojaksoilla. Tällöin prosessin suoritus vaatii pidemmän ajan, mutta prosessin tarvitsemasta piipinta-alasta tulee pienempi. Jos prosessilta vaaditaan suurta nopeutta pinta-alasta tulee suurempi, koska tarvitaan enemmän resursseja rinnakkaiseen laskentaan.

”Synopsys BC”:n versio 3.3b:ssä synteesin resurssien jako tapahtuu prosessikohtaisesti, joten eri prosessien välillä ei voida käyttää yhteisiä rekistereitä. SA:n pieniä minispesifikaatioita voidaan tarvittaessa helposti yhdistää suuremmiksi prosesseiksi. Rekisterien koolla on synteesissä vähemmän merkitystä verrattuna esimerkiksi kertoja- tai summauskomponenttien kokoon.

Synteesin tuloksiin vaikuttavat myös synteesityökalun muuttujat (variable) ja attribuutit (attribute). ”Vhdlout”- ja ”hdlin”-alkuiset muuttujat vaikuttavat VHDL:n lukuun ja kirjoitukseen. Synteesissä synteesityökalua voidaan ohjata koodista käsin attribuuttien avulla. Esimerkiksi ”dont_unroll”-attribuutin asettaminen estää sillä määritetyn silmukan avaamisen. Nämä menetel-

mät yhdessä koodin rakenteen kanssa vaikuttavat synteesin tulokseen ja simuloitavan mallin rakenteeseen huomattavasti.

2.2.4 VHDL-mallien simulointi

Simuloinnilla pyritään varmistamaan, että tehty malli toimii oikein. Suunnitelman prosessit ja simulointiprosessi kytketään yhteen testipenkillä. Periaatteena simuloinnissa on antaa suunnitelmalle herätesignaaleja simulointiprosessista. Simulointiprosessissa voidaan lähettää määrättynä simulointiaikana heräte suunnitelmaan ja vastaanottaa ja kuvata suunnitelmasta tulevaa tietoa. Simulointityökalun avulla ohjataan simuloinnin kulkua ja tulostetaan aikagraafiin suunnitelmasta valittujen signaalien arvoja.

Simulaatioita tehdään useita suunnitelman jokaisella abstraktiotasolla. Käyttäytymistason synteesin jälkeen simuloidaan RTL-tasolla ja viimeiseksi logiikkasynteesin jälkeen porttitasolla. Ideana on, että samoilla herätteillä voitaisiin simuloida jokaisella tasolla käyttäytymisen vertailemiseksi. Kuitenkin SA-simuloinnissa käytettyihin herätteisiin tarvitaan muutoksia syntetisoituvaa koodia simuloitaessa mm. kello, alustussignaalien ja kättelyiden toteuttamiseksi.

SA-tason ja syntetisoituva VHDL-koodi voidaan simuloida millä tahansa standardia tukevalla simulaattorilla. Suunnittelutyössä oli käytettävissä PC-työasemassa V-SYSTEM, Unix:ssa vanhempi VHDL2000 ja Synopsys VSS -simulaattorit. Syntetisoiduista käyttäytymis- ja logiikkatason kuvauksista tuotetut vhdl-kuvaukset simuloidaan Synopsys VSS -simulaattorilla.

2.2.5 Uudelleenkäyttömenetelmät SA/VHDL-suunnittelussa

Tässä työssä uudelleenkäytöllä tarkoitetaan SA/VHDL-mallien uudelleenkäyttöä. VTT:llä valmistetun kirjastointiohjelman avulla voidaan SA-hierarkioita tai yksittäisiä tietomuunnoksia siirtää kirjastoon uudelleenkäytettäväksi.

Ohjelmistopuolella uudelleenkäytettävien objektien käytössä on havaittavissa seuraavia ongelmia: ymmärtäminen, löytäminen, muuttaminen ja liittäminen [14]. Samat ongelmat löydetään helposti SA/VHDL-malleja uudelleenkäytettäessä. SA-mallien ymmärtämistä ja muuttamista helpottaa graafinen ja tiettyjä sääntöjä käyttäen piirretty kuvaustapa. Esimerkiksi tilakoneiden kuvaus on graafinen, jolloin niiden muokkaaminen on helpompaa kuin, jos tilakone olisi pelkkää VHDL-kieltä.

SA-mallien koko on suhteellisen pieni ennen Velvet-käännöstä. Tästä seuraa, että uudelleenkäytettävien mallien muokkaus sujuu nopeammin suhteessa siihen, että uudelleenkäyttöä suoritettaisiin alemmalla tasolla, jossa

koodin määrä voi olla viisikin kertaa suurempi ja koodissa on jo voitu ottaa kantaa toteutukseen.

”Sv-manager” on VTT:ssä valmistettu SA-mallien kirjastointiohjelma [15]. Ohjelmassa on apumenetelmiä objektien löytämiseksi ja muuttamiseksi. Työkalussa tallennetaan osia tai hierarkioita SA-kuvista myöhempää käyttöä varten. Tallennetut mallit sijaitsevat yleisessä kirjastossa (global library), josta malleja siirretään käytettäväksi paikalliseen kirjastoon (local library). Malleja voidaan hakea kirjastosta tallennettujen avainsanojen perusteella. Malleista on tallennettu erilaista tietoa, kuten toimintakuvaus, toteutustapa ja mallin koko.

Malleja uudelleenkäytettäessä mallin ulospäin näkyvien ja sisäisten signaalien nimet voidaan vaihtaa. Signaalien nimiä voidaan muuttaa määrittelemällä esimerkiksi nimiin lisättävän etu- tai takaliitteen. Tästä on apua, jos samaa komponenttia käytetään useassa paikassa ja jos suunnitelmaan ei haluta samannimisiä SA-osia.

SA-kaavioiden tallennustyövaiheet ovat seuraavat:

1. Prosalla tehdään kirjaston tallennus -komento, jossa tallennettava alue rajataan. Tallennettavalle komponentille annetaan nimi ”obj.dfd” ja se tallennetaan liitehakemistoon.
2. Kirjastointiohjelma käynnistetään, jolloin komponentti luetaan kirjastoon.
3. Ohjelmassa komponentti nimetään, kirjoitetaan avainsanoja ja kuvaus toiminnasta.
4. Malli lisätään yleiseen kirjastoon.

SA-kaavioiden käyttöönottaminen tehdään seuraavasti:

1. Yleisestä kirjastosta haetaan tarvittava komponentti, joka siirretään paikalliseen kirjastoon antamalla komponentille sopiva hierarkianumero.
2. Signaalien nimet muokataan sopiviksi.
3. Lisätään kaavio SA-malliin muokattavaksi.

2.3 YLLÄPITOMENETELMIÄ

Tässä työssä ylläpitoa tarkastellaan pääosin ohjelmistosuunnittelun kautta. Tarkoituksena on löytää menetelmiä uudelleenkäytön ja synteesisuunnittelun apumenetelmiksi.

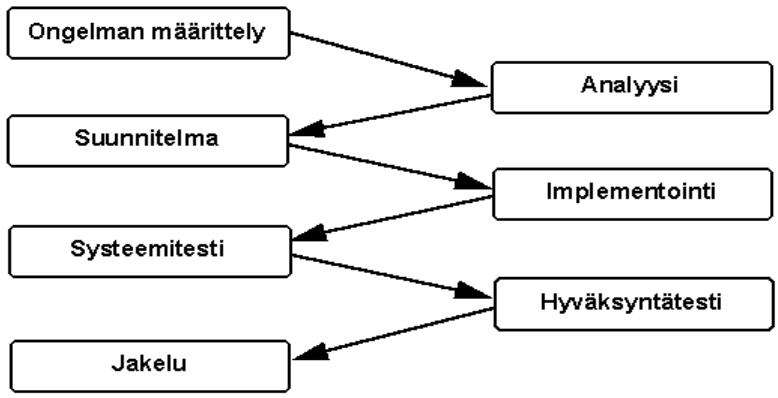
Ohjelmistotekniikassa ylläpidolla tarkoitetaan ohjelman muuttamista sen julkaisun jälkeen virheiden korjaamiseksi tai tehokkuuden tai muun ominaisuuden parantamiseksi tai ohjelman sopeuttamiseksi uuteen ajoympäristöön.

Ylläpitoa on olemassa 4 eri tyyppiä: korjaava, sovittava, parantava ja kiireellinen ylläpito [16]. Korjaavaa ylläpitoa on ohjelmassa havaittujen virheiden korjaaminen tai puuttuvien toimintojen lisääminen. Sovittavalla ylläpidolla ohjelma muutetaan toimimaan uuteen käyttöympäristöön. Parantava ylläpito suoritetaan, kun ohjelmaa muutetaan helpommin ylläpidettäväksi. Kiireellinen ylläpito on vian korjaamista ilman dokumentaatiota systeemin pitämiseksi toiminnassa.

Tässä työssä ylläpidoksi katsotaan VHDL-mallien muuttaminen uudelleenkäyttötilanteessa tiedostojenhallintaan tarkoitetun työkalun avulla, jolla voidaan myös parantaa suunnitelman hallintaa muuallakin suunnitteluprosessissa. Seuraavissa kappaleissa esitetään ohjelmistojen ylläpitoa, ylläpidon yleisiä ongelmia, tiedostojenhallintamenetelmä ja VHDL-kielen ylläpitoa. Luonnollista nimeämistä käsitellään, koska se liittyy uudelleenkäytettävyyden parantamiseen.

2.3.1 Ylläpito ohjelmistotekniikassa

Ohjelmistojen ylläpito on standardin mukaan jaettu osatekijöihin (kuva 7). Ylläpidon avulla pidennetään ohjelmistoprojektissa tuotetun tuotteen elinkaarta muuttamalla sitä erilaisten menetelmien avulla [16].



Kuva 7. Ylläpitoprosessin tehtävät.

Ongelma, joka ylläpidon avulla yritetään ratkaista, määritellään muutosvaatimuksessa (modification request). Ylläpitotehtävästä muodostetaan vaatimusmäärittely ja määrätään ongelmalle prioriteetti verrattuna muihin samanaikaisiin ylläpitotehtäviin. Määritellään myös mitä ylläpitomenetelmää käytetään ongelman ratkaisussa, sekä estimoidaan ongelman korjauksen vaatimaa resurssitarvetta.

Analyysivaihe jakaantuu kahteen osaan: toteutettavuuden (feasibility) analysointiin ja lopulliseen (detailed) analyysiin. Toteutettavuutta analysoidaan

arvioimalla muutoksen vaikutukset ja hyödyt ohjelmistolle, etsitään vaihtoehtoisia ratkaisuja ongelmaan ja arvioidaan muutoksen aiheuttama lyhyen ja pitkän aikavälin hinta edellisten arviointien perusteella. Lopullisessa analyysissä tehdään lopullinen vaatimusmäärittely, tarkastetaan muutoksen tarvitsemat resurssi- ja aika-arviot, suunnitellaan testausstrategia, tuotetaan implementointisuunnitelma, lista tehtävistä muutoksista ja niiden toteuttamissuunnitelma.

Suunnitelmavaiheessa paikannetaan ohjelmamoduulit, joita muutos koskee. Muutetaan moduulien dokumentaatio vastaamaan muutosta, luodaan testitapaukset uudelle suunnitelmalle ja täydennetään implementointisuunnitelmaa.

Implementaatiovaiheessa lähdekoodiin tehdään suunnitelman mukaiset muutokset, suoritetaan integrointi, riskien analysointi ja testataan muutoksen valmius. Implementaatiovaihetta voidaan toistaa useita kertoja. Riskien analysoinnissa arvioidaan muutosprosessin etenemistä aikataulussa ja ohjelman vioittumisen todennäköisyyttä. Implementaation tuloksena saadaan uusi ohjelmajulkaisu, dokumentaatio, testausdokumentaatio, käyttöohjeet ja harjoitusmateriaali, sekä testaussuunnitelma.

Systeemitestissä suoritetaan testaussuunnitelman mukaan toiminnallinen testaus. Hyväksyntätesti suoritetaan oikeassa käyttöympäristössä kuluttajien toimesta. Testien osoittaessa muutoksen toimivaksi aloitetaan uuden version jakelu. [16].

2.3.2 Ylläpidon ongelmat

Ylläpidolle tuottavat ongelmia seuraavat tilanteet [17]:

- ohjelmasta on vähän tai huonosti tehtyä dokumentaatiota käsillä,
- ohjelmaa ei ole suunniteltu muutettavaksi,
- ohjelman moduulien määrä on pieni ja koko suuri,
- ohjelmasta ei ole versiohistoriaa ja
- ylläpito on kallista.

Ohjelman ylläpitäminen on helpompaa, jos dokumentaatio on olemassa. Dokumentaation ulkoasu vaikuttaa myös ohjelman ylläpidettävyyteen dokumentaation ymmärtämisen kautta.

Ylläpidon helppouteen vaikuttaa ohjelman rakenne. Mitä suurempia kokonaisuuksia ohjelmamoduulit sisältävät, sitä vaikeampaa on niiden muuttaminen. Ylläpidettävyys ohjelmistotekniikassa voidaan määritellä ylläpitomenetelmien suorittamisen helppoutena. Ylläpidettävän ohjelman toiminta on helppoa ymmärtää, korjata ja parantaa ilman suuria kustannuksia.

Ylläpidon kalleuteen vaikuttaa olennaisesti ylläpitotapahtuman suorittamisen nopeus, johon vaikuttavat ohjelman rakenteen ja toiminnan analysoimiseen menevä aika sekä ylläpitoon liitettyjen resurssien määrä.

2.3.3 Tiedostojenhallinta ylläpitoprojektissa

Ohjelman konfiguraation hallintaa (SCM) tarvitaan ylläpitovaiheiden aikana. Sen avulla ylläpitoon osallistuvalla on kohdeohjelman tiedostoista aina uusin versio käytössään. SCM-käsitteeseen sisältyvät versionhallinnan lisäksi versiohistoria, työkalujen konfiguraation ylläpito, riippuvuuksien seuranta ja automaattinen kääntäminen, tietoturva ja työtilan hallinta. SCM:n avulla ylläpitoprosessissa tallennetaan suunnitelmasta seuraavia tiedostoja: lähdekoodien eri versiot, lähdekoodin systeemi- ja käyttäjädokumentaatiot, testausdokumentaatiot, ylläpidon muutoslistat, implementointi-, analyysi- ja suunnitteluraportit ja spesifikaatiot [18].

Versionhallinnan ja historian avulla suunnitteludokumentista pidetään tallessa eri versioita ja tärkeitä tietoja versioista: esimerkiksi mitä muutoksia on tehty, kuka ne on tehnyt ja milloin. Ohjelman konfiguraatio käsittää joukon ohjelmadokumentteja, jotka yhdessä muodostavat tuotteen. Erilaisia konfiguraatioita voivat olla esimerkiksi eri käyttöjärjestelmille suunnatut ohjelmaversiot tai ohjelman vanhemmat versiot. Konfiguraatio sisältää tiedon jokaisen siihen kuuluvan suunnitteluobjektin versiosta. Riippuvuuksien seurannassa huolehditaan eri suunnitteluobjektien välisistä viittauksista, joiden avulla esimerkiksi oikea kääntämis- ja linkitysjärjestys määrätään. Näiden viittausten avulla voidaan automatisoida käännös ja linkitys sekä oikeiden kirjastojen käyttö ja tarvittavien työkalujen suoritus. Ryhmätyöskentelyssä tietoturvan avulla rajataan käyttäjien pääsyä suunnitteluobjekteihin salasanojen, tiedoston etapin tai muun seikan avulla. Työtilan hallinnassa pyritään välttämään toimintojen monimutkaisuutta.

PVCS-ohjelma tukee hajautetussa ympäristössä tapahtuvaa versionhallintaa. Ohjelmisto on tarkoitettu nimenomaan ohjelmistopuolelle, mutta sen ominaisuuksia voi käyttää hyvin kaikenlaisen tiedon käsittelyssä [19].

2.3.4 VHDL-kielen ylläpito

ASIC-suunnittelussa ylläpidoksi voidaan katsoa esimerkiksi olemassa olevan ASIC-piirin kuvauksen muuttaminen toiselle teknologialle tai uusien piiriversioiden tuottaminen käyttäen apuna olemassa olevien kuvauksen osia. Piirikuvauksen muuttaminen voidaan rinnastaa ohjelmistotekniikan sovittavaan ylläpitoon.

Seuraavaksi esitellään joitakin European Space Agencyssa (ESA) käytössä olevia menetelmiä VHDL-kielen ylläpidettävyyden parantamiseksi. Niiden mukaan VHDL-prosessista pitäisi dokumentoida seuraavat asiat [20]:

- suunnitteluyksikön nimi,
- tiedoston nimi,
- prosessin tarkoitus ja selitys sen toiminnasta,
- rajoitukset ja todetut virheet,
- suunnittelukirjasto, jolle koodi on tehty,
- lista analyysissä vaikuttavista riippuvuuksista,
- prosessin kirjoittaja,
- käytetty simulaattori,
- ympäristö ja
- muutoslista eri versioista.

VHDL-kieltä käytettäessä piirikuvauksien kirjoittaminen on eräänlaista ohjelmointia, jolloin siihen voidaan soveltaa ohjelmistotekniikan alueilla käytettäviä menetelmiä.

2.3.5 Luonnollisen nimeämisen käyttäminen ohjelmistotekniikassa

Ohjelmistoprojekteissa luonnollisten nimien käyttäminen on havaittu hyväksi menetelmäksi. Tutkimuksissa on todettu pitkien nimien mm. auttavan nimien muistamisessa projektin aikana, helpottavan tietyn kohdan etsimistä lähdekoodista Nimien keksiminen on auttanut ongelman analysoinnissa ja luonnolliset nimet ovat useiden mielestä vähentäneet ajatteluprosessia verrattuna lyhenteiden ja lyhyiden nimien käyttämiseen. [21].

Luonnollisen nimeämisen käytön taustalla on ajatus, että lähdekoodi soveltuisi suoraan dokumentiksi hyvän esitystapansa ja luettavuutensa vuoksi. Menetelmällä tehtyjen ohjelmien ylläpito olisi halvempaa, koska koodia muuttavan henkilön olisi helpompi ymmärtää ohjelman toiminta ja tehdä siihen tarvittavat muutokset.

Luonnollisen nimeämisen periaate

Luonnollisia nimiä käytettäessä ovat voimassa seuraavat säännöt: nimet kirjoitetaan kieliopillisesti oikein ja lyhenteiden käyttöä vältetään. Objektin nimi kuvaa objektin käyttötarkoituksen tai objektin suorittaman tehtävän tai objektin sisältämän tiedon niin hyvin, että nimestä nähdään objektin osuus suunnitelmassa.

Esimerkiksi ohjelmassa muuttujan nimiä voisivat olla

a) buf
b) buffer
c) key_buffer
d) Input_Key_Buffer
e) User_Input_Key_Buffer

Näistä a, b ja c ovat liian lyhyitä kuvatakseen tarpeeksi puskurin tarkoitusta tai sen sisältämää tietoa, d ja e ovat tarpeeksi kuvaavia [21].

Luonnollisten nimien käyttäminen

Luonnollisella nimillä saavutetaan seuraavia etuja: kommenttirivien määrää voidaan vähentää, ohjelman ymmärtäminen paranee ja ohjelmalistauksiin voidaan saavuttaa yhtenäisyyttä.

```
#define PMAX 100              /* MAX Amount Of Phone Number Items */
int PArray[PMAX];             /* Array Of Phone Numbers    */
int Pindex=0;                 /* Index to Phone Number Array */

void show_numbers() {
        for(PIndex = 0 ; PIndex < PMAX ; PIndex++ )
        {
                printf(" %d \n" , PArray[ PIndex ] );
        }
}
```

Esimerkki 4. Tavallisesti käytettyä nimeämistä.

Esimerkeissä 4 ja 5 havainnollistetaan luonnollisen nimeämisen käyttöä verrattuna lyhyiden nimien käyttämiseen. Niistä havaitaan, että pitkät luonnolliset nimet eivät tarvitse lisäselvityksiä kommenttirivien avulla. Pitkien nimien avulla ohjelman kirjoittajan huomio siirtyy kryptisten nimien muistamisesta enemmän suunnitelman toteuttamisen ongelmaan. Ohjelmoijan ei tarvitse assosioida esimerkiksi ”PIndex”-muuttujaa taulukkoon osoittavaksi indeksiksi, vaan hän voi johtaa nimen vaikka ”Phone_Number_Array”-nimisestä taulukosta. Samalla nimien kirjoitusvirheet vähenevät, koska ne havaitaan helpommin.

```
#define MAXIMUM_AMOUNT_OF_NUMBERS 100

int  Index_to_Phone_Number_Array = 0;
int  Phone_Number_Array[ MAXIMUM_AMOUNT_OF_NUMBERS ];

void Show_Phone_Numbers_On_Display() {
for(
  Index_to_Phone_Number_Array = 0 ;
  Index_to_Phone_Number_Array<MAXIMUM_AMOUNT_OF_NUMBERS ;
  Index_to_Phone_Number_Array ++   )
{
  printf(" %d \n", Phone_Number_Array [ Index_to_Phone_Number_Array ] );
 }
}
```

Esimerkki 5. Luonnollista nimeämistä.

Lyhenteiden käytöllä lähdekoodista tulee vaikeasti luettavaa ja siten vaikeammin muuteltavaa. Toisaalta voi tuntua helpommalta käyttää tutuksi tulleita lyhenteitä koodin kirjoittamisen nopeuttamiseksi. Kuitenkin lyhenteillä kirjoitettua koodia on vaikeampi muuttaa myöhemmin kun ohjelman toiminnan unohtumista on tapahtunut tai kun ohjelman toimintaa ei ennestään tunneta.

Suurissa projekteissa yhteistyöllä on suuri merkitys. Yhteistyö lisää myös kommunikaation määrää ohjelmoijien välille. Kommunikaatiotilanteissa on helpompaa lausua luonnollisia nimiä kuin esimerkiksi nimeä ”hWnd”.

Nimeämissääntöjen ja ulkoasultaan yhtenäisen lähdekoodipohjan ja lähdekoodin kirjoitussääntöjen avulla saavutetaan etua lähdekoodin uudelleenkäytössä, uusien versioiden tuottamisessa ja erilaisissa ylläpitotapahtumissa.

3 SUUNNITTELUPROSESSIN MALLINTAMINEN

Suunnittelu- ja ylläpitomenetelmien kuvaaminen DMM-työvuoksi aloitettiin tutkimalla eri työvaiheita ja niiden riippuvuuksia toisistaan CAN-kontrollerin syntetisointiprojektin kautta [22]. Tämän jälkeen etsittiin mahdollisuuksien mukaan uudelleenkäytettävyydelle ja ylläpidolle työvaiheet ja niiden mahdollinen sijainti työvuossa.

Valmistetun työprosessimallin avulla suunnittelija pystyy mallintamaan, verifioimaan ja syntetisoimaan ASIC-piirejä työvuota käyttämällä.

