**KAUNO TECHNOLOGIJOS UNIVERSITETAS**
**INFORMATIOS FAKULTETAS**
**PROGRAMŲ INŽINERIJOS KATEDRA**

**Vaidotas Pečkys**

# Sparčiosios magistralės aukšto abstrakcijos lygio modelio sudarymas ir analizė

Magistro darbas

**Vadovas**
**doc. E. Bareiša**

**KAUNAS, 2005**

**KAUNO TECHNOLOGIJOS UNIVERSITETAS**
**INFORMATIOS FAKULTETAS**
**PROGRAMŲ INŽINERIJOS KATEDRA**

**TVIRTINU**
**Katedros vedėjas**
**doc. E. Bareiša**

# Sparčiosios magistralės aukštame abstrakcijos lygyje analizė ir tyrimas

Informatikos magistro baigiamasis darbas

**Kalbos konsultantė**
**Lietuvių k. katedros lekt.**
**dr. J. Mikelionienė**

**Vadovas**
**doc. E. Bareiša**

**Recenzentas**
**doc. R. Marcinkevičius**

**Atliko**
**IFM-9/5 gr. stud.**
**V. Pečkys**

**KAUNAS, 2005**

# SUMMARY

In this work was studying literature related to object orientated programming tools for hardware design, capabilities for modeling and synthesis of high-level models of abstraction. It was founded-out the operating principles of high-speed bus and created prototype of such bus in TLM level. It was created methodology for transformation of high-speed bus prototype to RTL level. This methodology was used for transformation of high-speed bus prototype to RTL level. Transformed module was synthesized to gate level. Simulation speed of high-speed bus model in TLM was compared with simulation speed of model in behavioral level. It was demonstrated universality and reuse capabilities of TLM models.

# SANTRAUKA

Šio darbo metu susipažinta su literatūra, susieta su aparatūrinės įrangos projektavimu objektinio programavimo priemonėmis, tirtas šių aprašų modeliavimo ir sintezavimo galimybės. Susipažinta su sparčiosios magistralės veikimo principais ir sukurtas sparčiosios magistralės prototipas TLM lygyje, Sukurta metodika sparčiosios magistralės aprašo transformavimui iš operacijų lygio į tarpregistrinį. Ši metodika naudota sparčiosios magistralės TLM lygio prototipo transformacijai. Transformuotas aprašas susintezuotas ir gauti ventiliniai vaizdai. Palygintas TLM ir RTL lygio modulių simuliacijos greitis. Pademonstruotas TLM modulių universalumas, pakartotinio panaudojimo galimybės.

# Turinys

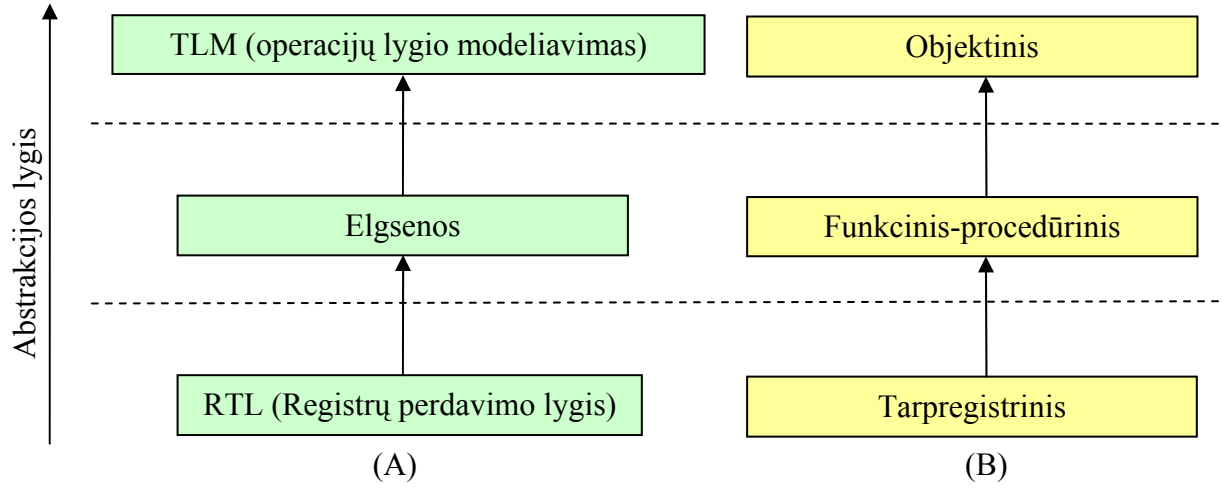
# Paveikslų sąrašas



# Lentelių sąrašas

# 1 Įvadas

Keliolika metų atgal aparatūrinės ir programinės įrangos kūrimas buvo visiškai skirtingos sąvokos. Programuotojas prie žalių kompiuterių ekranų ir aparatūros specialistas su lituokliu rankose neturėjo nieko bendra. Tačiau su laiku išryškėjo daug bendrumų tarp šių sričių. Programinės ir aparatūrinės įrangos projektavime atsirado daug sąlyčio taškų. Galų gale aparatūros projektavimo kai kurie etapai tapo tokie patys, kaip ir programinės įrangos projektavimo. Dabar gi, aparatūros projektavimas neįsivaizduojamas be programavimo ir kodo sintezavimo.




**1 pav. Perėjimas nuo žemo iki aukšto abstrakcijos lygio. (A) Aparatinės įrangos. (B) Programinės įrangos.**

Aparatūros ir programinės įrangos specialistai visada stengėsi supaprastinti ir pagreitinti projektavimo procesą, sumažinti didelių sistemų sudėtingumą, kuo labiau sistemos dalis pagaminti vieną nuo kitos nepriklausomas, supaprastinti naujų komponentų prijungimą, kelti abstrakcijos lygį. Tai buvo viena iš svarbiausių problemų ir klausimų tobulinant kūrimo procesą. Kaip programinė įranga turėjo savo abstrakcijos lygio kilimo kelią nuo Assembler programavimo kalbos iki Turbo Pascal, Java ar C#, taip ir aparatinės įrangos projektavimas turi savo kelią ir istoriją link aukštesnio abstrakcijos lygio. Projektuojant sudėtingas sistemas elgsenos ir RTL lygiuose, atsirado poreikis aukštesniam abstrakcijos lygiui, kad būtų galimybė projektuoti sistemas nesigilinant į smulkmenas, o koncentruoti dėmesį tik į bendras ir svarbiausias sistemos operacijas.

Aparatūroje yra žinomi trys abstrakcijos lygiai – RTL (*register transfer level)*, elgsenos (*behavioral*) ir TLM (*transaction level modeling*). Pirmieji du yra apibrėžti ir standartizuoti. Tuo tarpu trečiasis – TLM - dabartiniu metu bandomas apibrėžti, patvirtinti kaip standartą. Ypatingas dėmesys skiriamas sintezės galimybėms studijuoti. O kol tiesioginių sintezavimo įrankių TL lygiui nėra, šiame lygyje modeliuojamos, verifikuojamos ir testuojamos sistemos dėl šio abstrakcijos lygio spartos ir lankstumo.

Šio darbo tikslas – susipažinti su aparatūros modeliavimu abstrakčiame lygyje, palyginti TLM modulio simuliavimo greitį ir efektyvumą lyginant su žemo abstrakcijos lygio greičiu bei efektyvumu, sukurti metodiką aukšto lygio duomenų magistralės aprašo sintezei, bent iš dalies padengiant šioje srityje esančia spragą.

Šio tikslo pasiekimo uždavinį galima suskirstyti į tokias dalis:

- Susipažinti su literatūra, susieta su aparatūrinės įrangos projektavimu objektinio programavimo priemonėmis
- Ištirti šių aprašų modeliavimo ir sintezavimo galimybes
- Susipažinti su sparčiosios magistralės veikimo principais ir sukurti sparčiosios magistralės prototipą TLM lygyje
- Pademonstruoti TLM modulių universalumą, lankstumą, pakartotinio panaudojimo galimybes
- Sukurti metodiką sparčiosios magistralės aprašo transformavimui iš operacijų lygio į tarpregistrinį
- Šią metodiką panaudoti TLM lygio sparčiosios magistralės prototipo transformacijai.
- Palyginti TLM ir RTL lygio modulių simuliacijos spartą
- Susintezuoti transformuotą elgsenos lygio sparčiosios magistralės aprašą

# 2 Analitinė dalis

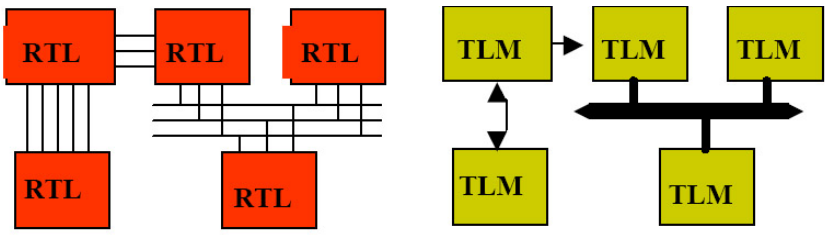
## 2.1 Modeliavimas tarpregistrinio perdavimo lygyje

SoC projektuotojai tradiciškai aparatinę įrangą projektuoja funkciniame lygyje. Tipiškai elgsenos ir RTL aprašas yra pradinis taškas architektūros realizacijai silicyje, nes procesai, įrankiai, metodologija nuo RTL aprašo iki silicio mikroschemos yra subrendęs, ištobulintas ir palaikomas EDA įrankių skaičiavimams, analizei, sintezei ir gamybai. RTL ir elgsenos lygio modelių aprašai dažnai tebenaudojami kaip analizės, verifikavimo ir trasavimo sujungimo taškas, kas labai sulėtina mikroschemų projektavimo procesą. Dėl šio lygio aprašų lėtos simuliacijos šio tipo aprašai nėra ypač tinkami verifikavimui. Iš ties, RTL lygio programavimo kalbos buvo suprojektuotos modelių kūrimui, kurie tiksliai atlieka savo funkciją, realizuoja eigą, ciklus, laiką, kas įtakoja, jog loginė sintezė iš tokio aprašo bus sąlyginai nesudėtinga. Tačiau, šios detalės turi keletą nepageidaujamų efektų, kuomet prasideda modelio verifikavimas ir su trasavimu susijusios operacijos – tai lėta simuliacija, milijonai nesuprantamų laikinių diagramų, kas apsunkina būtiną atlikti analizę ir įvertinimą.

## 2.2 Modeliavimas operacijų lygyje

Augant projektuojamų įrenginių sudėtingumui, ventilių skaičiui, atsirado didelis poreikis sistemas projektuoti aukštame abstrakcijos lygyje ir taip supaprastinti projektavimo procesą. Aparatūros projektavime tai vadinama modeliavimu operacijų lygyje (*Transaction Level Modeling*).

Aukštas sistemos abstrakcijos lygis – tai svarbi sistemos savybė, kurią išnaudojant labai supaprastinamas sudėtingų sistemų projektavimas, realizavimas, testavimas ir verifikavimas. *Behavioral* ir *RTL* abstrakcijos lygiai padeda paslėpti modelio detales esančias žemiau laikrodis-ciklas (clock-cycle) lygio, tokias kaip ventiliai, skaičiavimo vėlavimai. Tačiau struktūriškai šie *RTL* ir elgsenos lygiai yra išvadų tikslumo (*pin-accurate*), t.y. sujungimai ir registrai yra patalpinti ant struktūros ribų, tuo apribodami kelią, kuriuo duomenys gali įeiti ir išeiti iš modulio.




**2 pav. RTL ir TLM lygio modulių sujungimo paprasčiausia schema**

Projektuotojai TLM lygyje gali modeliuoti modulių komunikavimą abstrakčiame lygyje atskiriant bendravimo realizaciją nuo pačių modulių funkcionalumo. Šis atskyrimas supaprastina modelį, pagreitina simuliavimą, palengvina komponentų pakartotiną panaudojimą, tuo pačiu sudėtingų sistemų projektavimą.

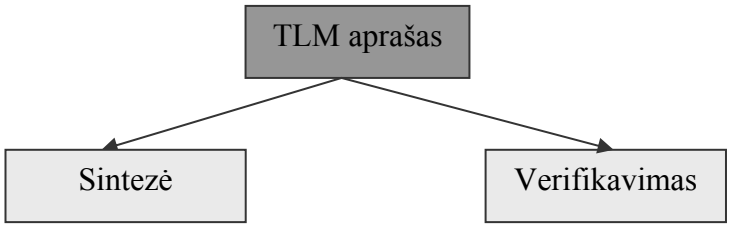
TLM lygyje operacija pradedama iškviečiant sąsajos funkciją iš modulio kanalo. Šios funkcijos savyje turi žemo lygio detales reikalingas informacijos mainams. Kitais žodžiais tariant – operacijų lygyje labiau koncentruojamasi į duomenų perdavimo funkcionalumą – kokie duomenys iš kur ir į kur perduodami – ir mažiau į jų tikrą realizavimą (t.y. į apsikeitimo informacija protokolą).

Šis požiūris įgalina kiekvieno komponento atskirą modeliavimą įvairiuose abstrakcijos lygiuose. Taip pat programuotojui palengvina eksperimentavimo procesą, pavyzdžiui pajungiant savąjį modelį prie skirtingų architektūrų magistralių (kurios palaiko reikalingas sąsajas) be jokių modelių kodo koregavimų tam, kad galėtų „susikalbėti“ su kitais moduliais per skirtingas magistrales. Per duomenų magistralę bendraujama kreipiantis į bendras sąsajas.

Modulius, kurie naudoja komunikavimo kanalus, lengviau projektuoti. Projektuojant elgsenos, TLM ir RTL, nereikia galvoti apie tokios detales, kaip operacijų vėlavimai, bet vis tik, dar reikalinga, keičiant bendravimo protokolą mažiausia modifikuoti susijungimo signalus. Tuo metu, kai TLM lygio moduliams modifikavimas gali būti lengvai atliktas naudojant sąsajas ir kanalus.

Modeliavimas TLM lygiu spartesnis dėl to, kad paslepiamos „neįdomios“ detalės. Pavyzdžiui, realiame pasaulyje ilga monopolinė (*burst)* operacija gali trukti daugelį laikrodžio ciklų. Daugelio šių ciklų metu magistralė dirbs rutininį darbą, o tie magistralės klientai, kurie lauks priėjimo prie magistralės, paprasčiausia lauks. Jeigu mes į monopolinę operaciją pažvelgtume kaip į vieną operaciją, tada nereikėtų veltui leisti laiką tiems „neįdomiems“ magistralės sisteminio laikrodžio ciklams. Priklausomai nuo to, ar reikalingas ciklo tikslumo modeliavimas ar ne, gali būti pasirinkta keletas strategijų, kurias pritaikius gaunamas žymus simuliacijos greičio sumažėjimas. Netgi, jeigu TLM lygiu modeliuojant prisireikia simuliuoti ciklo tikslumu, simuliacija vyksta daug sparčiau nei RTL ar elgsenos modelio. TLM sintezei kol kas mažai naudojamas dėl sintezės įrankių nebuvimo.

Paprastai TLM modeliai naudojami funkciniam modeliavimui (tiek laikiniam, tiek belaikiui), platformų modeliavimui, testavimo aplinkų konstravimui, verifikavimui. Taigi, apibendrinant TLM taikymus galima sutraukti į dvi sritis – sintezę ir verifikavimą.




**3 pav. Labiausia paplitusios TLM aprašų pritaikymo sritys**

Priežastys, kodėl TLM lygio modeliai yra tokie svarbūs:

- Sąlyginai paprasta programuoti, suprasti ir išplėsti
- Įmanoma tiksliai modeliuoti tiek aparatinės, tiek programinės įrangos sistemų komponentus
- Galimybė sukonstruoti veikiančią sistemą labai ankstyvoje projektavimo fazėje, kas įgalina išbandyti įvairias architektūros alternatyvas ir rasti architektūros kompromisą ankščiau, nei architektūros keitimas tampa pernelyg vėlus arba tampa per brangu tai atlikti
- Pakankamai greita ir tikslu programinės įrangos patvirtinimui (*validate*) ankščiau, nei detalios aparatūrinės įrangos modelių realizacijos yra parengtos

Didelės apimties modeliams reikalingas labai greitas simuliacijos greitis tam, kad būtų galima didelės apimties programinę įrangą vykdyti lygiagrečiai su aparatinės įrangos modeliu. Be to, modeliai turi veikti ciklo tikslumu tam, kad tiek aparatinės, tiek programinės įrangos specialistai galėtų pasitikėti vieni kitų kuriamais modeliais. Taigi, tai leidžia ankstyvą architektūros suderinimą gerokai iki to, kol prasidės modelių detalaus realizavimo procesas.

## 2.3 TLM lygio aprašų savybės

TLM lygyje modelių bendravimui naudojami ne signalai, o operacijos. Operacijos modeliuojamos naudojant kreipinį į funkciją, kuri perduoda tiek kontrolės informaciją, tiek pačius duomenis iš vieno modelio į kitą. TLM modelis nėra išvadų tikslumo (*pin-accurate)* – visi duomenys perduodami operacijos metu, dėl to informacija gali būti supakuota ir perduota daug efektyviau.

TLM modelis naudoja aukšto lygio duomenų tipus dažniau, nei žemo lygio bitų vektorius, keturių reikšmių bitus – tai yra tuos kurie dažniausia naudojami aparatinės įrangos modeliavime.

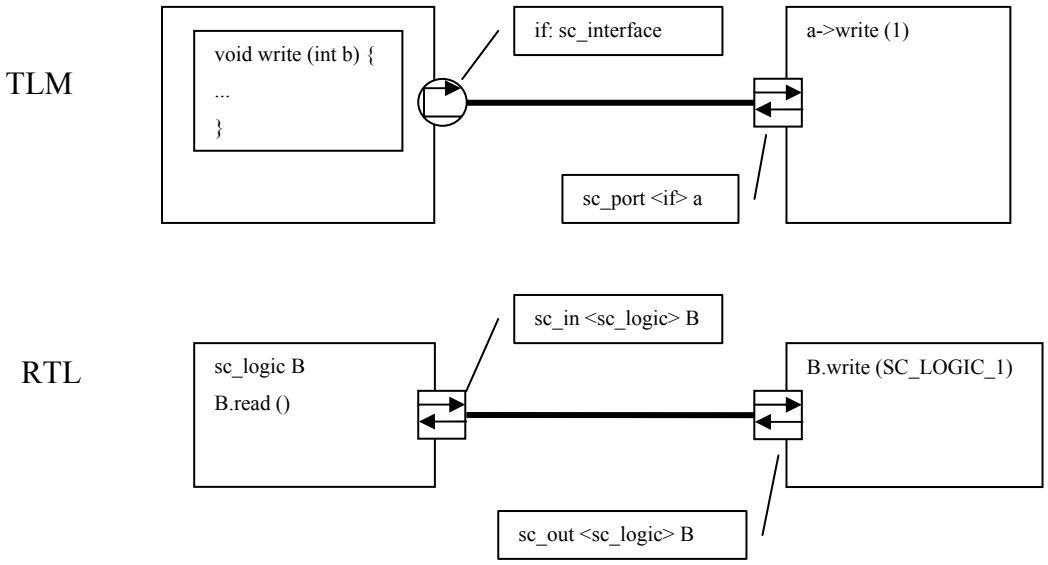
Nuorodos į duomenis perduodamos iš vieno modelio į kitą operacijos metu. Dėl to vienam modeliui yra daug paprasčiau duomenų blokus nukopijuoti iš kito modulio.

SystemC dinaminio jautrumo (*dynamic sensitivity*) savybė naudojama pašalinti nereikalingus procesų aktyvaciją. Tai leidžia pvz. suspenduoti procesą jam pasiuntus reikalavimą (*request*) tol, kol pasibaigs operacijos vykdymas. Procesas nebus aktyvuotas nepriklausomai nuo to, ar operacija vyks vieną ar daugiau ciklų ir nepriklausomai ar modulį pasieks sinchronizacijos signalas ar ne. Tipinis RTL ciklo tikslumo modelyje procesas yra aktyvuojamas laikrodžio nepriklausomai nuo to ar jis turi darbo. Naudojant dinaminį jautrumą mes gauname ryškų modeliavimo paspartėjimą lyginant su RTL modeliu, nes modelio procesai dabar aktyvuojami tik tada, kai turi realaus darbo.

Kai kurie modeliai RTL lygyje turėdavo savo nuosavą procesą. Tačiau dažnai TLM lygyje jie tampa tokie paprasti, jog nuosavas procesas tampa nereikalingas. Visas reikalingas darbas atliekamas gavus reikalavimo signalą (*request*).

TLM lygyje, kur įmanoma naudojami SC_METHOD procesai vietoje SC_THREAD. SC_METHOD paprastai naudoja daug mažiau atminties ir veikia kur kas greičiau nei SC_THREAD.

Aukščiau aprašytų TLM modeliavimo savybių išnaudojimas lemia labai greitą TLM modelių simuliaciją lyginant su tipiniu RTL modeliu.




**4 pav. Duomenų mainų principo TLM ir RTL lygiuose palyginimas**

Pagrandinis TLM išskirtinumas yra tai, kad bendravimas tarp modulių vyksta ne per signalus ir prievadus, o per kanalus (*channel*) ir sąsajas (*interface*). Paveiksle Nr. 1 galima matyti, jog RTL komponentai yra sujungti paprasčiausiais signalais. Tuo tarpu TLM moduliai sujungti kanalais ir sąsajomis. TLM lygyje nėra fizinio ryžio tarp modulių, kaip RTL – signalai. Tam yra sąsajos funkcijos.

## 2.4 TLM modelių sintezė

Kadangi TLM modeliavimas įmanomas praktiškai tik su SystemC, taigi, kalbant apie TLM lygio aprašo lygio sintezavimą, kalbėsime apie TLM lygio SystemC aprašus.

Aparatūrinei įrangai projektuoti iki pasirodant SystemC buvo naudojamos dvi programavimo kalbos VHDL ir Verilog.

SystemC turi nemažai privalumų prieš senąsias aparatūrinės įrangos projektavimo kalbas. Pirmiausia ji yra laisvai platinama. Sintaksė analogiška C++. Tai suartina programinės ir aparatūrinės įrangos specialistus. Programuojant abstrakčiame lygyje jie gali dirbti vienu metu. Priešingai nei su SystemC, naudojant standartinius VHDL ir Verilog neįmanomas modeliavimas TLM lygyje. Tai ir yra didelis SystemC privalumas.

Tačiau tai išryškina kitą problemą. Aparatūros aprašo neužtenka sumodeliuoti. Turi egzistuoti galimybė jį sintezuoti iki ventilinio lygio. Nes priešingu atveju, jeigu aprašo negalima paversti loginiais elementais, modeliavimas TLM lygyje netenka dalies patrauklumo. Tai yra silpniausia SystemC TLM modelių vieta – sudėtingas sintezavimas. Nes šiuo metu TLM modeliai dažniausiai naudojami tik modeliavimui dėl sintezės įrankių nebuvimo. Synopsys CoCentric kompiliatorius palaiko SystemC, bet tik RTL ir elgsenos lygiuose.

Galbūt tai ir yra priežastis, kodėl SystemC plitimas nėra toks intensyvus kaip norėtųsi. Projektavimui RTL ir elgsenos lygiuose plačiausia tebenaudojami VHDL ir Verilog. Tikėtina, kad panaši situacija išliks tol, kol nebus sukurta objektinius SystemC aprašus palaikančių sintezatorių.

Vis tik, SystemC rado savo nišą. Tokios kompanijos kaip AMD, Samsung, Fujitsu, Amasso naudoja SystemC verifikavimui, testavimo aplinkoms programuoti.