3.1 TYÖVAIHEIDEN HAHMOTTAMINEN MALLIPROJEKTIN AVULLA

Seuraavissa kappaleissa esitellään CAN-kontrolleri ja kontrolleriprojektissa käytettyjä työmenetelmiä suunnittelun eri vaiheissa. Vaiheet on jaoteltu niihin käytetyn työajan ja siten työmäärän mukaan erillisiksi kokonaisuuksiksi. Kokonaistyömäärä kontrolleriprojektissa oli noin 9 kuukautta. CAN-kontrollerimallinnuksesta saadun tiedon kautta pyritään myöhemmin tuottamaan apuvälineitä syntetisoituvan koodin tuottamiseksi SA-spesifikaation pohjalta.

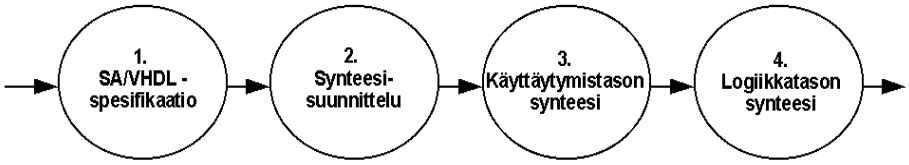
3.1.1 CAN-kontrolleri

CAN-kontrollerin avulla lähetetään ohjauskomentoja tai dataa eri päätelaitteille parikaapelia pitkin kehysten (frame) avulla. CAN-protokollasta on olemassa kaksi standardin mukaista toteutusta, ne eroavat vain kehysten tunnisteiden lukumäärällä. Kontrollereita voidaan käyttää eri väylätopologioissa ja väylällä voi olla useita johtajia [*23*].

CAN-kontrollerin käyttömahdollisuudet ovat hyvät. CAN-verkko on halpa, luotettava ja nopea sarjamuotoinen siirtomenetelmä. Komponenttivalmistajia on useita ja väylään on helppo liittää uusia kontrollereja. Käyttökohteita ovat erityisesti ahtaat tilat. Esimerkiksi erilaiset hissit ja nosturit, auton sähkölaitteiden, erilaisten venttiilien ja hydrauliikan ohjaus.

3.1.2 SA/VHDL-korkean tason synteesi työvaiheina

Työvaiheet olivat karkeasti jaoteltuina SA-mallin tekeminen, synteesisuunnittelu ja mallin syntetisointi. Kuva 8 esittelee synteesiprosessin päätyövaiheet.

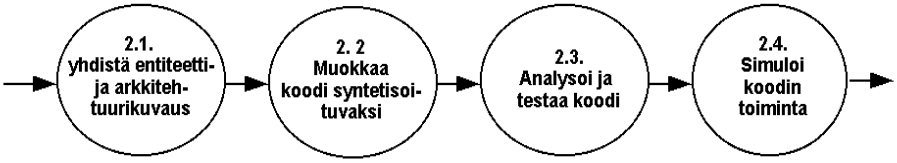


Kuva 8. Käytetyt perustyövaiheet synteesissä.

SA/VHDL-mallin tekemisessä käytettiin yksinomaan Prosaa. Kuvauksessa tuotetut minispesifikaatiot simuloitiin pääosin hierarkiatasoittain. Velvet-käännöksen jälkeen simuloitava malli löytyy kahdesta eri hakemistosta "../vhdl" ja Velvetin käynnistyshakemistosta. Lisäksi Velvet kokoaa suunnitelman prosessien suunnitteluyksikön esittely- ja rakennekuvaukset omaansa (entities.vhd) ja arkkitehtuurikuvaukset omaan tiedostoon (archits.vhd).

Mallien simuloinnit suoritettiin työkalulla VHDL2000. Simulointia varten tarvittava herätetiedosto (stimul-b.vhd) muokattiin käsin jokaiselle simuloitavalle hierarkialle. Eri herätteet tallennettiin muuttamalla tiedoston nimi kuvaamaan kyseessä olevaa hierarkiatasoa. Testipenkki generoitiin tarvittaessa uudelleen Velvetin avulla vastaamaan muutettua suunnitelmaa.

Paljon tapahtumia sisältävä SA-mallin simulointikierros kestää vain muutaman minuutin, jos herätteet ovat edellisestä kierroksesta valmiina.



Kuva 9. Synteesisuunnittelun työtehtäviä.

Kuva 9 esittelee synteesisuunnittelussa käytettyjä työtehtäviä. Käyttäytymistason koodimalliin siirryttiin SA/VHDL-mallista käyttämällä hyväksi Velvetillä tuotettuja tilakone-, rakenne-, sekä esittelykuvauksia, että itse kirjoitettuja minispesifikaatioita. Tiedostojen muokkauksen helpottamiseksi pyrittiin yhdistämään eri tiedostoissa olevat esittely- ja arkkitehtuurikuvaukset samaan tiedostoon. Tiedostojen määrä oli suuri, jolloin muunnostyön tekemisessä havaittiin käytännölliseksi suorittaa yhden hierarkiatason muokkaus omassa hakemistossaan.

SA-mallista saatuja pieniä osia yhdistettiin muutosvaiheessa. Haara- ja tietovarastomuunnokset voitiin usein liittää toisiin prosesseihin. Samoin useiden arkkitehtuurien sisältämiä prosessikuvauksia siirrettiin yhden arkkitehtuurin alle. Velvetin tuottamia tilakoneita muutettiin myös käsin: usein eri muunnoksiin meneviä käynnistys- ja pysäytyssignaaleja voitiin yhdistää yhdeksi.

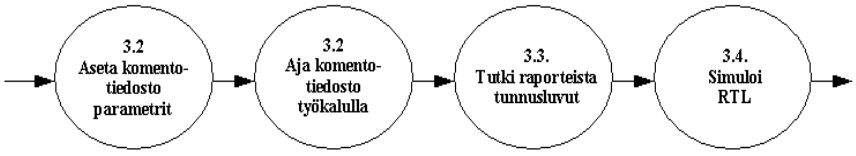
Prosessien väliseen tiedonsiirtoon tuotettiin kättelysignaalit ja tieto siirrettiin tietynkokoisissa paloissa prosessilta toiselle niiden avulla. Muutosten tekeminen tietysti muutti suunnitelmassa olevia tietorakenteita ja prosesseista ulospäin näkyviä signaaleita, jolloin myös rakennekuvauksia päivitettiin.

Synteesityökalua ohjataan ja se tukee käytäntöä, jossa peräkkäiset komennot annetaan komentotiedostosta (script). Työkalun sisällä on myös komentotulkki, jonka avulla voidaan valmistaa monimutkaisiakin ”ohjelmia”. Tässä työssä komentotiedostoissa on eniten käytetty työkalun sisäisiä muuttujia ja niille tehtäviä vertailuoperaatioita.

Valmiiden kuvausten analysointi suoritettiin BC:lle tehdyllä komentotiedostolla, jolla jokainen muutettu arkkitehtuurikuvaus analysoitiin ja allokoitiin skedulointia varten. Analyysin tuloksena suunnittelija saa raportin koodin mahdollisista virheistä. Syntetisoituvia tilakonekuvauksia ei skeduloida ja ne vain testattiin ja allokoitiin logiikkasynteesiä varten.

Hierarkiatason valmistuttua se simuloitiin käyttämällä samoja herätteitä kuin SA-mallissa lisäämällä niihin kello- ja alustussignaalit. Tiedonsiirtoa varten herätteet muokattiin vastaamaan tarvittaviin kättelysignaaleihin. Joihinkin prosesseihin oli tehtävä muutoksia simulaattorin takia, jolloin ne jouduttiin analysoimaan uudelleen ennen simulointia.

Käyttäytymistason mallin valmistuttua voitiin lähteä syntetisoimalla etsimään RTL-tason kuvauksia.



Kuva 10. Käyttäytymistason synteesin työvaiheita.

Kuva 10 havainnollistaa suunnittelun etenemistä käyttäytymistason synteesissä. Komentotiedostojen teon jälkeen tarvitsi vain tutkia, onnistuiko synteesi halutulla tavalla vai pitikö esimerkiksi lähdekoodia muuttaa.

Synteesiä varten asetettiin parametrina kellojakson pituus, suunnitteluyksikön nimi ja haluttu prosessin kellojaksojen määrä. Käännöstä ohjattiin tarvittaessa asettamalla VHDL-koodiin kääntäjälle ohjeita attribuuttien muodossa ja muuttamalla kääntäjän asetuksia. Jokainen arkkitehtuurilohko vietiin erikseen synteesiin. Tunnuslukuina synteesiraporteista saatiin prosessin kokoarvio, suoritusaika kellosykleinä, resurssien määrä ja niiden käyttötietoa kartan muodossa. Muutamista prosesseista etsittiin kokeilemalla nopeudeltaan ja kooltaan erilaisia toteutusvaihtoehtoja parametreja muuttamalla. Useimmat kontrollerin prosesseista jäivät pieniksi, jolloin niistä ei pystytty tuottamaan kovin erilaisia toteutuksia.

Toteutusvaihtoehdot tallennettiin työkalun omana tietorakenteena tietokantamuodossa (db) ja RTL-kuvauksena, joka simulointiin toiminnan tarkistamiseksi. Simuloitaessa käytettiin synteesisuunnittelussa tuotettuja rakenne-, komponentti- ja tilakonekuvauksia. Simulointi suoritettiin VHDL2000 ja VSS simulaattoreilla. RTL-kuvaukset ovat suurehkoja, jolloin on parasta käyttää yhteensopivuuksien ja nopeuden takia VSS simulaattoria.

RTL-tason malli kuljetettiin synteesiin hierarkiatasoittain omissa hakemistoissaan. Tämä siksi, että syntyvien tiedostojen määrä kasvaa ja niiden hallinta käsin vaikeutuu nopeasti. Synteesin jälkeen yhdestä prosessitiedostosta tuotetaan raporttitiedosto, tietokantatiedosto, simuloitava kuvaus ja kolme muuta synopsyksen tarvitsemaa tiedostoa.

Käyttäytymistasolta valitut toteutukset ajettiin logiikkasynteesin läpi, jossa niitä optimoitiin synteesin eri vaihtoehdoilla. Työmenetelmät olivat pääosin samat kuin käyttäytymistason synteesissäkin: komentotiedoston muuttamista ja raporttien tutkimista. Raporteista tarkistettiin ainakin ajoituksien onnistuminen prosessin minimi- ja maksimipoluille, resurssien määrä, prosessin koko ja oliko kaikki osat komponentoituneet.

Käännösvaiheessa pystytään tuottamaan erilaisia toteutusvaihtoehtoja ajamalla logiikkasynteesikomento (compile) erilaisilla kahvoilla. Optimoidut tulokset kirjoitettiin synteesityökalusta vhdl-muodossa simulointia varten.

Viimeisenä työvaiheena synteesissä on erillisten prosessien kokoaminen yhteen. Tämä voidaan tehdä lukemalla rakenne- ja syntetisoituja kuvauksia hierarkiatasoittain Synopsyksen Design Analyzerilla ja lopuksi poistamalla suunnitelmasta kaikki hierarkiatasot ja simuloimalla koko mallista kirjoitettua kuvausta.

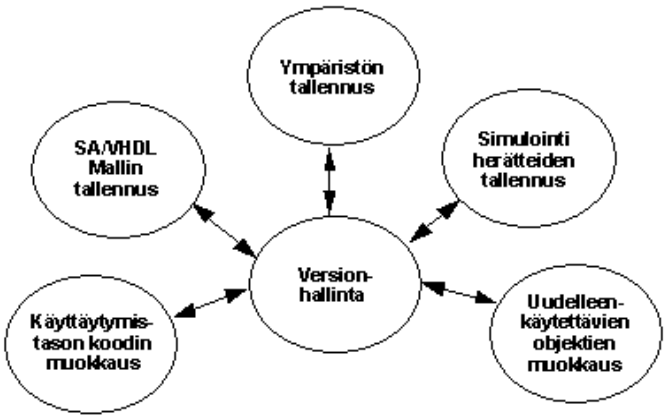
3.2 YLLÄPITOMENETELMÄT JA UUDELLEENKÄYTTÖMAHDOLLISUUDET

Versionhallintaa käytettäessä malli pidetään versionhallinnassa, josta sen saa muutettavaksi PVCS:n projektissa tai kansiossa määriteltyyn hakemistoon. PVCS:llä projektin tiedostoja muutettaessa ne pitää ensin ottaa versionhallinnasta ulos ja sen jälkeen tehdä muokkausoperaatiot ja lopuksi palauttaa tiedostot takaisin versionhallintaan.

Versionhallinnassa eri tiedostot voidaan kansioida (folder), jolloin esimerkiksi kansion sisältämien tiedostojen tuominen muokattavaksi tiettyyn hakemistoon onnistuu helpommin [19].

Projektissa voidaan määrätä, pidetäänkö työtiedostoista kirjoitussuojattu kopio työhakemistossa vai ei. Esimerkiksi SA-malli voidaan pitää koko ajan hakemistossa kirjoitussuojattuna ja syntetisoituvaa mallia tehtäessä valmis

mallin hierarkiataso viedään kokonaan versionhallintaan, jotta hakemistossa olevien tiedostojen määrä pysyy pienempänä ja suunnittelussa tarvitaan vain yksi hakemisto.



Kuva 11. Versionhallinnan mahdollisuudet.

Mallien ylläpitoon ja uudelleenkäyttöön saatiin apua PVCS-ohjelmasta. Kuva 11 esittää versionhallinnan mahdollisia käyttökohteita tässä työssä. Versionhallinnan avulla voidaan ylläpitää hyvin erilaisia tiedostoja suunnitteluprosessissa.

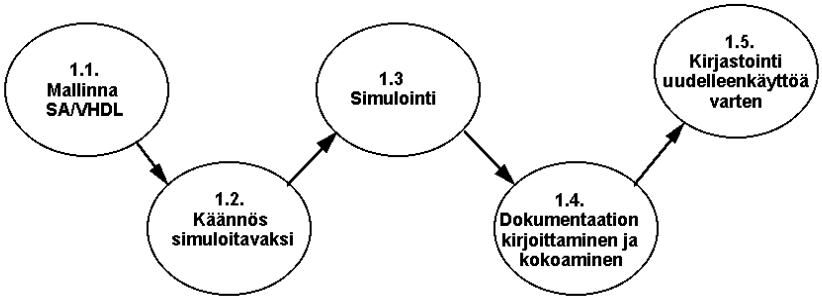
Simuloinnin päätteeksi testipenkit ja herätteet voidaan kansioida siten, että kunkin hierarkiatason eri herätteet on myöhemmin noudettavissa. Samalla herätteisiin voidaan liittää mukaan kuvaus, mitä herätteellä simuloitiin.

SA/VHDL-malli voidaan kokonaisuudessaan säilyttää yhdessä kansiossa. Syntetisoituvaa koodia muokatessa SA/VHDL-mallin käyttökelpoiset osat voidaan kopioida kansioihin tai siirtää uuteen versionhallintaprojektiin. Muokattava malli kannattaa tallentaa kansioihin hierarkiatasoittain. Tämän jälkeen voidaan hallitusti aloittaa hierarkian muokkaus alimmalta tasolta lähtien.

Uudelleenkäytettäviä malleja muutettaessa malli viedään ensin versionhallintaan luotuun kansioon ja muokataan mallia Prosalla, kunnes se voidaan liittää suunnitelman osaksi.

Työvuomallin ohjelma- ja ympäristöasetuksia voidaan tallentaa erilliseen versionhallintaprojektiin, josta ne voidaan tarvittaessa palauttaa.

Kuva 12 esittää uudelleenkäytettävän mallin tuottamisen työvaiheina. Mallit simuloidaan toiminnan tarkastamiseksi ja tämän jälkeen tuotetaan mallista dokumentaatio. Dokumentaatioksi voidaan liittää selitys mallin toiminnasta ja tarkoituksesta, rajoituksista, tiedot kirjoittajasta, simulaattorista ja mallin käyttökohteesta. Toiminta voidaan selittää esimerkiksi sanallisesti kirjastointityökaluun tai tietomuunnoksiin kommenttien muodossa. Toiminnassa pitäisi kertoa lähtö- ja tulovoiden merkitys ja käytetyt tietotyypit eri prosesseissa.



Kuva 12. Mahdollinen uudelleenkäyttötyöpolku SA-suunnittelussa.

Uudelleenkäytettävää mallia tuotettaessa käytetään luonnollisia ja kuvaavia nimiä mallin eri osissa. Malli piirretään myös huolellisesti, jotta vuot, tilat ja niiden nimet erottuvat kaaviosta selvästi.

Seuraavaksi esitetään luonnollisen nimeämisen kautta asioita, jotka vaikuttavat kaavioiden ymmärtämiseen ja luettavuuteen.

3.2.1 Luonnollisen nimeämisen soveltaminen SA/VHDL-suunnittelussa

Seuraavassa listassa on kokemuksia esiin tulleista ongelmista luonnollisilla nimillä suoritetusta SA/VHDL-suunnittelusta:

- Kuvissa käytettävien nimien on esiinnyttävä suunnitelmassa vain kerran,
- ylimmän tason tietomuunnoksien kuvaava nimeäminen on vaikeaa,
- pitkät nimet tarvitsevat lisätilaa kuvissa,
- luonnollisen nimen keksimiseen kuluu enemmän aikaa ja
- nimien vaihtaminen suunnitelmaan myöhemmin on työlästä.

Suunnittelussa samaa tai samankaltaista tietoa kuljettava vuo voi kulkea useaan eri muunnokseen, jolloin voiden nimeäminen voi olla ongelmallista. Tietomuunnoksen minispesifikaatiossa tuleva vuo tai signaali liitetään muuttujaan (variable) operaatioita varten. Operaation tulokset liitetään toiseen muuttujaan, joka minispesifikaation lopussa liitetään muunnoksesta lähtevään vuohon. Lisäksi jokaiselle vuolle määrätään sen sisältämä tietotyyppi. Suunnitelmassa itse määriteltyjä tietotyyppejä on yleensä runsaasti ja niiden nimeäminen lisää samankaltaisten asioiden nimien keksimisongelmaa.

Luonnollista nimeämistä voidaan käyttää esimerkiksi seuraavalla nimeämistyylillä:

1. Vuolle keksitään ensin kuvaava nimi: sanat kirjoitetaan yhteen ja suurilla alkukirjaimilla. Esimerkiksi UserTransmitRequest.
2. Jos vuolle määritellään oma tietotyyppi, niin Prosassa vuon tietotyypiksi annetaan vuon nimi kirjoitettuna yhteen tai yhdellä tai useammalla väliviivalla ja nimen loppuun liitetään *type*. Edelliselle vuolle tietotyypiksi tulee esimerkiksi *UserTransmitRequest_type*.

```
subtype primitive_type is bitvector(0 to 1);
subtype....
type UserTransmitRequest_type is record
        primitive       : primitive_type;
        data_identifier         : data_identifier_type;
        datalength      : datalength_type;
        datafield       : datafield_type;
end record;
```

Esimerkki 6. Tietotyyppien nimeäminen SA/VHDL-mallissa.

Esimerkissä 6 on ote CAN-kontrollerin ”common.vhd”-tiedostosta suunnitelman tietotyyppien määrittämisestä luonnollisella kielellä.

```
architecture behavior of sample is begin process
        variable User_Transmit_Request   :UserTransmitRequest_type;
        variable primitive : primitive_type;
        begin
                wait on  UserTransmitRequest'transaction;
                User_Transmit_Request  :=  UserTransmitRequest;
                primitive := User_Transmit_Request.primitive;
                ....
        end process
end behavior
```

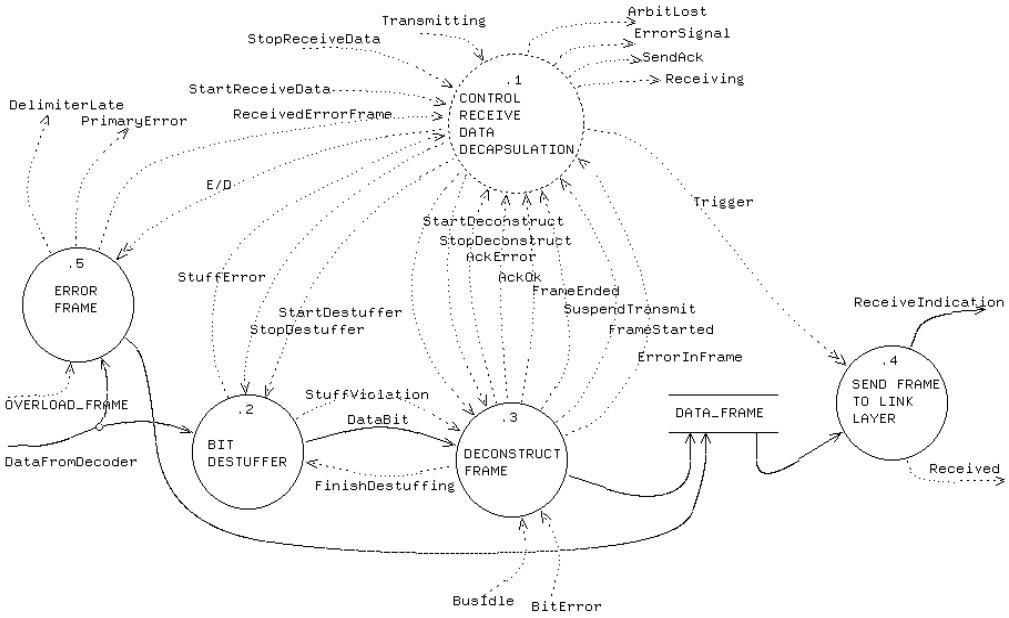
Esimerkki 7. Nimeämistä SA/VHDL-prosessissa..

Tietomuunnoksen minispesifikaatiossa muuttujien nimenä voidaan käyttää tietotyypistä ja vuon nimestä eroavaa nimeä kuten esimerkissä 7 on tehty.

3. Samaa tietoa kuljettavat vuot voidaan erotella esimerkiksi liittämällä nimeen vastaanottaja- tai lähettäjämuunnoksen nimeä tai niiden kuvaaman laitteen tai käsitteen nimeä kuten esimerkiksi ”ToBus” ja ”FromBus”.

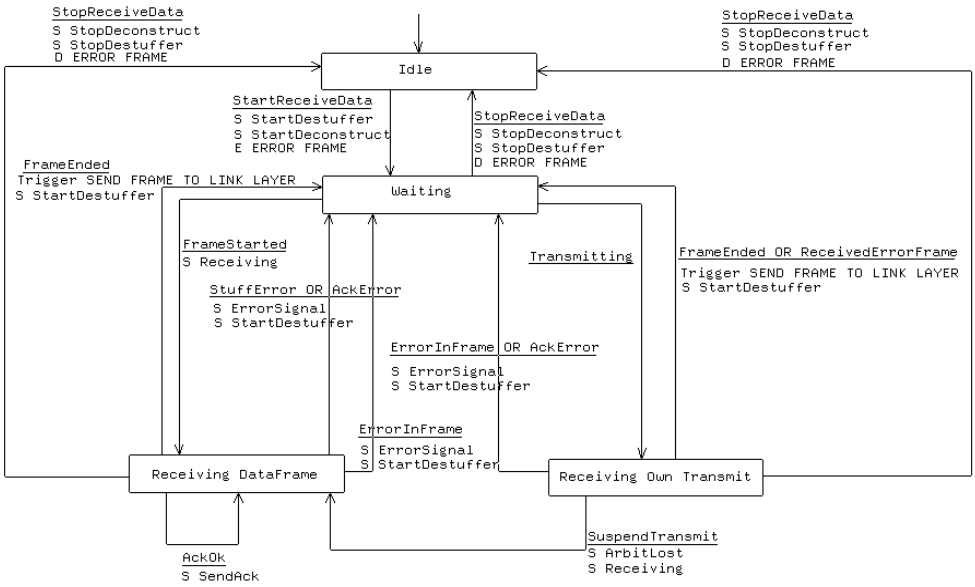
Kaavioiden koko SA/VHDL suunnittelussa on rajallinen: Velvetin avulla tehtävässä suunnittelussa suunnitelman on mahduttava 9 tietomuunnokseen [11]. Jokainen tietomuunnos voi tietysti sisältää 9 uutta muunnosta. Tästä on seurauksena, että ylimmällä tasolla muunnos voi sisältää useita erillisiä toimintoja, joten muunnokselle on joissakin tapauksissa vaikeata keksiä hyvä nimi. Nimeämistä voidaan helpottaa jakamalla suunnittelua korkeammalla tasolla kokonaisuuksiin: CAN-työssä esimerkiksi muunnos ”receive data decapsulation” sisältää pelkästään vastaanottoon liittyviä muunnoksia, joissa vastaanotetusta kehyksestä poistetaan lähetyksen ajaksi lisättyjä osia.

Tietomuunnosten välille tarvitaan runsaasti signaali- ja tietovoita. Näiden nimeäminen pitkiä, luonnollisia nimiä käyttämällä voi aiheuttaa vaikeasti luettavan kaavion. Samaan suuntaan kulkevia voita voidaan yhdistää yhdeksi vuoksi tilan säästämiseksi, mutta yhdysvuolle sisältöä kuvaavan nimen keksiminen voi olla vaikeaa ja toiminnan selvittäminen kuvasta hankaloituu. Muunnosten määrä kannattaakin pitää mahdollisimman alhaisena: 3 - 5 tietomuunnosta näyttävät sopivan hyvin yhteen kuvaan. Muunnosten määrää voi rajoittaa jakamalla toimintoja alemmille hierarkiatasoille.



Kuva 13. Nimien sijoittelu tietovuokaaviossa.

Nimien sijoittelulla voidaan parantaa kuvan luettavuutta (kuva 13). Jos signaaleja on piirretty paljon ja niiden nimet menevät useiden signaalien päälle, niin signaalin nimi asetetaan signaalin kohdalle. Lisäksi luettavuuden parantamiseksi on syytä piirtää samaan suuntaan menevät signaalit vierekkäin, sekä muunnokseen tulevat signaalit vasemmalle puolelle ja lähtevät oikealle. Kaaviosta ylemmälle tasolle menevien voiden nimet asetetaan esimerkiksi vuon päälle tai nuolen kärjen kohdalle kuten kuvassa 14 on tehty.



Kuva 14. Esimerkki tilakaavion nimien käytöstä.

Tilojen havainnollinen nimeäminen auttaa mallin toiminnan selvittämisessä, jos tilan nimestä näkee heti mitä toimintaa malli tiloissaan suorittaa (kuva 14). Esimerkiksi tilassa ”Receiving Own Transmit” CAN vastaanottaa omaa lähetystään.

Kaaviosta voi tehdä luettavamman, jos välttää ehtojen ja toimintojen sijoittamista tilansiirtymänuolen päälle (kuva 14). Tilansiirrot on pyritty asettelemaan siten, että tilasta toiseen menevät ja sieltä palaavat nuolet ovat kaaviossa vierekkäin: palaava nuoli oikealla ja paluuehto, sekä toiminnot heti nuolen oikealla puolella, samoin vasemmalla puolella tilasta lähtevä siirros ja sen ehto- ja toimintolauseet. Jos tilojen väliin jätetään lisää tilaa ja koko kaavion alue käytetään hyväksi, saadaan kuvasta entistä selvempi.

Pitkät nimet ehkäisevät tehokkaasti virheitä, joissa samaa nimeä on käytetty suunnitelman eri paikoissa. Nimikonfliktit aiheuttavat ylimääräisiä suunnittelukierroksia, kun suunnitelmaa aletaan simuloida.

3.2.2 Luonnollisten nimien tuottaminen SA-menetelmässä

Laajoissa suunnitteluissa on tärkeää eri osien kuvaava nimeäminen. Paljon tietovoita ja -muunnoksia sisältävässä suunnitelmassa alkavat lyhyet nimet sekoittua toisiinsa ja niiden muistaminen vaikeutuu. Suunnitelman toiminnan esittely siihen ennen perehtymättömälle on aikaavievää, jos jokaisen vuon ja signaalin merkitys joudutaan kertomaan erikseen. Kuvaavista luonnollisista nimistä voidaan päätellä suoraan kohteen käyttötarkoitusta.

Luonnollisten nimen tuottaminen SA:n eri osille on hieman erilaista. Seuraavaksi esitellään CAN-kontrollerin suunnittelussa käytettyjä nimeämisme-

netelmiä tietovoiden, signaalien, tietomuunnosten, tietovarastojen ja tilojen nimeämiseen.

Tietovuot

Tietovoiden nimeäminen on tärkeää vuon sisältämän tiedon tarkoituksen tai rakenteen ymmärtämiseksi nopeasti. Tietovuon nimen tuottaminen voidaan jakaa esimerkiksi seuraaviin osiin:

1. Tutki vuon sisältämää tietoa.

   Esimerkiksi CAN-kontrollerille menee käyttäjältä kaksi kehystä samaa vuota pitkin: Etäkyselykehys ja Datakehys. Näissä on myös useita samoja tietokenttiä. Luonteeltaan ne ovat pyyntöjä (request) kontrollerille, joka lähettää kehyksen, kun siihen on tilaisuus.

2. Tuota muutama tietoa kuvaava avainsana

   Esimerkiksi data, transmit, frames, requests, send, user

3. Muodosta sanoista nimiä ja valitse mielestäsi kuvaavin

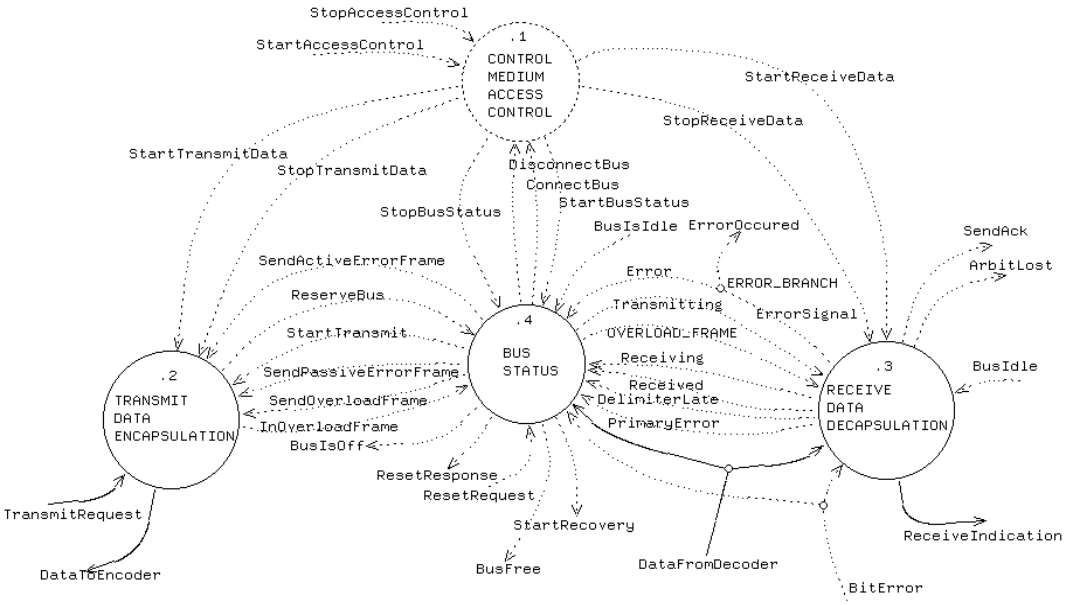
UserFrames	DataFrames
DataRequests	SendRequests
UserSendRequests	UserTransmitRequests
UserDataRequests	UserDataFrames

4. Onko nimi jo varattu vuon, tilan tai muunnoksen nimenä?

Signaalien nimeäminen

Tietovuokaaviossa kuvassa 15 on nimettyjä signaaleja paljon. Signaalit nimettiin käyttötarkoituksen mukaan. Niitä käytettiin tapahtumien käynnistämiseen, pysäyttämiseen, tapahtuman suorituksen onnistumisen ilmoittamiseen, virhetilanteista ilmoittamiseen ja niin edelleen. Signaalit voidaan jaotella esimerkiksi ohjaussignaaleihin tai ne voivat välittää tietoa jostakin tapahtumista.

Poikkeuksena ovat muunnoksien aktivointi- ja pysäytyssignaalit: ne nimettiin alimmalla tasolla E/D ja ylemmillä tasoilla Start<muunnoksen nimi> ja Stop<muunnoksen nimi>.



Kuva 15. Esimerkkejä signaalien nimistä.

Signaalien nimeämisen perusaskeleet:

1. Tutki signaalin tarkoitusta: Onko informatiivinen vai ohjaussignaali?
2. Tuota nimiehdotuksia.
3. Valitse tarkoitusta kuvaava ja yksilöllinen nimi.

Esimerkiksi signaalit SendActiveErrorFrame ja ReserveBus ovat ohjaustyyppisiä ja Receiving, Transmitting ja BusIsOff informatiivisia (kuva 15).

Tietomuunnokset

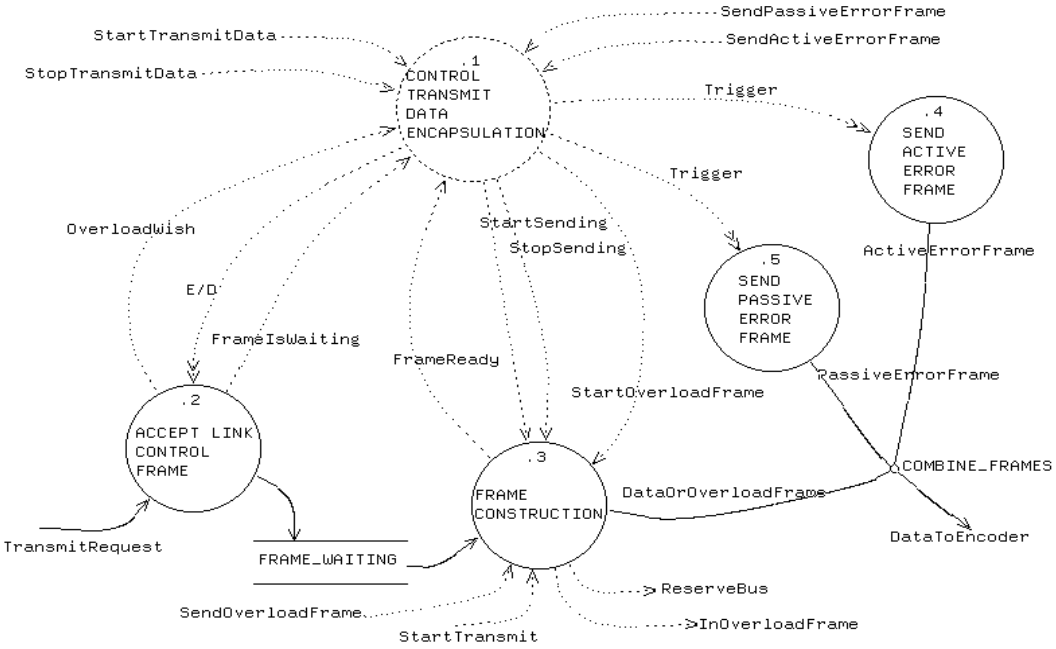
Velvetillä käännettävään tietomuunnokseen saa mennä 1 aktivoiva vuo ja siitä voi lähteä tarvittava määrä voita ja signaaleja. Tietomuunnoksella on jokin tehtävä, jonka se tekee tulevien tietovoiden sisältämälle tiedolle. Tietomuunnos voi myös koota useita tietomuunnoksia yhteen.

Tietomuunnosten nimeäminen ei ole ongelma, jos nimen voi tuottaa muunnoksen tekemästä toiminnasta. Esimerkiksi kontrollerissa on spesifikaation esittämä bittivirheen havaitsemismenetelmä kuvattu tietomuunnoksessa Bit Error Detector, jossa väylälle kirjoitettua bittiä verrattiin väylältä luettuun bittiin ylikirjoituksen eli virheen havaitsemiseksi.

Muunnoksia kokoavien muunnosten nimeäminen voi avautua muunnosten toimintojen yhteisestä tekijästä: Miten toimintoja kuvataan yhteisellä nimellä tai minkä ylemmän tason toiminnon muunnokset suorittavat? CAN-kontrollerissa on nimiä haettu myös OSI-jaottelusta, joka oli esitelty sen spesifikaatiossa.

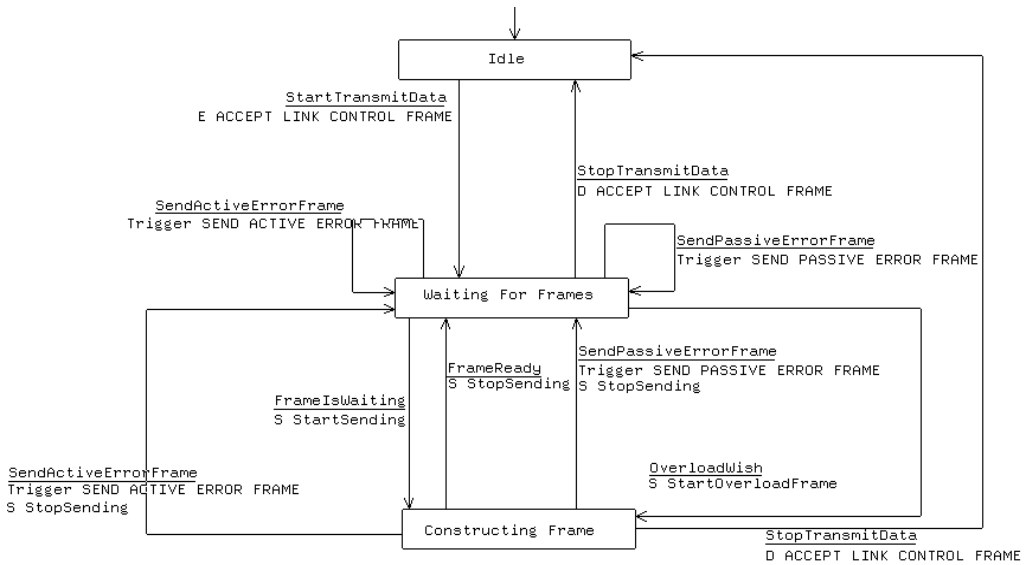
Tietovarastot ja puskurit

Tietovarastojen ja puskureiden nimeämisessä sääntönä voidaan pitää nimen tuottamista tietovaraston sisältämästä tiedosta. FRAME_WAITING-tietovarastossa oleva kehys odottaa väylän vapautumista lähetystä varten (kuva 16). Muunnoksen numeroinnista johtuen, tästä muunnoksesta numero 2 lähtevän vuon nimeksi tulee FRAME_WAITING_322.



Kuva 16. Tietovarastojen nimeäminen.

Tilakoneet



Kuva 17. Tilojen nimeäminen edelliselle kaaviolle.

Tilojen nimeäminen on tärkeää synteesivaiheessa tilarakenteiden muokkauksen helpottamisessa. Tilan nimi voisi kuvata tilassa tapahtuvaa toimintaa, jota tilassa aktiivisina olevat tietomuunnokset suorittavat tai ilmaista pelkästään tilassa odotettavaa tapahtumaa. Kuva 17 esittää, että tietovuokaaviossa on 2 päätilaa. Toisessa odotetaan kehystä ja toisessa kehys valmistetaan lähetettäväksi. Tilakoneen tila, jossa mitään toimintaa ei tapahdu voidaan nimetä ”Idle”:ksi eli alkutilaksi. Eri tilakoneissa olevat samannimiset tilat eivät aiheuta nimikonfliktia.

Nimitaulukot

Taulukoihin 2 ja 3 on koottu muutamia datavoiden, signaalien ja tietomuunnosten avainsanoja. Nimitaulukoiden avulla voidaan nopeuttaa luonnollisten nimen keksimistä suunniteltaessa.

Taulukko 2. Avainsanoja CAN-työssä käytetyistä nimistä

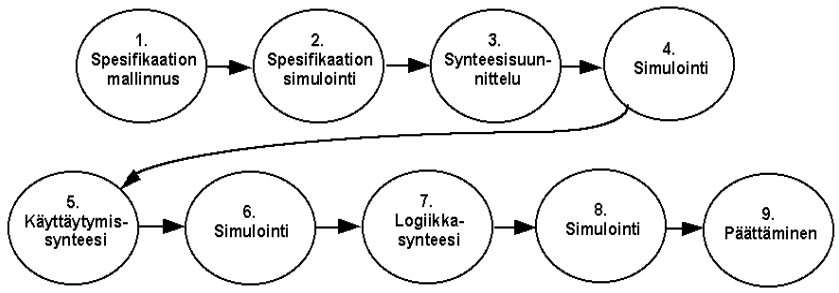
Lähetys / Vastaanotto	Tietomuunnokset	Tietovuot
Acknowledge	Accept	Bit
Confirm	Calculate	Data
Connect	Check	Field
From	Compare	Frame
Get	Construct	Information
Receive	Counter	Packet
Recovery	Decoder	Reference
Request	Decrease	Section
Send	Detect	Status
Syncronize	Encoder	Message
Take	Increase	Sequence
To	Status	Length
Transfer	Store	
Transmit	Stuffer	

Taulukko 3. Erilaisten signaalien avainsanoja

Ohjaus	Informatiivisia	
(Dis)Connect	Finished	Done
Receive	Receiving	Error
Request	Started	Indication
Reserve	Succeeded	Lost
Send	Transmitting	Ready
Start Stop	Changed	Response
Suspend	Waiting	
	Wish	

3.3 TYÖVUON SUUNNITTELU

Valmistettava työvuo käsittää kuvassa 18 esitetyt työvaiheet. Työvaiheiden järjestys ei aina ole kuvasta huolimatta peräkkäinen, vaan esimerkiksi vaiheista 5 ja 7 voidaan palata muokkausvaiheeseen. Samoin simuloinneista palataan useammin edelliseen vaiheeseen kuin jatketaan seuraavaan.



Kuva 18. Synteesiprosessin työaskeleet kokonaisuudessaan.

Jokainen työvaihe piirretään omaan kaavioonsa, jolloin ne voidaan testata helpommin erikseen ja liittää yhdellä ylimmän tason kaaviolla toisiinsa. Osin simuloinneista kannattaa tehdä hierarkkiset kaaviot, koska niissä kuitenkin tehdään samat työvaiheet.

Työvuossa syntyvät tiedostot muokataan pelkästään UNIX-ympäristössä. Tällä vältetään mahdolliset yhteensopimattomuudet tekstitiedostoissa. Esimerkiksi Prosalla mallien muokkaus sekaisin eri ympäristöissä ei onnistu.

Mallissa käytetään apuna tarkistuslistoja, joissa luetellaan työvaiheisiin liittyviä tärkeimpiä tehtäviä ja tarkistettavia kohtia. Tarkistuslistaprosessit toteutetaan prototyyppiohjelmalla nimeltään ”Dmmtable”. Ohjelma näyttää sillä tallennetut listat, joihin käyttäjä ruksaa tehdyt työvaiheet [24]. Ohjelmalla voidaan estää työvuossa eteneminen, jos kaikkia työvaiheita ei ole kuitattu.

3.3.1 SA/VHDL-suunnittelun työvaiheet

SA/VHDL-suunnittelussa käytettiin Prosaa ja sen avulla ajettavaa tekstieditoria. Työskentelyn joustavuuden helpottamiseksi Prosan käynnistysprosessi voidaan asettaa aina käynnistettäväksi.

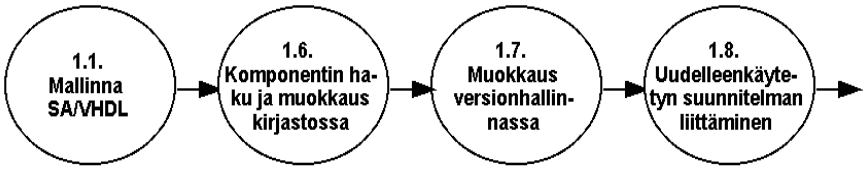
Prosan jälkeen työvuo käännetään Velvetillä, joka tarvitsee tiedostonimen, josta käännös aloitetaan. Lisäksi Velvet vaatii käyttäjältä vastauksen herätetiedoston ylikirjoittamiseksi. Velvetin jälkeen muokataan, joko uutta herätetiedostoa tai muutetaan edellisen suunnittelukierroksen herätteitä.

Herätteiden tallentamiseksi ja hakemiseksi tarvittiin versionhallinnan käynnistävä prosessi. Kun herätetiedosto on valmis, malli testataan simuloimalla, jonka jälkeen palataan takaisin SA-suunnitteluun. Simuloinnin jälkeen teh-

tävät muutokset voivat olla pieniä, jolloin simulointiprosessista tehdään sellainen, että työkalua ei ole pakko sulkea simuloinnin jälkeen ja sitten käynnistää jokaisella ”Prosa-Velvet-simulointi”-kierroksella. Useita simulointeja voidaan suorittaa samoilla herätteillä, joten herätteiden muokkausprosessista tehtiin myös valinnaisesti ajettava.

Koska versionhallintaa ja Prosaa tarvitaan satunnaisesti, vuohon tehtiin aina ajettavissa oleva prosessi pelkästään työkalujen ajamiseksi, jossa suoritettava työkalu valitaan kyselykaavakkeesta.

Uudelleenkäytettävän mallin tuottaminen tehdään simuloinnin jälkeen. Mallin muokkaamiseen ja dokumentointiin tarvittiin erilliset prosessit. Mallia voidaan muokata mm. lisäämällä kommentteja koodiin ja muuttaa koodia luettavammaksi. Vasta uudelleenkäytettävän mallin ollessa liitehakemistossa voidaan kirjastointiohjelman käynnistävä prosessi ajaa ja tallentaa uudelleenkäytettävälle suunnitelmalle liityntä- ja toimintadokumentaatio.



Kuva 19. Uudelleenkäytettävän komponentin haku kirjastosta.

Työvuon rakenteessa oli huomioitava, että uudelleenkäytettävien mallien hakeminen kirjastosta pitää pystyä ajamaan milloin tahansa SA/VHDL-mallinnuksen aikana. Tällöin kuvassa 19 esiteltyjen työvaiheiden on oltava käynnistettävissä aina kun SA-suunnittelu on aktiivinen. Kuvan 19 vaihe 1.6 voidaan aloittaa kirjastointiohjelman käynnistyksellä. Ohjelmalla muutetaan voiden nimet ja hierarkian tasonumerot suunnitelmaan sopivaksi. Kun uudelleenkäytettävä malli on uudelleenkäyttöhakemistossa, se tallennetaan versionhallintaan. Tämän jälkeen käynnistetään toinen Prosa, jolla suoritetaan mallin muokkaus erillään varsinaisesta suunnitelmasta. Lopuksi, jos malli halutaan siirtää suunnitelmaan, se tallennetaan ensin suunnitelman versionhallintaprojektiin ja sen jälkeen muokattavaksi suunnitelman hakemistoon.