Egzistuoja trys būdai, kaip galima sintezuoti objektinį TLM lygio SystemC kodą:

- Rankomis SystemC TLM modelį transformuoti į sintezuojamą RTL ar elgsenos kodą. Ir šį kodą sintezuoti. Šis būdas gana sudėtingas ir reikalaujantis daug darbo ir pastangų. Nors ir egzistuoja metodikos, kaip tai galima atlikti. Bet ji gana paini ir komplikuota. Be to, lengva įvelti naujų klaidų.
- Automatiškai SystemC TLM kodą transformuoti į sintezuojamą RTL ar elgsenos lygį ir jam naudoti įprastus sintezavimo įrankius. Tam galima naudoti SystemC kalbos praplėtimą SystemC-Plus. Vis tik, čia taip pat yra reikalingas rankinis kodo koregavimas.
- Tiesiogiai naudoti elgsenos (*behavioral*) sintezės įrankius SystemC TLM lygio modeliui. Ši galimybė sąlyginė ir labai ribota, nes pvz. Synopsys CoCentric

kompiliatorius siūlo vos keletą sintezuojamų objektinių SystemC konstrukcijų. Aprašą su šiomis konstrukcijomis netgi sunku pavadinti objektiniu.

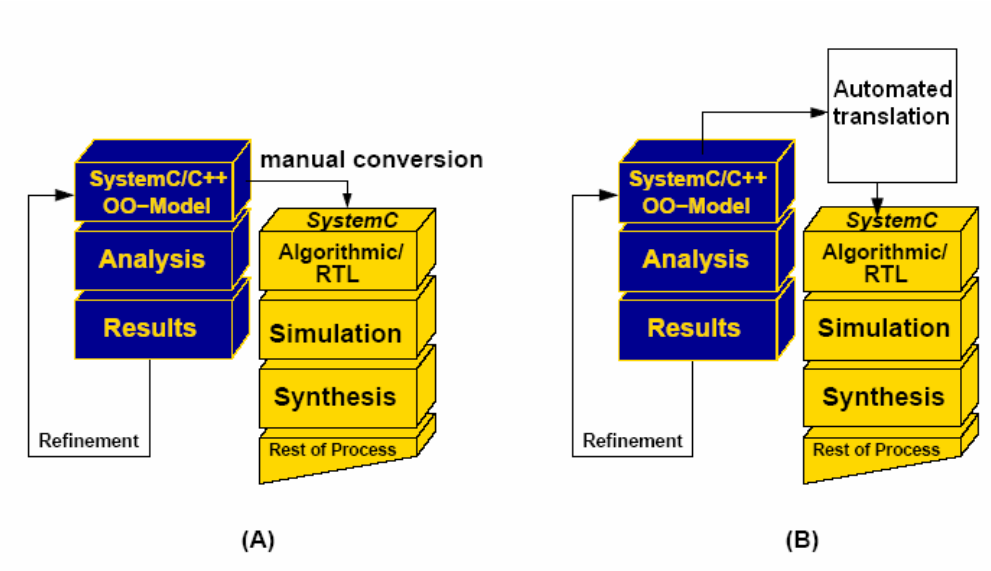
Darant išvadas galima pasakyti, kad iš esmės tradiciniai sintezės algoritmai negali suprasti objektiškai orientuotų aprašų.

Žemiau nupieštame paveiksle pavaizduotas sistemos kūrimo procesas nuo architektūros iki realizacijos silicyje t. y. iki aprašo sintezavimo fazės.

| | | |
|---|---|---|
| UML, Matlab | Algoritmo aprašas<br>Pagrindas: funkcijos | Kreipiniai į funkcijas,<br>funkcinis |
| SystemC | Programuotojo požiūrio aprašas<br>Pagrindas: atminties žemėlapis | Bendra magistralė,<br>architektūrinis |
| | Programuotojo požiūrio aprašas +<br>sinchronizavimas<br>Pagrindas: sinchronizuotas protokolas | Bendra magistralė<br>Apytikris sinchronizavimas |
| | Ciklų kvietimas<br>Pagrindas: laikrodžio frontas | Operacijos žodžiais<br>Ciklo tikslumas |
| VHDL, Verilog | RT lygis<br>Pagrindas: aparatinės įrangos realizacija | Signalas/bitas<br>Ciklo tikslumas |

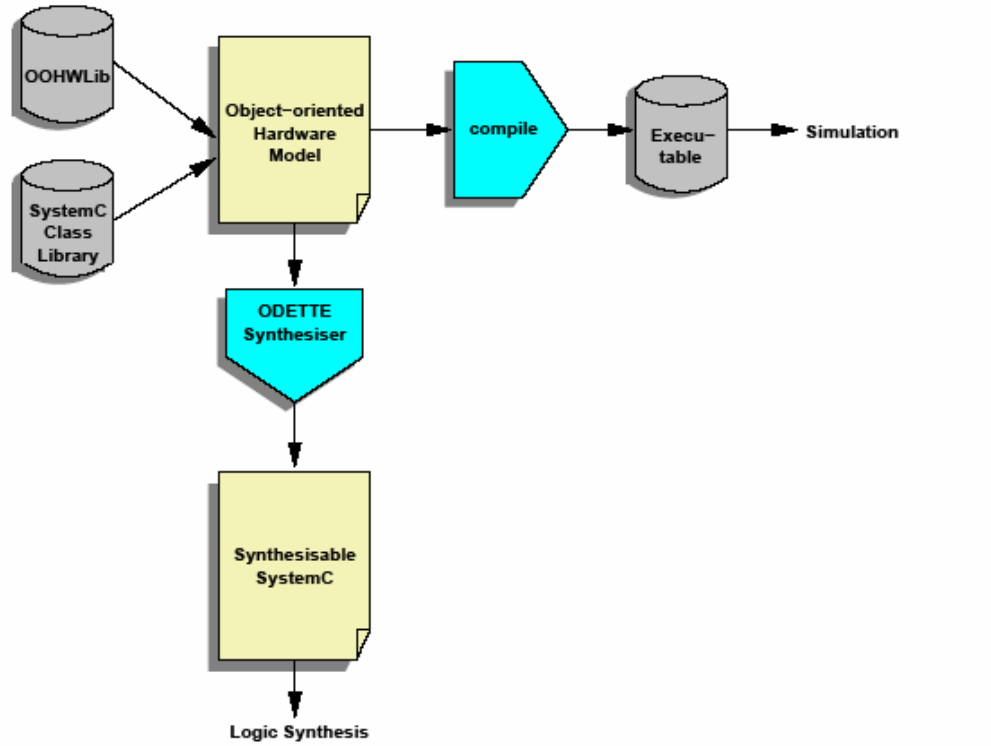
**5 pav. Sistemos aprašo kitimas per abstrakcijos lygius. Perėjimas nuo abstraktaus sistemos aprašo iki aparatūrinės įrangos realizacijos paruoštos sintezei**

Ieškant sprendimo sintezuoti TLM modelius, šalia visiems įprasto tiesioginio kodo sintezavimo į ventilių lygį atsirado ir kitas požiūris. Objektinio aprašo sintezavimą pavyko suskaldyti į dvi dalis: TLM kodo transformavimą į sintezuojamą RTL ar elgsenos lygio aprašą, o vėliau visiems įprastu būdu jį sintezuoti į ventilinio modelio lygį.




**6 pav. (A) Tradicinė SystemC objektinio modelio sintezė (B) Automatizuota SystemC objektinio modelio sintezė**

Ieškant būdų automatizuotam TLM modelių sintezei buvo suprojektuotas automatinis vertimo iš SystemC TLM aprašo į sintezuojamą RTL įrankis - SystemC-Plus. SystemC-Plus tai Europos komisijos IST FP5 ODETTE projekto rezultatas.




**7 pav. ODETTE siūlomas objektinio SystemC aprašo sintezės procesą**

ODETTE sintezatoriui pateikiamas SystemC TLM aprašas turi tam tikrus reikalavimus [3]. Aprašas sintezuojamas su C++ kompiliatoriumi naudojant SystemC ir OOHWLib bibliotekas. Po sėkmingo modeliavimo, šis aprašas su ODETTE įrankiu gali būti sintezuojamas į RTL aprašą. SystemC-Plus

palaikomas konstrukcijų rinkinys – tai visiškai palaikomas objektinis aprašas. Konstrukcijų sąrašą galima rasti [11].

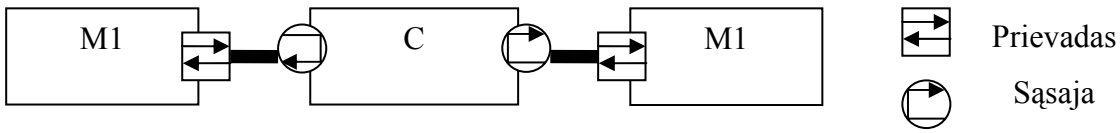
## 2.5 Bendravimo tobulinimas

Kaip ankščiau buvo minėta, SystemC TLM lygio modelius sintezuoti galima keletu būdų. Vienas iš jų – rankinis aprašo transformavimas. Tai ko gero sudėtingiausias kelias – bet vienintelis šiuo metu prieinamas. Bet SystemC turi puikias priemones tokiam procesui - skirtingų abstrakcijų lygio modelių vienoje sistemoje veikimo palaikymą, kas ypač aktualu testuojant ir verifikuojant tarpiniais, iš skirtingo abstrakcijų lygio modulių sujungtas sistemas.

Modulių bendravimo tobulinimas – tai dažniausiai abstraktaus komunikavimo protokolo išplėtimas iki detalios realizacijos. Šio proceso metu abstrakti komunikavimo schema arba yra sumontuojama ant duotos tikslo architektūros arba tiesiog tobulinama iki norimos efektyvesnės realizacijos.

Išties modulių bendravimo tobulinimas – komplikuotas projektavimo uždavinys: čia nėra vieningos metodologijos kaip vieną modulį galima transformuoti iš vieno abstrakcijos lygio į kitą. Galima aptikti tik bendras tendencijas ir bendrus principus. Kiekvienam atskiram atvejui reikalingas skirtingas sprendimo būdas. Pabandysim trumpai pailiustruoti šį procesą.

Tarkim mes turime du aukšto lygio modulius M1 ir M2, kurie bendrauja per kanalą C abstrakčiu protokolu. Mūsų tikslas – patobulinti modulius M1, M2 iki žemesnio lygio aprašo.



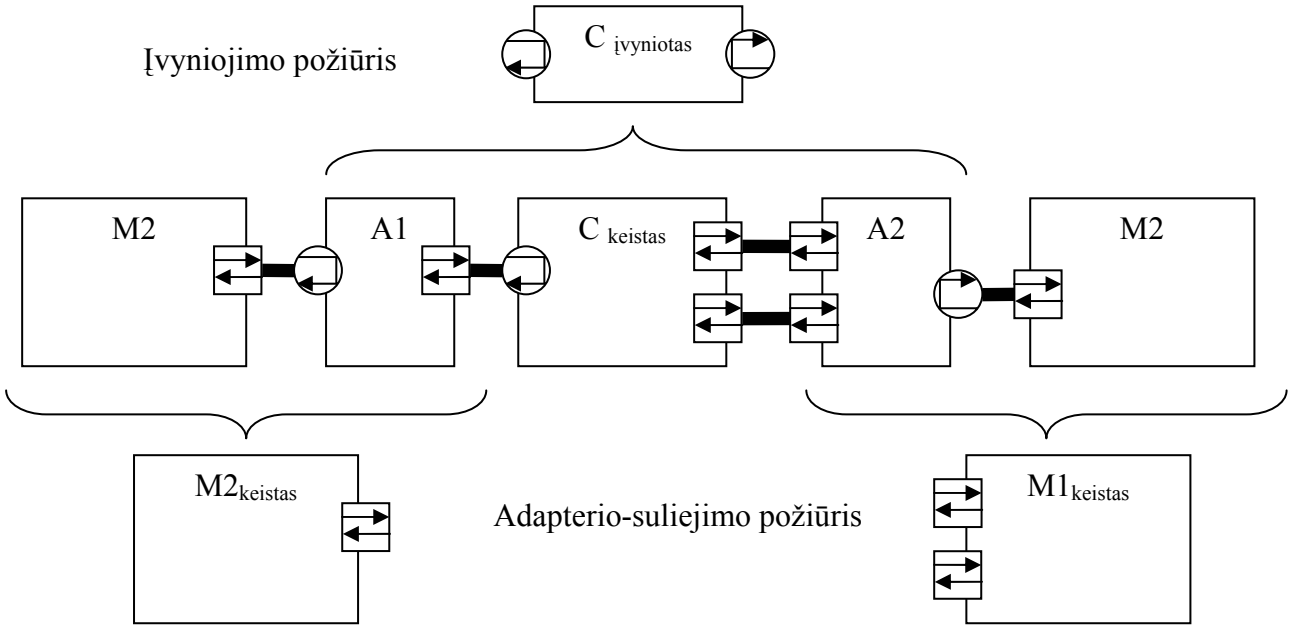

**8 pav. Moduliai M1 ir M2 bendraujantys per kanalą C**

Šį procesą galima suskaidyti į keletą gana abstrakčių etapų:

- Išrinkti atitinkamą bendravimo schemą, kuri bus naudojama realizacijoje
- Pakeisti abstraktų bendravimo kanalą C patobulintu kanalu, kuris realizuoja patobulinto kanalo $C_{keistas}$ schemą.

Įgalinti modulius M1 ir M2 bendrauti per patobulintą kanalą $C_{keistas}$ sekančiais būdais:

- Prie kanalo $C_{keistas}$ pajungiant papildomus modulius A1 ir A2 taip, kad naujas gautas kanalas $C_{įvyniotas}$ derintųsi prie modulių M1 ir M2 sąsajų (įvyniojimo požiūris).
- Modulius M1 ir M2 patobulinti iki $M1_{keistas}$ ir $M2_{keistas}$ taip, kad atitinkama sąsaja derintųsi prie $C_{keistas.}$ (adapterio-suliejimo požiūris)




**9 pav. Bendravimo tobulinimo dvi skirtingos strategijos**

Matome, kad po antro sistemos tobulinimo žingsnio, modulių ir kanalo sąsajos nebesiderina. Trečiojo žingsnio užduotis – išlyginti šį skirtumą keičiant vieną iš sujungimo taškų. Arba pradinį kanalą įmontuoti į sudėtinį kanalą C $_{įvyniotas}$, kurio sąsajos lygiai tokios pačios kaip ir pradinio kanalo C. Arba modulius M1 ir M2 pakoreguoti taip, kad būtų suderinami su pakoreguotu kanalu C $_{keistas}$ .

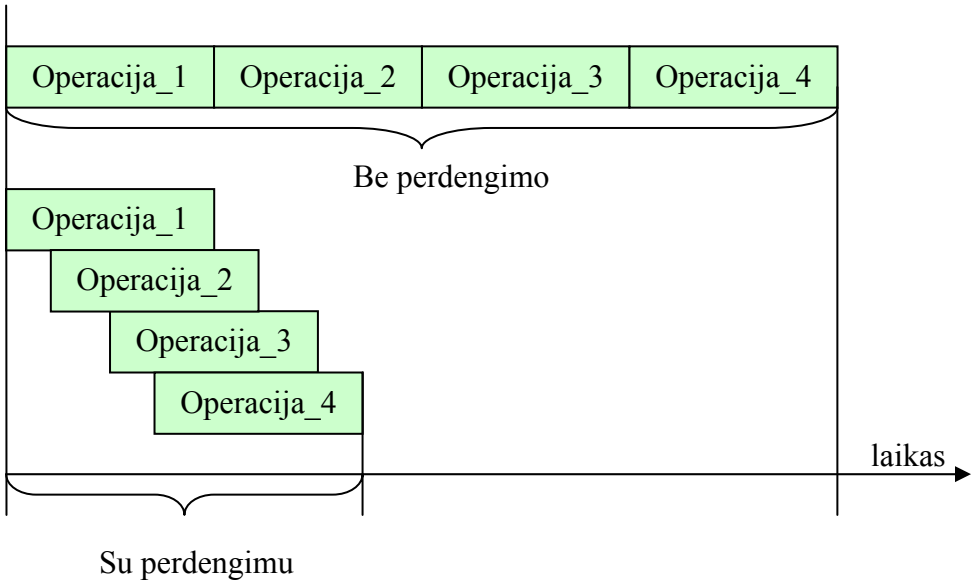
Šios abi alternatyvos yra panašios tuo, kad jas abi sudaro tobulinti kanalai ir jie yra sumontuoti ant pradinio abstraktaus kanalo.

Kanalo tobulinimo procesui palengvinti, galima įvesti dar vieną tarpinį žingsnį – adapterio konstravimą.. Adapteris – tai modulis, kuris gali vieną modulio sąsają išversti į kito modulio kitokią sąsają. Pagal SystemC terminologiją adapteris – tai hierarchinis kanalas, kuris realizuoja vieną ar daugiau sąsajų ir vieną ar kelis prievadus. Šis modulis išverčia sąsają į kreipinius į prievadus.

## 2.6 Srautinis veikimo principas

Sparčiųjų duomenų magistralių sparta paprastai būna pagrįsta jos srautiniu (*pipelining*) veikimo principu. Tai tokia savybė, kai sudėtinės instrukcijos (etapai) yra perdengiamos vykdymo metu. Visa magistralės operacija yra padalinta į kelis etapus. Kiekvieno etapo dalis yra perdengiama su sekančio etapo dalimi. Perklojamos tos dalys, kurios gali būti vykdomos lygiagrečiai, nevykdomi jokie tarpusavio duomenų mainai, ir skaičiavimui reikalingi duomenys neveikia vieni kitų – t.y. visiškai nepriklausomi procesai. Monopolinį principą magistralėje įgalina tai, jog magistralės sistemoje vienu metu vykdomi keletas nepriklausomų procesų. Tad jie lygiagrečiai gali manipuliuoti skirtingais duomenimis ir vienu metu gauti iš jų visiškai nesusijusios rezultatus. Etapai sujungiami vienas su kitu

suformuojant kanalą – instrukcijos pradedamos viename gale, praeina visus etapus ir rezultatas gaunamas kanalo kitame gale.




**10 pav. Nesrautinės (be perdengimo) ir srautinės (su perdengimu) monopolinių operacijų palyginimas**

Monopolija nesumažina laiko skirto vienos instrukcijos vykdymui. Vietoje to, šio principas išnaudoja sistemos galimybę vienu metu vykdyti keletą instrukcijų.

# 3 Projektinė dalis

Pilną AMBA AHB specifikaciją galima rasti [1] dokumente, čia apžvelgtos tik esminės AMBA AHB savybės. Šioje projektinėje dalyje didžiausias dėmesys skiriamas originaliems TLM aprašo projektavimo ir realizavimo sprendimams.

## 3.1 Trumpai apie AMBA AHB magistralės protokolą

Paprastai AMBA AHB sudaro tokie komponentai:

**AHB valdantysis įrenginys** (*angl. master*). Magistralės valdantysis įrenginys gali inicijuoti rašymo ir skaitymo operacijas. Tai atliekama siunčiant adresą ir kontrolės informaciją. Tik vienas valdantysis įrenginys gali vienu metu naudoti magistralę.

**AHB pavaldusis įrenginys** (*angl. slave*). Magistralės pavaldusis įrenginys reaguoja į skaitymo ir rašymo operacijas tam tikrame adresų intervale. Pavaldusis įrenginys siunčia atgal atsaką valdančiajam įrenginiui apie sėkmingą operaciją, klaidą arba apie laukimą duomenų perdavimo.

**AHB teisėjas. (***angl. arbiter***)**Magistralės teisėjas garantuoja, kad vienu metu tik vienas valdantysis įrenginys gali inicijuoti ir naudoti magistralę. Nepaisant to, kad teisėjavimo protokolas yra apibrėžtas, teisėjavimo algoritmas gali būti apibrėžtas kiekvieną kartą skirtingai, priklausomai nuo atskirų reikalavimų.

**AHB iškodavimo įrenginys. (***angl. decoder***)**AHB iškodavimo įrenginys naudojamas iškoduoti kiekvieno duomenų perdavimo adresą ir siųsti duomenis pavaldžiajam įrenginiui, kuriam tai yra skirta.

1 - ame priede yra pateikti ir trumpai aprašyti AMBA AHB magistralės signalai. Pateikti pavadinimai, jų formatai ir trumpas aprašas apie jų paskirtį

Magistralės darbą lengviausia įsivaizduoti turint jos darbo pavyzdžius. 2 – ame priede yra pateikta trejetas esminių pavyzdžių.

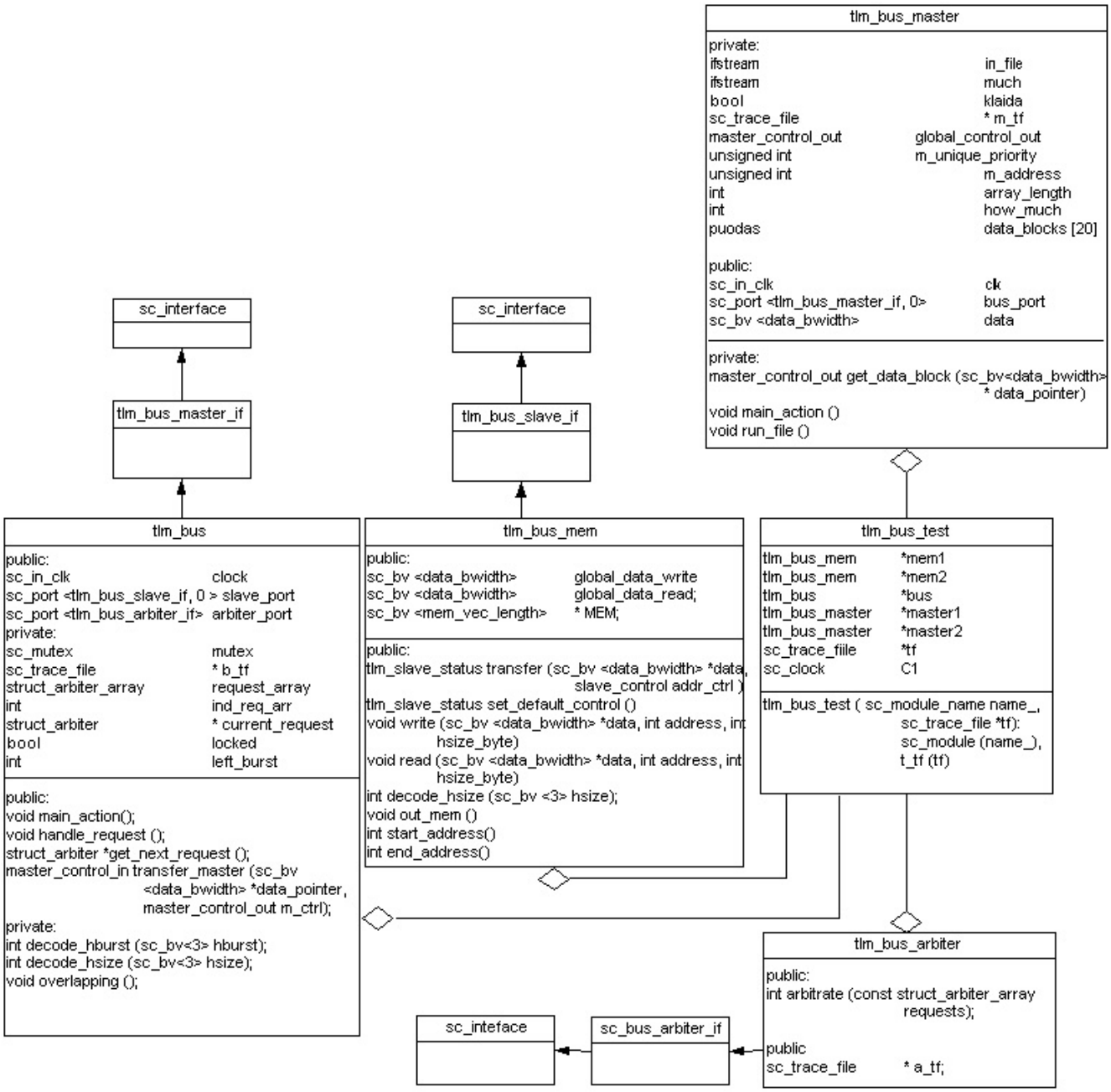
Pirmame paveiksle pateiktas pats paprasčiausias duomenų perdavimo pavyzdys.