Kaavioon tarvitaan kaksi tarkistuslistaprosessia: toinen ennen käännöstä SA-mallin varmentamiseksi ja toinen uudelleenkäytettävän mallin valmistamisen yhteyteen. SA-mallin tarkistuksia ja vaatimuksia voidaan kuvata tarkistuslistaan, jolloin SA-mallin tekemiseen saadaan enemmän kontrollia. Ennen käännöstä tehdään vielä seuraavat tarkistukset Prosan avulla. Tarkistetaan, onko kaikille voille annettu tietotyypit ”dat”-raportin avulla. Samalla tarkistetaan, onko kaavioissa virheitä ”balance”-raportin avulla. Varmistetaan myös, onko tarvittavat tietotyypit esitelty ”common.vhd”-tiedostossa. Uudelleenkäytettäessä varmistetaan, että malli on tallennettu liitehakemistoon, dokumentaatiossa mainitaan vaaditut asiat ja tietotyypit ovat mukana.

SA-mallin valmistuttua se siirretään kokonaan versionhallintaan ja revisiot merkataan esimerkiksi versioksi 1.0 (version label). Versionumeron avulla spesifikaatio on myöhemmin haettavissa.

3.3.2 Synteesisuunnittelun työtehtävät

Synteesisuunnittelussa tarvitaan SA-kuvauksesta tilakone-, arkkitehtuuri- ja rakennekuvaukset sekä suunnitteluyksiköiden esittelyt. Koottu malli jaetaan hierarkiatasoittain eri kansioihin muokkausta varten. Näin jaotellut hierarkiat on helpompi tuoda versionhallinnasta kansioon määriteltyyn muokkaushakemistoon. SA-simuloinnissa syntyneet hajallaan olevat herätteet voidaan siirtää ko. tason kansioon.

Muokkauksen apuna tarvitaan SA-kuvaus, josta voidaan palauttaa mieleen muokattavan tason toiminta. Tämän takia Prosan käynnistävä prosessi asetettiin valinnaisesti ajettavaksi.

Työvuo aloitetaan versionhallintaprosessilla, josta tuodaan työtiedostoja muutettavaksi. Sen jälkeen ajetaan muokkausprosessi versionhallinnasta otetuille tiedostoille. Itse muokkaus voidaan toteuttaa UNIX:n tiedostonhallinnalla (filemgr), jonka avulla käyttäjä voi sujuvasti avata muokattavat tiedostot.

Analyysivaiheessa käyttäjällä on valittavana tilakone- ja prosessikuvauksille erilliset komentotiedostot, niihin asetetaan analysoitavan suunnitteluyksikön nimi ja prosessissa käytetyn kellosignaalin nimi. Komentotiedostossa kellosignaalin jakso on asetettu alkuarvoltaan 50 nanosekuntiin, jolla suurin osa prosesseista voidaan analysoida.

Komentotiedostoille tarvitaan muokkaus- ja ajoprosessit, näiden käynnistyessä valitaan kumpaa tiedostoa muokataan. Muokkaus tehdään tekstieditorilla ja komentotiedostot ajetaan työkalulla.

Analyysin tuloksena saadaan tiedosto työkalun komentojen tulostuksista. Näistä tarkastetaan, ettei virheitä tapahtunut tai jos tapahtui, mitä pitää korjata. Usein tarvittava koodin muutos on pieni raportin osoittamaan kohtaan, joten vuosta tehtiin sellainen, ettei käyttäjän tarvitse hyppiä kahden prosessin välillä.

Käyttäjälle tulokset ja analyysiajon kulku näytetään X-ikkunassa. Raportit näytetään komentotiedostosta käsin, koska sinne on määritelty, mihin tiedostoon komentojen tulostus on ohjattu. Jos raporttitiedoston näyttäminen toteutettaisiin vuosta käsin, tarvittaisiin yksi prosessi lisää, jolle annettaisiin parametrinä näytettävän raporttitiedoston nimi. Toisaalta työskentelyssä voidaan tarvita tämänkaltaista raporttien selailu- ja poistomahdollisuutta. Nämä voidaan tehdä PVCS:llä tai tiedostonhallinnalla.

Analysointiprosessin jälkeen voidaan siirtyä koodin muokkausprosessiin tai simuloida muutettua koodia. Koska analysointiparametreja ei tarvitse jokaisella analyysikierroksella muuttaa, niin komentotiedoston muokkausprosessin ajamisesta tehdään valinnaista. Kun kaikki tilakone- ja prosessitiedostot on analysoitu, päivitetään rakennekuvauksen komponenttikuvaukset vastaamaan suunnitteluyksikön esittelyä simulointia ja suunnitelman kokoamista varten.

Tarkistuslista tarvittiin ainakin muokkauksen yhteyteen. Muokattaessa tarkistetaan, että prosessin osat on nimetty oikein, suunnitteluyksikön esittely on siirretty samaan tiedostoon arkkitehtuurin kanssa, suunnitelma on synkroninen, alustussilmukassa alustetaan signaalit ja muuttujat, signaaleihin ei kirjoiteta kahdesti kellojaksossa jne.

3.3.3 Käyttäytymistason synteesin työtehtävät

Synteesin tarkoituksena on etsiä käyttäytymistasolla suunnitelman prosessien erilaisia toteutusvaihtoehtoja. Toteutusvaihtoehdot voidaan tuottaa asettamalla suunnitelmalle rajoituksia.

Suunnitelmat syntetisoidaan ja rajoitukset asetetaan työkaluun sopivan komentotiedoston kautta. Käyttäjä asettaa synteesiä varten vähintään seuraavia tietoja: suunnitteluyksikön nimen, kellosignaalin nimen, kellojakson, maksimimäärän kellojaksoja eri operaatioiden välille tai prosessissa oleville silmukoille ja synteesin kestoa ja laatua ohjaavan parametrin (effort).

Tässä voidaan käyttää samaa menetelmää kuin analyysitasossakin eli työvuohon sijoitetaan komentotiedoston muokkaus- ja ajoprosessit. Suunnitelman analyysivaiheessa syntyneet synteesitietokannat sijaitsevat muokkaushakemistossa, josta työkalu käy ne lukemassa. Työvuossa on oltava mahdollista ajaa useita skedulointeja erilaisilla parametreilla ja selata eri ajojen tuottamia raportteja.

Synteesiajon tuloksien perusteella käyttäjä määrää, mitä tehdään. Vaihtoehtoja ovat esimerkiksi uusi skedulointi eri parametreillä, nykyisen tiedoston koodin muokkaaminen ja analyysin kautta suunnitelman tuonti uudelleen skeduloitavaksi tai simulointiin siirtyminen.

Synteesissä käsitellään nyt vain prosessikuvaukset, ei tilakonekuvauksia. Synteesin tuloksena saadaan raportit suunnitelman skeduloinnista, arvio pinta-alasta ja nopeudesta. Komentotiedosto kirjoittaa valmiin suunnitelman tietokannaksi logiikkasynteesiä varten ja vhdl-muodossa simulointia varten.

Simulointiprosessiin siirrytään, kun kaikki suunnitelman hierarkiatason prosessit on syntetisoitu onnistuneesti. Simuloidessa käytetään analysoitaessa tuotettuja rakennekuvauksia ja herätteitä, joihin ei tarvitse tehdä muita muutoksia kuin asettaa komponenttikuvaus osoittamaan tuotettuun syntetisoituun vhdl-kuvaukseen.

Työskentely nopeutuu, jos synteesiprosessien aikana voidaan esimerkiksi muokata seuraavan hierarkiatason koodia. Työvuota suunnitellessa tähän oli muutamia vaihtoehtoja, kuten työvuon haaroittaminen, työvuon prosessin päättymistä ei odoteta tai työkalut käynnistetään siten, että DMM-työkalu tulkitsee ne välittömästi suoritetuksi. Haaroittaminen on eri ratkaisuista yleensä paras.

3.3.4 Logiikkasynteesin työtehtävät

Logiikkasynteesissä käytettävät komentotiedostot lukevat skedulointivaiheen tietokantatiedoston ja kirjoittavat synteesin tuloksen raportteineen. Synteesiä varten komentotiedostoon asetetaan syntetisoitavan suunnitelman nimi, kellosignaalin nimi ja Synopsyksen synteesikomennon parametreja ja muita synteesiin liittyviä rajoituksia.

Työvuossa tarvitaan prosessit komentotiedoston muuttamiselle, ajamiselle, tiedostojen selailulle ja versionhallinnalle. Samoin kuin käyttäytymistasollakin. Simuloitaessa logiikkatason suunnitelmia käytetään samaa työvuota kuin RTL-tasollakin.

Logiikkasynteesissä käytetään kahta komentotiedostoa, joilla analyysivaiheen tilakoneet ja skeduloidut suunnitelmat optimoidaan. Synteesiä varten asetellaan tarvittavat parametrit tiedostoihin ennen niiden ajamista. Tuloksista tarkastellaan onnistuminen ja päätetään seuraavaan työvaiheeseen siirtyminen. Synteesin jälkeen pitäisi olla yhteys synteesisuunnittelukaavioon, käyttäytymistason kaavioon, uuteen synteesiin ja suunnitelman kokoamiseen.

Synteesivaihe voi kestää kauankin prosessin koosta riippuen. Synteesin aikana voidaan esimerkiksi valmistella uutta synteesiajoa, simulointiherätteitä tai muokata toista hierarkiatasoa.

3.3.5 Suunnitelman kokoaminen ja loppusimuloinnit

Tässä vaiheessa suunnitelman VHDL prosessit on käsitelty erillään, jotta niiden syntetisointi olisi nopeampaa ja suunnitelmassa säilyisi yhteys SA-spesifikaatioon.

Kun prosessit on syntetisoitu, ne voidaan yhdistää yhdeksi suunnitelmaksi syntetisointityökalulla. Yhdistely aloitetaan alimmalta hierarkiatasolta. Kaikki tason synteesitiedostot luetaan Design Analyzerilla. Tämän jälkeen luetaan tason rakennekuvaus, jolloin työkalu muodostaa irrallisista prosesseista hierarkkisen kuvauksen. Nyt hierarkia voidaan poistaa, kun suunnitelmasta valitaan kaikki osat ja ajetaan valikosta ungroup-komento. Tämän jälkeen voidaan poistaa kaikki muut paitsi ylimmän tason kuvaus. Seuraavaksi voidaan jatkaa lukemalla sisään seuraavat hierarkiatasot.

Jokaisella tasolla voidaan ajaa tarkistus, ettei mitään jäänyt huomaamatta. Lopuksi jäljelle jää suurehko kuvaus, jota voidaan yrittää vielä optimoida laiterajoitusten puitteissa. Suunnitelmasta voidaan tuottaa vhdl-kuvaus, jota vielä simuloidaan edellisen vaiheen ylimmän tason herätteillä.

Lisäksi työvuossa tarvitaan versionhallintaprosessia, työkalua tiedostojen siirtelemiseksi ja selailemiseksi, synteesityökalua ja liityntää simuloitiin.

3.3.6 Simulointien suunnittelu

SA-suunnittelussa simulointiprosessit sijoitetaan samaan kaavioon, koska silloin saadaan liityntä kätevimmin uudelleenkäyttöprosesseihin. Synteesisuunnittelussa käytetään samoja simulaattoreita kuin SA-tasolla. RTL- ja porttitasolla ja integroinnissa on käytettävissä vain Synopsys VSS.

Kaikilla tasoilla tarvittiin prosessit herätteiden tallentamiseen ja etsimiseen versionhallinnan avulla. Valittujen herätetiedostojen muokkausta voidaan tehdä yhtä aikaa simulaation ollessa käynnissä. Kun simulointi on suoritettu suljetaan molemmat prosessit ja herätteet voidaan tallentaa. Versionhallinnasta tehdään valinnaisesti ajettava, koska se on voitu sulkea tai se on jo päällä simulointiin siirryttäessä.

SA-tasolla simuloinnista palataan SA-mallin muokkaukseen, uudelleenkäytettävän komponentin suunnitteluun tai edetään synteesisuunnitteluun. Synteesisuunnittelutason simuloinnista palataan esimerkiksi koodin muokkaukseen, versionhallintaan tallentamaan toimiva suunnitelma tai edetään synteesissä eteenpäin.

Synteesi- ja integrointikaavioissa simulointi suoritetaan Synopsys VSS -simulaattorilla, joka löytää simuloinnissa tarvittavat kirjastot ja on yhteensopiva ”dc_shell”:n tuottaman koodin kanssa.

VSS-simulaattoria varten määritellään ylin (top) rakennekuvaus, jonka nimi (tässä ”simulate”) tarvitaan kutsuttaessa simulointityökaluja. Rakennekuvaus sisältää vain komponenttiviittauksen testipenkkiin, joka sisältää kaikkien simuloinnissa tarvittavien komponenttien esittelyt.

Simulointi VSS:llä tapahtuu kolmessa vaiheessa. Ensin vhdlan-nimisellä työkalulla analysoidaan kaikki simuloinnissa tarvittavat tiedostot mukaanlukien ylin rakennekuvaus tietyssä järjestyksessä. Tämän jälkeen voidaan käynnistää vhdldbx-työkalu, joka lukee analysoinnin tuottamat tiedostot. Sillä voidaan valita seurattavat signaalit ja ajaa halutun mittainen simulointi.

4 MALLIN TOTEUTUS

Työvuomallia tehtäessä oli ratkaistava muutamia ongelmia, kuten työkalukomentojen vieminen NT:stä UNIX:iin ja niiden käynnistäminen oikeassa hakemistossa, parametrien välittäminen komennon mukana, komennon aikaansaaman palautteen näyttäminen työvuon käyttäjälle, hakemistorakenteen organisoiminen ja vuon siirrettävyyden mahdollistaminen.

Työvuossa komentojen ajaminen toteutettiin seuraavanlaisella tavalla. Pääosin NT:ssä olevasta mallista ohjattiin UNIX-työkalujen käynnistys tiettyyn UNIX-hakemistoon, jossa näillä työkaluilla operoitiin. Joihinkin komentoihin oli liitettävä useitakin parametreja. Näiden parametrien antamiseksi oli kaksi mahdollisuutta, joko koko komentorivi rakennettiin NT:ssä kyselykaaviossa ja lähetettiin sieltä UNIX:ssa ajettavaksi ”exceed”-ohjelman kautta tai toisena mahdollisuutena oli lähettää UNIX komentona alias, joka sisälsi kaikki työkalun tarvitsemat komennot.

Komennon muodostaminen tehtiin jälkimmäisellä tavalla, koska DMM-työkalun tällä versiolla ei pystytty tuottamaan tarvittavia komentorivejä NT:ssä ”exceed”:llä ajettavaan muotoon. Asiaa puolsi myös se, että nyt tarvittavat hakemisto- ja aliasmäärittelyt voitiin tehdä UNIX:ssa ja siksi niiden suunnitelmakohtaisesta muuttamisesta tuli helpompaa verrattuna siihen, että ne olisivat työvuokaavion ”sisällä”.

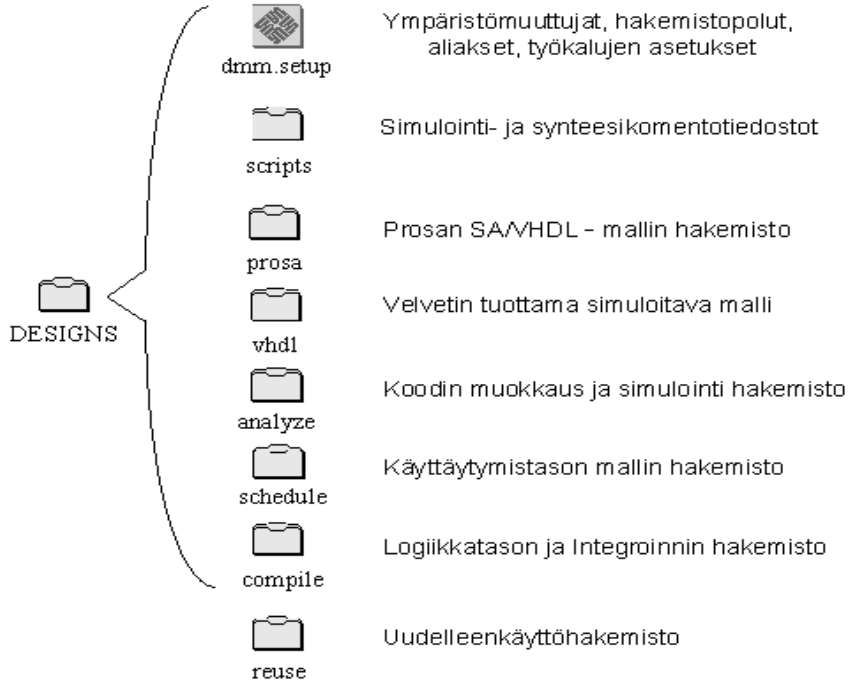
Haittapuolena tässä on, että prosessia ajettaessa ei voida kysyä mitään muuttuvaa tietoa, esimerkiksi tiedoston nimeä, koska sitä ei saada välitettyä UNIX:ssa olevaan komentotiedostoon tai työkalukomentoon. Tämä johtuu siitä, että exceed:n ”xstart”-ohjelma, joka mahdollistaa X-ikkunoiden avaamisen NT:ssä, vaatii UNIX:lla ajettavan komentorivin lainausmerkkien sisään [25]. Dmmexec:n ja kyselykaavakkeen avulla tätä ei voitu tehdä ohjelmassa olevan virheen takia. Siksi päädyttiin asettamaan esimerkiksi synteesin tarvitsemat parametrit suoraan komentotiedostoihin.

Suunniteltaessa syntyvät tiedostot pidetään UNIX-levyllä seuraavaksi esiteltävässä hakemistorakenteessa ja työkaluja kutsuttaessa siirrytään kutsun yhteydessä suoraan kyseessä olevaan hakemistoon myöhemmin esiteltävien aliaksien ja ympäristömuuttujien avulla.

Kuva 20 esittää työvuossa käytettyä hakemistorakennetta. ”Scripts”-hakemisto sisältää synteesissä ja simuloinnissa tarvittavia komentotiedostoja ja synteesissä käytettävän asetustiedoston ”.synopsys_dc.setup”:n.

”Dmm.setup”-tiedosto (Liite A) sisältää mallinnetun työvuon tarvitsemia määrittelyjä: UNIX-hakemistopolkuja, ympäristömuuttujia ja työvuosta kutsuttavat aliasmäärittelyt. Hakemistopolkuja käytettiin osoittamaan suunnitteluhakemistoon BC:n komentotiedostoissa ja siirtymään oikeaan hakemistoon työkalujen käynnistysaliaksissa. Jokaisessa alias-kutsussa DISPLAY-muuttuja asetetaan osoittamaan työasemaan. Tiedostossa on

myös Prosan, Synopsys BC:n ja muiden ohjelmien tarvitsemia ympäristömuuttujia.

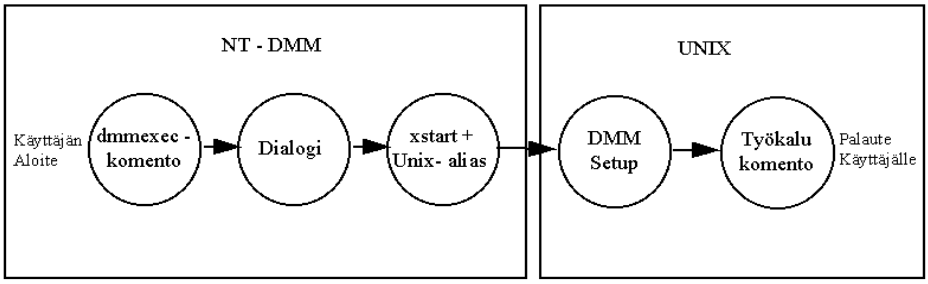


Kuva 20. Työssä käytetty hakemisto- ja tiedostorakenne.

Valmiiksi ”dmm.setup”:ssa on asetettu seuraavat ympäristömuuttujat hakemistopolkuja varten: DESIGNS, SADIR, SASIMULATE, ANALYZEDIR, REUSE_DIR, BEHAVIOUR_CODE, SIMULATEDIR, ja LOGIC_CODE.

DESIGNS on suunnitelmakohtainen hakemistopolku suunnitelmahakemiston juureen. Oletusarvona kaikki muut suunnitelman hakemistot sijaitsevat tässä hakemistossa. SADIR-muuttuja määrittelee SA/VHDL-mallinnusta varten ”prosa”-hakemiston DESIGNS-hakemistoon. SASIMULATE-määrittelee SA/VHDL-mallin simulointihakemiston. REUSEDIR osoittaa samaan hakemistoon, joka on määrätty kirjastointiohjelmassa liitehakemistoksi. ANALYZEDIR-hakemistossa muokataan syntetisoituvaa koodia ja SIMULATEDIR taas osoittaa synteesissa syntyvien ja analyysivaiheen vhdl-kuvauksien simulointihakemistoon. Oletusarvoiltaan ANALYZEDIR ja SIMULATEDIR osoittavat samaan hakemistoon, joten analyysivaiheessa suunnitelmaa ei tarvitse kopioida käsin toiseen hakemistoon.

Käyttäytymistason malli ja logiikkatason malli sijaitsevat omissa hakemistoissaan: ”schedule” ja ”compile”. PVCS:ää ja VHDL2000:tta käytettäessä joudutaan projektien työhakemistot asettamaan oikeiksi työkaluissa. Hakemistonimet ovat tietenkin oletusarvoja, jotka voidaan haluttaessa muuttaa.



Kuva 21. Työkalukomennon muodostuminen työvuosta.

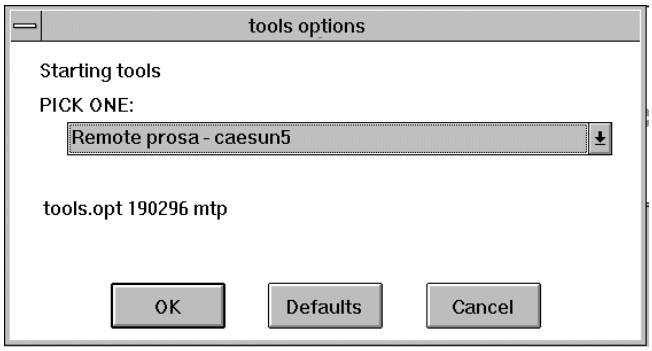
Kuvassa 21 havainnollistetaan komennon muodostumista tehdyssä työvuomallissa. Esimerkiksi Prosan avaaminen suunnitelman ”prosa” hakemistoon etenee seuraavasti. Työvuossa olevassa prosessissa on määritelty seuraava komento (dmmexec -oe -w -i=%o -p=”StartTools” ”tools.exe” ”ok” ”ok”), jossa ”-oe”-kahvalla luetaan valintatiedosto, ”-w”:llä määrätään, että ohjelma avaa oman ikkunan, jolloin ohjelma ei avaa erillistä ikkunaa, ”-i=%o”:llä määrätään komennon kohdehakemistoksi vuohon liitetty hakemisto, ”-p”-kahva ilmoittaa ajettavan prosessin nimen ja ”tools.exe” on ajettavan ohjelman nimi, jonka jälkeen luetellaan vähintään 2 prosessin lopetusviestiä. Ohjelmaa ei tarvitse löytyä, se vain ilmaisee valintatiedoston nimen.

Edellä oleva prosessin komentorivi näyttää käyttäjälle kyselykaavakkeen (kuva 22) perustuen seuraavaan ”tools.opt”-tiedoston sisältöön (esimerkki 8).

```
TEXT: "Starting tools"
CHOOSEONE: "PICK ONE" REQUIRED
"PVCS version control" TRUE=1="EXECUTABLE:pvcs.exe" FALSE="
 dcaesun5.xs -c runprosa&"
"Remote prosa-caesun5" TRUE=2="EXECUTABLE:xstart.exe"
```

Esimerkki 8. Tools.opt tiedoston sisältö.

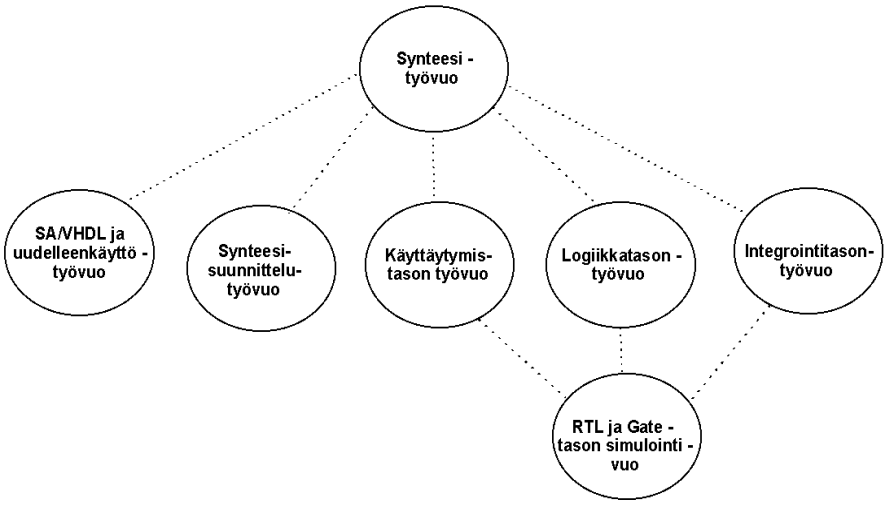
Kun käyttäjän valitsee kaavakkeesta Prosan, komentorivi muutetaan suorittamaan ”xstart dcaesun5.xs -c runprosa” -komento, jonka parametreina annetaan yhteyden muodostamista varten ”.xs”-loppuinen tiedosto ja komentona UNIX-alias. Komento käynnistää UNIX:ssa komentotulkin, joka lukee ympäristöasetukset ja ”dmm.setup”-tiedoston ”source”-komennon avulla ”.cshrc”-tiedostosta käsin. UNIX suorittaa ”runprosa” aliaksen ja ajaa komennot (set_display;cd $SADIR;/usr/prosa/prosax context.dfd) käynnistäen Prosan oikeaan hakemistoon.



Kuva 22. Kaavakkeen muodostuminen kuvauksen avulla.

Työvuossa olevat noin 18 kyselykaavaketta ovat rakenteeltaan samankaltaisia kuten kuvassa 22. Osassa on myös mahdollista valita toisella valikolla käytettävän UNIX-palvelimen nimi.

Kuva 23 esittää valmistetun työvuomallin hierarkian. Tällaisen hierarkiarakenteen avulla kaavioiden koko pysyi suhteellisen pienenä ja kaavioiden testaus oli helpompaa verrattuna tilanteeseen, jossa kaikki työvaiheet olisi tehty samassa kaaviossa. Eri synteesivaiheet erottuvat myös selvästi ja niiden välillä voidaan liikkua.



Kuva 23. Työvuomallien hierarkia.

Malli käsittää 6 työvuokaaviota, jotka on yhdistetty toisiinsa yhdellä ylimmän tason kaaviolla. Yhteensä kaaviot sisältävät 63 prosessia ja niistä ohjataan 13 työkalua erilaisilla parametreillä käyttämällä komentoja (liite I). Toteutetun työvuon kaaviot esitellään seuraavissa kappaleissa.

4.1 YLIMMÄN TASON-TYÖVUO

Ylimmän tason kaaviolla (liite B) kytkettiin eri työvuokaaviot toisiinsa. Mallissa hierarkioiden välissä olevat prosessit sisältävät kuittauskomennot, joilla käyttäjä jättää viestin tiedostoon aina kokonaisuuden päätyttyä. Synteesikaavioista päästään haluttaessa synteesisuunnittelu- eli koodin muokkauskaavioon, jos se jossakin vaiheessa havaitaan tarpeelliseksi. Logiikkasynteesistä on myös mahdollista siirtyä esimerkiksi ajamaan jonkun prosessin skedulointi uudelleen.

4.2 SA/VHDL-TYÖVUO

SA/VHDL-vuohon (liite C) on mallinnettu SA/VHDL-uudelleenkäyttötyövaiheet ja SA-spesifikaation simulointi.

Vuo aloitetaan ajamalla työkalujen käynnistysprosessi, josta käynnistetään Prosa- tai versionhallintatyökalu kyselykaavakkeen avulla (kuva 22). Tämä prosessi on aina ajettavissa. Seuraavana oleva suunnittelun laadintaprosessi ei sisällä mitään työkalua, sen tarkoituksena on ilmaista, että SA-mallia suunnitellaan ja samalla prosessi toimii paluuprosessina kaaviossa. Tämänkaltaiset tyhjät prosessit toteutettiin käärityllä komennolla, joka kokoaa prosessin lopetusviesteistä kaavakkeen, josta voidaan valita prosessin lopetusviesti. Tähän vastaamalla siirrytään TAI-operaattorin kautta, joko uudelleenkäytettävien SA-mallien etsimiseen tai SA-mallin kääntämiseen Velvetillä.

Tarkistuspisteprosessi sisältää tarkistuslistan SA-mallin tarkastamiselle ennen käännöstä. Listassa on lueteltu seuraavia asioita: ”Balance” ja ”dat” raportit ovat puhtaita, ”common.vhd” tiedosto on päivitetty, signaalit ovat booleantyyppisiä, E/D vuo on aktivointityyppiä, Start/Stop vuot ovat signaaleita, ”vanhat” simulointiherätteet on tallennettu ja tilakoneessa on alkutilan osoittava nuoli.

TAI-operaatiosta lähtevä uudelleenkäytettävien SA-mallien etsiminen aloitetaan käynnistämällä kirjastointiohjelma, jonka sulkeuduttua uusi objekti muokataan erillään varsinaisesta suunnitelmasta uudelleenkäyttöhakemistossa.

Kolmantena vaihtoehtona on kääntää SA/VHDL-malli ajamalla X-ikkunan avaava prosessi suunnitteluhakemistoon, jolloin haluttu hierarkiataso voidaan kääntää komennolla: ”Velvet dNRO.dfm”. Velvetin tuottamasta tekstistä Prosalla korjataan mallissa olevat mahdolliset viat. Velvetin ajon jälkeen X-ikkuna suljetaan, jolloin voidaan palata takaisin tai jatkaa simulointiin.

Simulointiin siirryttäessä voidaan valita, käynnistetäänkö simulointityökalu. Mikäli se on jo päällä edellisestä kierroksesta voidaan muuttunut malli ana-

lysoida ja simuloida. Simulaattorin käynnistysprosessissa käyttäjä voi valita 2 simulointityökalusta. Näistä VHDL2000 käynnistetään kuten aiemminkin ”ViewTool”-nimisen hallintaohjelmiston kautta, jonne luodaan simulointi-hakemistoon osoittava projekti.

Herätteiden muokkausprosessista käynnistyy tiedostonhallinta SA-mallin hakemistoon. Sillä voidaan muokata ja esimerkiksi nimetä uudelleen simuloinnissa tarvittavat heräte- ja testipenkkitiedostot. Simulaattoria ja tiedostonhallintaa ei tarvitse sulkea simulointiprosessista poistuttaessa.

Simulointiprosessissa on jälleen tyhjä komentorivi. Sen ajo tulostaa kyselyn, jossa vaihtoehtoina ovat uuden käännöksen ajaminen, paluu SA-mallin suunnitteluun, siirtyminen uudelleenkäyttöobjektin valmistamiseen tai kuittaamaan SA-malli valmistuneeksi. Ensimmäisellä valinnalla päästään ajamaan uusi Velvet käännös. Toisella ”Fail”-tyyppisellä polulla siirrytään takaisin SA-mallinnukseen, kolmannella siirrytään tuottamaan uudelleenkäytettävä SA-malli ja viimeisenä vaihtoehtona on siirtyminen syntetisoitavan koodin valmistamiseen.

SA/VHDL on valmis -prosessissa on tarkistuslista asioista, kuten SA-mallin simuloinneista, mallin jakamisesta ja tallentamisesta versionhallintaan ja simulointituloksien tallentamisesta.

4.2.1 Uudelleenkäyttömenetelmät SA-työvuossa

Uudelleenkäyttöobjektin valmistusprosessissa tarkoituksena on valmistella malli uudelleenkäytettäväksi: mallista etsitään tietotyypit ja suunnitellaan dokumentaatiota, joka liitetään myöhemmin kirjastotyökalun toimintakuvausosaan. Seuraavassa prosessissa tallennetaan haluttu SA-kaavion osa nimellä ”obj.dfd”.

Tarkistusprosessissa sijaitsevat tarkistettavat asiat ja tiedot, jotka dokumentaatiossa tulisi mainita kuten mallin käyttäytyminen ja liityntä on dokumentoitu, kommentteja lisätty, nimet ovat luonnollisia, tietotyypit on siirretty ”common.vhd”-tiedostosta ja kirjastointiohjelman hakemistot on asetettu.

Uudelleenkäytettävän objektin tallennusprosessiin ei päästä ennenkuin kummatkin JA-operaation haarat ovat suoritettu. Tässä prosessissa käynnistetään kirjastointiohjelma, jolloin uuden mallin pitäisi näkyä paikalliskirjaston sivulla. Ohjelmassa täytetään mallille tarvittavat tiedot ja siirretään se yleiseen kirjastoon käytettäväksi.

4.3 SYNTEESISUUNNITTELUTYÖVUO

Synteesisuunnittelutyövuossa (liite D) on mallinnettu syntetisoitavan koodin muokkaus, versionhallinta ja koodin analysointiprosessit. Työvuohon liitettiin irralleen prosessit Prosan ja Synopsyksen ohjeen käynnistämiseksi.

Koska työvuohon on kolme tuloa ylemmältä tasolta synteesityövoista, jouduttiin piirtämään myös kolme liityntänuolta terminaalista versionhallintaprosessiin.

Työvuossa on tarkoituksena pitää muokattu koodi versionhallinnassa ja tuoda sieltä aina yksi hierarkiataso kerrallaan analysointihakemistoon muokattavaksi. Versionhallinnan työhakemisto määrätään joko projektissa tai kansiossa valinnan mukaan.

Työvuossa versionhallintaohjelma jää käynnistyksen jälkeen päälle, jolloin muokkausprosessi voidaan käynnistää. Muokkausprosessin käynnistyessä valitaan kaavakkeesta tiedostonhallinnan ja X-ikkunan väliltä. Kumpikin käynnistyy suoraan analysointihakemistoon. Versionhallinta ja muokkausprosessia ei tarvitse sulkea käynnistyksen jälkeen ”Start-Start”-yhteyden ansiosta.

Kun prosessi on muokattu, voidaan tarkistuslistaprosessissa tarkistaa, että suunnitteluyksikön esittely ja arkkitehtuurikuvaus on liitetty ”dNRO-b.vhd” nimiseen tiedostoon, prosessi on synkroninen, kello ja alustussignaali on muodostettu, suunnitteluyksikön esittely on päivitetty, prosessissa alustetaan kaikki muuttujat ja lähtevät vuot ja sinne on nimetty prosessi, alustus- ja pääsilmukka.

Tämän jälkeen on mahdollisuus ajaa VHDL-kuvausten tarkistus ja analyysi. Komentotiedostoihin asetettavat parametrit ja muutokset tehdään valinnaisesti ajettavalla muokkausprosessilla. Analyysissä käytettävien komentotiedostojen nimet ovat aina samat. Tiedostoista valitaan toinen muokkaus- ja ajoprosessin käynnistyessä.

Analyysiajo kestää korkeintaan muutaman minuutin. Tänä aikana voidaan esimerkiksi muokata seuraavaa prosessia. Tämän jälkeen näyttöön ilmestyy raporttitiedosto. Suunnitelman muuttaminen voidaan tehdä samanaikaisesti raportin ollessa esillä. Raporttitiedoston sulkemisen jälkeen valitaan seuraavista viesteistä: uusi analyysi, simuloi tai odota tässä. Ensimmäisen valinta vie takaisin koodin muokkausprosessiin ja käynnistää sen automaattisesti uudelleen. Uusien työkalujen ajo voidaan estää vastaamalla ”cancel” esiin tulevassa kaavakkeessa. Viimeisen viestin valinta pysäyttää vuon tähän, koska prosessille ei ole tätä viestiä asetettu ja DMM jättää sen huomioimatta. Prosessi jää käynnissä-tilaan ja suunnittelija voi heti ajaa uuden analyysin.

Analyysiprosessin jälkeen voidaan siirtyä simulointikaavioon simulaation valmisteluprosessin kautta. Tämä vaihe lisättiin siksi, että suunnitelman tie-

dostot tarkastettaisiin ennen simulointikaavioon siirtymistä. Tässä tarkistetaan esimerkiksi, onko kaikki tiedostot analysoitu, rakennekuvaukset päivitetty ja herätetiedostot muutettu. Käyttäjä voi tässä valita kahdesta lopetusviestistä: simuloi tai palaa muokkaukseen.

Simulointi suoritetaan samaan tapaan kuin SA-tasollakin. Herätteiden muokkausprosessi avaa tiedostonhallinnan nyt synteesiä varten varattuun simulointihakemistoon, jonka oletusarvo oli sama kuin analysointihakemisto. Nyt versionhallinnasta voidaan hakea viimeistään tarvittavia herätteitä ja rakennekuvauksia päivitettäväksi.

Simulointiprosessi käynnistetään herätteiden muokkausprosessin jälkeen. Kun simulointi on suoritettu vastataan lopetusviestikyselyyn ja siirrytään simulointi valmis -prosessiin. Tässä prosessissa ei tehdä muuta kuin valitaan yksi kolmesta lopetusviestistä: palaa muokkaukseen, tallenna hierarkia ja etene synteesissä. Ensimmäisellä palataan koodin muokkausprosessiin, toisella versionhallintaprosessiin tallentamaan nykyinen suunnitelma ja ottamaan sieltä seuraava hierarkia muokattavaksi. Viimeisellä viestillä voidaan edetä logiikkasynteesiin.

Simuloinnin tarkistusprosessi sisältää muokatun koodin simulointivaiheessa tarkistettavia asioita kuten, että alustussignaali aloittaa simulaation, vuot alustuivat, simuloitavat laskut menivät oikein, signaalit kestävät yhden kellojakson, kättelyt, kello ja eri suorituspolut on simuloitu.

4.4 KÄYTTÄYTYMISTASON TYÖVUO

Päätarkoituksena käyttäytymistason työvuolla (liite E) on etsiä hierarkian prosesseille riittävän hyvä toteutus suunnitelmarajoituksia muuttamalla ja varmentaa syntetisoidun hierarkiatason toiminta simuloimalla.

Käyttäytymistasolla päätyövaiheiksi tulivat komentotiedoston muokkausprosessi ja syntetisointiprosessi. Hierarkkisessa simulointiprosessissa on RTL- ja porttitason simulointityövuo.

Komentojen muokkausprosessi avaa komentotiedoston käyttäjälle. Sinne asetetaan aiemmin analysoidun suunnitelman nimi, kellojakson pituus, kellojaksojen määrä prosessin pääsilmukalle, kellosignaalin nimi ja synteesin teho (effort). Synteesiä varten on valittavana ympäristöolosuhteet synteesikirjastosta ja suunnittelija voi määrätä, tallennetaanko kokeilussa syntynyt syntetisoitu simuloituva ja tietokantamalli.

Ennen synteesiajoa työvuossa on suoritettava tarkistuslistaprosessi vähintään yhden kerran. Prosessi sisältää tarkistuslistan parametrien asettamisesta, prosessien erilaisten skedulointien tekemisestä, vertailemisesta ja skedulointitulosten tallentamisesta.

Suunnittelijan etsiessä erilaisia vaihtoehtoja skeduloinnin teho voi olla pienempi. Kun sopiva kellojakso ja jaksojen määrä on löydetty suoritetaan viimeinen skedulointi suuremmalla teholla ja asetetaan komentotiedosto tallentamaan suunnitelma.

Työvuossa irrallaan oleva muistioprosessi ajaa tekstieditorin, jonne voidaan tallentaa eri vaihtoehtojen tuloksia vertailemista varten.

Syntetisointiprosessi ajaa käyttäytymistason synteesin näyttäen sen etenemisen X-ikkunassa. Skeduloinnin viemä aika riippuu operaatioiden määrästä ja vaatimusten tasosta. Jos suoritus onnistuu ilman virheitä käyttäjälle näytetään raportti skedulointituloksista. Raportteja tallentuu yksi jokaista skedulointia kohti. Ylimääräiset raportit poistetaan tai niitä vertaillaan suunnitelman selausprosessin avaamalla tiedostonhallinnalla.

Skedulointia suoritettaessa komentotiedoston muokkausprosessi voidaan pitää käynnissä ja valmistella esimerkiksi seuraavan skedulointia. Synteesiprosessin ajon jälkeen valitaan neljästä viestistä: muokkaa komentotiedosto, siirry koodin muokkaukseen, siirry simulointiin tai odota tässä. Viimeinen näistä on jälleen ”virheellinen” viesti, jonka valinta jättää synteesiprosessin päälle ja uudelleen käynnistettäväksi. Työvuohierarkiassa palataan takaisin synteesisuunnittelutyövuohon ”siirry koodin muokkaukseen” -viestin valinnalla.

Kaavion viimeiseen prosessiin siirrytään jos simulointi onnistui. Tässä prosessissa ajetaan versionhallinta, jonne voidaan tallentaa esimerkiksi tarvittavat raportit, simuloituvat tiedostot ja tietokantatiedostot.

4.5 LOGIIKKATASON TYÖVUO

Logiikkatason työvuossa (liite F) syntetisoidaan analysoidut tilakonekuvaukset ja skeduloidut prosessit logiikkatason kuvaukseksi. Työskentelyvaiheet ovat samankaltaiset kuin käyttäytymissynteesivuossakin. Simuloinnista palataan, joko ajamaan uutta synteesiä tai palataan synteesisuunnittelutyövuohon esimerkiksi muuttamaan jotain prosessia. Tarkistuslistaprosessiin kerättiin synteesin tuloksista tarkistettavia asioita.

Komentojen muokkausprosessilla muokataan kyselykaaviosta valittua tilakoneille tai prosesseille tarkoitettua komentotiedostoa. Siinä asetetaan mm. suunnitteluyksikön nimi, tehokkuus, kellosignaalin nimi, määrätään tallennetaanko suunnitelma, poistetaanko hierarkia ja synteesikomennon parametreja. Lisäksi tilakoneita varten määrätään tilavektorin nimi ja pituus, jonka avulla SA-mallin tilakone syntetisoidaan tilakoneeksi [26]. Tilavektorin ominaisuudet löytyvät esimerkiksi analyysivaiheen raporttitiedostosta.

Syntetisointiprosessi ajaa X-ikkunassa suunnitelman synteesin. Synteesin loputtua suunnittelijalle näytetään raportti tai lokitiedosto. Käyttäjän on

tämän jälkeen valittava 4 eri viestistä: siirry simulointiin, muuta komentotiedostoa, palaa käyttäytymisynteesi- tai synteesisuunnittelutyövuohon.

Irrallaan olevaa muistioprosessia käytetään jälleen tallentamaan muistettavia asioita. Selausprosessilla päästään muokkaamaan hakemistoa ja lukemaan raportti- ja lokitiedostoja.

4.6 INTEGROINTITASON TYÖVUO

Integrointitason työvuossa (liite H) on tavoitteena koota erilliset prosessit yhdeksi suunnitelmaksi käyttäen synteesisuunnittelussa päivitettyjä rakennekuvauksia. Rakennekuvaukset ja suunnitelman tiedostot haetaan versionhallintaprosessin avulla. Synopsyksen graafinen työkalu käynnistetään logiikkasynteesiä varten varattuun hakemistoon, jossa työkalun avulla suunnitelmasta voidaan poistaa hierarkia ja eri tasoja voidaan esimerkiksi optimoida. Hierarkian poiston jälkeen tuotetaan vhdl-muodossa kuvaus koko suunnitelmasta ja se vielä simuloidaan simulointiprosessin työvuossa.

Tarkistuslistaprosessissa tarkistetaan mm. että hierarkia on poistettu, vhdl-malli on kirjoitettu ja tarvittavat raportit on tehty.

4.7 RTL- JA PORTTITASON SIMULOINTITYÖVUO

Simulointityövuo (liite G) on yhteinen kolmelle työvuokaaviolle ja siinä tavoitteena on simuloida hierarkiataso kerrallaan käyttäen apuna aiemmin valmistettuja heräte- ja testipenkkitiedostoja Synopsys VSS -simulaattorilla. Simulointia varten suunnitelmasta tuotetut simuloitavat tiedostot pitää analysoida ”vhdlan”-nimisellä ohjelmalla. Tätä varten komentotiedostoprosessissa kirjoitetaan analyysikomennot suunnitelman tiedostoille: rakennekuvaus-, heräte-, testipenkkitiedostoille sekä VSS:n vaatimalle konfiguraatiotiedostolle.

Komentotiedoston ajoprosessi ajaa analyysikutsut UNIXin komentotulkissa simulointihakemistossa. Suorituksen tulokset tallennetaan tiedostoon, joka näytetään lopuksi käyttäjälle. Virheettömän analyysiajon jälkeen voidaan siirtyä simulointiprosessiin analyysi onnistui -viestillä. Analyysissä tulkitaan syntetisoidusta suunnitelmasta kirjoitettua vhdl-koodia, jonka analyysi onnistuu hyvin todennäköisesti. Varalle työvuohon liitettiin mahdollisuus siirtyä takaisin edelliseen vaiheeseen suunnitelmaa viestillä ”analyysi epäonnistui”.

Testipenkkiä käytettäessä komentotiedostohakemistossa oleva konfiguraatiotiedosto on aina samansisältöinen ja siinä määritellyn ”simulate”-nimen avulla kutsutaan ”vhdldbx”-työkalu simulointiprosessissa.

Simulointiprosessin loputtua on mahdollista ajaa uusi simulointi valitsemalla se lopetusviestiksi, jolloin siirrytään takaisin herätteiden muokkausprosessiin. Prosessissa käynnistetään tiedostonhallinta simulointihakemistoon ja versionhallinnasta voidaan hakea uusia herätteitä. Herätteet otetaan simulointiin mukaan ajamalla analysoinnit ja simulaattori uudelleen.

Tarkistuslistaprosessissa on lista simuloitaessa tarkistettavista asioista samaan tapaan kuin synteesisuunnittelutyövuon simuloinnissakin. Lisänä listassa on maininta analysointikomentojen kirjoittamisesta.