Tolesniame paveiksle atspindėta vienas iš esminių magistralės darbo režimų – tai monopolinė operacija, kurios metu duomenys perduodami keletu etapų, kiekviename takte po vienodą duomenų kiekį. Tai leidžia perduoti didesnius duomenų kiekius vienu metu, neperjunginėti kiekvieną kartą duomenų magistralės ir neanalizuoti prioritetų. Tai lemia greitą magistralės darbą.

Paskutiniame 2 - o priedo pavyzdyje pateiktas atvejis, kai pasibaigusi magistralės operacija inicijuoja VĮ ir PĮ prioritetų analizę ir perjungimą, t. y. kaip keičiasi magistralės signalai perjungiant įrenginius.

## 3.2 Magistralės TLM modelis

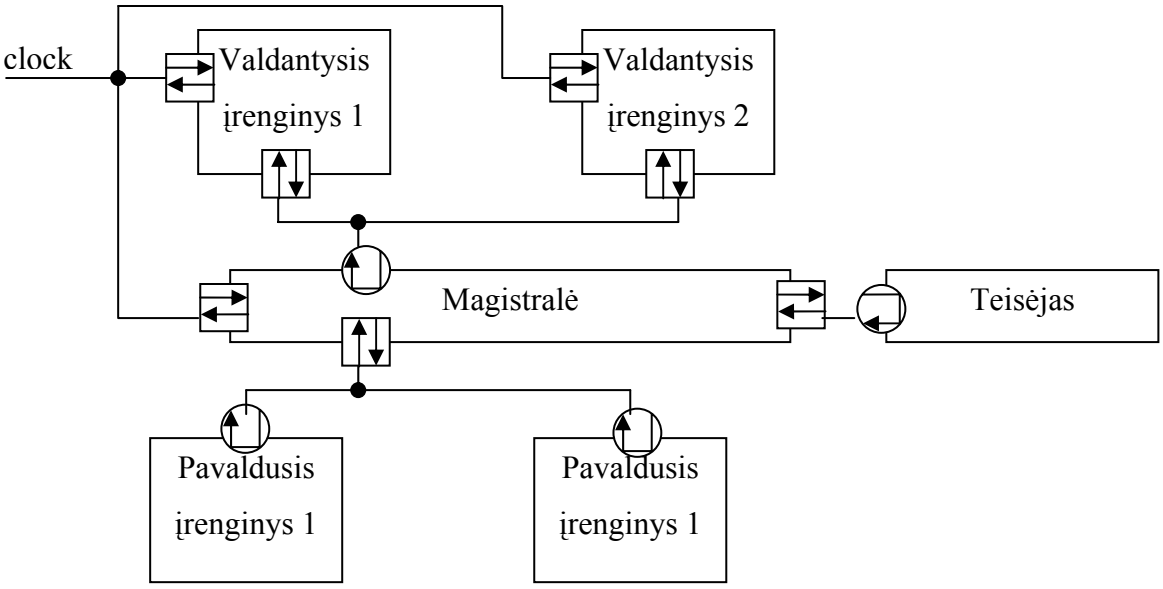
### 3.2.1 Klasių diagrama




**11 pav. Magistralės TLM aprašo UML klasių diagrama**

Duomenų magistralės TLM lygio sistemoje yra sukurtos aštuonios klasės. Iš jų dvi – virtualios, naudojamos kaip sąsajos su kitais sistemos objektais. Startinė klasė, kuri sukuria visus likusius objektus – tai *tlm_bus_test.* Jame sukuriami objektai – *tlm_bus, tlm_bus_mem, tlm_bus_arbiter, tlm_bus_master.* Visos virtualios klasės paveldi SystemC standartinę sąsajos klasę s*c_interface.*

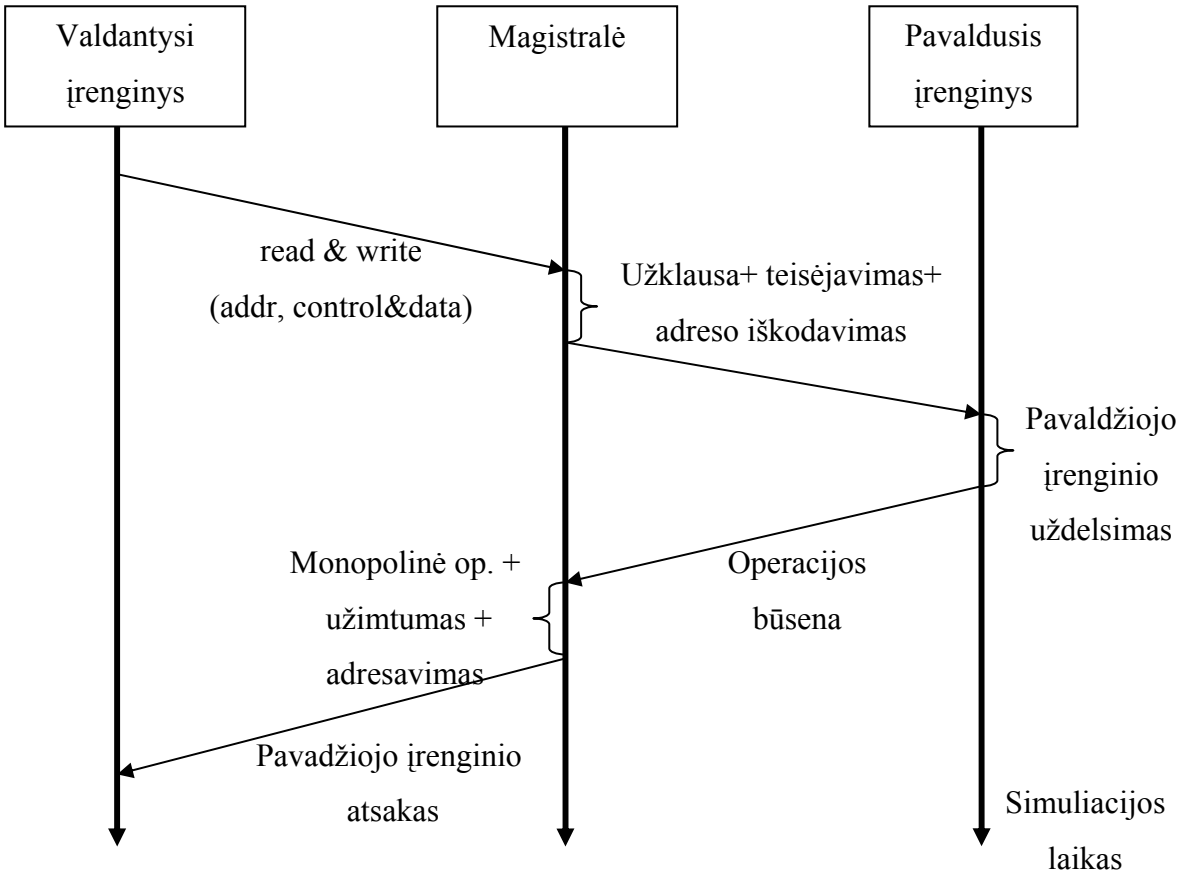
### 3.2.2 Blokinė diagrama




**12 pav. TLM AMBA AHB magistralės blokinė schema**

Sistemoje yra šeši komponentai. Valdantieji įrenginiai kreipiasi į magistralės sąsają per prievadus. Kreipinys į rašymo sąsają atrodo taip: bus_port->transfer_master (&data, global_control_out). Atsaką gauna per kintamąjį *data*. Magistralė kreipiasi į pavaldžiuosius įrenginius taip pat per prievadą. Sąsajos funkcijos yra realizuotos pačiuose pavaldžiuose įrenginiuose. Magistralė bendrauja su teisėju per prievadą, kurio sąsaja realizuota teisėjo modulyje.. Nuo laikrodžio *clock* sinchronizuoti valdantieji moduliai ir pats magistralės modulis.

### 3.2.3 Modulių sąveikos diagrama




**13 pav. Sistemos modulių sąveikos diagrama**

Sistemos darbe ypač svarbi jos komponentų sinchronizacija – užklausų ir atsakymų seka, informacijos apdorojimo etapai, jiems sugaištas laikas, vėlavimai. Sistemos darbą arba operaciją inicijuoja valdantysis įrenginys kreipdamasis į magistralę. Ji apdoroja užklausą, paskirsto ją atitinkamam pavaldžiajam įrenginiui, įrenginys suformuoja atsaką, pasiunčia magistralei, kuri nukreipia šį paketą atgal valdančiajam įrenginiui reikiama kryptimi.

### 3.2.4 Modulių specifikacija

#### 3.2.4.1 Tipų aprašai

```
enum { addr_bwidth =32 }                Adresų magistralės adresų plotis
enum { data_bwidth =32 };               Duomenų magistralės plotis
enum { mem_vec_length =8 };
enum       tlm_slave_hresp       {      Pavaldžiojo įrenginio būsena po operacijos.
TLM_SLAVE_OK = 0,
              TLM_SLAVE_ERROR,
              TLM_SLAVE_RETRY,
```

```
            TLM_SLAVE_SPLIT };
```

```
struct tlm_slave_status {
     sc_logic           HREADY;
     tlm_slave_hresp    HRESP;
};
```

Struktūra naudojama pavaldžiojo įrenginio atsakui valdančiajam įrenginiui

```
struct slave_control {
     sc_logic           HSELx;
     sc_bv<data_bwidth> HADDR;
     sc_logic           HWRITE;
     sc_uint <2>        HTRANS;
     sc_uint <3>        HSIZE;
     sc_uint <3>        HBURST;
};
```

Struktūra naudojama perduoti kontrolės informaciją pavaldžiajam įrenginiui

```
struct master_control_in {
     sc_logic      HGRANTx;
     sc_logic      HREADY;
     sc_uint <2>   HRESP;
     sc_logic      HRESETn;

};
```

Struktūra naudojama reakcijos duomenims nusiųsti valdančiamjam moduliui

```
struct master_control_out {
     sc_logic           HBUSREQx;
     sc_bv <2>          HTRANS;
     sc_bv<addr_bwidth> HADDR;
     sc_logic           HWRITE;
     sc_uint<3>         HSIZE;
     sc_bv <3>          HBURST;
};
```

Struktūra naudojama “supakuoti” valdančiojo modulio siunčiamą kontrolės duomenis

```
struct struct_arbiter {
     slave_control      *s_ctrl;
     sc_bv<data_bwidth> *data;
     sc_event           transfer_done;
     int                how_old;
};
```

Struktūra naudojama nusiųsti teisėjui visą informaciją, kuri reikalinga nuspręsti, kuriam įrenginiui suteikti priėjimą prie magistralės

```
typedef      struct_arbiter      *
struct_arbiter_array [30];
```

Duomenų tipas skirtas saugoti užklausoms laukiantiems priėjimo prie magistralės

```
struct puodas {
     sc_bv <data_bwidth>     data;
     master_control_out
     control_out;
};
```

Duomenų tipas skirtas užklausų masyvui magistralei saugoti

### 3.2.4.2 Magistralė

```
sc_in_clk clock
```

Sinchronizacijos signalas

```
sc_port     <tlm_bus_slave_if,     0     >
slave_port
```

Prievadas ryšiui su pavaldžiaisiais moduliais

```
sc_port              <tlm_bus_arbiter_if>
arbiter_port
```

Prievadas ryšiui su teisėju

```
sc_mutex mutex
```

Priemonė kontroliuoti priėjimą prie *request_array* ir *ind_req_arr*

```
struct_arbiter_array request_array
```

Saugo laukiančias valdančiųjų įr. užklausas

```
Int ind_req_arr
```

Laukiančių užklausų kiekis

```
struct_arbiter * current_request
```

Apdorojama užklausa

| | |
|---|---|
| `int left_burst;` | Likę neatlikti monopolinės operacijos ciklai |
| `void main_action()` | Pagrindinis procesas |
| `void handle_request ()` | Apdoroti priėjimo laukiančių eilę |
| `struct_arbiter *get_next_request ()` | Kas sekantis prie magistralės? |
| `master_control_in transfer_master (sc_bv <data_bwidth> *data_pointer, master_control_out m_ctrl)` | Sąsajos funkcija |
| `int decode_hburst (sc_bv<3> hburst)` | Kiek monopolinės operacijos. ciklų? |
| `int decode_hsize (sc_bv<3> hsize)` | Koks duomenų perdavimo plotis? |

#### 3.2.4.3 Valdantysis įrenginys

| | |
|---|---|
| `sc_in_clk clk` | Sinchronizacijos signalas |
| `sc_port <tlm_bus_master_if, 0> bus_port` | Prievadas ryšiui su magistrale |
| `ifstream in_file` | Pradinių duomenų failas |
| `ifstream much` | Pradinių duomenų failas |
| `master_control_out global_control_out` | Siunčiamas kontrolės duom. paketas |
| `int array_length` | Masyvo ilgis |
| `int how_much` | Kiek liko monopolinės operacijos ciklų? |
| `puodas data_blocks [20]` | Duomenys, kurie bus siunčiami magistralei |
| `master_control_out` | |
| `get_data_block (sc_bv<data_bwidth> * data_pointer)` | Paima duomenų bloką iš *data_blocks* masyvo |
| `void main_action ()` | Pagrindinis procesas |
| `void run_file ()` | Priskiria inicializavimo reikšmes |

#### 3.2.4.4 Pavaldusis įrenginys

| | |
|---|---|
| `sc_bv <data_bwidth> global_data_write` | Rašomų duomenų buferis |
| `sc_bv <data_bwidth> global_data_read` | Skaitomų duomenų buferis |
| `void out_mem ()` | Funkcija trasavimui |
| `int start_address()` | Operacijos pradžios adresas |
| `int end_address()` | Operacijos pabaigos adresas |
| `sc_bv <mem_vec_length> * MEM` | Atmintis |
| `tlm_slave_status transfer (sc_bv <data_bwidth> *data, slave_control addr_ctrl )` | Sąsajos funkcijos realizacija |
| `tlm_slave_status set_default_control ()` | Būklės nustatymas į pradinę būseną |

| | |
|---|---|
| `void write (sc_bv <data_bwidth> *data, int address, int hsize_byte)` | Rašymas į atmintį |
| `void read (sc_bv <data_bwidth> *data, int address, int hsize_byte)` | Skaitymas iš atminties |
| `int decode_hsize (sc_bv <3> hsize)` | Duomenų perdavimo plotis |

#### 3.2.4.5 Teisėjas

| | |
|---|---|
| int arbitrate (const struct_arbiter_array requests) | Pagal gautus duomenis nusprendžia kurį valdantįjį įrenginį prileisti prie magistralės |

### 3.2.5 Testavimas

Testavimas atliktas magistralės sistemoje, kurią sudaro pati magistralė, du valdantieji ir du pavaldieji įrenginiai. Testavimo aplinka sistemai nebuvo reikalinga. Jos funkcijas perėmė valdantieji įrenginiai – jie patys nuskaito duomenis iš tekstinių failų, išsisaugoja atskirame buferyje, ir viso testavimo metu, duomenis ima iš vidinio buferio.

Testavimo metu magistralei rankiniu būdu buvo paruošti testiniai rinkiniai, kurie tikrina ir apima visą AMBA AHB magistralės funkcionalumą.

Žemiau pateiktos pagrindinės magistralės funkcijos iš testuotų jos savybių:

- Paprasta rašymo operacija
- Paprasta skaitymo operacija
- Monopolinė įvyniojimo operacija
- Monopolinė nuosekli operacija
- Teisėjavimo principas
- Sinchronizacija, srautinė architektūra
- Valdančiųjų ir pavaldžiųjų įrenginių reakcijos į užklausas korektiškumas
- Pertraukimo įvykis (*split*)
- Operacijos pratęsimas po pertraukimo

Testavimo rezultatai pateikti prieduose 2, 3, 4. laikinėmis diagramomis Tai yra per tris puslapius suskaidytas vienas testavimas. Juos sieja bendra laiko linija.

Aptarsime šias laiko diagramas. Pradiniu laiko momentu visi signalai nustatyti į pradines reikšmes pagal nutylėjimą. Jas nustato modulių konstruktorius. Perskaitę duomenis iš tekstinių failų valdantieji įrenginiai paprašo priėjimo prie magistralės. Kadangi prieš tai nebuvo atlikta jokių operacijų, pagal

nutylėjimą priėjimas patvirtintas pirmam valdančiajam įrenginiui. Jis pradeda vykdyti srautinę 32 bitų pločio 4-ių taktų trukmės rašymo į pavalduįį įrenginį operaciją. Siunčia vieną po kito adresas&kontrolė informaciją ir gauna atsakymus. Magistralė pagal iš VĮ gautą adresą nusprendžia, į kurį PĮ kreiptis. Toliau tas pats valdantysis įrenginys vykdo iš eilės vieną po kitos, keičiant parametrų HSIZE, HRESP reikšmes, paprastas, ne tęstines operacijas – 3 kartus rašo ir 2 kartus skaito. Taip pat su skirtingais parametrais atlieka 4-ių taktų nuoseklią ir įvyniojimo monopolines operacijas. Toliau, po 917200 ps. laiko žymos, valdymą perima antrasis valdantysis įrenginys. Ir atlieka su įvairiais parametrais 4-ių ir 2 – ų taktų nuoseklias monopolines operacijas.

Nagrinėjant laikines diagramas derėtų prisiminti, jog magistralė veikia srautiniu (*pipelining)* principu t. y. magistralė neturi tuščių ciklų laukimui, kol bus paruošti kontrolės signalai, duomenys ir t.t.. Duomenys kiekvienai operacijai paruošiami paskutinės operacijos metu. Projektavimas srautiniu principu sumažina signalų vėlavimą, padidina magistralės pralaidumą.

# 4 Tyrimo dalis

## 4.1 Įžanga

Šio tyrimo tikslai:

- Rasti ir aprašyti kuo bendresnę metodiką, pagal kurią iš abstraktaus lygio sparčiosios magistralės būtų galima gauti sintezuojamą modelį. Tai galime pavadinti TLM transformacija iki RTL lygio.

Bendrąją metodiką bandysime aprašyti remiantis AMBA AHB magistralės TLM lygio prototipą tobulinant (*refine*) iki tokios pat specifikacijos magistralės elgsenos lygyje. Tai galima vadinti protokolo, bendravimo transformacija. Šį konkretų pavyzdį galima laikyti labai artimu bendrajai metodikai, nes TLM aprašo tobulinimas – tai iš esmės yra bendravimo tarp modulių tobulinimas. Padarius nedidelius šio proceso apibendrinimus, galima teigti, jog šią metodiką galima naudoti bet kokį TLM modelį tobulinant iki RTL lygio.

Tyrimas pradėtas tobulinant magistralę nuo operacijų lygio (TLM), realizuotą pagal programuotojo požiūrį su sinchronizuotu protokolu iki RTL sintezuojamo aprašo (žr. 5 pav.). Tiek periferiniai moduliai (atmintis, valdantieji įrenginiai), tiek pati magistralė, pradžioje veikia operacijų lygyje. Po tobulinimo proceso magistralė veikia pagal tą pačią specifikaciją, bet jau elgsenos lygyje (RTL).

## 4.2 Tyrimo aplinka

Aptarsime priežastis dėl kurių tyrimui pasirinkta *AMBA AHB* magistralė.

Žinant, jog mūsų vienas iš darbo tikslų susintezuoti TLM lygio aprašą bei palyginti elgsenos ir TLM aprašų modeliavimo greitį, reikėjo rasti tokį modelio sprendimą, kad su jo pagalba būtų galima atlikti abu eksperimentus. Šiam tikslui puikiausiai tinka duomenų magistralė. Pradinėje fazėje ją gana paprasta įsivaizduoti tiek abstrakčiame, tiek žemame lygyje: aukštame lygyje moduliai – tai klasės, kurios duomenimis keičiasi naudodamos sąsajas ir jais bendraudami su kitais moduliais; žemame lygyje moduliai sujungti signalais ir informacijos mainai tarp modulių vyksta siunčiant duomenis šiais signalais. Be to, laikinės charakteristikos duomenų magistralei – tai esminis parametras, kuris apibūdina duomenų mainų spartą. Iš šio parametro galima spręsti apie magistralės efektyvumą ir kokybę.

Toliau seka klausimas, kodėl pasirinkta būtent šios architektūros – *AMBA AHB* – magistralė? Pirmiausia - magistralės specifikacija laisvai prieinama internete, dėl to sumažėja laiko ir pastangų sąnaudos projektavimo uždaviniui spręsti. Šio tipo magistralė yra vienas iš naujausių sprendimų

įrenginių komunikacijos srityje. Tai viena sparčiausių ir plačiausiai paplitusių magistralių tarp naudojamų Soc specialistų dėl savo spartos ( tai įtakoja jos srautinė architektūra) ir lankstumo. Be to, magistralės prototipas puikiai tinka TLM iliustracijai.

Modeliavimui pasirinkta SystemC kalba. Ši kalba turi gausų rinkinį primityvų bendravimui ir sinchronizavimui – kanalai, prievadai, sąsajos, įvykiai, signalai, laukimo būsenų įterpimas. Lygiagretūs procesai vykdomi atskiromis gijomis, o jų vykdymas valdomas planuotojo. Kaip minėta ankščiau, tai, ko gero vienintelė aparatūrinei įrangai programuoti pritaikyta kalba palaikanti TLM abstrakcijos lygį. Be to, tai - laisvo kodo kalba puikiai tinkanti modeliuoti tiek TLM, tiek RTL lygiuose. Kadangi SystemC pagrįsta C++ klasių biblioteka, jos aprašai objektiniai ir moduliniai, bei leidžiantys duomenų apjungimą – visa tai palengvina IP (*intelectual property*) plitimą, pakartotinį panaudojimą, pritaikymą prie skirtingo abstrakcijos lygio.

Kaip sintezavimo įrankis RTL aprašui pasirinktas *Synopsys Behavioral* ir *RTL* kompiliatoriai. Tik šie įrankiai yra laisvai prieinami universiteto laboratorijoje. Laisvai platinamų sintezavimo įrankių rasti nepavyko. Per daug komplikuota, brangi tokios programinės įrangos gamyba. Negalima atmesti ir finansinio faktoriaus – iš to uždirbami didžiuliai pinigai.

Šis tyrimas gali sudominti studentus, aparatūros projektuotojus, kurie domisi ir kuria dideles bei komplikuotas sistemas ir nori paspartinti jų testavimo bei verifikavimo procesą, projektavimo procesą pradedant nuo TLM abstrakcijos lygio ir vėliau po žingsnį, pritaikant šiame tyrime pateiktą metodiką ir tobulinimo būdą, abstraktų modelį „nuleisti“ iki RTL lygio – aparatinio modelio realizacijos. Tai turėtų padėti apsispręsti abejojantiems operacijų lygio modeliavimo teikiamais privalumais: verifikavimo sparta, projektavimo abstrakcija, „natūralesniu“ projektavimo procesu.

## 4.3 Testavimas tobulinimo proceso eigoje

Ypač svarbu, kad visame transformavimo procese, po kiekvieno tobulinimo etapo, įrenginys veiktų pagal specifikaciją, lygiai taip pat kaip ir prieš pastarąjį tobulinimo etapą.