Viimeisen simulointi tehty -prosessin tehtävänä on olla etappina työvuossa, sen suorituksen jälkeen valitaan simulointitasolle lopetusviesti, jonka perusteella ylemmällä tasolla siirrytään synteesissä eteenpäin, palataan vaiheeseen, jossa oltiin ennen simulointiin siirtymistä tai palataan synteesisuunnittelutyövuohon.

4.8 YLEISTÄ KOMENTOTIEDOSTOISTA

Yhteistä synteesissä käytettävissä komentotiedostoissa on menetelmät, joilla käyttäjä antaa synteesin tarvitsemia parametreja. Parametrit sijoitellaan tiedoston alussa olevassa osiossa synteesityökalun sisäisiksi muuttujiksi, joita käytetään eri komennoissa. Parametrien lisäksi tiedostoihin määritetään muitakin muuttujia; esimerkiksi raportti- ja lokitiedostojen nimet voidaan haluttaessa muuttaa 8 merkkiin sopiviksi.

Kaikissa komentotiedostoissa luetaan ”dmm.setup”-tiedostossa määriteltyjä UNIX-ympäristömuuttujia. Näiden avulla työkalu löytää oikean hakemiston kun synteesin tiedostoja käsitellään.

Jos synteesissä havaitaan virheitä, käyttäjälle näytetään ”.log”-loppuinen tiedosto, jonne tallentuvat eri komentojen tulostukset ja virheilmoitukset. Mikäli synteesissä ei havaittu virheitä, näytetään käyttäjälle synteesin tuloksia raporttitiedostosta. Komentojen ajon jälkeen tuloksien näyttäminen hoidetaan komentotiedostosta käsin. Edelliset loki- ja raporttitiedostot kirjoitetaan yli. Aikaleimat tulevat raporttitiedostojen alkuun ja loppuun.

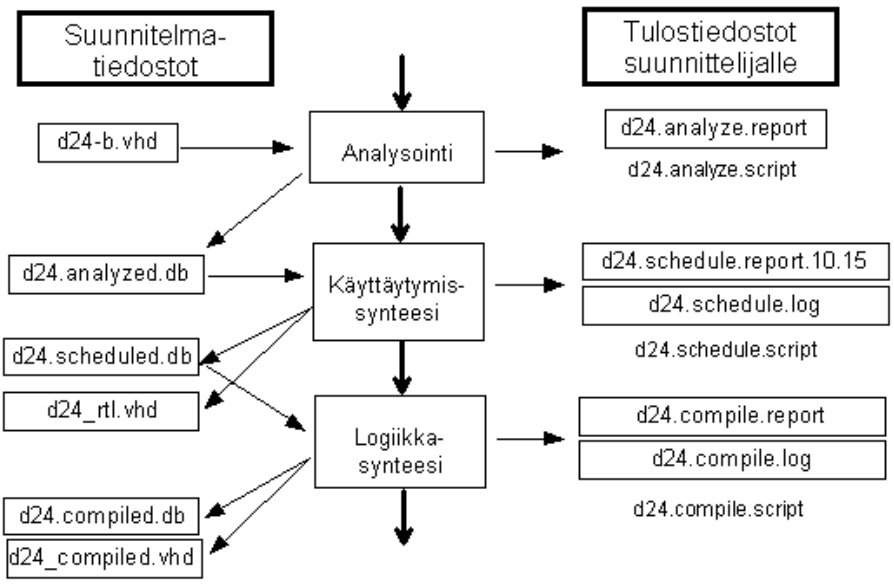
Komentotiedostojen säilytys -hakemistossa sijaitsevat seuraavat tiedostot: ”analyze.(state.)script”, ”schedule.script”, ”compile.(state.)script” ja VSS:llä tehtävää simulointia varten UNIX:lle tehty ”simulate.script”. Synteesissä kirjoitettavat raportti-, suunnitelma- ja lokitiedostot on nimetty kuvan 24 esittämällä tavalla.

Analyysin läpäissyt tiedosto tallennetaan nimelle ”<ent>.analyzed.db” ja analyysiraportti on nimeltään ”<ent>.analyze.raport”. Näissä <ent> on prosessin nimi. Skeduloitaessa luetaan analyysissä tallennettu tiedosto ja sen tulokset tallennetaan skedulointihakemistoon. Skeduloidut suunnitelmat nimetään <ent>.scheduled.db.

Skeduloinnissa syntyvät raportit nimetään määriteltyjen rajoitusten mukaan: Esimerkiksi ”d24.schedule.raport.10.15” on d24-suunnitteluyksikön skedulointiraportti suoritettuna kellojakson pituudella 10 ja pääsilmukan kellojaksojen määrällä 15. Simuloitavat mallit tallennetaan niin haluttaessa simulointihakemistoon nimelle ”<ent>_rtl.vhd” ja tietokanta skedulointihakemistoon nimelle ”<ent>.scheduled.db”.

Tilakoneiden logiikkasynteesissä analysoidut suunnitelmat etsitään analysointihakemistosta. Muille prosesseille tarkoitettu komentotiedosto etsii skeduloidut suunnitelmat skedulointihakemistosta. Logiikkasynteesiajo käynnistetään logiikkasynteesihakemistoon, jonne tallentuvat tiedostot <ent>.compiled.db ja simuloitava malli <ent>_compiled.vhd.

Lokitiedosto on ”<ent>.compile.log” ja synteesitulokset ovat tiedostossa ”<ent>.compile.report”. Tilakoneista tulostetaan myös suunnitelmasta erotettu tilakonekartta tiedostoon ”<ent>_fsm.st”.



Kuva 24. Työvuokaavion synteesissä syntyvät tiedostot.

Komentotiedostot kirjoittavat kaikki suorittamansa komennot tiedostoihin <ent>.analyze.script, <ent>.compile.script ja <ent>.schedule.script. Näiden avulla voidaan aiemmin suoritettu vaihe toistaa ”dc_shell -f”-komennolla.

4.9 MALLIN KÄYTTÖÖNOTTO

Ennen mallin käyttämistä on asennettava ja konfiguroitava tarvittavat ohjelmat NT-työasemaan. Sen jälkeen luodaan projektihakemistot UNIX-palvelimelle ja muutetaan ”dmm.setup”-tiedostosta suunnitelman hakemistopolkuasetukset.

DMM asennetaan verkon yli palvelinkoneesta ”install”-hakemistosta. NT työasemassa vuon käyttäjälle määrätään työvuon käyttöoikeudet ja liitetään käyttäjä DMM-työryhmään.

Työasemassa tarvittavat ohjelmat asennetaan siten, että ne löytyvät käyttäjän tunnuksen PATH-asetuksesta ennen vuon liittämistä työhakemistoon. DMM:n tarvitsemat valintatiedostot kopioidaan esimerkiksi kotihakemistoon.

Työasemassa tarvittavat ohjelmat ovat Sv-manager, PVCS, dmmtable, V-SYSTEM (valinnainen), Notepad ja Exceed versio 5 NT:lle. UNIXissa tarvittavia ohjelmia ovat Velvet, VHDL2000, tiedostonhallinta, tekstieditori, Prosa ja Synopsys-työkalut.

Kirjastointiohjelmassa asetetaan yleinen ja paikallinen kirjastopolku ja liitehakemisto. UNIXssa määriteltävä uudelleenkäyttöhakemisto, paikallinen ja liitehakemisto asetetaan samoiksi. Kirjastointiohjelma tekee paikallishakemistoon uusille malleille lisää hakemistoja.

Tarkistuslistaohjelma voidaan ajaa verkon yli jaettujen hakemistojen kautta. Työvaiheisiin tallennetut tiedot ovat ”dmmtable.ini”-tiedostossa.

Exceed asennuksen yhteydessä luodaan 2 konfigurointitiedostoa nimille ”dsutsun1.xs” ja ”dcaesun5.xs”, joihin määritetään UNIX koneen nimi (sutsun1 tai caesun5), käyttäjätunnus ja salasana. Kielletään kysymästä salasanaa käynnistyksessä ja käynnistysmenetelmäksi asetetaan REXEC. Komentorivi voidaan jättää tyhjäksi. Koneiden nimet tulee kirjata ”xhosts.txt”-tiedostoon tai käyttää numero-osoitetta.

”Dsutsun1”-asetusta käytetään ajettaessa synteesikomentoja. Synteesiajo voidaan jättää taustalle, kun asetuksiin määrätään ”Connection settings”-osaan ”Close timeout” riittävän suureksi, esimerkiksi 20 sekunniksi. Tällöin Xstart-ohjelma sulkeutuu X-ikkunan käynnistymisen jälkeen, jolloin DMM tulkitsee komennon suoritetuksi ja vuossa päästään etenemään. Tämän avulla suunnittelija voi esimerkiksi jättää skedulointiajon päälle ja siirtyä synteesisuunnitteluun valmistelemaan seuraavaa hierarkiaa.

UNIXissa ”.cshrc”-tiedostoon lisätään ”source <path>/dmm.setup” komento, jolla tiedostossa olevat asetukset otetaan käyttöön. ”Dmm.setup”-tiedostossa määritellyt hakemistot luodaan ja ”set_display”-aliaksessa määritelty koneen nimi tarkistetaan.

4.10 HUOMIOITA TYÖVUOMALLISTA JA MALLINNUKSESTA

Työn tekemisessä oli vaikeuksia jo DMM:n versio 14 toimittamisessa ja asennuksessa. Toimivan työvuon tekeminen ja testaus oli oletettua suurempi työ, jota vaikeutti DMM:n ohjekirjojen liian yleinen taso ja ohjelman sisäl-

tämät virheet, heikko käyttöliittymä vuon suunnittelijalle ja testaustoimintojen puutteet. Näiden asioiden vaikuttaessa kaikkia DMM:n ominaisuuksia ei ehditty käyttää. Siksi osa vuon työvaiheista jäi suuriksi ja esimerkiksi suunnittelijan tekemää työtä ei tarkisteta eri työvaiheissa.

Työvuon mallintaminen olisi pitänyt aloittaa jo työprosessin suunnitteluvaiheessa, kun työvuon prosessit olivat hahmottuneet. Työvuosta olisi tullut yksityiskohtaisempi, jos työvuon mallinnustyökalu olisi toiminut UNIXissa. Eri käyttöjärjestelmissä toimivien ohjelmien kontrollointi DMM:llä on hankalaa ja toimivan mallin tuottaminen aikaavievää. Samalla työvuomallissa tarvittavat ohjelmat ja niiden asetukset hajaantuvat, jolloin toiminnan ylläpitäminen vaikeutuu.

DMM:llä tehtävät työvuot sopivat parhaiten peräkkäisten eri työkaluilla tehtävien työvaiheiden mallintamiseen. Tämä on havaittavissa esimerkiksi SA/VHDL-työvuota tarkasteltaessa. Siinä Prosalla tehtävä työ, joka on tärkein ja aikaa vaativin työvaihe, jää peittoon yhden prosessin alle. Kuitenkin mallin varmennus pystyttiin mallintamaan käännös-, herätteiden muodostus- ja simulointityövaiheissa, ja SA-mallin tarkistamiseen tuotettiin ohjeita tarkistuslistaan.

SA-mallia tehtäessä muokattavat prosessit kannattaa jättää mahdollisimman suuriksi kokonaisuuksiksi. Mitä pienempiä prosesseja synteesiin tulee sitä enemmän suunnitelmasta löytyy päällekkäisiä käyttämättömiä resursseja. Paras tulos synteesityökalun 3.3 versiolla saavutetaan, kun suunnitelmassa on suuria prosesseja ja kun kaikki prosessit toimivat mahdollisimman rinnakkaisesti.

Tarkistuslistaprosessien avulla malliin saatiin lisää ohjeita ja kontrollia eri työvaiheisiin. Tarkistuslistoja voidaan muokata työvuota käytettäessä myöhemmissäkin synteesiprojekteissa ja näin kerätä niihin lisää tietoa. Työvuomallin muussa ylläpidossa päähuomion voi kiinnittää työkalukomentojen oikeellisuuteen valintatiedostoissa ja ”dmm.setup”-tiedostossa.

5 POHDINTA

Työssä toteutetun suunnitteluprosessin suunnittelun, prosessin mallintamisen, komentotiedostojen luonnin ja ohjelmien asetusten kokoamisella on vaikutusta mallin avulla suoritettavassa synteesiprosessissa. Näitä asioita perustellaan seuraavissa kappaleissa. Tässä työssä mallin ominaisuuksia ei voida perustella mittaustuloksilla, vaan mallia verrataan työmenetelmien eroavaisuuksiin ilman mallia tehtävään suunnitteluun nähden. On muistettava, että kuvattu syntetisointiprosessi oli uusi ja että CAN-projekti oli syntetisointimenetelmien etsimistä ja synteesikokeilua kappaleessa 4.1 kuvatulla tavalla.

5.1 YLLÄPITOMENETELMIEN VAIKUTUS SUUNNITTELUPROSESSISSA

Suurin muutos suunnitteluprosessissa on suunnitelman tiedostojenhallinta, joka mahdollistaa versiohistorian, paluun virhetilanteissa ja estää tiedostojen häviämistä. Ilman versionhallintaa suunnitelmasta kopioidaan käsin osia eri hakemistoihin tai otetaan käsin varmuuskopioita. Tällöin on vaarana esimerkiksi, että suunnitelma kopioidaan väärään paikkaan tai vanha versio otetaan käyttöön. Versionhallinnan avulla suunnittelutiedostot voidaan palauttaa aina takaisin tilaan, josta virheelliseksi paljastunutta muutosta alettiin tehdä. Tämä on tärkeää, sillä suurta suunnitelmaa muutettaessa ei aina voida olla varmoja, että koko suunnitelma toimii muutoksen jälkeen, vaan muutoksen toimivuus voidaan todeta vasta simulointivaiheessa.

Versionhallinta mahdollistaa uudelleenkäytettävien suunnitelmien muuttamisen hallitusti. Uudelleenkäyttömenetelmät lisättiin työvuohon nopeuttamaan spesifikaation suunnittelua. Menetelmien lisäys toteutettiin dokumentaatiovaatimusten tallentamisella työvuohon ja työvaiheiden mallintamisella. Tuotettu työvuomalli ei vaadi uudelleenkäytön käyttämistä, mutta jos mallia käytettäessä tuotetaan komponentteja, voi käyttäjä motivoitua tekemään tarpeellisen dokumentaation helpommin kuin ilman mallia. Motivaatiota lisää tarvittavien työkalujen käynnistyminen valmiiksi työpöydälle, kun uudelleenkäyttöprosesseihin siirrytään.

Luonnollisten nimien käytöllä voidaan saavuttaa helpommin ymmärrettäviä suunnitelmia, ja kuvaavista nimistä on hyötyä syntetisoitavaa mallia valmistettaessa prosessin toiminnan ja suunnitteluratkaisujen muistamisessa. Pitkillä nimillä ei todettu olevan vakavia haittavaikutuksia synteesissä tai simuloinneissa. Ainoastaan VHDL2000-simulaattorissa pitkät nimet haittaavat tuloksien tarkastelua, koska ohjelma näyttää nimestä vain 8 merkkiä kerrallaan.

Vaikka luonnolliset nimet ja kaavioiden piirtoseikat eivät liity varsinaiseen prosessin mallintamiseen, niillä on vaikutusta suunnitelman ymmärrettävyy-

den ja toiminnan muistamisen parantumisessa, vian etsimisen, herätteiden suunnittelun ja muutoksen tekemisen nopeutumisessa uudelleenkäyttötilanteissa, simuloinneissa ja syntetisoituvaa mallia tehtäessä.

Työvuomallissa käytetyssä versionhallintatyökalussa oli paljon ominaisuuksia usean käyttäjän ohjelmistoprojektin hallitsemiseen. Ohjelma oli laaja yhden käyttäjän prosessin tarpeisiin, mutta sen graafinen käyttöliittymä oli helppokäyttöinen ja ohjelma toimii luotettavasti ja nopeasti. Kirjastointityökalu on jäänyt ajastaan jälkeen ylläpidon puutteessa. Tehokkaan käytön aikaansaamiseksi sitä pitäisi päivittää uudemmille ajoympäristöille. Erityisesti työkalussa pitäisi olla tekstin leikkaa- ja liitätoiminnot tai mahdollista lukea dokumentaatio suoraan tiedostosta sekä tuki uudemmille Prosan kaavioversioille. Samalla korjaustoimet voitaisiin tehdä muutaman ohjelmavirheen korjaamiseksi. Työvuomallissa uudelleenkäyttöä ei voida tehokkaasti soveltaa ennen kirjastointiohjelman ylläpitoa.

Mallissa käytettävät ylläpitomenetelmät muuttivat suunnittelun luonnetta huomattavasti. Tiedostoja käsitellään graafisen käyttöliittymän kautta, jolloin kirjoittamisen määrä vähenee, virheitä sattuu vähemmän ja tiedostojenhallinta on helpompaa.

5.2 MALLIN EDUT JA HAITAT

Valmistettu työvuomalli toimii samalla periaatteella kuin komentoriviltä tehtävä suunnittelukin. Mallin kautta tuotetaan samat komennot ja ajetaan samat työkalut. Käyttöympäristönä toimii suunnittelijan PC niin kuin ennenkin ja suunnittelutiedostot voidaan tallentaa projektille varatulle levylle.

Mallinnetulla työvuolla tehtävässä suunnittelussa on havaittavissa seuraavat eduiksi luettavat seikat:

- työvuot ovat graafinen käyttöliittymä prosessiin,
- työprosessi on kuvattu suoritettavassa muodossa,
- työvaiheet ovat esillä ja järjestyksessä,
- työohjeet on liitetty työvaiheeseen,
- työkalukomennot ovat piilotettuja,
- työskentely on turvallista,
- työmenetelmiä opitaan työvuota käytettäessä,
- työprosessiin on mahdollista tallentaa lisää tietoa,
- työprosessista saadaan suunnittelukohtaista tietoa,
- prosessin suoritusta voidaan seurata reaaliajassa,
- synteesiprosessi on muutettavissa suunnitelmakohtaisesti,
- työvuot hallitaan palvelimessa,
- työprosessi asennetaan käyttäjälle ja
- komentotiedostot ovat osana asennusta.

Mallinnetulla työvuolla havaittuja haittapuolia ovat seuraavat:

- työkalujen käyttäminen on hitaampaa verkon yli,
- asetustiedostot ovat hajallaan,
- mallissa voidaan käyttää vain sinne määriteltyjä työkaluja,
- UNIX-ohjelmien ajaminen NT:stä hidasta,
- vuo perustuu laajasti Exceed-ohjelman käyttöön,
- suunnittelijan työtä ei tarkisteta,
- osaa tiedostojen nimistä ei voi muuttaa,
- kyselykaavaketiedostot ovat erillään työvuosta,
- työvuo tarvitsee ylläpitoa,
- vuon muuttaminen on aikaavievää,
- osa käytetyistä työkaluista ei sovellu hyvin työvuokäyttöön ja
- vuot tarvitsevat pysyvästi palvelinkoneen.

Mallinnetun työvuon käyttäminen on helppoa graafisen käyttöliittymän ansiosta. Suunnitteluprosessin eri työvaiheet ovat näkyvissä prosesseina, joiden avulla työkalujen ajaminen hiirellä on helpompaa kuin komentoriviltä.

Graafisesta kuvauksesta on etua suunnittelun tilan hahmottamisessa ja DMM:stä on hyötyä suunnittelun seurannassa. Suunnittelun edistymistä voidaan haluttaessa seurata reaaliajassa toiselta asiakaskoneelta ja työkalu tallentaa suunnittelukohtaista tietoa, kuten esimerkiksi SA-mallin aloitus- ja lopetusajankohdan, syntetisointiin käytetyn ajan ja syntetisointikertojen määrän.

Työvuossa suunnittelu on jaettu pieniin työvaiheisiin, joissa tarvittavat työkalut käynnistyvät ja työvaiheen ohje voidaan lukea käynnistetystä prosessista. Työohjeet mallinnetussa työvuossa ovat lähinnä prosessissa tehtävän työn selittämistä. Malliin upotetut tarkistuslistat täydentävät hyvin useasta prosesseista koostuvan osan tavoitteiden tarkistamista ja ne mahdollistavat myös lisätiedon tallentamisen listaan osaksi prosessia. Tehdyssä mallissa prosessit on valittu tarkoituksella mahdollisimman yleisiksi suunnittelun pääosien hahmottamiseksi, jolloin paljon yksityiskohtaista tietoa, esimerkiksi synteesikomennot, jäävät irralleen työprosessimallista. Tämä mahdollistaa synteesikomentojen muokkaamisen halutuiksi suunnittelukohtaisesti.

Työvaiheisiin määritetyn työkalun ajaminen on käytännöllistä niin kauan kuin käyttäjä haluaa käyttää juuri tarjottuja työkaluja. Käyttäjän on vaikeaa vaihtaa esimerkiksi työvuon prosessissa määriteltyjen työkalujen tai tiedostojen nimiä. Työprosessin suorittaminen onnistuu ilman työohjeita, jos prosessi on ennestään tuttu, mutta silloin esimerkiksi suunnittelumenetelmistä uusien tarkistettavien asioiden löytyminen voi jäädä pelkästään suunnittelijan tiedoksi.

Työvuohon tallennetut komennot ovat turvallisempia kuin käsin annettavat. Väärien komentojen antaminen työvuota suorittamalla on vaikeaa. Ainoa vikamahdollisuus on, että suunnittelija ei suorita kaikkia ajamiaan prosesseja, jolloin suunnittelutiedostoja puuttuu.

Mallinnetussa työvuossa ei tarkisteta tehtyä työtä. Vaikka DMM antoikin siihen mahdollisuuden, kaikki käytetyt menetelmät eivät sitä tukeneet: esimerkiksi verkon kautta UNIX:ssa käynnistettävien ohjelmien suorituksen onnistumista ei voida tietää tai simuloinnin onnistumisen varmistaminen ohjelmallisesti olisi ollut vaikeaa.

UNIX-ohjelmien ajaminen NT:stä perustui suurelta osin Exceed-ohjelman käyttöön. Toisen X-ikkunoinnin mahdollistavan ohjelman käyttöönotto aiheuttaisi komentojen uudelleenkirjoittamisen työvuon prosesseissa ja kyselykaavaketiedostoissa. Mielestäni suurin puute mallinnetulle työvuolle on NT - UNIX välille sovitettu toiminta, joka aiheutti lisätyötä ja mallin hajaantumista. Samoin graafisten työkalujen käynnistäminen verkon kautta on hitaampaa verrattuna konsolilla työskentelemiseen. NT:n käyttäminen palvelimena aiheuttaa sen, että palvelinkone on käytännössä kokonaan varattava työvuolle, jolloin se on poissa muusta hyötykäytöstä.

Ennen käyttöönottoa työprosessi on asennettava käyttäjälle. Tämä on samalla puute ja etu. Asennukseen käytetty aika ja vaiva maksavat itsensä takaisin asennuksen yhteydessä tehtyjen valmiiden työkaluasetusten, komentotiedostojen ja työvoiden kautta. Asennuksen yhteydessä lisätään puuttuvat ohjelmat ja asetetaan ympäristöasetukset asiakaskoneeseen. Huono puoli mallissa on asennuksen hajanaisuus: kyselykaavakkeet asennetaan asiakaskoneeseen, UNIX-komennot työkalujen ajoon soveltuvaan koneeseen ja muita työkaluja ajetaan paikallisesti tai palvelinkoneelta. Lisäksi asetukset vaativat paikallista konfigurointia hakemistojen ja verkko-osoitteiden muodossa.