Patartina po kiekvieno didesnio žingsnio atlikti testavimą. Yra gerai turėti jau paruoštą testų rinkinį su charakteringais testų deriniais, testuojantį nuo paprasčiausių operacijų iki sudėtingesnių (pvz.: mūsų atveju tai būtų paprasčiausias nuskaitymas, įrašymas, monopolinės rašymo ir skaitymo operacijos su įvairiais parametrais, teisėjavimo teisingumas, operacijų pertraukimas ir pan.)

Nėra reikalaujama kiekvieną kartą atlikti pilną testavimą ir verifikavimą, tačiau patikrinti pagrindines magistralės vykdomas operacijas visos transformacijos eigoje yra būtina, idant esminės klaidos nebūtų pastebėtos tik vėlyvoje projektavimo stadijoje, kai jos ištaisymas jau reikalauja daug laiko ir pastangų.

Pilnas tarpinių sistemų verifikavimas visos transformacijos metu gali būti atliktas keletą kartų, priklausomai nuo sistemos sudėtingumo. Jeigu sudėtingumas mažesnis, pilnas tarpinis sistemos verifikavimas nėra būtinas. Sekanti nelygybė išreiškia priklausomybę tarp pilnam ir nepilnam testavimams sugaišto laiko ir „įsisenėjusioms“ klaidoms ištaisyti reikalingo laiko.

$$k * A + t * a < \alpha_1 + \alpha_2 + ... + \alpha_n$$

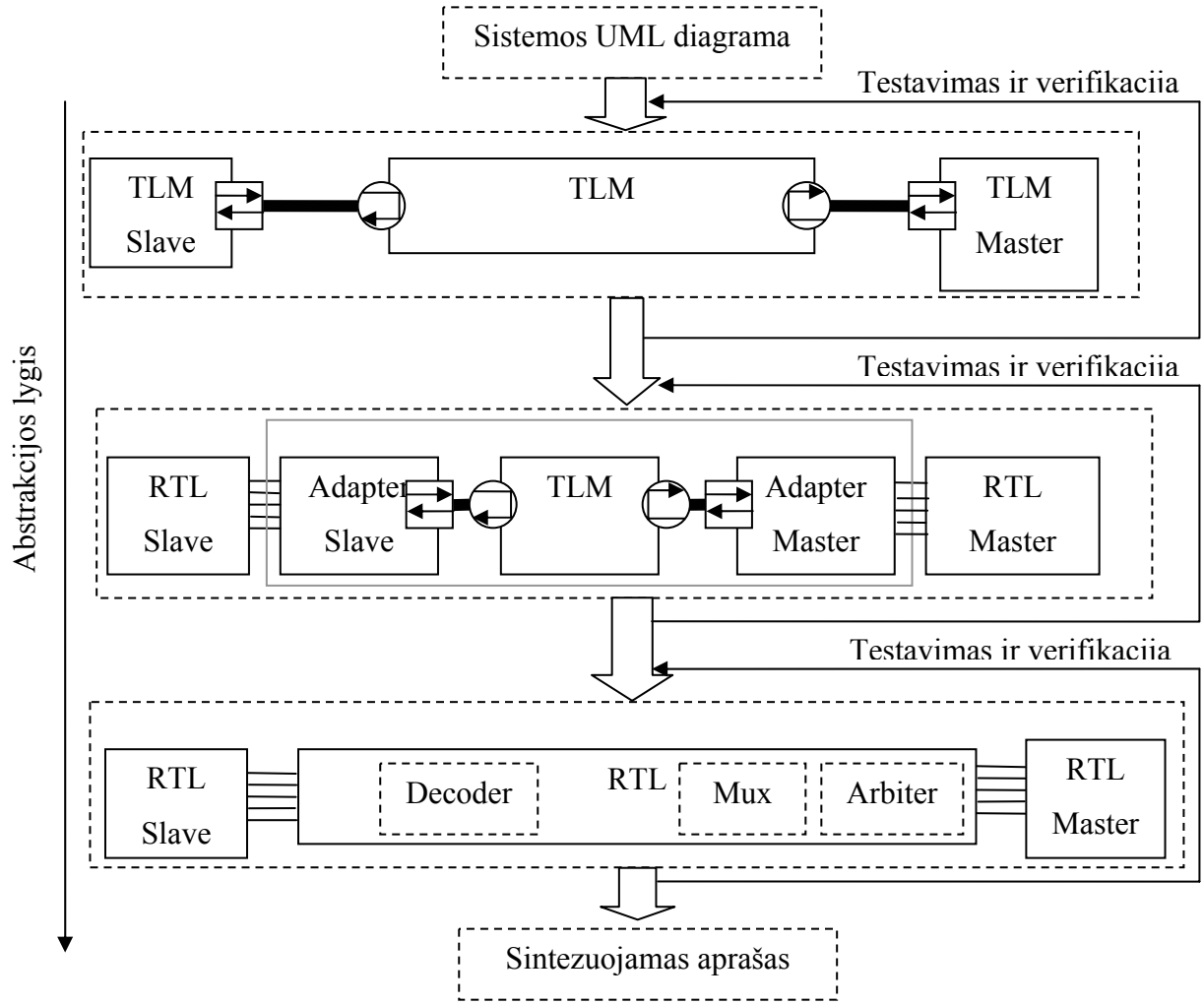
A – laiko sąnaudos pilnam sistemos testavimui
a – laiko sąnaudos ne pilnam sistemos testavimui
k– kiek kartų atliktas pilnas sistemos testavimas
t– kiek kartų atliktas ne pilnas sistemos testavimas
$\alpha_1$, $\alpha_2$, $\alpha_n$ – laikų sąnaudos „įsisenėjusių“ klaidų taisymui

Kuo dažniau bus atliekami testavimai, tuo mažiau klaidų liks transformuojamame kode. Didinant testavimo dažnį ir sudėtingumą pasiekiamas toks lygis, kai dažnesnis testavimas jau neduoda rezultatų t.y. klaidų nelieka arba jų lieka nykstamai mažai. Laiko sąnaudos pilnam ir nepilnam testavimams atvirkščiai proporcingos „įsisenėjusių“ klaidų taisymui sugaištam laikui. Tai reiškia, kad didinant testavimų kiekį, mažėja klaidų skaičius. Kintamųjų k ir t parinkimas optimalus, kai gaunamas didžiausias skirtumas tarp abiejų nelygybės pusių.

## 4.4 Transformacijos procesas

Skirtumai tarp TLM ir RTL sparčiųjų magistralių modelių transformacijos procesui pasibaigus yra tokios:

- Magistralės modulį TLM apraše sudaro viena esybė. Atskiri veiksmai suskaidyti į funkcijas ir procedūras. Tarpusavyje šie elementai bendrauja kreipiniais vienas į kitą. Dėl supaprastėjusios struktūros tai tapo įmanoma. Transformacijos metu teko kai kuriuos TLM aprašo struktūrinius elementus transformuoti į atskirus esybės modulius su sava sinchronizacija ir su savais prievadais.
- Esybės bendrauja jau nebe per kreipinius viena į kitą, o siunčia informaciją signalais.




**14 pav. TLM magistralės sistemos tobulinimo nuo TLM iki RTL lygio pagrindiniai 3 etapai**

Bendruoju atveju, kaip jau buvo aptarta anksčiau, egzistuoja du tobulinimo proceso keliai – adapterius prijungti prie pačios magistralės arba prie periferinių įrenginių, tokių kaip atmintys, CPU, fifo ir pan.. Abu būdai yra panašūs ir rezultatas yra tas pats. Prieš pradedant tobulinimo procesą patartina turėti RTL lygyje bent vienos kategorijos komponentus – arba periferinius įrenginius arba pačią magistralę. Jeigu tikslas yra suprojektuoti RTL lygio magistralę, tada transformavimo pradžiai reikalinga turėti jau suprojektuotus RTL lygio periferinius įrenginius. Jie gali būti paimti kaip užbaigti moduliai su reikalinga sąsaja iš intelektualinės nuosavybės saugyklų arba juos reikia susiprojektuoti pačiam atskiro projektavimo proceso metu, kuris mums dabar nėra aktualus ir kurio čia nenagrinėsime. Tik, prieš pradėdami transformavimo procesą, padarysime prielaidą, jog tokius modulius mes jau turime. Taigi, pradžioje reikalinga turėti:

- Magistralės TLM modulį
- Periferinių įrenginiu modulius S1, S2, ..., Sn; M1, M2, .., Mn TLM lygyje
- Periferinių įrenginių S1, S2, ..., Sn; M1, M2, .., Mn analogiški įrenginiai RTL lygyje

Viršutiniame paveiksle pažymėta tik du periferiniai įrenginiai. Tai padaryta norint supaprastinti paveikslėlį. Periferinių įrenginių gali būti tiek, kiek leidžia magistralėje realizuoto protokolo specifikacija. Daugelis protokolų leidžia prijungti 16 periferinių įrenginių Magistralės testavimui pakanka keturių įrenginių – dviejų pavaldžiųjų ir dviejų valdančiųjų.
TLM lygio periferiniai įrenginiai naudojami magistralės pilnam testavimui. Šiuos įrenginius mes galima pavadinti tam tikra testavimo aplinka, nes jie tiekia testinius duomenis, kviečia duomenų operacijas tokias kaip rašymas ir skaitymas. O tikroji testavimo aplinka šiai sistemai reikalinga tik modulių objektų sukūrimui, laikinių laikrodžio parametrų nustatymui ir laikinių diagramų failo sukūrimui.
Pirmiausia trumpai aptarsime magistralės derinimo eigą. Transformacijos procesą galima suskaidyti į keletą dalių. Tai tik patys pagrindiniai etapai. Pagal poreikį, šių etapų skaičių galima didinti. Tai priklauso nuo sistemos sudėtingumo, apimties, programuotojų įgūdžių ir pan..

Pradinis etapas – tai duomenų magistralės sistema TLM lygyje, pilnai ištestuota ir veikianti pagal specifikaciją. Joje negali būti klaidų, nes tai yra pradinis taškas visam sistemos transformavimo procesui. Nuo jos teisingo veikimo priklauso viso derinimo proceso sėkmė.

Antras etapas – RTL lygio periferinių įrenginių prijungimas. Jų sąsajos nesiderina prie TLM magistralės, taigi prie magistralės sąsajų reikia prijungti adapterį, kuris ją priderintų prie RTL modelių jungčių. Šiuo momentu patartina žinoti, kiek periferinių įrenginių bus norima jungti prie magistralės. Nes vėlesniuose etapuose papildomos periferijos prijungimas tampa gana sudėtingas. Sujungus, vėlgi rekomenduotina atlikti pilną sistemos testavimą.

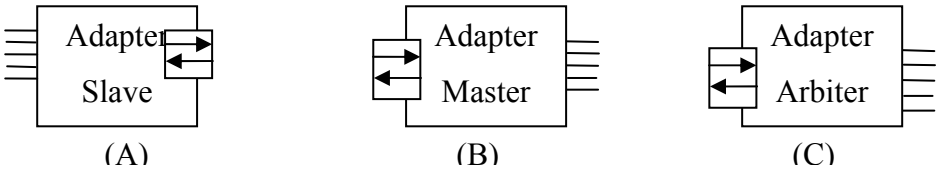
Trečias etapas – adapterių ir TLM magistralės modulio suliejimas į vieną modulį, tuo būdu gaunant magistralės elgsenos lygio aprašą. Jeigu mūsų tikslas būtų tik periferinių įrenginių verifikavimas, o ne sintezė, šių adapterių ir magistralės junginys būtų kompromisinis variantas verifikuojant periferiją.
Tačiau mūsų tikslas – magistralės sintezė. Įterpus adapterius, tarp jų ir TLM modulio atsirado papildomos jungtys, kurias reikalinga eliminuoti. Šis procesas yra pats sudėtingiausias visame transformavimo procese, jis imlus laikui, pastangoms ir klaidoms. Svarbus momentas suliejimo eigoje - transformuojamo modelio restruktūrizavimas. Mūsų atveju magistralės modulį TLM apraše sudaro viena esybė. Atskiri veiksmai suskaidyti į funkcijas ir procedūras. Tarpusavyje šie elementai bendrauja kreipiniais vienas į kitą. Dėl palyginus paprastos TLM aprašo struktūros tai yra įmanoma. Tačiau RTL lygyje šios funkcijos realizuotos atskirais moduliais Transformacijos metu dažnai tenka kai kuriuos TLM aprašo struktūrinius elementus transformuoti į atskirus esybės modulius su sava sinchronizacija ir su savais prievadais.

Ketvirtas etapas – elgsenos lygio modelio sintezė, ir sintezuoto įrenginio konvertavimas į verilog ar VHDL RTL lygio aprašą. Taip gaunamas galutinis sistemos RTL lygio aprašas. Taigi, iš elgsenos aprašo iki RTL lygio aprašo procesas automatizuotas EDA priemonėmis.

Turėdami TLM modelį, veikiantį pagal specifikaciją, galima pradėti tobulinimo procesą. Testavimo rezultatus geriausia išsisaugoti, tam kad po vėlesnių transformacijos etapų testavimų, juos būtų galima sulyginti.

## 4.5 Adapterių kūrimas

Adapteris skirtas tiesiog paimti vieno formato duomenis, juos paversti į kitą formatą ir persiųsti toliau. Adapterio konstravime esminis momentas yra sąsajų su moduliais, tarp kurių atnaujinamas bendravimas, formavimas. Iš vienos pusės modulį pasiekia duomenys iš RTL modulio, iš kitos pusės duomenimis keičiamasi su TLM moduliu. Arba atvirkščiai. Duomenų magistralės atveju reikalingi dviejų rūšių adapteriai, dėl duomenų judėjimo skirtingomis kryptimis iš RTL į TLM ir iš TLM į RTL modulius. Vieni naudojami prijungti valdančiuosius (B), kiti pavaldžiuosius įrenginius (A) ir teisėją (C) prie magistralės.




**15 pav. Skirtingi adapterių tipai – teisėjui, valdantiesiems ir pavaldiesiems įrenginiams**

Adapteris prie pavaldžiojo įrenginio ir teisėjo modulio atliks kanalo funkciją. Jame yra realizuota sąsajos funkcija, kuri iš magistralės gautus kontrolės duomenis „išlukštena“ iš struktūros ir siunčia signalais pavaldžiajam įrenginiui. Kai šis atsiunčia atgal savo reakcijas, adapteris jas vėl „supakuoja“ į struktūrą, perskaitytus duomenis įrašo tam tikru adresu ir pasiunčia atgal magistralei. Šis adapteris jautrus visiems į jį ateinantiems iš pavaldžiojo įrenginio signalams. Adapterio pseudokodą galima pasižiūrėti žemiau:

```
class tlm_adapt_slave:  public tlm_bus_slave_if, public sc_module {
     sc_out <sc_bv<addr_bwidth> >          HADDR1;
     sc_out <sc_logic>                     HWRITE1;
     .
     .
     sc_in <sc_bv <data_bwidth>            HRDATA1;
     .
     .

     public:   tlm_adapt_slave (sc_module_name name_):    sc_module (name_) {

               SC_METHOD (transfer);
               sensitive << HRDATA1 << ..<<..;
               tlm_slave_status transfer (sc_bv <data_bwidth> *data, slave_control
addr_ctrl )
          {
          HADDR1.write = addr_ctrl.HADDR;
          HWRITE1.write = addr_ctrl.HWRITE;
          .
          .
          .
          wait ();
          *data = HRDATA1.read ();
          .
          .
          return (HRESP1.read ());
};
```

**1 pvz. Adapterio prie pavaldžiojo modulio pseudokodas**

Adapteris prie valdančiojo įrenginio veikia truputį kitaip. Jis jau nėra kanalas, kuris realizuoja sąsają. Šis adapteris tarsi perima valdančiojo įrenginio funkciją – kreipiasi į sąsajos funkciją, kuri realizuota magistralėje. Jis turi savyje išėjimo prievadą *sc_port <master_if>* per kurį ir kreipiasi į magistralę. Adapterio funkcija – perskaityti duomenis iš valdančiojo modulio signalų ir per sąsają juos pasiųsti magistralei ir laukti atsakymo. Žemiau pateikti šio adapterio karkasas:

```
#include "tlm_bus_master_if.h"
.
.
.

class tlm_bus_master_adapt: public sc_module {
public:
     sc_in <addr_bwidth>                   HADDR;
     sc_in <sc_logic>                      HWRITE;
          .
          .
          .
     sc_out <sc_bv <data_bwidth> >         HRDATA;
     sc_port <tlm_bus_master_if, 0>        bus_port;

     SC_HAS_PROCESS (tlm_bus_master);

     tlm_bus_master_adapt (sc_module_name name_, sc_module (name_){

          SC_THREAD (main_action);
          sensitive_pos << HADDR << HWRITE << .. << ...;
     }
```

```
     void main_action () {
          control_out.HADDR = HADDR.read();
          control_out.HWRITE = HWRITE.read ();
               .
               .
               .
          control_in = bus_port-> transfer_master (&data, control_out);
          HRDATA.write = data;
               .
               .
               .
     };
     sc_bv <data_bwidth>          data;
     master_control_in       control_in;
     master_control_out      control_out;
};
```

**2 pvz. Adapterio prie valdančiojo modulio pseudokodas**

Adapterių formavimo priežastis – apibrėžti sąsajas, kuriomis vėliau magistralės modulis bus sujungtas su periferiniais įrenginiais. TLM modelyje visa kontrolinė informacija ir duomenys siunčiami supakuoti į struktūras. Žinodami, kad *CoCentric RTL* sintezatorius nepalaiko SystemC *struct* operatoriaus, formuodami RTL magistralės sąsają visas struktūras turime išskleisti iki SystemC duomenų tipo prievadų. Pavyzdžiui tokį išėjimo prievadą

```
sc_out <master_control_in> master_out
```

tenka išskleisti iki tiek prievadų, kiek *master_out* struktūra turi signalų nuosavo SystemC tipo:

```
     sc_out <sc_logic>       HGRANTx;
     sc_out <sc_logic>       HREADY;
     sc_out <sc_uint <2> >   HRESP;
     sc_out <sc_logic>       HRESETn;
```

Tai tenka padaryti su visais įeinančiais ir išeinančiais signalais.
Sistemoje atsiradus RTL lygio modulių, reikalinga startiniame modulyje, kuriame sukuriami sistemos moduliai, sujunginėti signalus, mūsų atveju, signalus tarp adapterių ir periferinių įrenginių – pavaldžiųjų ir valdančiųjų įrenginių.

## 4.6 Adapterių ir TLM modulio suliejimas

Šis procesas reikalauja gana daug kodo keitimo. Todėl tai turi būti ypač kruopščiai atlikta. Šio etapo metu eliminuojami adapteriai kaip atskiri moduliai – jie transformuojami į magistralės modulio vidines funkcijas.

*RTL sąsajų perkėlimas į TLM magistralę.*

Pirmiausia reikalinga adapterių RTL sąsajas perkelti į magistralės modulio sąsajų dalį. Valdančiojo įrenginio sąsaja įkeliama į magistralės TLM modulį. Pavaldžiojo įrenginio sąsają taip pat reikalinga įkelti. Prie to, dar reikalinga iš TLM modulio pašalinti prievadą `sc_port <tlm_bus_slave_if, 0>`, modeliuojant TLM lygyje naudotą kaip sąsają su pavaldžiaisiais įrenginiais, kurį keičiame iš pavaldžiojo adapterio įkeliamu RTL skirtu prievadų rinkiniu.

```
sc_port <tlm_bus_slave_if, 0>      sc_out <sc_bv <2 > >              HTRANS
                                   sc_out <sc_bv <add_bwidth > >     HADDR
                                   sc_out <sc_logic >                HWRITE
                                   sc_out <sc_uint <3 > >            HSIZE
                                   sc_out <sc_bv <3 > >              HBURST;
                                   sc_out <sc_bv <data_bwidth > >    HWDATA;
                                   sc_in <sc_uint <2 > >             HRESP_1
                                   sc_in <sc_bv <data_bwidth > >     HRDATA_1;
                                   sc_in <sc_logic >                 HREADY_1;
```

**3 pvz. Prievadų sc_port keitimas signalais sc_in ir sc_out**

*Buferių sukūrimas*

Įeinantiems ir išeinantiems signalams reikalinga sukurti buferį jiems saugoti, taip pat, realizuoti srautinę architektūrą. Po kiekvieno sinchronizacijos signalo iš visų *sc_in* prievadų nuskaitomos reikšmės ir surašomos į buferius. Kontrolės informacija ir duomenys periferiniams įrenginiams į signalus *sc_out* taip pat nuskaitoma iš buferio

```
sc_bv <4 >                  delay_hmaster;
sc_bv <2 >                  temp_htrans;
sc_bv <add_bwidth >         temp_haddr;
sc_bv <data_bwidth>         temp_data;
sc_logic                    temp_hwrite;
sc_bv <3 >                  temp_hsize;
sc_bv <3 >                  temp_hburst;
```

**4 pvz. Buferis signalams saugoti**

*Adapterių konvertavimas į funkcijas*

Į magistralės modulius sukėlus prievadus signalams iš periferinių įrenginių, galima adapterio, kaip modulio, funkcionalumą perkelti į magistralės modulį kaip procesą. Šio proceso sinchronizacijai patartina sukurti atskirą sinchronizacijos signalą, kurio dažnis būtų gerokai didesnis už visos magistralės darbinį dažnį. Po kiekvieno sinchro signalo jis nuskaito duomenis iš prievadų ir įrašo jį į buferį arba atvirkščiai - skaito iš buferio ir rašo į signalus – priklausomai nuo adapterio tipo. Iš buferio visą informaciją nuskaito pagrindinis magistralės procesas *transfer*. Teks modifikuti ir pagrindinį procesą. Jis duomenis taip pat turėtų skaityti ir rašyti duomenis į buferį.

Skaitymas ir rašymas į buferį – tai apibendrintas atvejis. Skirtingais atvejais galima rašyti ne į buferį, o tiesiai į atitinkamą prievadą išvengiant nereikalingo rašymo/skaitymo. Tad verta permąstyti kiekvieno signalo galimybes būti rašomam ne į buferį, bet tiesiogiai į signalą.

Šio etapo metu, adapterius konvertuojant į funkcijas, eliminuojami sąsajų *tlm_bus_slave_if*, *tlm_bus_master_if* ir *tlm_bus_master_arbiter* paveldimumas. Į magistralės modulyje realizuotas skaitymo/rašymo funkcijas dabar kreipiamasi iš konvertavimo funkcijos. Tad paveldimos sąsajos funkcijos čia tampa nebereikalingos.

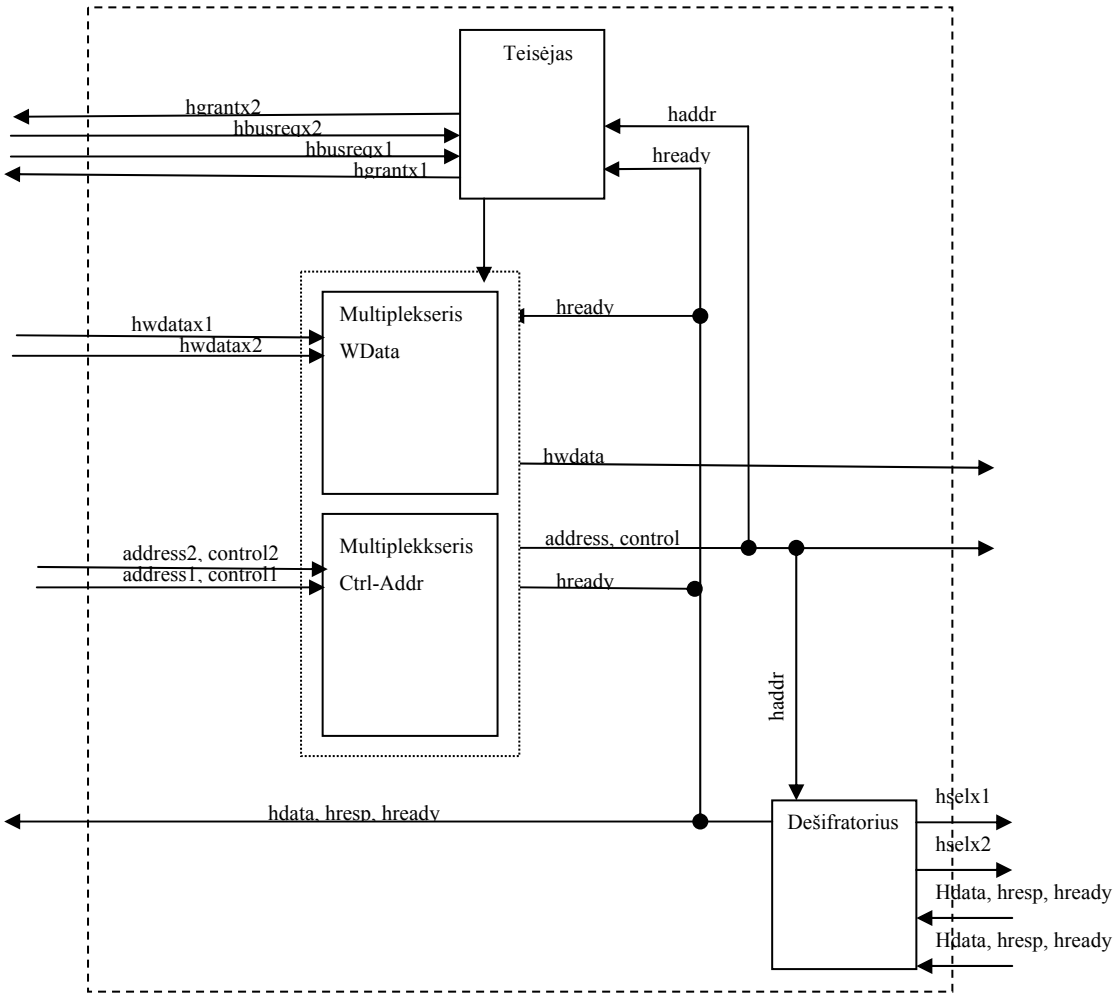
Po šio etapo patartina atlikti išsamų testavimą. Ypatingą dėmesį būtina atkreipti į sinchronizaciją.

Eliminavus paveldimumą, magistralė tampa RTL lygio. Šioje fazėje, atlikus testavimą, būtų galima bandyti aprašą sintezuoti, tačiau apraše yra palikta daug „pritempimo“ prie RTL lygio priemonių, pvz. - adapterių funkcijos, dėl ko „geležies“ poreikis bus didesnis, nei galėtų. Todėl prieš sintezuojant rekomenduojama modulį „išvalyti“. Sekančiu etapu bandysime tai atlikti.

Apibrėžti is anksto kiek bus periferijos

## 4.7 Restruktūrizacija

Šis etapas skirtas eliminuoti, nuo TLM lygio aprašą transformuojant iki elgsenos lygio, likusias papildomas proceso priemones. Ir modifikuoti struktūrą sufleruojant sintezatoriui modulio sudėtį. Stengtasi išskirti labiausia struktūriškai nepriklausomas dalis. Restruktūrizacijos procesas kiekvienam transformacijos atvejui yra individualus – ypač sinchronizacija, mūsų atveju taip pat srautinio principo išlaikymas. Magistralės funkcionalumą galima suskaidyti į tris modulius – multiplekserį, dešifratorių ir teisėją. Multiplekserio funkcija – pagal teisėjo sprendimą, valdančiojo įrenginio kontrolės duomenis nusiųsti atitinkamam pavaldžiajam įrenginiui. Dešifratorius pagal adresą nusprendžia, kokiais signalais siųsti kontrolės duomenis ir iš kur priimti rezultatus. Paveiksle matyti, kaip pakito magistralės struktūra po restruktūrizacijos.




**16 pav. Magistralės modulis po restruktūrizacijos**

Po šio proceso vienas magistralės modulis suskaldomas į tris dalis. Dešifratorius ir teisėjas dirba pagal sisteminį laikrodį. Centrinis multiplekseris – pagal didesnio dažnio laikrodį. Šie trys moduliai tarpusavio sinchronizacijai privalo turėti kontrolės signalus. Vietoj jų, kartais įmanoma panaudoti esamus signalus, tiesiog prijungus prie papildomo išsišakojimo. Dešifratorius su multiplekseriu turi būti sujungti per adresų magistralę, kad dešifratorius pagal adresą galėtų nuspręsti, kurį pavaldųjį įrenginį aktyvuoti magistralės operacijai. Teisėjui aktualu žinoti, kada pavaldusis įrenginys pabaigia darbą ir pagal tai aktyvuoti sekantį valdantįjį įrenginį. Multiplekseris gauna duomenis iš teisėjo, kurią informaciją išrinkti iš ateinančių signalų

## 4.8 Periferinio įrenginio pajungimas

TLM lygio apraše periferinio įrenginio prijungimas yra labai paprastas. Pirmiausia aprašomas ir sukuriamas naujas komponentas pagrindiniame startiniame RTL sistemos modulyje *tlm_bus_test.* Jam priskiriamas unikalus numeris, kad būtų galima jį atpažinti. Čia naujas modulis gali pradėti dirbti.

RTL lygio apraše periferinio įrenginio prijungimas yra sudėtingas. Prijungimą komplikuoja ir sudėtingos periferinių įrenginių sąsajos. Atitinkamus signalus reikalinga prijungti prie visų trijų magistralės modulių. Duomenų ir kontrolė&adresas signalai prijungti prie centrinio multiplekserio, priėjimo prie magistralės signalai prie teisėjo, pavaldžiųjų įrenginių įėjimo signalai prijungti prie dešifratoriaus.

# 5 Eksperimentinė dalis

## 5.1 Įžanga

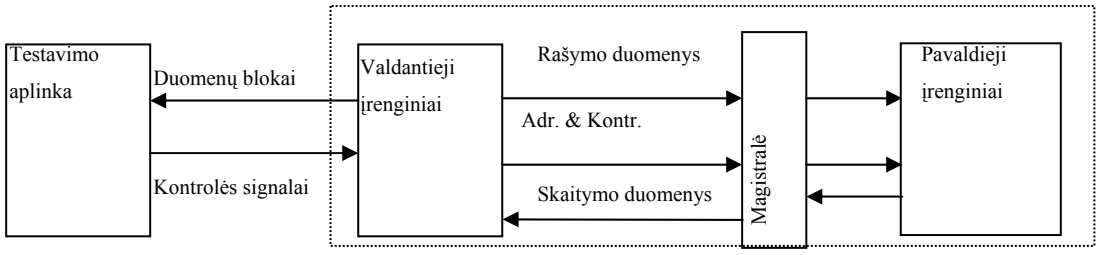
Tyrimo metu buvo suformuluota metodika operacijų lygio aprašui transformuoti į elgsenos lygį ir šį aprašą sintezuoti iki RTL aprašo *Verilog*'e ir ventilinių vaizdų. Eksperimentinėje dalyje bus pateiktas AMBA AHB magistralės aprašas elgsenos lygyje, kuris gautas pagal tiriamojoje dalyje suformuluotą metodiką, taip pat magistralės TLM ir elgsenos lygių modeliavimo laikų sulyginimas ir sintezavimo rezultatai, kurie patvirtina metodikos teisingumą.

## 5.2 Magistralės elgsenos lygio modelis

### 5.2.1 Architektūros apžvalga

Teisingai veikiantį magistralės prototipą sudaro dvi dalys: AMBA AHB magistralė ir aplinka skirta jai testuoti, kuri darbo pradžioje siunčia sistemai inicializavimo informaciją, vėliau – duomenų blokus duomenims perduoti. Testavimo aplinka yra sujungta tik su valdančiaisiais įrenginiais. Šiuo atveju AMBA AHB sistemą sudaro 2 valdantieji, 2 pavaldieji, 2 iškodavimo įrenginiai, teisėjas, centrinis multiplekseris signalų perjunginėjimui. Sistema turi du skirtingo dažnio laikrodžio signalus. Vienas iš jų sinchronizuoja valdančiųjų, pavaldžiųjų įrenginių, teisėjų ir testavimo aplinkos darbą. Kitas valdo centrinio multiplekserio darbą. Šis laikrodis yra daug didesnio dažnio nei pirmasis.

Magistralė veikia srautiniu (*pipelining)* principu t. y. magistralė neturi tuščių ciklų laukimui kol bus paruošti kontrolės signalai duomenys ir t.t.. Duomenys kiekvienai operacijai paruošiami paskutinės operacijos metu. Projektavimas srautiniu principu sumažina signalų vėlavimą, padidina magistralės pralaidumą.



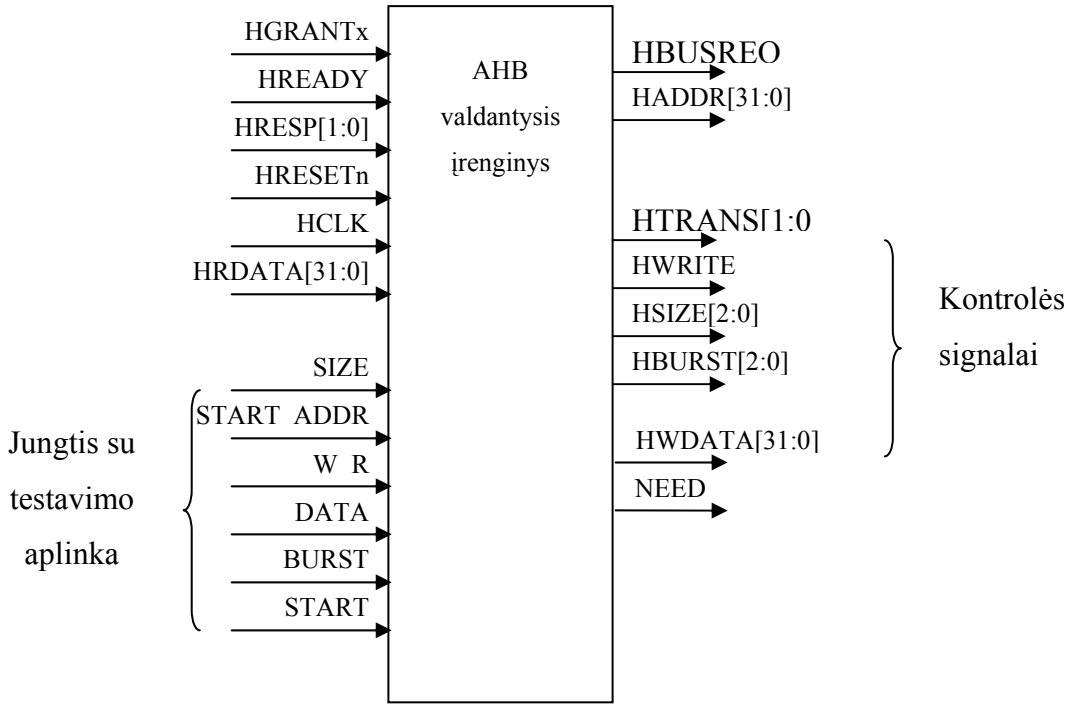

**17 pav. Bendra magistralės schema**

### 5.2.2 Duomenų srautų diagrama

Pilną duomenų srautų diagramą galima rasti priede 1. Virš rodyklių surašyti signalų vardai. Signalai sujungia atitinkamus modulius. Punktyrinės linijos stačiakampiai reiškia modulius, kurie yra įtraukti į vieną esybę.

### 5.2.3 Modulių sąsajos

#### 5.2.3.1 Valdantysis įrenginys



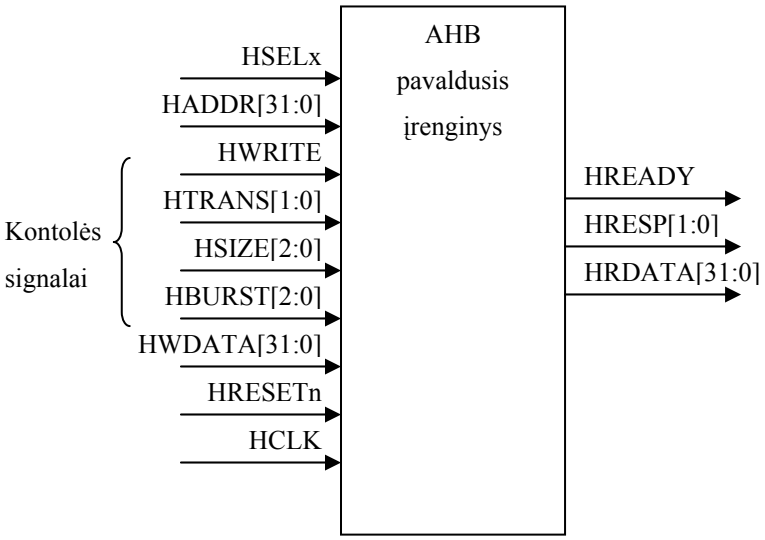

**18 pav. Valdančiojo įrenginio sąsaja**

Valdantysis įrenginys yra jautrus *HCSLK* signalui, kuris prijungtas prie *clk* laikrodžio.

Signalai *SIZE, START_ADDRESS, W_R, DATA, BURST, START* naudojami gauti naujus duomenų blokus magistralės darbui. Duomenų blokai - tai duomenys skirti simuliacijai. Į šį bloką įtraukta visa informacija reikalinga imituoti magistralės darbą. Paprasto vienetinio arba pirmojo monopolinės operacijos takto atveju valdantysis įrenginys paprasčiausiai persiunčia informaciją iš šių prievadų į atitinkamus signalus, prijungtus „tiesiogiai“ su pavaldžiuoju įrenginiu. Jeigu tai nėra pirmasis monopolinės operacijos takto ciklas, duomenų blokus pavaldžiajam įrenginiui valdantysis įrenginys generuoja pats. Prieš kiekvieną paprastą duomenų perdavimą arba prieš paskutinį monopolinės operacijos taktą, valdantysis įrenginys nusiunčia signalą į prievadą *NEED*, su tikslu duoti žinią testavimo aplinkai, kad ji jam jau gali siųsti naują duomenų porciją. Signalas *START* naudojamas

inicijuoti valdančiojo įrenginio darbą. Po šio signalo valdantysis įrenginys siunčia *HBUSREQx* signalą teisėjui.

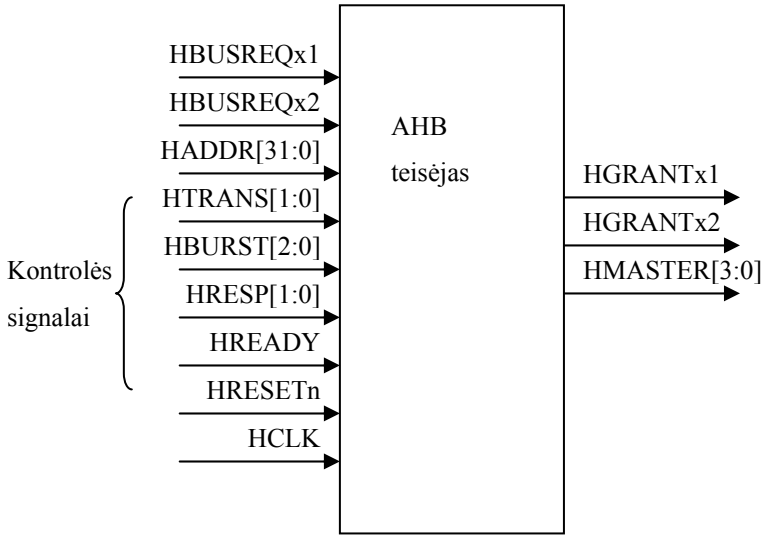
#### 5.2.3.2 Pavaldusis įrenginys




**19 pav. Pavaldžiojo įrenginio sąsaja**

Pavaldusis įrenginys yra jautrus *HCLK* signalui, kuris yra prijungtas prie laikrodžio *clk*.

#### 5.2.3.3 Teisėjas



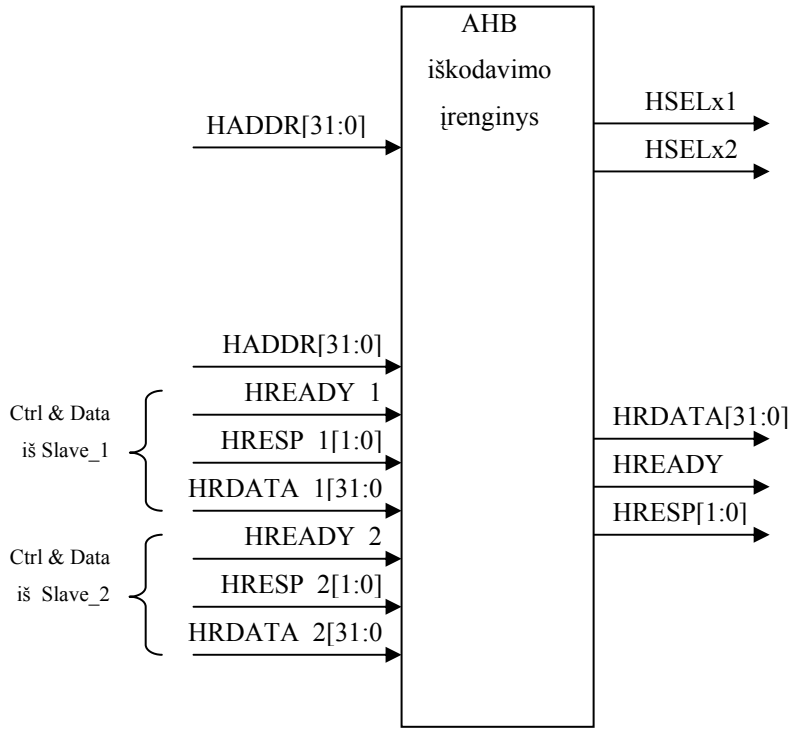

***20 pav. Teisėjo sąsaja***

Teisėjas yra jautrus į *HCLK* signalą, kuris yra prijungtas prie *clk* šaltinio.

Mūsų atveju, teisėjavimo principas yra labai paprastas. Jeigu daugiau nei vienas valdantysis įrenginys išsiunčia reikalavimus priėjimui prie magistralės, teisėjas patikrina, kuris valdantysis įrenginys ilgiausią laiką nenaudojo magistralės. Jeigu šie dydžiai vienodi, priėjimą prie magistralės gauna tas valdantysis įrenginys, kuris turi mažiausią numerį. Pirmas valdantysis įrenginys pagal nutylėjimą yra su numeriu 1.

Jungiant sudėtingesnius valdančiuosius ir pavaldžiuosius įrenginius ir turint specifinius reikalavimus, šį algoritmą galima lengvai modifikuoti.

#### 5.2.3.4 Iškodavimo įrenginiai

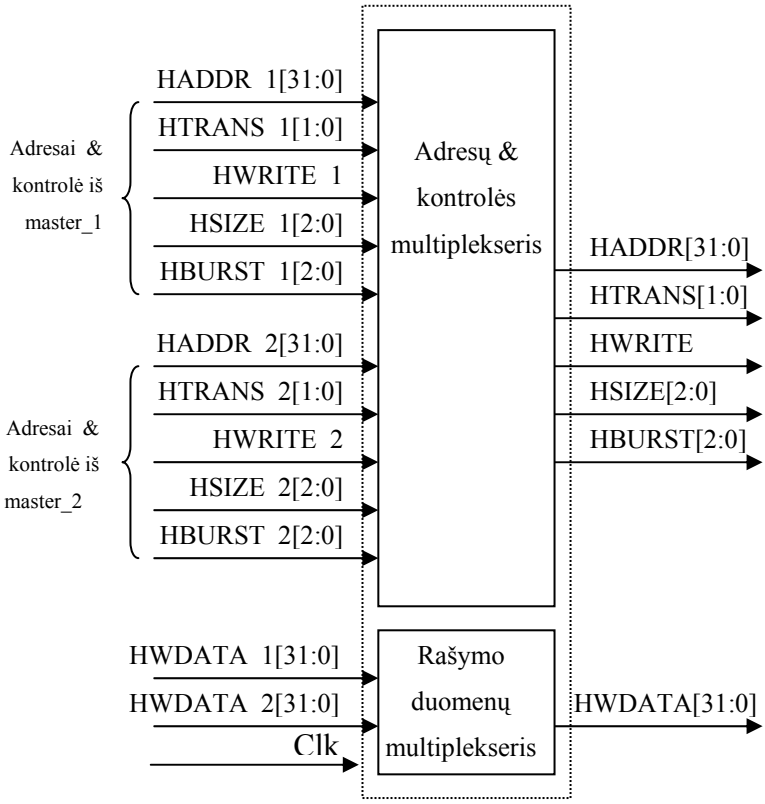



**21 pav. Iškodavimo įrenginio sąsaja**

Iškodavimo įrenginys yra jautrus *HADDR* signalams.

Iškodavimo įrenginio specifikacijoje buvo apibrėžtas tik įėjimo signalas – *HADDR*. Taigi, multiplekseris negali pasakyti, kurios rūšies duomenų perdavimas dabar vyksta (skaitymas ar rašymas). Šita svarbu žinoti, kai dešifravimo įrenginys išrinkinėja perskaitytą informaciją iš pavaldžiųjų įrenginių. Ši dešifravimo įrenginio dalis turėtų dirbti tik tada, kai *HWRITE* yra žemo lygio. Be rašymo-skaitymo informacijos dešifratorius iš pavaldžiųjų įrenginių perjunginėja informaciją nepriklausomai nuo dabar vykstančio proceso tipo – rašymo ar skaitymo.

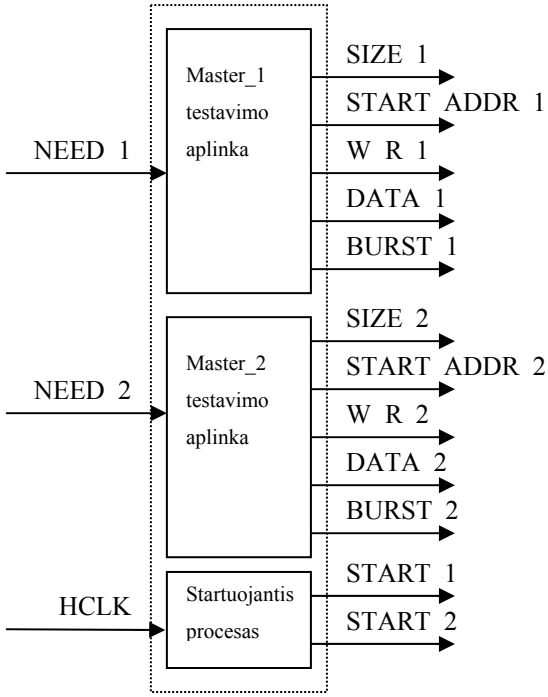
#### 5.2.3.5 Centrinis multiplekseris




**22 pav. Centrinio multiplekserio sąsaja**

Centrinis dešifratorius yra jautrus į antrąjį sistemos laikrodį – *clk2*. Taigi, ši magistralės dalis dirba daug greičiau, nei likusieji magistralės komponentai. Multiplekseris viduje savęs turi du procesus. Jie vykdo skirtingas užduotis. Vienas iš jų išrenka atitinkamą HWDATA prievadą, kitas išrenka vieną iš adresas&kontrolė signalų rinkinių. Multiplekseris yra padalintas į dvi dalis, nes dėl pipelining architektūros yra reikalinga išrinkinėti skirtingus valdančiųjų įrenginių duomenų rinkinius: rašymo duomenis ir adresas&kontrolė. Signalo uždelsimas einant per multiplekserį lygus laikrodžio *clk2* periodui.