Malli tarvitsee ylläpitoa. Uusien ohjelmien lisääminen työvuohon voi onnistua pelkästään lisäämällä työkalukutsut kyselykaavakkeeseen ja aliaslistaan tai sitten työkalun lisääminen aiheuttaa prosessien lisäämistä tai modifiointia, jolloin työvuo käännetään uudelleen. Mallinnettu työvuo on kuitenkin kohtuullisen yksinkertainen, eikä pienten muutosten tekemiseen kuluva aika ole suuri.

5.2.1 SA/VHDL-työvuo

SA/VHDL-työvuon joustava toteutus oli vaikeaa, koska Prosalla tehtävä työ on piilossa DMM-työkalulta, Velvet, versionhallinta ja kirjastointiohjelma olivat tässä kaaviossa työkaluja, joiden keskinäistä käyttöjärjestystä ei voitu asettaa tai sitoa tiettyyn työvaiheeseen. Tässä kohtaa työvuo toimii ohjelmien käynnistäjänä ja simuloinnin tukena. Simulointityövaiheissa työvuo toimii paremmin kääntämis-, herätteiden muokkaus-, mallin

simulointi- ja uudelleenkäytettävän komponentin tallennusprosesseina. Näillä prosesseilla oli yhteys toisiinsa, ja niiden käyttöjärjestys voidaan määrätä.

Velvetin kaltaisten ohjelmien kontrollointi työvuosta aiheuttaa vaivaa, koska ohjelma vaatii käyttäjältä vastauksen kysymykseen ajon aikana, jolloin ohjelmalle oli avattava erillinen X-ikkuna. Työvuohon liitettävissä ohjelmissa olisi aina oltava mahdollisuus asettaa kaikki tarvittavat parametrit komentoriviltä.

Versionhallinnan, simulaattorin ja Prosan käynnistys- ja herätteiden muokkausprosessit asetettiin irralleen työvaiheista, jolloin työvuon käytettävyyttä parannettiin. Tällöin työvuossa ei tarvitse jatkuvasti sulkea ja käynnistää työkaluja, jotta työvaiheesta päästäisiin seuraavaan. Toisaalta samalla työvuossa menetettiin kontrollia, koska prosesseissa tehtävää työtä ei tarkisteta ja koska ne voidaan ohittaa itse työtä tekemättä.

5.2.2 Synteesityövuot

DMM soveltui hyvin synteesityövoiden mallintamiseen. Analysointitasolla ja synteeseissä oli selvät riippuvuudet työvaiheiden järjestyksestä, jolloin työvaiheiden laatiminen ja prosessin liittäminen toisiinsa oli helppoa ja työvoista tuli johdonmukaisempia.

Irrallisten prosessien, kuten ohjelman sähköisen ohjeen tai muun satunnaisesti tarvittavan prosessin käyttäminen on joustavampaa, kun prosessi on kokonaan irti päätyövaiheista. Yksinään työvuossa olevat prosessit toimivat hyvin työkalujen käynnistäjinä eivätkä ne haittaa työvuon etenemistä. Näin työvuosta saadaan selvempi kuin jakamalla työvuo operaattoreilla.

Synteesityövoista tuli ulkonäöltään samankaltaisia, mutta prosesseista suoritettavat komennot ovat erilaisia, ja ne on piilotettu käyttäjältä suunnittelun aikana. Synteesin suorittaminen työvuon kautta on joustavampaa kuin komentoriviltä. Synteesivuot mahdollistavat seuraavan VHDL-prosessin valmistelun synteesin aikana, jolloin saavutetaan samat rinnakkaisen työskentelyn edut kuin komentoriviltä tehtävässä työskentelyssäkin.

Synteesissä parametrien asettamiseksi olisi voitu käyttää kyselykaavaketta, jolloin se olisi ollut helppokäyttöisempi. Toisaalta tässä tapauksessa synteesi olisi sidottu liiaksi työvuon kyselytiedostoihin, jolloin uusien parametrien asettaminen olisi ollut vaikeampaa kuin nyt komentotiedostoissa tehtävä asettelu on. Osa parametreista, esimerkiksi suunnitteluyksikön nimi, olisi voitu asettaa kyselykaavakkeessa, jolloin analysointityövaiheista olisi tullut yksinkertaisempia.

Synteesin suorittaminen työvuon ja komentotiedostojen avulla on huomattavasti helpompaa kuin komentoriviltä tapahtuva komentotiedostojen ajaminen ja raporttien lataaminen tekstieditoriin.

5.3 TYÖVUON KÄYTTÖKOHTEET

Mallinnettavalla työvuolla saavutetaan parhaat ominaisuudet, joustavuus, selkeys ja käytön helppous, kun mallinnettavassa prosessissa käytetään useita työkaluja, työkaluilla tehdään määrätty tehtävä, tehtävät ovat toisistaan riippuvia ja työprosessilla on useita käyttäjiä. Malliin upotettavilla työkaluilla tulisi olla mahdollisimman paljon komentoriviltä annettavia parametreja, jolloin niiden käyttämisellä työvuon kautta saavutetaan eniten etuja käytön helppoutena. Mallinnettaville työvaiheille olisi oltava mahdollista tehdä tarkistusohjelmia, jolloin saavutettaisiin kontrolli työn tekemiselle. Jos työvuon työvaiheilla on selkeä riippuvuus, työvuosta saadaan selkeämpi ja helpommin käytettävä. Työvaiheissa saa olla useitakin lähtöjä, kunhan työvaiheiden järjestys on mielekkäästi määrättävissä. Kaikkien edellä mainittujen ominaisuuksien ei tarvitse toteutua, mutta niiden toteutuessa työvuosta voidaan olettaa saatavan eniten hyötyä.

Työvoiden avulla voidaan tuottaa prosessimalleja opetustarkoituksiin, joista prosessin osat näkyvät helposti ja osien mukana on työohjeita. Mallien avulla prosessin työmenetelmät voidaan oppia miellyttävämmin kuin paperisia ohjeita lukemalla.

Työvuomallit ovat periaatteessa siirrettäviä ja työvuonhallintatyökalut ovat halpoja, joten mallien valmistus myyntitarkoituksissa on myös mahdollista. Tällöin mallin teemana voisi olla vaikkapa yrityksen dokumentointi- tai arkistointimenetelmien kuvaaminen tai jokin muu yritykselle ominainen prosessi.

6 YHTEENVETO

Tässä työssä suunnitellun ja mallinnetun työvuon avulla suunnittelija voi suorittaa ASIC-suunnittelun spesifikaatiosta toteutukseen lähetettäväksi piirisuunnitelmaksi saakka. Työvuonhallintaohjelmisto tarjoaa suunnittelijalle graafisen työskentelytilan, jonka kautta työvaiheissa tarvittavat työkalut avataan.

Hallintaohjelmisto tarjoaa apu- ja kontrollointimenetelmiä työskentelyssä käytettävän työprosessin seuraamiseksi, mittaamiseksi ja parantamiseksi. Työtehtävistä saadaan raporttien muodossa esimerkiksi käynnistyksien lukumäärä ja prosessien kesto. Työvuokaavion kuittausprosessien leimoista voidaan määrittää esimerkiksi SA/VHDL-spesifikaatioon käytetty suunnitteluaika.

Työskenneltäessä työvuo pakottaa suunnittelijan tekemään tärkeät ja välttämättömät työtehtävät uudelleenkäytettävien mallien valmistuksessa ja synteesissä. Mahdollisuuksien mukaan suunnittelijalla on myös vaihtoehtoja työkalujen valinnassa ja työtehtävien suoritusjärjestysten valitsemisessa.

VHDL-kielisen suunnitelmanhallintaan löydettiin uusia menetelmiä ohjelmistotekniikkaan tarkoitetulla työkalulla. Luonnolliset nimet, nimien sijoittelu ja piirtotekniikat auttavat suunnitelmien ymmärtämisessä, spesifikaatioita uudelleenkäytettäessä ja niiden pohjalta suoritettavassa synteesisuunnittelussa.

Graafisten työvoiden avulla tuotettiin kokonaisuus, jossa on tallennettu synteesiprojektissa tarvittavaa tietoa myöhemmin toistettavaan muotoon.

LÄHDELUETTELO

[1] Design Methodology Management User’s Guide (1995). Intergraph Corporation, Huntsville, USA.

[2] Design Methodology Management Quick Start (1995). Intergraph Corporation, Huntsville, USA.

[3] A Methodology for Interactive Synthesis (1996). In: The European Design & Test Conference 1996, March 11-14, Paris, France, 1996, s. 273 - 275.

[4] Behavioral Compiler User's Guide, version 3.3b (1995). Synopsys, Inc.

[5] Behavioral Compiler Technology Backgrounder (1994). Synopsys, Inc.

[6] Kauppi M. (1994) Toiminnallinen mallinnus SA/VHDL-menetelmässä. Oulu: Oulun Yliopisto, sähkötekniikan osasto. S. 27 - 48. Lisensiaattityö.

[7] Kauppi M. (1992) SA/VHDL-menetelmä elektroniikkasuunnittelussa. Insko, 103-92 V.

[8] Juntunen, T., Kivelä, J. & Reinikka, A. & Sipola, M. & Soininen, J.-P. & Tiensyrjä, K. & Tikkanen, T. (1988) Real-Time Structured Analysis in System Level Design of Embedded ASICs, Microprocessing and Microprogramming, The Euromicro Journal, vol. 24.

[9] Sipola, M., Soininen, J.-P. & Kivelä, J. (1991) Systems Real Time Analysis with VHDL Generated from Graphical SA-VHDL. In: Proceedings of the Second European Conference on VHDL methods, EURO-VHDL'91, Stockholm, 1991.

[10] Kauppi, M., Soininen, J.-P. & Tiensyrjä, K. (1992) Velvet, a Tool for Translating System-Level SA-VHDL Model into VHDL for Interactive Modeling and Simulation, In: EuroDAC'92.

[11] Velvet User Manual: The Modelling Guidelines and the Usage of Velvet (1993). Oulu: VTT Electronics.

[12] VHDL Compiler Reference, version 3.3b (1995). Synopsys, Inc. S. 398 - 403.

[13] VHDL Compiler User's Guide, version 3.3b (1995). Synopsys, Inc.

[14] Biggerstaff, T. J. & Perlis, A.J. (eds.) Software Reusability, Volume I, Consepts and Models. New York: ACM Press, S. 1-17.

[15] SA-VHDL komponentointijärjestelmä (1991). Oulu: VTT Elektroniikka. Käyttöohje.

[16] IEEE Standard for Software Maintenance (1993). The Institute of Electrical and Electronics Engineers, Inc., New York, USA.

[17] Roger S. Pressman (1992) Software Engineering: A practitioner's Approach. McGraw-Hill Inc.

[18] Forte, Gene (1993) Software Configuration Management. Case Outlook 1993 vol 7, No. 2, s. 3 - 27.

[19] PVCS Version Manager Quick Start (1995). Intersolv, Inc.

[20] VHDL Modelling Guidelines (1994). European Space Research and Technology Centre, European Space Agency. S. 5 - 6.

[21] Laitinen, K. (1993) The principle of natural naming in software documentation. Espoo: VTT Research Notes 1498.

[22] Alapoikela, T (1996). Toiminnallinen mallinnus SA/VHDL-menetelmällä ja korkean tason synteesi. Oulu: Oulun Yliopisto, elektroniikan laboratorio. Diplomityö.

[23] Alanen, Jarmo (1995). CAN-ajoneuvojen ja koneiden sisäinen paikallisväylä. Tampere: VTT Automaatio. CAN väylä teollisuudessa -seminaari.

[24] Sipola, Matti (1996). Dmmtable-käyttöohje. Oulu: Oulun Yliopisto, sähkötekniikan osasto.

[25] Exceed X Server 5.0 Help (1995) Hummingbird Communications Ltd.

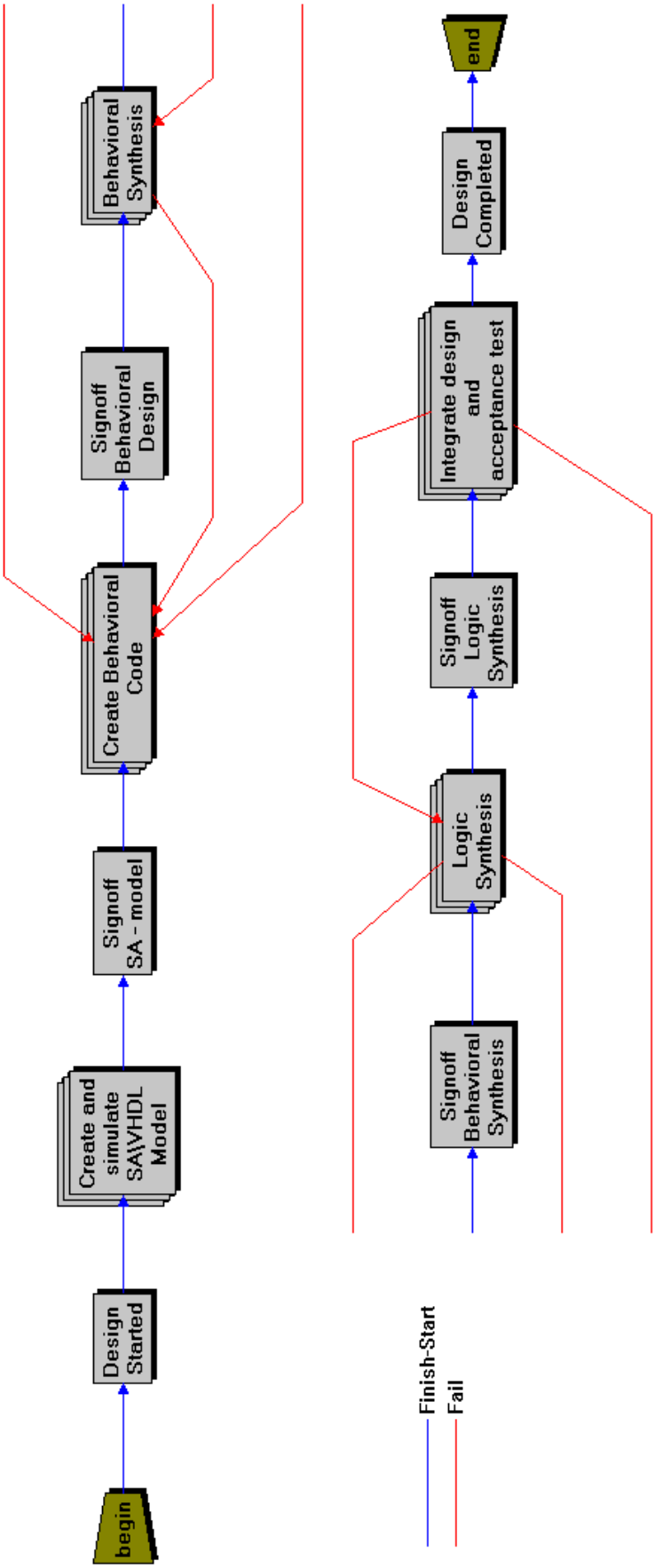
[26] VHDL Compiler reference (1995), version 3.3b. Synopsys, Inc., s.100 - 103.

Liite A: DMM.setup tiedosto

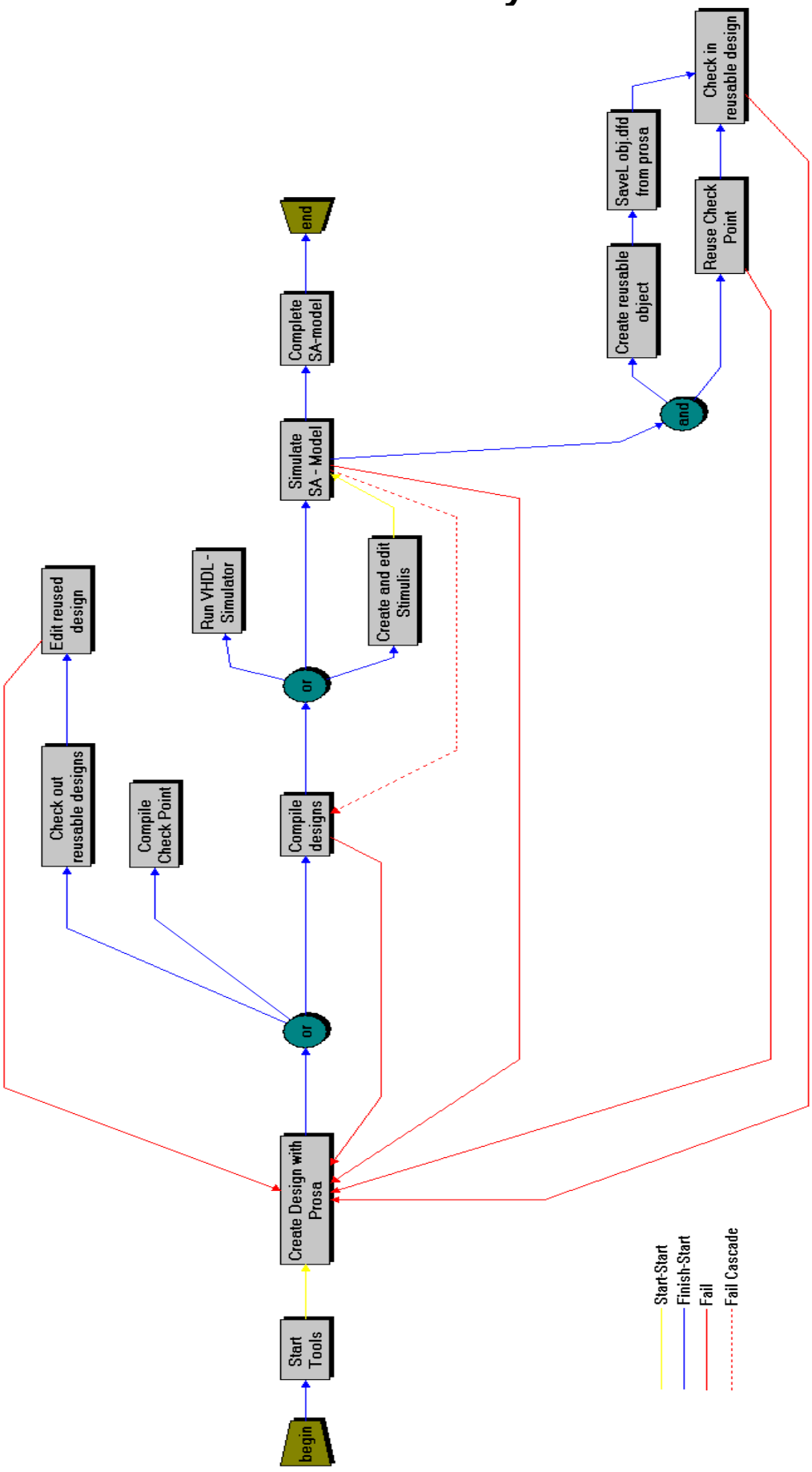
```
echo Reading DMM synthesis setup...

#DMM aliakset
setenv SYNOPSYS                 /usr/local/synopsys

#VSS simulator
setenv ARCH sparc
setenv SIM_ARCH sparc
setenv CDG_VHDLGEN $SYNOPSYS
setenv SGE_ROOT $SYNOPSYS/sparc/sge
setenv MWFONT_CACHE_DIR ~/.synopsys_windows/mwfc-4a5a
setenv LD_LIBRARY_PATH ".:$SYNOPSYS/sparc/sim/lib:/usr/openwin/lib:/usr/lib"

############################
# design root
############################

setenv DESIGNS ~/dmm_test
echo "Design path is "  $DESIGNS

############################
# Display for x-window
############################
alias set_display "setenv DISPLAY tko81:0"

#do not change names, just paths

setenv SCRIPTS $DESIGNS/scripts
setenv SADIR $DESIGNS/prosa
setenv SASIMULATE $DESIGNS/vhdl
setenv ANALYZEDIR $DESIGNS/simulate
setenv SIMULATEDIR $DESIGNS/simulate
setenv BEHAVIOR_CODE $DESIGNS/schedule
setenv LOGIC_CODE $DESIGNS/compile
setenv REUSE_DIR /pc/common/reuse

#for prosa
setenv PROSADIR /usr/prosa
setenv PROSAAUTH /usr/prosa
setenv PROSACFG $DESIGNS/prosa.cfg

#for creating behavioral code

alias EditAnalyzeScript "set_display;textedit $SCRIPTS/analyze.script"
alias EditStateAnalyzeScript "set_display;textedit $SCRIPTS/analyze.state.script"
alias XtermBehavior "set_display;cd $ANALYZEDIR;dxterm -T Schedule "
alias runanalyze "set_display;cd $ANALYZEDIR;dxterm -e dc_shell -f  $SCRIPTS/analyze.script"
alias runstateanalyze "set_display;cd $ANALYZEDIR; dxterm -e dc_shell -f
```

```
$SCRIPTS/analyze.state.script"
alias BehaviorEditFiles "set_display;cd $ANALYZEDIR;dxterm -T Edit_Files "
alias PrepareForSimulation "set_display;filemgr -d $ANALYZEDIR "
alias SynopsysHelp "set_display;$SYNOPSYS/iview2/bin/iview"

#for behavioral synthesis

alias EditScheduleScript "set_display;textedit $SCRIPTS/schedule.script"
alias runschedule "set_display;cd $BEHAVIOR_CODE;dxterm -e dc_shell -f $SCRIPTS/schedule.script"
alias BrowseBehavioral "set_display;filemgr -d $BEHAVIOR_CODE "

#for logic synthesis

alias EditCompileScript "set_display;textedit $SCRIPTS/compile.script"
alias EditStateCompileScript "set_display;textedit $SCRIPTS/compile.state.script"
alias runcompile "set_display;cd $LOGIC_CODE;dxterm -e dc_shell -f $SCRIPTS/compile.script"
alias runstatecompile "set_display;cd $LOGIC_CODE; dxterm -e dc_shell -f $SCRIPTS/compile.state.script"
alias BrowseCompile "set_display;filemgr -d $LOGIC_CODE "

#for simulation
alias EditStimulis "set_display;filemgr -d $SIMULATEDIR"
alias EditStimulisSA "set_display;filemgr -d $SASIMULATE"
alias runVHDL2000 "set_display;cd $SIMULATEDIR;red"
alias dxterm "xterm -sl 1000 -s -sb -j "
alias AnalyzeForVSS "set_display;cd $SIMULATEDIR;dxterm -e $SCRIPTS/simulate.script"
alias RunVHDLDebugger "set_display; cd $SIMULATEDIR;vhdldbx simulate"
alias EditSimulateScript "set_display;textedit $SCRIPTS/simulate.script"

#for running tools
alias runprosa "set_display;cd $SADIR;/usr/prosa/prosax context.dfd&"
alias RunVelvet "set_display;cd $SADIR;dxterm -T Velvet;"
alias velvet "~/tools/velvet/runvelvet"
alias RunDesignAnalyzer "set_display;cd $LOGIC_CODE;design_analyzer -f $SCRIPTS/.synopsys_dc.setup"
alias EditReuse "set_display;cd $REUSE_DIR;prosax"
echo "setup done."
```