#### 5.2.3.6 Testavimo aplinka




**23 pav. Testavimo aplinkos sąsaja**

Testavimo aplinka atsakinga už teisingų duomenų, reikalingų magistralės modeliavimui pateikimą valdantiesiems įrenginiams. Šis modulis turi 3 procesus. Vienas iš jų naudojamas inicijuoti valdančiųjų įrenginių procesus siunčiamas jiems *START_x HIGH* signalą ( x – valdančiojo įrenginio numeris). Šis procesas yra jautrus į laikrodžio *clk* signalą.

Testavimo aplinka duomenų tiekimui kiekvienam valdančiam įrenginiui turi po atskirą procesą. Norint prijungti dar vieną valdantįjį įrenginį, tektų sukurti naują procesą. Taigi, antras ir trečias procesai naudojami tam tikslui. Šie procesai turi skirtingus jautrumo sąrašus. Antrasis yra jautrus į *NEED_1* signalą, trečias į *NEED-2* signalą. Kiekviename cikle jie siunčia naujus duomenų blokus atitinkamiems valdantiesiems įrenginiams.

### 5.2.4 Modulių specifikacija

Modulių išeinančius ir įeinančius prievadus galite rasti 5.2.3 skyriuje.

### 5.2.4.1 Valdantysis įrenginys

```
AutomatState            state          Saugo valdančiojo įrenginio būseną
int                     made_beats     Saudojamas monopoliniai opercijai
int                     MastNumb       Valdančiojo įrenginio numeris
bool                    temp_need      Indukuoja naujų duomenų poreikį
sc_logic                t_hwrite       BUFERIAI skirti saugoti iš testavimo
sc_bv <data_bwidth >    t_hwdata       aplinkos gautus duomenų blokus
sc_uint <add_bwidth >       t_haddr
sc_uint <3 >                t_hsize
sc_bv <3 >                  t_hburst
sc_bv <2 >                  t_htrans
```

void master_proc () - pagrindinis VĮ procesas valdantis vidines būsenas

void temp_load_to_bus () - procedūra skirta duomenų siuntimui iš vidinio duomenų buferio į magistralę.

void TB_load_to_temp () - procedūra skirta duomenų priėmimui iš testavimo aplinkos į vidinį buferį

void reset_signals () – nustato VĮ vidinę būseną ir prievadus į pradinę būseną.

void put_data_to_right_place () – siunčiamus į magistralę duomenis perkelia į reikalaujamas menų šynos linijas.

bool INCR_WRAP () – *incremental* ar *wraping* operacija ?

int burst_beat () – nustato kelių taktų operacija bus vykdoma.

bool end_SEQ () – funkcijos reikšmė teigiama, kai baigiasi *sequencial* operacija

void fix_temp () – vidinio buferio korekcijos

int decode_hsize () – pagal *hsize* signalą suformuoja *hsize* dydį baitais.

int get_f2 () – nustato nuo kurio bito duomenų šynoje prasideda siunčiami duomenys.

### 5.2.4.2 Pavaldusis įrenginys (PĮ)

```
enum { SelNumb = 5 };
enum { data_bwidth = 32 };              duomenų šynos plotis
enum { add_bwidth = 32 };               adresų šynos plotis
enum { mem_size = 32 };                 atminties vektorių kiekis
enum { mem_vect_length = 8 };           atminties vektoriaus ilgis
enum AutomatState { idle, transfer};    PĮ vidinė būsena


sc_bv<mem_vect_length>  MEM [mem_size]     PĮ atmintis (RAM)
sc_bv <add_bwidth >     address_buffer;    buferis saugoti adresui
sc_uint <3 >            hsize_buffer;      buferis saugoti hsize signalui
sc_logic                hwrite_buffer;     buferis saugoti hwrite signalui
AutomatState            state;             PĮ automato būsena
bool                    before_last;       priešpaskutinis burst operacijos taktas?
```

void slave_proc() – pagrindinis pavaldžiojo įrenginio procesas.

void sim_read_data () – paprasta skaitymo operacija.

void sim_write_data () – paprasta rašymo operacija

void read_addr_ctrl () – adreso ir kontrolės signalų nuskaitymas

void reset_signals () - vidinės būsenos ir signalų nustatymas į pradinę būseną

int decode_hsize () - hsize signalą suformuoja hsize dydį baitais.

int get_f2 () – nustato nuo kurio bito duomenų šynoje prasideda siunčiami duomenys.

#### 5.2.4.3 Teisėjas

void arbiter_proc () – pagrindinis teisėjo procesas

void change_grant () – prieinančio prie magistralės valdančiojo įrenginio keitimas

int burst_beat () – kelintas burst operacijos taktas?

#### 5.2.4.4 Iškodavimo įrenginiai

void decoder_proc () – pagrindinis dekodavimo procesas

#### 5.2.4.5 Centrinis multiplekseris

```
sc_bv <2 >              temp_htrans;
sc_bv <add_bwidth >     temp_haddr;         Laikinas buferis ateinančių signalų
sc_logic                temp_hwrite;        saugojimui
sc_bv <3 >              temp_hsize;
sc_bv <3 >              temp_hburst;
```

void mux_proc_data() - pagrindinis centrinio multiplekserio procesas.

void mux_proc_ctrl_addr() - procesas adresų ir kontrolės signalams perjunginėti.

#### 5.2.4.6 Testavimo aplinka

```
sc_logic                sig_reset;          // pirmagrįžos signalas.
```

void testbench_proc_1() – procesas atsakingas už duomenų blokų pristatymą pirmajam valdančiajam įrenginiui.

void testbench_proc_2()– procesas atsakingas už duomenų blokų pristatymą antrajam valdančiajam įrenginiui.

void start_proc () – procesas skirtas pastartuoti sistemos darbą.

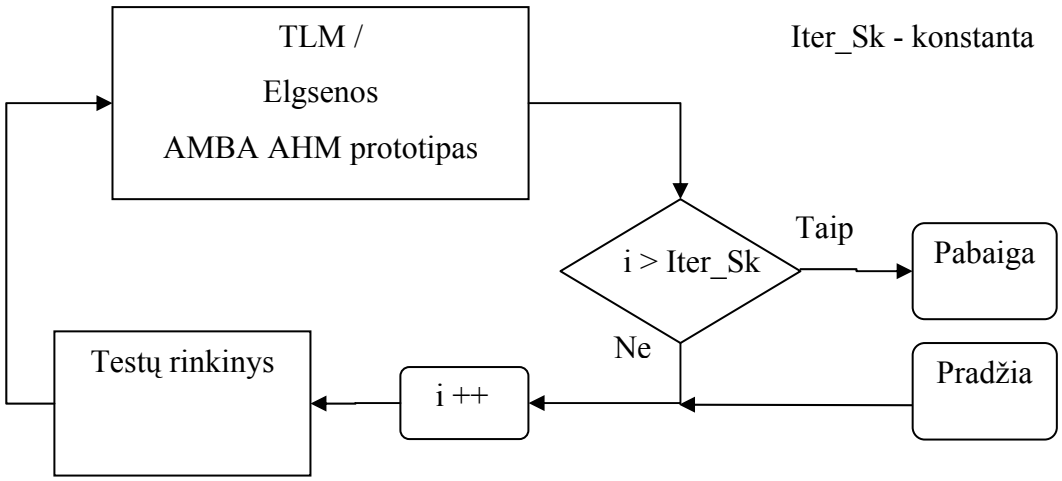
sc_bv<32 > get_addr (int, int ) – formuoja adresą valdančiajam įrenginiui.

### 5.2.5 Testavimas

AMBA AHB magistralės testavimo rezultatus – laiko diagramas - galima rasti priede nr. 2.. Modeliavimo laiko diagramos gautos iš SystemC priemonėmis (sc_trace()) suformuotų vcd failų. Vcd failų vaizdavimui naudotas programinis įrankis *GTKwave_Win.*

## 5.3 Elgsenos ir operacijų lygio modelių simuliacijos greičių palyginimas

Prieš atliekant simuliacijos greičio testavimus, iš aprašų pašalintos visos I/O nereikalingos operacijos ankščiau naudotos kodo trasavimui- tokios kaip *cout, printf,* taip pat standartinė SystemC priemonė *sc_trace.* Tai reikalinga, kad testavimo rezultatai nebūtų iškreipti į magistralės funkcionalumą neįeinančių operacijų. Kodas paliktas tik tas, kuris būtinas realizuoti *AMBA AHB* protokolą.
Testai buvo atlikti ant universiteto serverio *kopustas.elen.ktu.lt.* Matavimo metu serveryje nepastebėta žymių apkrovimų. Procesorius visų matavimų metu būdavo apkrautas iki 1% vidinių serverio procesų.




**24 pav. TLM ir elgsenos prototipų simuliacijos spartos matavimo schema**

Tiek elgsenos, tiek TLM lygio modelių testavimui naudojami tokie pat duomenys, kaip ir naudoti sistemų testavimui atskirai. Kad išvengtume didelio kiekio testinių rinkinių generavimo, šie testiniai rinkiniai siunčiami pakartotinai nurodytą kiekį kartų. Simuliacijos greitis matuojamas operacijų kiekiu per laiko vienetą – sekundę. Operacija vadinsime vieną kreipinį į pavaldųjį įrenginį. Tad pvz. keturių taktų monopolinę operaciją sudarys keturios operacijos. Paprastas skaitymas/rašymas – viena operacija. Kiekvieno modelio testavimas atliktas keletą kartų su skirtingu operacijų kiekiu. Tai reikalinga išlyginti simuliavimo greičio svyravimus dėl nenumatytų kompiuterio, ant kurio vyksta modeliavimas, apkrovimų. Su kiekvienu operacijų kiekiu atlikta po 5 bandymus. Lentelėje užrašyti šių

bandymų vidurkiai. Lentelės skiltyje vidurkis suskaičiuotas bendras visų bandymų vidurkis, pagal kurį nubraižytas spartų palyginimo grafikas.

***1 lentelė.***

**TLM abstrakcijos lygio modelio simuliacijos sparta**

| Atlikta operacijų | 500K | 800K | 1000K | 1200K | 15000 K | 1700K | 2000K | Vidurkis |
|---|---|---|---|---|---|---|---|---|
| Modeliavimo laikas, s | 3 | 5 | 6 | 8 | 10 | 11 | 13 | ---- |
| Modeliavimo sparta, op/sek | 366666 | 360000 | 366666 | 350000 | 350000 | 354545 | 353846 | 357389 |

Standartinis nuokrypis $S_{TLM}$= 6641.62

Vidurkis V= 357389

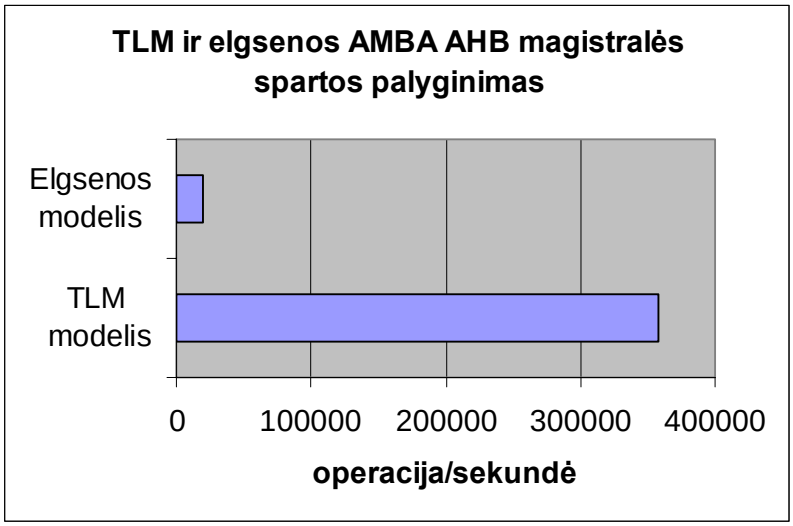
***2 lentelė.***

**Elgsenos abstrakcijos lygio modelio simuliacijos sparta**

| Atlikta operacijų | 100K | 200K | 300K | 400K | 500K | 1000K | Vidurkis |
|---|---|---|---|---|---|---|---|
| Modeliavimo laikas. s | 5 | 10 | 15 | 20 | 25 | 51 | ----- |
| Modeliavimo sparta, op/sek | 20000 | 20000 | 20000 | 20000 | 19230 | 19607 | 19806 |

Standartinis nuokrypis $S_{elgsenos}$= 295

Vidurkis $V_{elgsenos}$= 19806

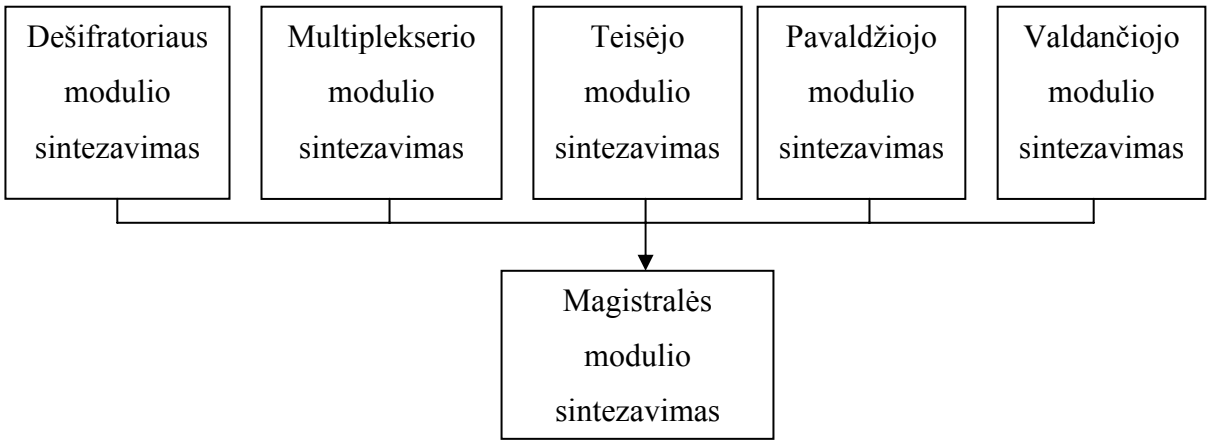



**25 pav. TLM ir elgsenos AMBA AHB magistralės prototipų spartos palyginimas**

Iš paveikslo nr. 13 galima pastebėti, kad TLM modelio simuliacija ~18 kartų spartesnė, nei elgsenos modelio. Jeigu mes prisimintume paveikslą nr. 4, pastebėtume, kad mes lyginome modelius realizuotus pagal programuotojo požiūrį su sinchronizacija ir modelį su ciklų kvietimu, jautrų laikrodžio frontams. Egzistuoja galimybė gauti didesnius skirtumus lyginant programuotojo požiūrio aprašą be sinchronizacijos ir RTL aprašą. Tačiau tai jau būtų kito darbo uždavinys.

## 5.4 Sintezė

### 5.4.1 Sintezės planavimas




**26 pav. Magistralės sintezavimo eiga ir etapai**

Elgsenos lygio magistralę sudaro penki komponentai: dešifratorius, multiplekseris, teisėjas, pavaldusis ir valdantysis įrenginiai. Jų sujungimo schema apibrėžta pagrindiniame magistralės modulyje. Tad sistemos sintezės procesas susideda iš dviejų etapų:

- Kiekvieno modulio atskiras sintezavimas ir išsaugojimas
- Susintezuotų modulių užkrovimas į atmintį ir sintezavimas apjungiant juos per magistralės modulį

Modelių sintezavimui, kad išvengtume rankinio sintezavimo komandų įvedinėjimo sintezatoriui, naudojami skriptai. 5 – ame priede pateiktas bendras skriptas, kurį naudojau sistemos sintezavimui.
Sintezavimui naudotos bibliotekos:
synthetic_library "dw01.sldb dw02.sldb"
target_library "tc6a_cbacore.db"
symbol_library "tc6a_cbacore.sdb"

### 5.4.2 Modelių paruošimas sintezei

Modeliuojama sistema retai kada būna iš karto sintezuojama. Ne visas konstrukcijas gali suprasti sintezatorius. Tenka modelių aprašą dar kartą peržiūrėti, modifikuoti sintezei netinkančias konstrukcijas. Keičiant konstrukcijas būtina išlaikyti tą patį funkcionalumą. Pakeitus netinkamą vietą būtina sistemą vėl ištestuoti ir sutikrinti rezultatus, kad rezultatai nepasikeistų. Ruošdamas aprašą sintezei susidūriau su keliomis problemomis.

Teko taisyti kodo vietą, kur iš bitų vektoriaus *fun* skirtingų vietų, bandžiau iškirpti 8 bitų pločio atkarpą. Bet sintezatorius reikalavo, kad iškerpamos atkarpos ribos būtų konstantės. Tad teko šią konstrukciją pakeisti range metodu su fiksuotais intervalo kraštais ir vektoriaus poslinkiu per 8 bitus į kairę. Funkcionalumas gautas tas pats. Sudėtingumas praktiškai nepakito.

```
sc_bv <data_bwidth > fun
int k = decode_hsize () / 8;

for ( int i=0; i< k; i++ ) {
     fun.range ( (get_f2()+8*(i+1)-1),
     };



// Nesintezuojamas kodas
```

```
sc_bv <data_bwidth > fun;
int k = decode_hsize () / 8;

for ( int i=0; i< k; i++ ) {
          fun = fun << 8;
          fun.range (7, 0) = MEM [ind+t-
1];
     };
};
fun = fun << get_f2 ();

//Sintezuojamas kodas
```

**27 pvz. Sintezės apribojimai range metodui. Range intervalai turi buti const int tipo ir negali kisti**

Taip pat teko pakeisti *for* ciklą, kurio iteracijų skaičius buvo kintamas. Sintezatorius reikalavo, kad fiksuoto iteracijų skaičiaus. Teko numatyti maksimalią galimą iteracijų skaičių pagal magistralės specifikaciją ir jų skaičių realizuoti cikle. Į ciklą taip pat įdėti sąlygos sakinį, kad perteklinėse ciklo iteracijose nebūtų vykdomi jokie veiksmai. Šis pataisymas yra priežastis perteklinių ciklo iteracijų, bet perkeliant programinį kodą į aparatūrinę įrangą tenka tai toleruoti , dėl specifinių reikalavimų.

```
Int k = decode_hsize ()/8;
for ( int i=0; i< k; i++ ) {
};



// Nesintezuojamas kodas
```

```
int k = decode_hsize () / 8;
for ( int t=128; t >0; t-- ){
     if (t<=k) {
          ……
     };
};
//Sintezuojamas kodas
```

**28 pvz. Sintezės apribojimai for ciklui. Iteracijų skaičius privalo būti fiksuotas**

Aprašo modeliavime nebuvo jokių problemų su *sc_logic* tipo kintamaisiais. Tai tipas kuris turi keturias galimas reikšmes: 0, 1, z, x. SystemC bibliotekoje tai apibrėžiama atitinkamai kaip SC_LOGIC_1,

SC_LOGIC_0, SC_LOGIC_Z, SC_LOGIC_X. Tačiau *CoCentric SystemC* kompiliatorius tokių tipų neranda. Tad teko tokio tipo priskyrimus modifikuoti ir SC_LOGIC_1 bei SC_LOGIC_0 pakeisti į atitinkamai *true ir false*.

```
Sc_out <sc_logic> HREADY;
HREADY.write ( SC_LOGIC_1 );

// Nesintezuojamas kodas
```

```
Sc_out <sc_logic> HREADY;
HREADY.write ( true );

//Sintezuojamas kodas
```

**29 pvz. Sintezatoriaus apribojimas priskiriamų reikšmių sc_logic tipo kintamajam**

Sintezatorius neleidžia modulio konstruktoriuje naudoti *dont_initialize()* ir kintamųjų ar signalų pradinių reikšmių priskyrimui. Iš konstruktoriaus teko paprasčiausiai jas išbraukti. Ir tai realizuoti atskirai HRESETn signale. Sistema savo darbo pradžioje privalo būti nustatoma į pradinę būseną su HRESETn signalu. Tada ji gali pradėti dirbti normaliu režimu.

```
SC_CTOR(master) {
  SC_METHOD(master_proc );
     sensitive_pos << clk;
  state = idle;
  temp_need = true;
  dont_initialize ();
}
```

```
SC_CTOR(master) {
  SC_METHOD(master_proc );
     sensitive_pos << clk;
}
```

**30 pvz.. Nesintezuojami yra dont_initialize() operatoriai konstruktoriuje. Taip pat inicializavimo reikšmių priskyrimas jame**

### 5.4.3 Sintezės rezultatai

#### 5.4.3.1 Ploto ataskaitos

Magistralės komponentuose logika palyginus paprasta, didžiausią plotą užima komponentų laikinoji atmintis naudojama buferiui. Tai ypatingai jaučiama *slave, master ir mux* komponentuose. Vertinant rezultatus reikia nepamiršti, kad sistemoje yra po 2 *slave* ir *master* modulius.

**3 lentelė.**

**Magistralės komponentų ir pačios magistralės plotų ataskaitų palyginimai**

| | Teisėjas | Dešifratorius | Multiplekseris | Pavaldusis įrenginys | Valdantysis įrengiys | Magistralė |
|---|---|---|---|---|---|---|
| Prievadų kiekis | 14 | 139 | 228 | 111 | 216 | 211 |
| Mazgų kiekis | 188 | 114 | 396 | 357 | 670 | 545 |
| Ląstelių kiekis | 150 | 78 | 243 | 284 | 503 | 7 |
| Skirtingų elementų kiekis | 28 | 8 | 7 | 31 | 43 | 5 |
| Kombinacinės logikos plotas | 259.250 | 65.500 | 228.720 | 359.039 | 801.899 | 2673.586 |
| Nekombinacinės logikos plotas | 368.049 | 87.840 | 817.281 | 400.679 | 1072.253 | 3850.9985 |
| Mazgų sujungimų plotas | 1972.968 | 672.031 | 3582.968 | 3266.718 | 6307.031 | 28667.343 |
| Bendras ląstelių plotas | 627.300 | 153.339 | 1046.000 | 759.720 | 1874.150 | 6524.5800 |
| Bendras plotas | 2600.268 | 825.371 | 4628.970 | 4026.438 | 8181.184 | 35191.929 |

#### 5.4.3.2 Laiko ataskaitos

Visi magistralės komponentai išskyrus *Decoder* yra sinchronizuoti pagal sisteminį *clk* signalą. Taigi jų visų laikinės charakteristikos vienodos. Tik modulis *Decoder* turi savo vėlavimą, nes jis jautrus tik į įeinantį adreso signalą. Taigi, jo vėlinimas skiriasi. Magistralės vėlinimas daug didesnis, nes įėjęs signalas sekančiame cikle dar nesukelia reakcijos.

**4 lentelė.**

**Magistralės komponentų laiko parametrų palyginimas**

| | Teisėjas | Dešifratorius | Multiplekseris | Pavaldusis įrenginys | Valdantysis įrenginys | Magistralė |
|---|---|---|---|---|---|---|
| Laikas, ms | 0.82 | 0.89 | 0.82 | 0.82 | 0.82 | 9.65 |

#### 5.4.3.3 Galios ataskaitos

Atskirų komponentų ir magistralės sistemos galios charakteristikos. Jos charakterizuoja sunaudojamą energijos kiekį. Derėtų nepamiršti, kad magistralės sistemoje yra po 2 *slave* ir *master* modulius.

**5 lentelė.**

**Magistralės komponentų galios parametrų palyginimas**

| | Teisėjas | Dešifratorius | Multiplekseris | Pavaldusis įrenginys | Valdantysis įrenginys | Magistralė |
|---|---|---|---|---|---|---|
| Galia, mW | 76.560 | 38.271 | 243.439 | 251.262 | 378.024 | 1695.470 |

Šia fiziniai parametrai (laikas, plotas, galingumas) nėra optimizuoti. Tiesiog tai faktas, įrodantis, kad sukurtos TLM lygio aprašų transformavimo į RTL lygį metodikos pagalba galima sparčiąją magistralę iš TLM lygio transformuoti į ventilių lygį. Sintezavimo rezultatų optimizavimas – tai jau kitas uždavinys.

# 6 Išvados

1. Analizuojant literatūra prieita prie išvados, kad projektavimas aukštame abstrakcijos lygyje – tai aparatūros projektavimo ateitis. Bet tam reikalinga metodika ir įrankiai šio lygio modelių sintezei.
2. Projektavimas aukštame abstrakcijos lygyje – greitas, lankstus ir intuityvus. Modelius paprasta prijungti, atjungti, pakartotinai naudoti.
3. TLM modulio transformaciją iki ventilio lygio sudaro tokie etapai:
    - Adapterių sukūrimas ir prijungimas
    - Adapterių ir TLM modelio suliejimas
    - Modelio restruktūrizacija
    - Elgsenos modulio sintezė
4. AMBA AHB magistralės TLM modelio simuliacija vyksta ~18 kartų greičiau, nei tokios pat specifikacijos elgsenos lygio modelio.
5. Gauti sintezuotos magistralės fiziniai parametrai: plotas – 35192; galia 1,695 W, dažnis 104 Hz

# 7 Literatūros sąrašas

[1] ARM AMBA (TM) Specification (Rev 2.0). <http://www.arm.com>

[2] Ernest F. Pipelining Concepts. – Ellen: Spertus, 2002.

[3] Grimpe E., Fandrey T., Timmermann B. User Guideline for Synthesis Tool. - OFFIS, 2003.

[4] Grimpe E., Fandrey T., Ashenden P.J., Biniasch R., Shubert A., Shubert T. Input language subset specification. - OFFIS, 2002.

[5] Grimpel E., Timmermann B., Fandrey T., Biniasch R., Oppenheimer F. SystemC Object-Oriented Extensions and Synthesis Features. - Oldenburg, Germany: OFFIS Research Institute.

[6] Grotken T., Liao S., Martin G., Swan S. SystemC Design with SystemC. - New York, Boston, Dordrecht, London, Moscow: Kluwer Academic Publishers, 2002. – 219 p..

[7]Grotken T., Synopsys Inc. Transaction Level Modeling with SystemC, 2002.

[8] Įvairūs autoriai Functional Specification for SystemC 2.0 - Final - Version 2.0. - M, 2001, 135 p..

[9] Įvairūs autoriai SystemC Version 2.0 Users's Guide. - 2001, 212 p..

[10] Schulz-Key C., Winterholer M., Schweizer T., Kuhn T., Rosenstiel W. Object-Oriented Modeling and Synthesis of SystemC Specifications. - Tuebingen, Germany: University of Tuebingen.

[11] Synopsys, Inc. CoCentric SystemC Compiler Behavioral Modeling Guide. – 2003.

[12] Synopsys, Inc. CoCentric SystemC Compiler RTL User and Modeling Guide. – 2003.

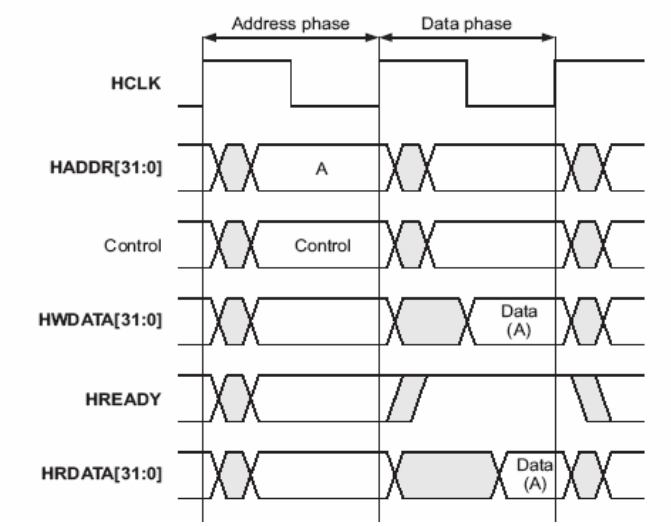
[13] Tabbara B., Hashmi N., Hashmi K. Transaction Level Modeling: verification leaps forward. – EDA Tech Forum, 2005.

# 1 Priedas. AMBA AHB magistralės signalų sąrašas

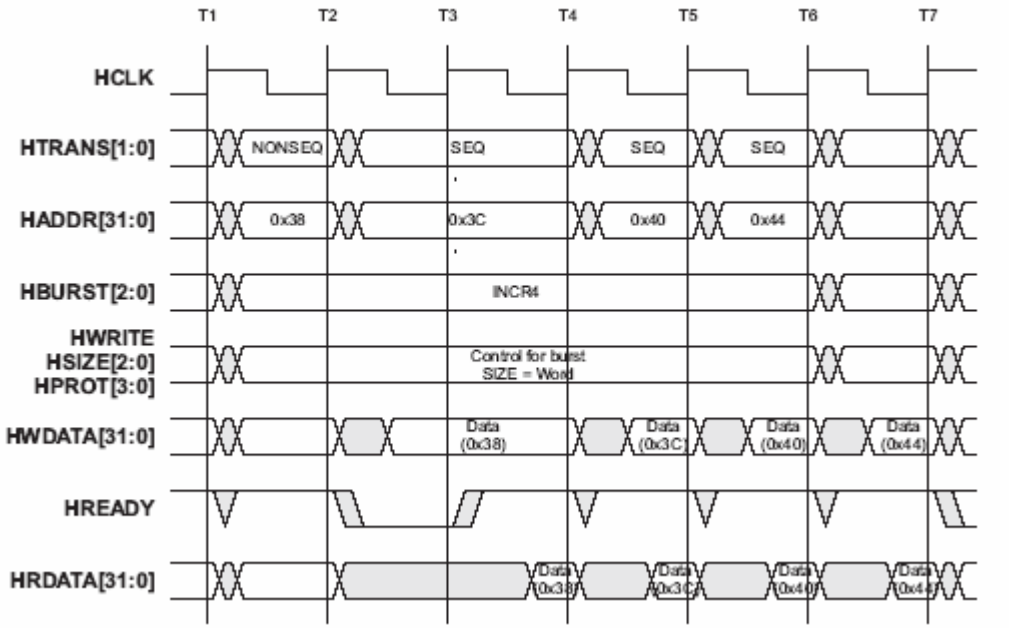
| Vardas | Šaltinis | Aprašymas |
|---|---|---|
| HCLK<br>Magistralės laikrodis | Laikrodžio šaltinis | Pagal laikrodį vykdomos visos magistralės transakcijos. Visi signalai dirba pagal priekinį HCLK frontą. |
| HRESETn<br>Numetimas | Numetimo valdiklis | Magistralės numetimo signalas yra aktyvus kai jis lygus LOW. Jis naudojamas sugrąžinit magsitralę į pradinę darbo būseną. |
| HADDR[31:0]<br>Adresų magistralė | Valdantysis įrenginys | 32 bitų sistemos adresų magistralė |
| HTRANS[1:0]<br>Perdavimo tipas | Valdantysis įrenginys | Indikuoja vykstančio duomenų perdavimo tipą, kuris gali būti NONSEQUENTIAL, SEQUENTIAL, IDLE arba BUSY |
| HWRITE<br>Perdavimo kryptis | Valdantysis įrenginys | Kai signalas lygus HIGH, jis identifikuoja, kad vyksta duomenų įrašymas. Kai lygus LOW – duomenų skaitymas |
| HSIZE[2:0]<br>Perdavimo dydis | Valdantysis įrenginys | Parodo operacijos dydį, kuris tipiniu atveju būna baitas (8 bitai), pusė žodžio (16 bitų), žodis (32 bitai). Protokolas leidžia operacijas iki 1024 bitų. |
| HBURST[2:0]<br>Monopolinės operacijos tipas | Valdantysis įrenginys | Parodo, kad esamu momentu vykdoma monopolinė operacija. Palaikoma keturių, aštuonių ar šešiolikos taktų monopolinė operacija. Monopolinė operacija gali būti *incremental* arba *wrap* tipo. |
| HWDATA[31:0]<br>Duomenų įrašymo magistralė | Valdantysis įrenginys | Rašymo magistralė naudojama duomenims iš valdančių įrenginių išsiųsti į pavaldžiuosius rašymo operacijos metu. Rekomenduojamas minimalus magistralės plotis – 32 bitai. Nepaisant to, plotį galima lengvai išplėsti leidžiant didesnio pločio reikalaujančias operacijas. |
| HSELx<br>Pavaldaus įrenginio išrinkimas | Iškodavimo įrenginys | Kiekvienas AHB pavaldusis įrenginys turi savo išrinkimo signalą ir šis signalas parodo, kad esamas duomenų perdavimas skirtas būtent tam pažymėtam pavaldžiajam įrenginiui. Šis signalas formuojamas paprasčiausiai iškoduojant informaciją iš adresų |

| | | |
|---|---|---|
| | | magistralės |
| HRDATA[31:0] Duomenų skaitymo magistralė | Pavaldus įrenginys | Skaitymo magistralė naudojama perduoti duomenims iš pavaldžiųjų įrenginių į valdančiuosius per skaitymo operaciją. Rekomenduojamas minimalus 32 bitų skaitymo magistralės plotis. Nepaisant to, plotį galima lengvai išplėsti leidžiant didesnio pločio reikalaujančias operacijas. |
| HREADY Perdavimas baigtas | Pavaldus įrenginys | HREADY HIGH signalas parodo duomenų perdavimo operacijos pabaigą. HREADY perjungiamas i LOW, kai norima išplėsti duomenų perdavimą. Pastaba: pavaldieji įrenginiai naudoja HREADY signalą tiek įvedimui, tiek išvedimui identifikuoti. |
| HRESP[1:0] Atsakas į duomenų perdavimą | Pavaldus įrenginys | Atsakas į duomenų perdavimą pateikia papildomą informaciją apie duomenų perdavimo operacijos būklę. Egzistuoja keturių rūšių reakcijos: OKAY, ERROR. |
| HBUSREQx Magistralės reikalavimas | Valdantysis įrenginys | Signalas iš magistralės valdančiojo įrenginio x į magistralės teisėją, kuris parodo, kad įrenginys reikalauja magistralės. Todėl kiekvienas valdantysis įrenginys turi savo HBUSREQx signalą. Valdančiųjų įrenginių gali būti iki 16. |
| HGRANTx Magistralės skyrimas | Teisėjas | Signalas parodo, kad šiuo metu valdantysis įrenginys x turi didžiausią prioritetą magistralės priėjimui. Adreso ir nuosavybės signalai pasikeičia, kai HREADY signalas tampa HIGH, taigi valdantysis įrenginys gauna priėjimą prie magistralės, kai abu signalai, HREADY ir HGRANTx yra HIGH. |
| HMASTER[3:0] Valdančiojo įrenginio numeris | Teisėjas | Šie teisėjo signalai parodo, kuris magistralės valdantysis įrenginys šiuo metu atlieka operaciją. Jį naudoja pavaldieji įrenginiai. HMASTER sinchronizuojamas pagal adreso ir kontrolės signalus. |

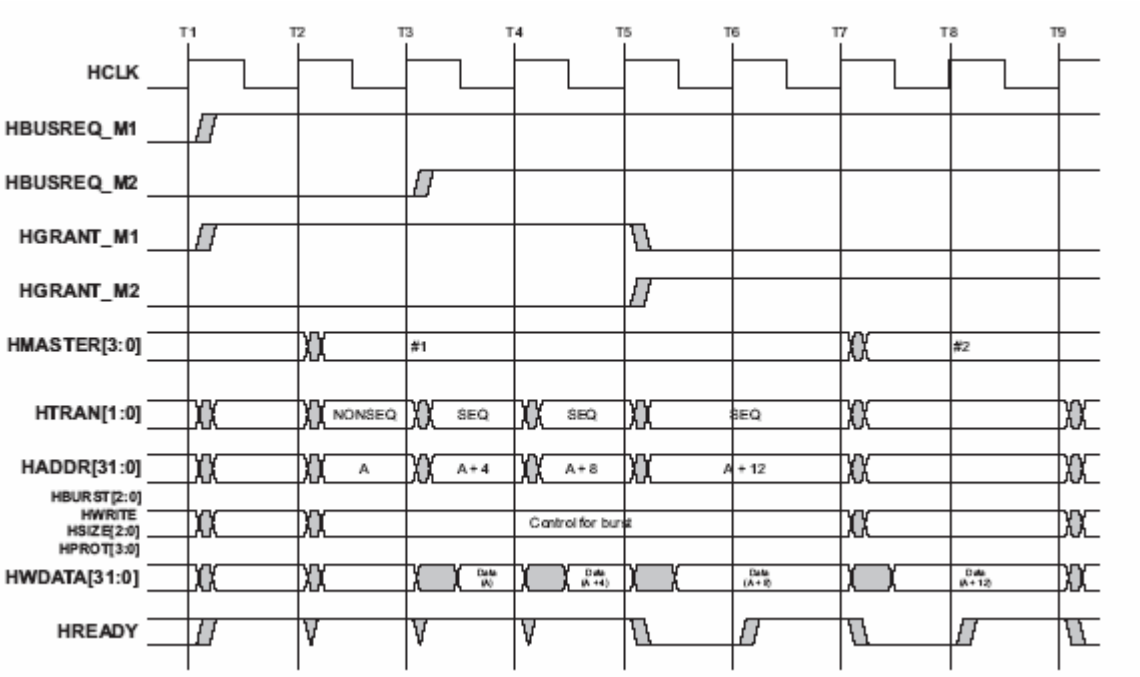
# 2 Priedas. AMBA AHB magistralės darbo pavyzdžiai




**Paprastas vieno takto duomenų perdavimas**

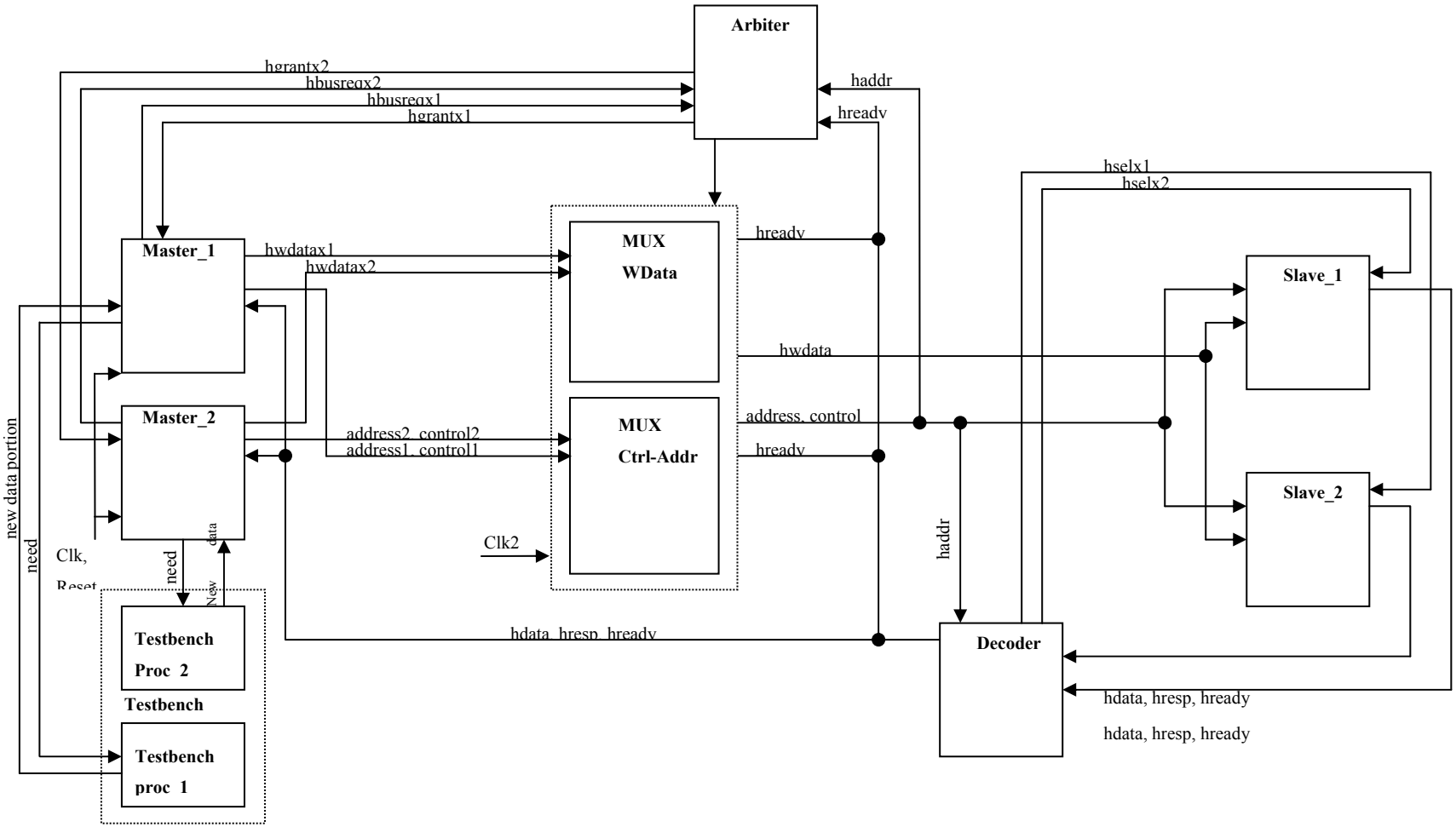



**Keturių taktų auganti monopolinė operacija**

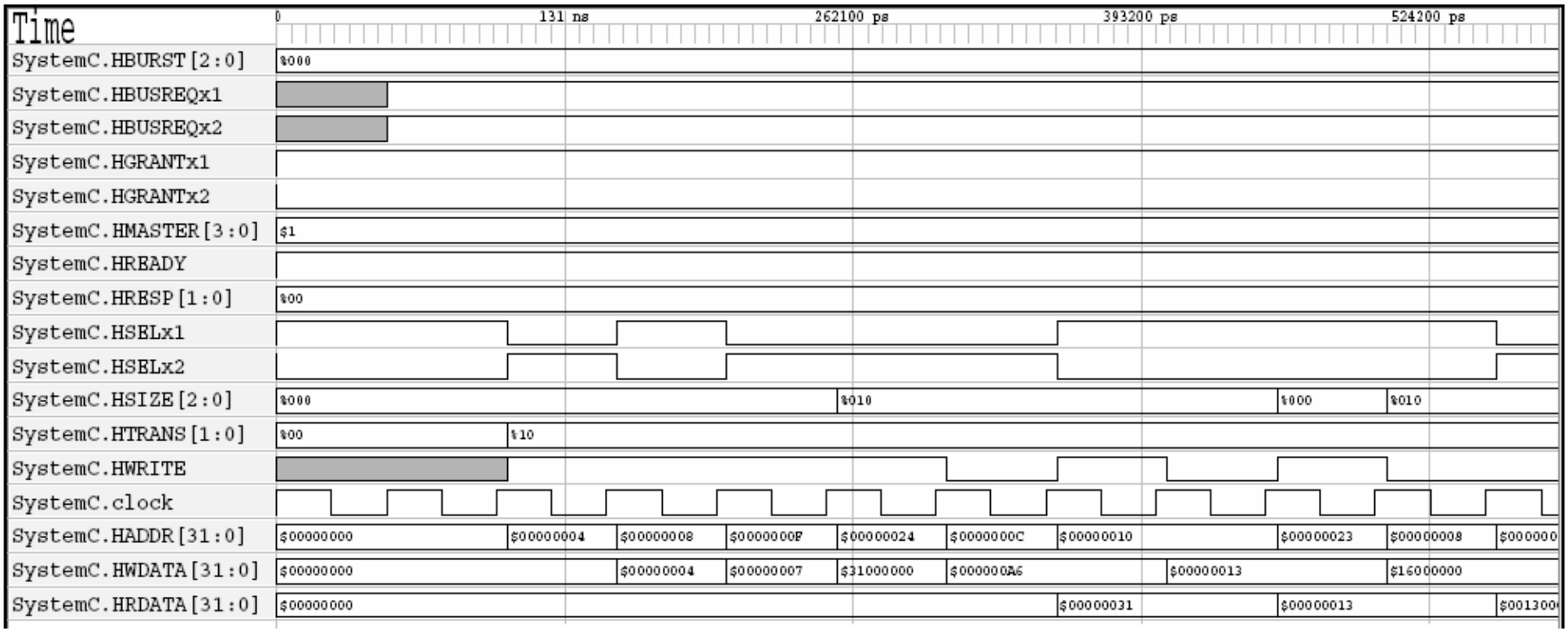



**Magistralės perjungimas po monopolinės operacijos**

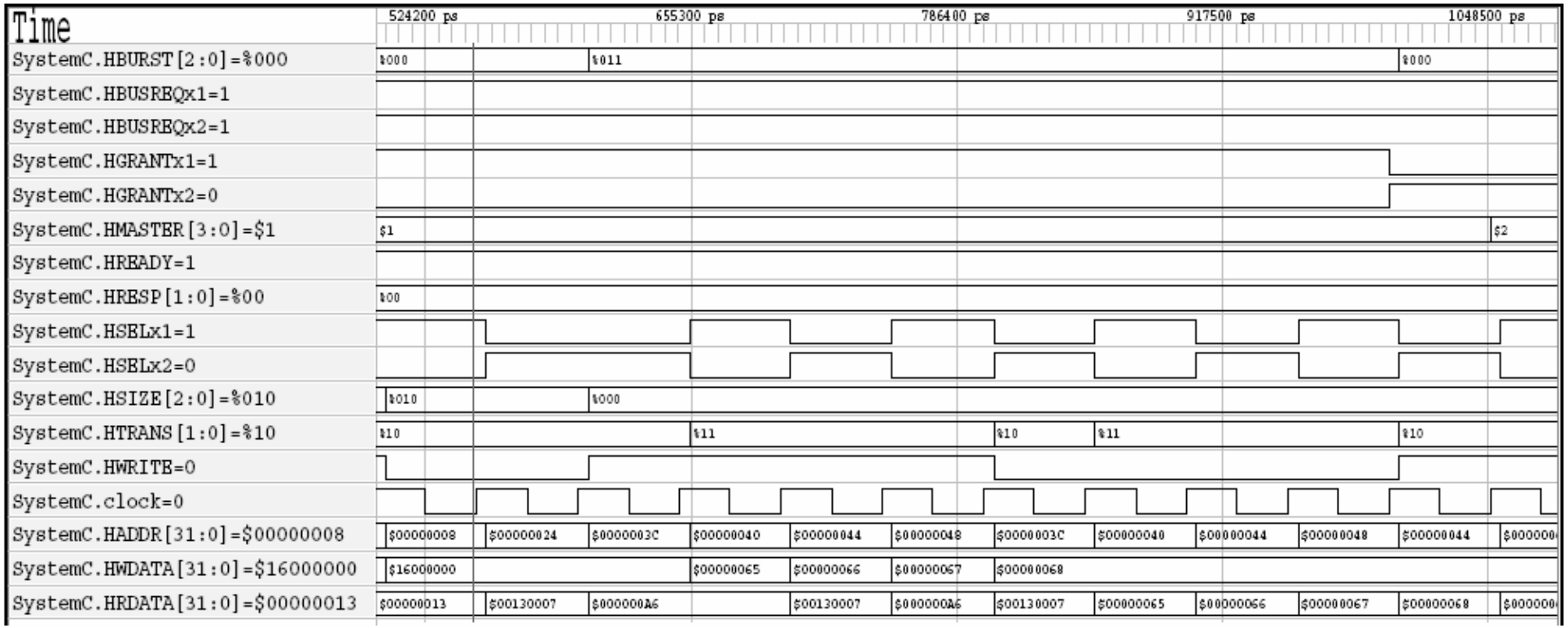
# 3 Priedas. RTL lygio magistralės blokinė schema