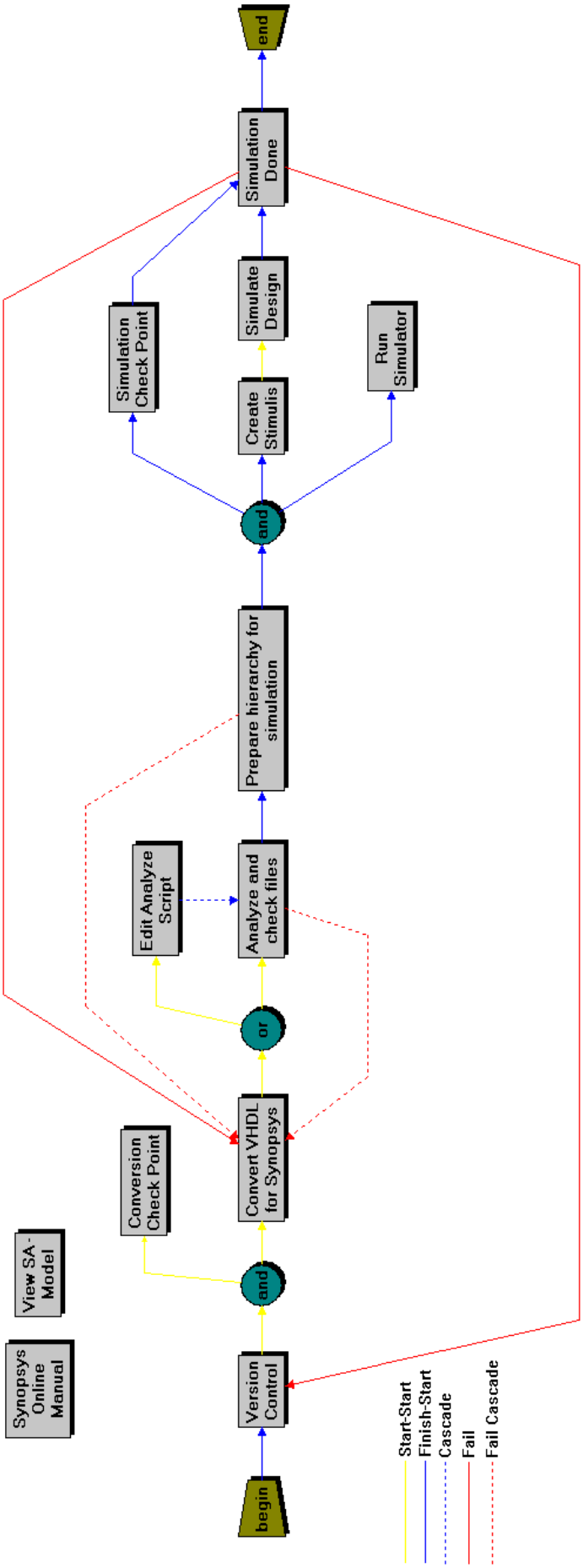
Liite B: Ylimmän tason työvuo



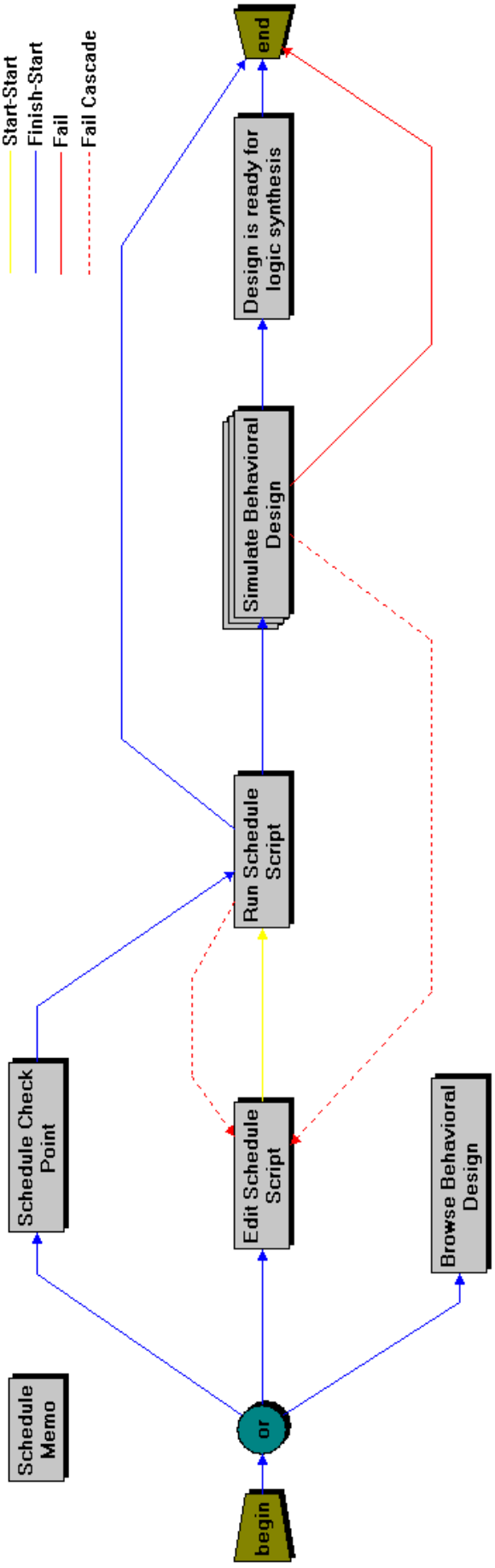
Liite C: SA/VHDL-työvuo



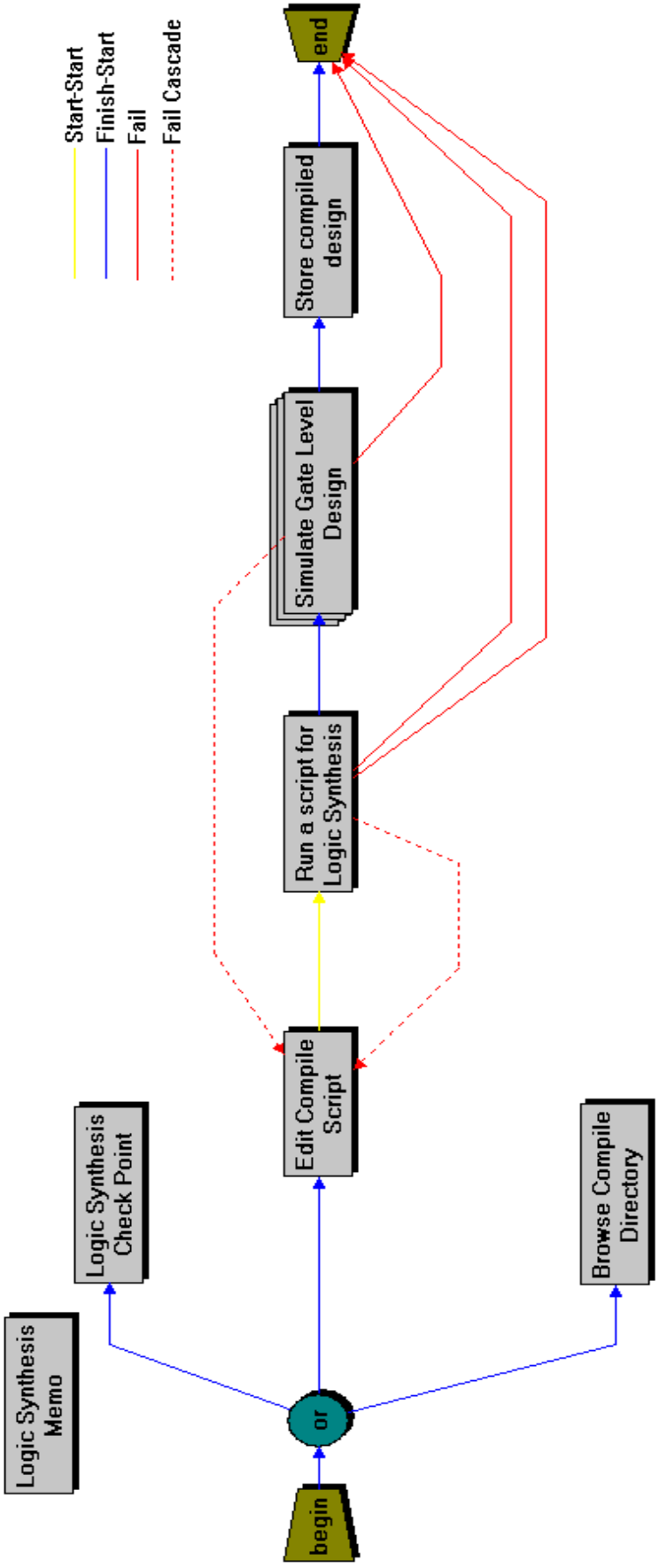
Liite D: Synteesisuunnittelutyövuo



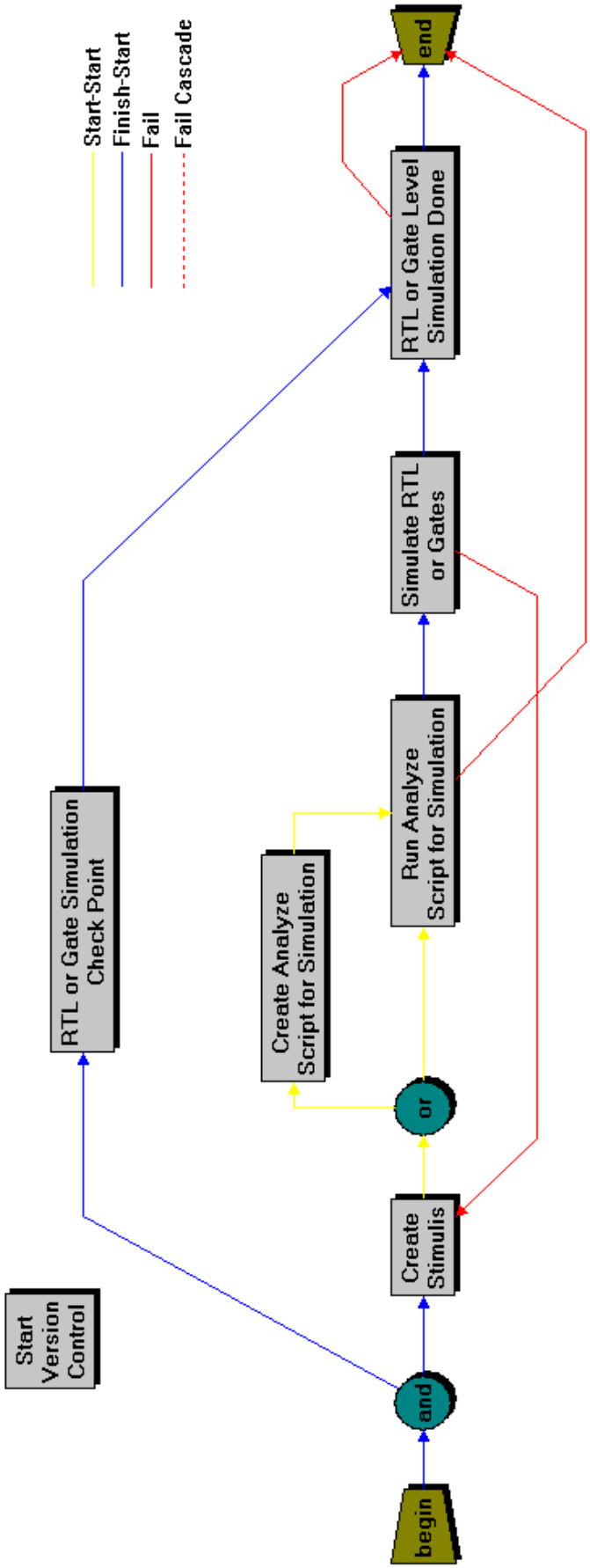
Liite E: Käyttäytymistason työvuo



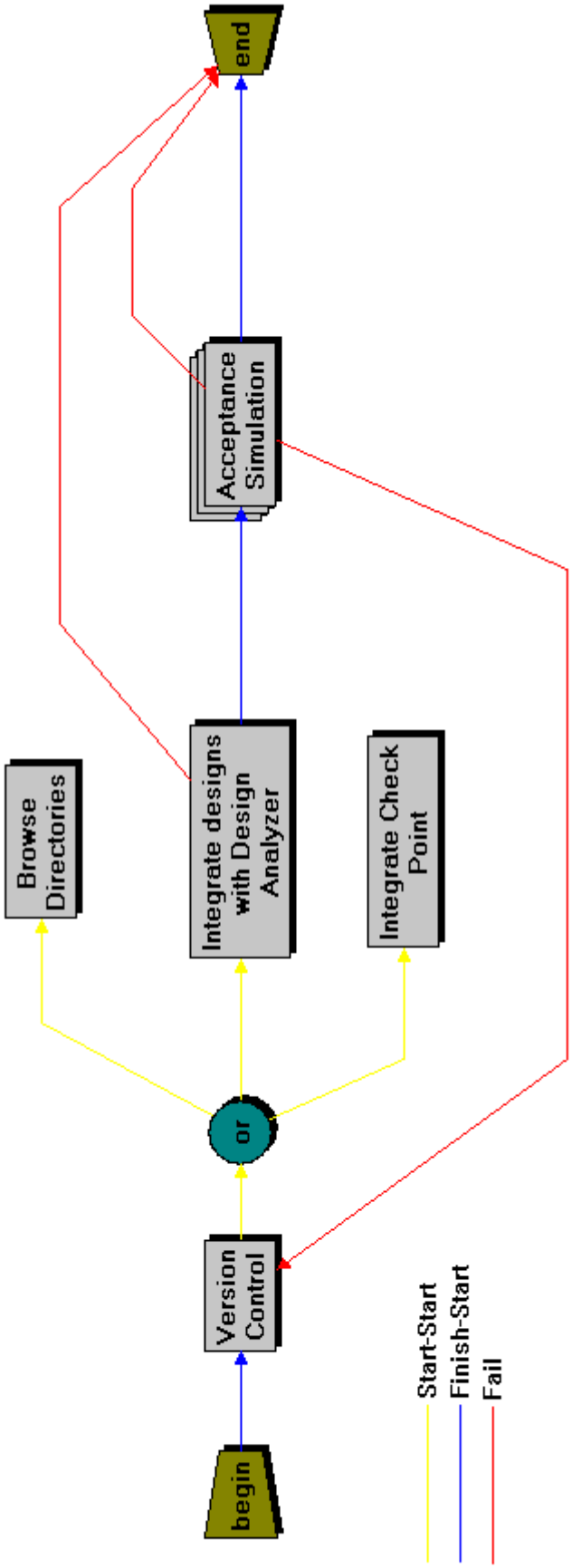
Liite F: Logiikkatason työvuo



Liite G: RTL- ja porttitason simulointityövuo



Liite H: Integrointitason työvuo



Liite I: Työvuoprosessien komennot ja viestit

Prosesseista on lueteltu prosessin nimi, lopetusviestit ja käärittykomento tai dmmexec-komento. Hierarkisissa prosesseissa pelkästään vuon nimi.

Ylimmän tason vuo

Design Started, Design Started, signoff started.txt
Create and simulate SA\VHDL Model, done, savhdl.wdu
Signoff SA - model, SA Model is ready, signoff sa_ready.txt
Create Behavioral Code, done, codecreate.wdu
Signoff Behavioral Design, Behavioral Design Ready, behavior.txt
Behavioral Synthesis, back , advance, behavior.wdu
Signoff Behavioral Synthesis, Behavioral Synthesis Ready, rtlready.txt
Logic Synthesis, done , back , logic.wdu
Signoff Logic Synthesis, Logic synthesis ready, gates.txt
Integrate design and acceptance test, Failed, advance, complete.wdu
Design Completed, design is ready, complete.txt

SA/VHDL - taso

Start Tools , OK, dmmexec -oe -w -i=%o -p="StartTools" "tools.exe" "Ok" "Ok" Create Design with Prosa, OK, wrapper ""
Compile Designs, dmmexec -oe -s -w -i=%o -p="CompileDesigns" "velvet.exe" "OK" "Failed"
Create and Edit Stimulis, dmmexec -n -w -i=%o -p="CreateStimulis" "xstart.exe dcaesun5.xs -c EditStimulisSA" "Done" "Done"
Run VHDL - Simulator, dmmexec -oe -w -i=%o -p "Simulator" "d:\users\dmm\simulate" "OK" "OK"
Simulate SA - Model, Run another compile Back to editing SA - model Create Reusable Object SA - model is completed.,wrapper ""
Complete SA - model, dmmexec -n -w -i=%o -p="CompleteSAModel" "d:\users\dmm\dmmtable\DMMTABLE 104" "OK" "OK"
Compile Check Point, dmmexec -n -s -w -i=%o -p="CompileCheckPoint" "DMMTABLE 101" "Check List OK" "Failed"
Edit and Check out reusable designs, dmmexec -n -w -i=%o -p="CheckOutDesign" "sa-manag" "Ok" "Ok"
Edit Reused Design, dmmexec -w -oe -i=%o -p="EditReusedDesign" "reuse.exe" "Continue" "Continue"
Create Reusable Object, Check in reusable design, wrapper ""
SaveL obj.dfd from prosa, ok, wrapper ""
Reuse Check Point, dmmexec -n -s -w -i=%o -p="ReuseCheckPoint" "DMMTABLE 102" "Check List Done" Failed
Check in reusable design, dmmexec -n -w -i=%o -p="CheckInDesign" "sa-manag.exe" "Done" "Done"

Synteesisuunnittelun - työvuo

Version Control, dmmexec -n -w -i=%o -p="CreateVersionControl" "pvcs.exe" "Ok" "Ok", avail
View SA - model, dmmexec -n -w -i=%o -p="StartProsa" "xstart dcaesun5 -c runprosa" "Ok" "Ok", avail
Synopsys Online Manual, dmmexec -n -w -i=%o -p="SynopsysOnLineManual" "xstart dsutsun1 -c SynopsysHelp" "Ok" "Ok"
Conversion Check Point, dmmexec -n -w -i=%o -p="ConversionCheckPoint" "DMMTABLE 103" "Check List OK" "Check List OK"
Convert VHDL for Synopsys, dmmexec -oe -w -i=%o -p="ConvertVHDL" "convert.exe" Ok Ok, (auto avail)
Edit Analyze Script, dmmexec -oe -w -i=%o -p="EditAnalyzeScript" "eanalyze.exe" "OK" "OK" (avail)
Analyze and check files, dmmexec -oe -s -w -i=%o -p="AnalyzeAndCheck" "analyze.exe" "New Analyze" "Simulate" "Wait here and try again"
Prepare Hierarchy for simulation, simulate, back to edit, wrapped ""
Simulation check point,dmmexec -n -w -i=%o -p="SimulateCheckPoint" "d:\users\dmm\dmmtable\DMMTABLE 106" "Check List OK" "Check List OK"
Create Stimulis, dmmexec -n -w -i=%o -p="CreateStimulis" "xstart.exe dcaesun5 -c EditStimulis" "Done" "Done" (avail)

Simulate Design, OK, wrapper ""
Run Simulator, dmmexec -oe -w -i=%o -p="RunSimulator" "simulate" "OK" "OK", avail
Simulation done, Back to edit code, Go back to version control, Advance Synthesis, wrapper ""

Käyttäytymistason vuo

Schedule Memo, dmmexec -n -w -i=%o -p="ScheduleMemo" "notepad.exe schedule.txt" "OK" "OK"
Browse behavioral design, dmmexec -n -w -i=%o -p="BrowseBehavioralDesign" "xstart.exe dsutsun1 -c BrowseBehavioral" "OK" "OK"
Edit Schedule Script, dmmexec -oe -w -i=%o -p="EditScheduleScript" "set_sche" "OK" "OK"
Run Schedule Script, Edit Script, Go to edit code, Scheduling done - simulate, dmmexec -oe -s -w -i=%o -p="RunScheduleScript" "schedule" "Edit script" "Go to edit code" "Scheduling done - simulate" "Wait here and try again"
Schedule Check Point, dmmexec -n -w -i=%o -p="ScheduleCheckPoint" "DMMTABLE 107" "Check List OK" "Check List OK"
Design is ready for logic synthesis, dmmexec -n -w -i=%o -p="DesignReadyForLogicSynthesis" "pvcs" "Advance to login synthesis" "Advance to login synthesis"
Simulate Behavioral Design, Back Failed Advance, sim_rtl.wdu

Logiikkasynteesin tyovuo

Logic synthesis Memo, dmmexec -n -w -i=%o -p="LogicSynthesisMemo" "notepad.exe compile.txt" "OK" "OK"
Browse compile directory, dmmexec -n -w -i=%o -p="BrowseCompileDirectory" "xstart.exe dsutsun1 -c BrowseCompile" "OK" "OK"
Edit Compile Script, dmmexec -oe -w -i=%o -p="EditCompileScript" "set_comp" "OK" "OK"
Run a script for logic synthesis, dmmexec -oe -s -w -i=%o -p="RunLogicSynthesis" "compile" "Edit script" "Synthesis successfull - simulate" "Go to behavioral synthesis" "Go to code create"
Simulate Gate Level Design, Advance, Back, Failed, sim_rtl.wdu
Logic Synthesis Check point, dmmexec -n -w -i=%o -p="LogicCheckPoint" "DMMTABLE 108" "Check List OK" "Check List OK"
Store Compiled Design, done, wrapper ""

RTL ja porttitason simulointivuo

Start Version Control,dmmexec -n -w -i=%o -p="SimulateVersionControl" "pvcs.exe" "Ok" "Ok"
Create Stimulis, dmmexec -n -w -i=%o -p="CreateStimulisRTL" "xstart.exe dcaesun5 -c EditStimulis" "Done" "Done"
Create Analyze Script for simulation, dmmexec -n -w -i=%o -p="CreateAnalyzeScript" "xstart.exe dcaesun5 -c EditSimulateScript" "Done" "Done"
Run Analyze Script for Simulation, dmmexec -n -w -i=%o -p="RunAnalyzeScriptForSimulation" "xstart.exe dcaesun5 -c AnalyzeForVSS" "Analyze successfull - simulate" "Analyze failed - edit script" "Analyze failed - go back to design"
Simulate RTL or Gates, dmmexec -n -s -w -i=%o -p="SimulateRTLorGates" "xstart dsutsun1 -c RunVHDLDebugger" "Simulations Done" "New Simulation"
RTL or Gate level simulation done, Go back to design, Go back to Edit Code, Advance Synthesis, wrapper ""
Integrointi vuo

Version Control, dmmexec -n -w -i=%o -p="IntegrateVersionControl" "pvcs.exe" "Ok" "Ok"
Integrate designs with Design Analyzer, dmmexec -n -s -w -i=%o -p="IntegrateDesigns" "xstart dsutsun1 -c RunDesignAnalyzer" "There is just one design file now" "Go back"
Integrate check point, dmmexec -n -w -i=%o -p="ScheduleCheckPoint" "DMMTABLE 109" "Check List OK" "Check List OK"
Acceptance simulation, Advance, Back, Failed, sim_rtl.wdu
Browse directories, dmmexec -n -w -i=%o -p="BrowseDirectories" "xstart.exe dsutsun1 -c BrowseCompile" "OK" "OK"