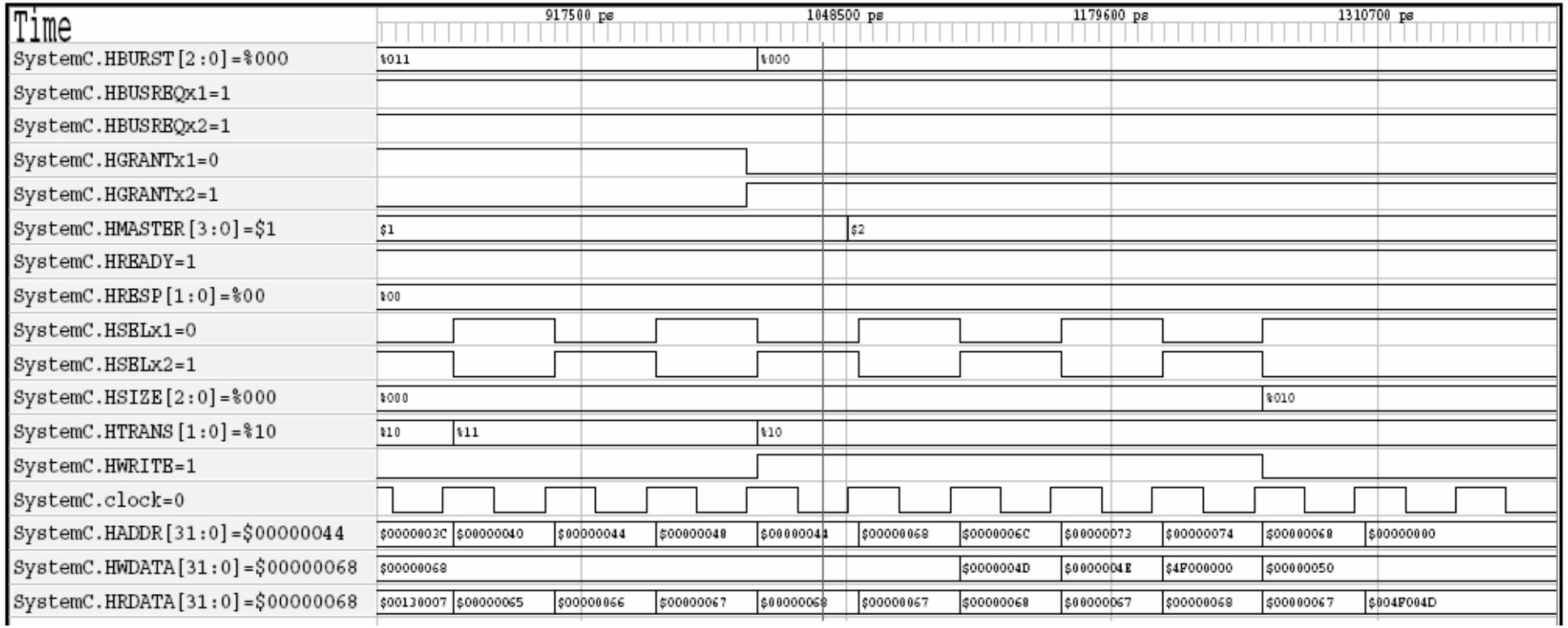
# 4 Priedas. Laikinė diagrama 1



# 5 Priedas. Laikinė diagrama 2



# 6 Priedas. Laikinė diagrama 3

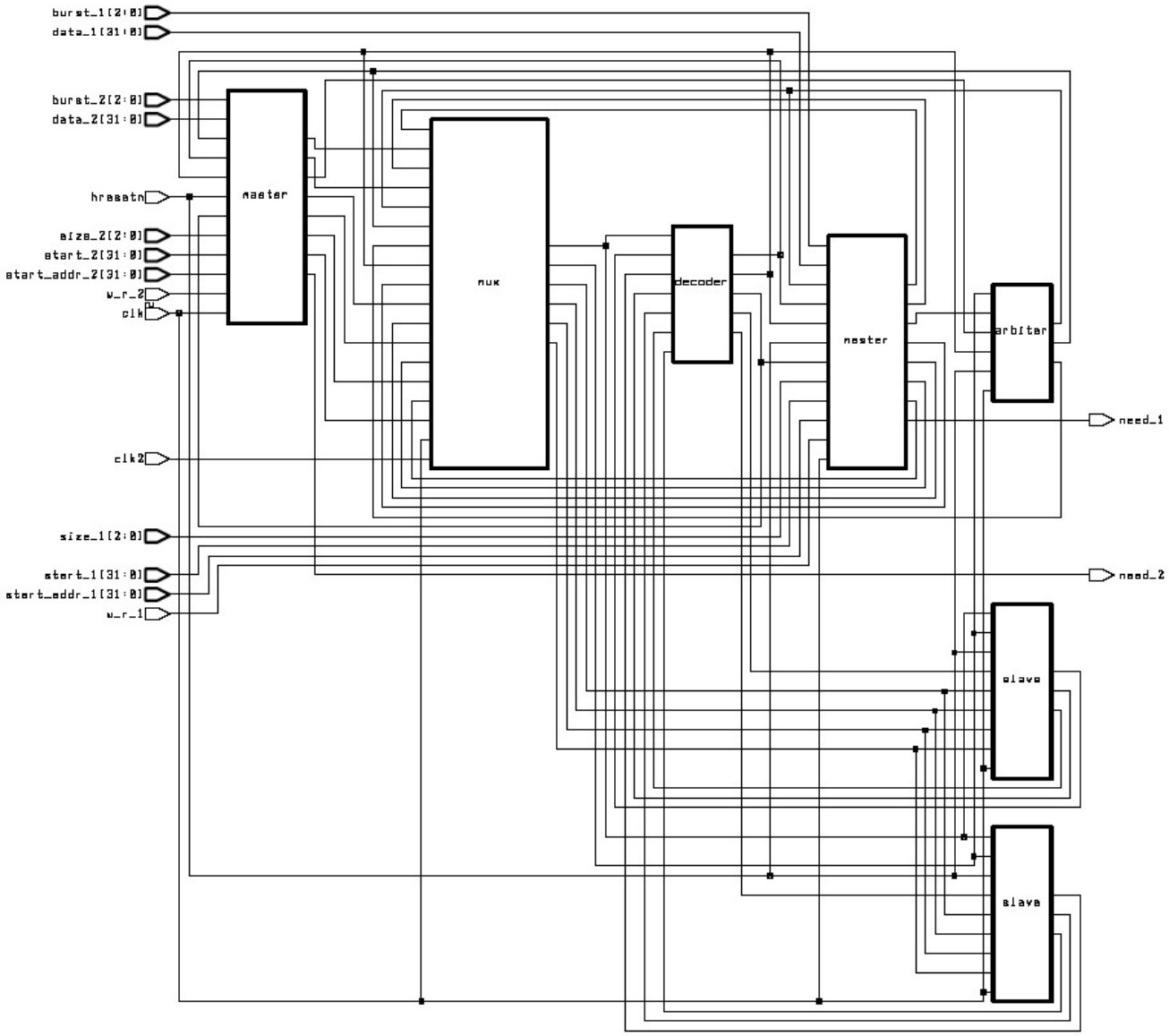


# 7 Priedas. AMBA AHB magistralės sintezavimo skriptas

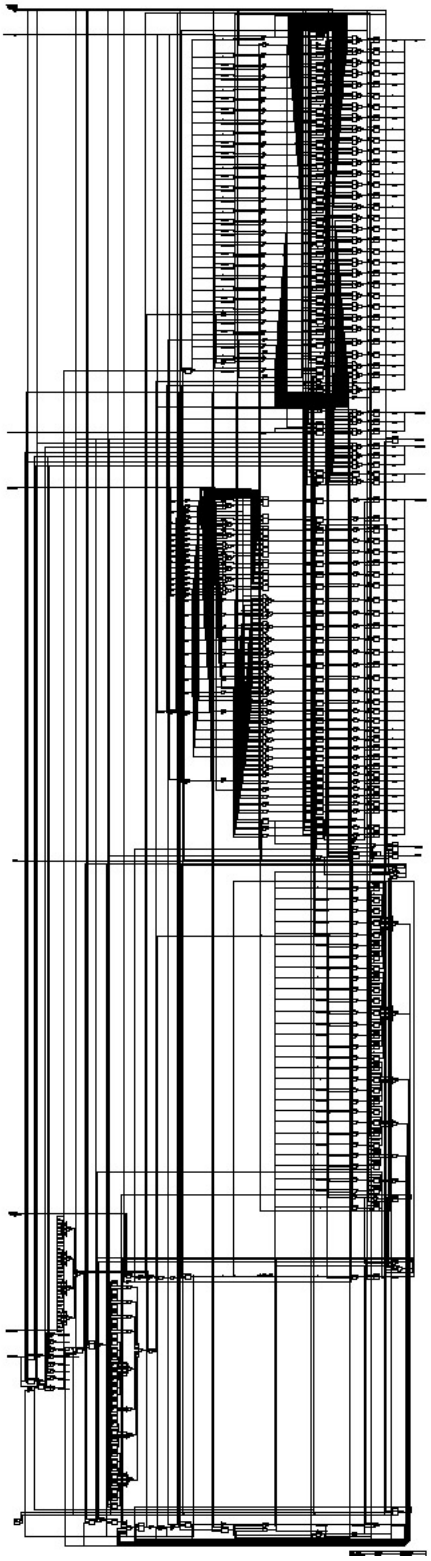
```
# Set configuration variables

set synthetic_library "dw01.sldb dw02.sldb"
set target_library "tc6a_cbacore.db"
set symbol_library "tc6a_cbacore.sdb"
set link_library "* $target_library $synthetic_library"

# Compile SystemC

#-------------decoder-----------------
compile_systemc -cpp "g++ -trigraphs -E -C -U__GNUC__ -U__GNYG__ -Wp -no-gcc -pedantic"
decoder.cc
write -output db/decoder.db -hierarchy

remove_design -all
read_file -format db db/decoder.db
check_design
check_timing
compile -map_effort medium -area_effort medium
write -hierarchy -format db -outpu db/decoder_opt.db

#-------------mux----------------
compile_systemc -cpp "g++ -trigraphs -E -C -U__GNUC__ -U__GNYG__ -Wp -no-gcc -pedantic"
mux.cc
write -output db/mux.db -hierarchy

remove_design -all
read_file -format db db/mux.db
check_design
check_timing
compile -map_effort medium -area_effort medium
write -hierarchy -format db -outpu db/mux_opt.db

#-------------arbiter----------
compile_systemc -cpp "g++ -trigraphs -E -C -U__GNUC__ -U__GNYG__ -Wp -no-gcc -pedantic"
arbiter.cc
write -output db/arbiter.db -hierarchy

remove_design -all
read_file -format db db/arbiter.db
check_design
check_timing
compile -map_effort medium -area_effort medium
write -hierarchy -format db -outpu db/arbiter_opt.db

#--------------slave-----------------
compile_systemc -cpp "g++ -trigraphs -E -C -U__GNUC__ -U__GNYG__ -Wp -no-gcc -pedantic"
slave.cc
write -output db/slave.db -hierarchy

remove_design -all
read_file -format db db/slave.db
check_design
check_timing
compile -map_effort medium -area_effort medium
write -hierarchy -format db -outpu db/slave_opt.db

#-------------master-----------------
compile_systemc -cpp "g++ -trigraphs -E -C -U__GNUC__ -U__GNYG__ -Wp -no-gcc -pedantic"
master.cc
```

```
write -output db/master.db -hierarchy

remove_design -all
read_file -format db db/master.db
check_design
check_timing
compile -map_effort medium -area_effort medium
write -hierarchy -format db -outpu db/master_opt.db

#-------------full_BUS------------------------
remove_design -all
read_file -format db db/arbiter_opt.db
read_file -format db db/decoder_opt.db
read_file -format db db/mux_opt.db
read_file -format db db/slave_opt.db
read_file -format db db/master_opt.db

compile_systemc -cpp "g++ -trigraphs -E -C -U__GNUC__ -U__GNYG__ -Wp -no-gcc -pedantic"
main.cc
check_design
check_timing
write -hierarchy -format db -outpu db/main.db

create_clock -name clk -period 40 -waveform { 0 12.5 } { clk }
create_clock -name clk2 -period 2 -waveform { 0 12.5 } { clk2 }

compile -map_effort medium -area_effort medium
write -output db/main_bus_opt.db -hierarchy

quit
```

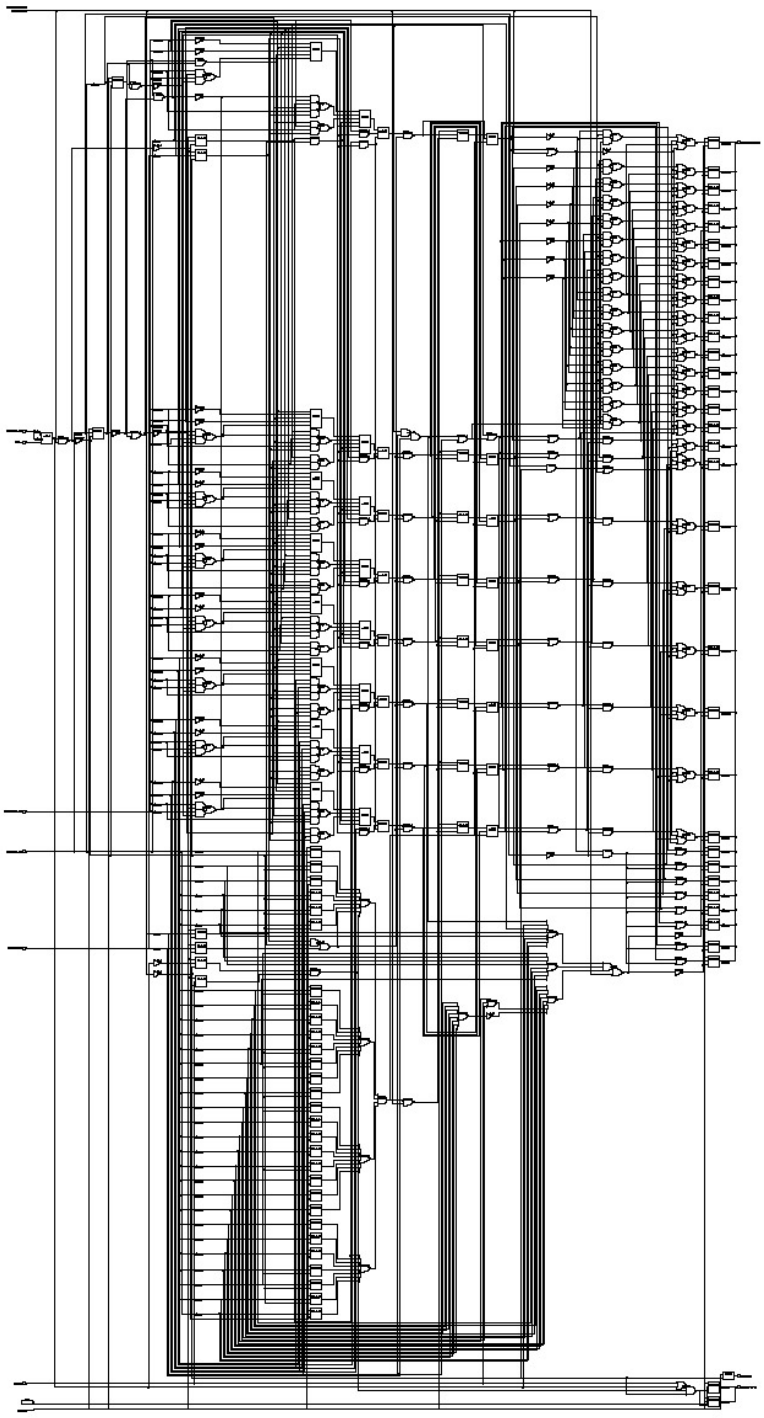
# 8 Priedas. AMBA AHB magistralės komponentė schema



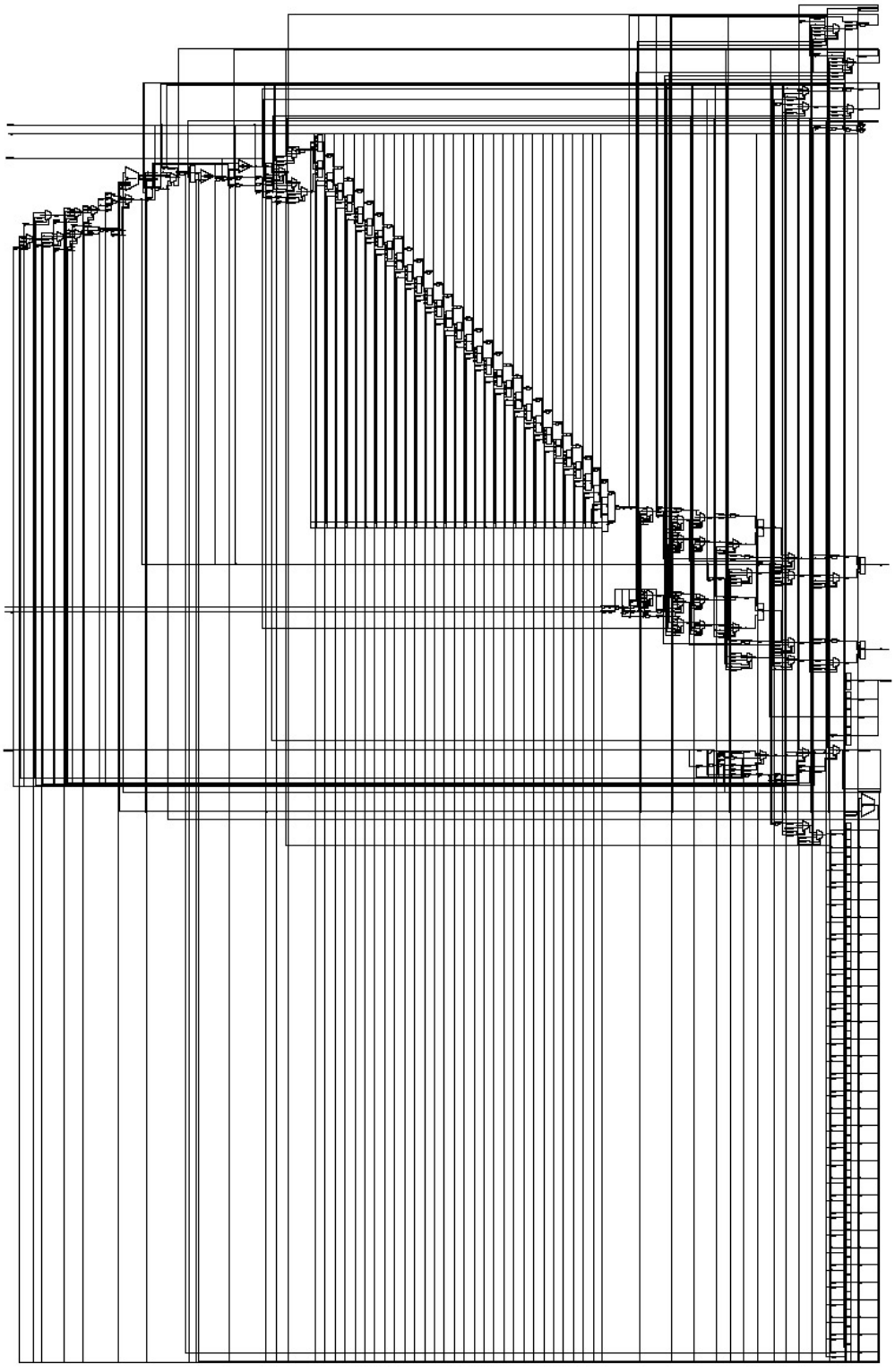
# 9 Priedas. Valdančiojo įrenginio ventilinė schema



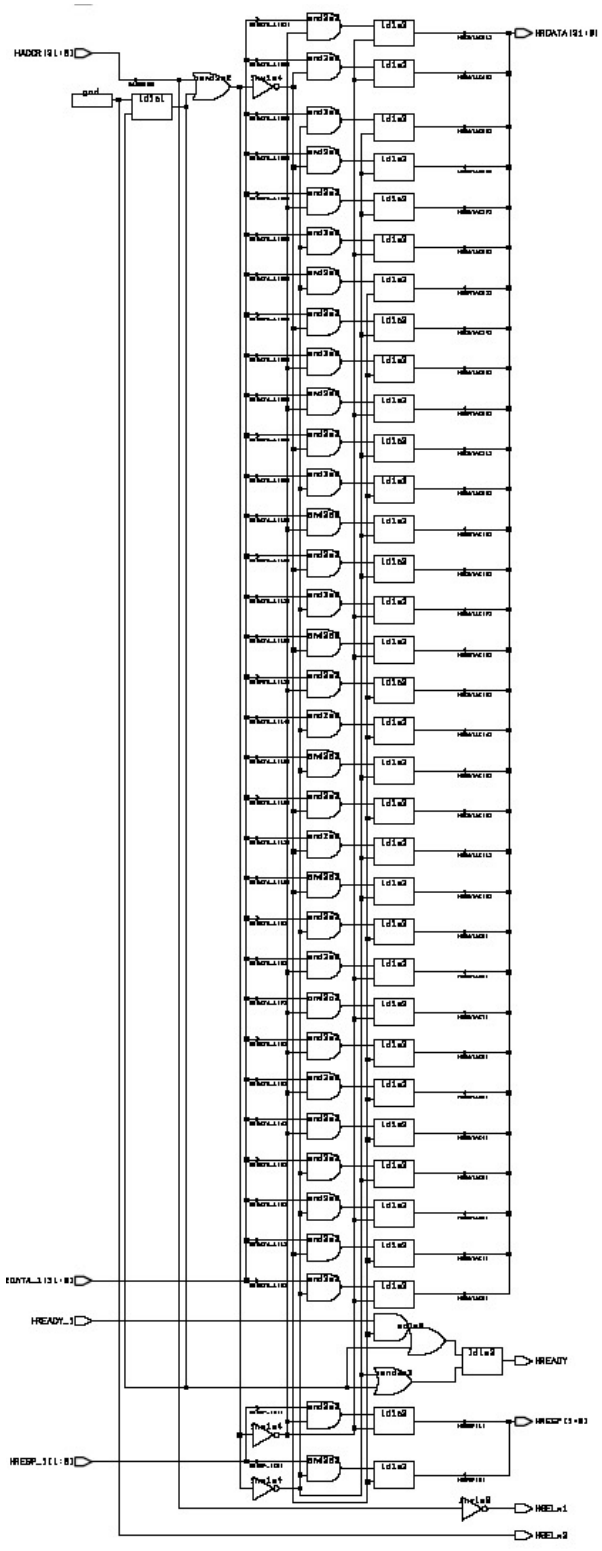
# 10 Priedas. Teisėjo įrenginio ventilinė schema



# 11 Priedas. Pavaldžiojo įrenginio ventilinė schema



# 12 Priedas. Dešifratoriaus įrenginio ventilinė schema



# 13 Priedas. Multiplekserio įrenginio ventilinė schema