VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

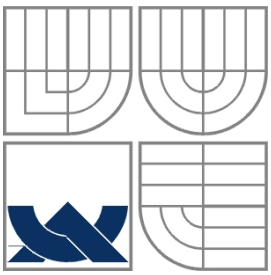
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁVRH KOMUNIKAČNÍHO PROTOKOLU PRO GENERICKÉ SIMULÁTORY MIKROPROCESORŮ

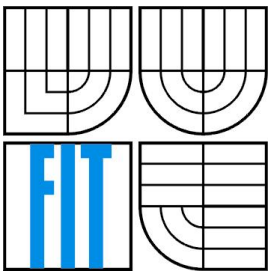
DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE BC. JIŘÍ MOSKOVČÁK
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁVRH KOMUNIKAČNÍHO PROTOKOLU PRO GENERICKÉ SIMULÁTORY MIKROPROCESORŮ

DESIGN OF COMMUNICATION PROTOCOL FOR GENERIC SIMULATORS OF MICROPROCESSORS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — BC. JIŘÍ MOSKOVČÁK
AUTHOR

VEDOUCÍ PRÁCE — ING. KAREL MASAŘÍK
SUPERVISOR

BRNO 2007

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2006/2007

Zadání diplomové práce

Řešitel: **Moskovčák Jiří, Bc.**
Obor: Informační systémy
Téma: **Návrh komunikačního protokolu pro generické simulátory mikroprocesorů**
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte s rozhraním pro předávání zpráv MPI (Message Passing Interface) a přenosovými síťovými technologiemi. Seznamte se s požadavky pro přenos dat mezi distribuovanými generickými simulátory mikroprocesorů, dále s jazykem ISAC pro popis architektury mikroprocesorů.

2. Navrhněte komunikační protokol umožňující synchronizaci distribuovaných generických simulátorů a umožňující předávání simulačních dat mezi simulátory během simulace.

3. Daný komunikační protokol implementujte.

4. Zhodnoťte svou práci z hlediska vlastností navrženého komunikačního protokolu.

Literatura:

- http://www-unix.mcs.anl.gov/mpi/mpich/
- Miroslav Popovic. Communication Protocol Engineering, CRC Press, 2006
- Holzmann, G.J. Kernighan, B.W. (ed.) Design and Validation of Computer Protocols Prentice-Hall, 1991

Při obhajobě semestrální části diplomového projektu je požadováno:

- Splnění prvních 2 bodů zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Masařík Karel, Ing.**, UIFS FIT VUT
Datum zadání: 28. února 2006
Datum odevzdání: 22. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Bc. Jiří Moskovčák**
Id studenta: 49434
Bytem: Anenská 2, 695 01 Hodonín
Narozen: 10. 04. 1983, Hodonín
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP:	Návrh komunikačního protokolu pro generické simulátory mikroprocesorů
Vedoucí/školitel VŠKP:	Masařík Karel, Ing.
Ústav:	Ústav informačních systémů
Datum obhajoby VŠKP:	.....

VŠKP odevzdal autor nabyvateli v:

tištěné formě	počet exemplářů: 1
elektronické formě	počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

Tato práce se zabývá návrhem komunikačního protokolu pro generické simulátory procesorů. Cílem práce bylo navrhnout a implementovat komunikační protokol, který jednak umožní simulaci víceprocesorového systému a zároveň běh simulace na svazku počítačů.

Klíčová slova

simulátor, SoC, ISAC, LISSOM, komunikační protokol, simulace, modelování procesorů, system on chip

Abstract

This work concerns about designing of communication protocol for generic processor simulator. The main objective of this work was to design a communication protocol which allows to simulate multiprocessor system on a cluster of computers.

Keywords

simulator,SoC, ISAC, LISSOM, communication protocol, simulation, processor modelling, system on chip

Citace

Jiří Moskovčák: Návrh komunikačního protokolu pro generické simulátory mikroprocesorů, diplomová práce, Brno, FIT VUT v Brně, 2007

Návrh komunikačního protokolu pro generické simulátory mikroprocesorů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Karla Masaříka. Další informace mi poskytl Bc. Zdenek Přikryl. Diplomová práce byla vypracována na základě veřejně přístupného kódu programu TinyXml a Eclipse. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jiří Moskovčák
20. května 2007

Poděkování

Na tomto místě bych rád poděkoval Ing. Karlu Masaříkovi za jeho cenné rady a trpělivost při vedení.

Obsah

1 Úvod

V dnešní době, kdy se různé vestavěné(embedded) systémy staly běžnou součástí života většiny lidí, je výroba takových systémů pro firmy na celém světě velmi lukrativní záležitost. Samozřejmě konkurence v této oblasti je velmi vysoká, a proto rychlost a cena návrhu nového systému je při vývoji kritická.

V praxi se nejčastěji používají nástroje, které mají za úkol zkrátit dobu činností, kterým se nelze při návrhu a testovaní vyhnout – jde hlavně o testování funkčnosti vytvořené architektury a instrukční sady bez jejich hardwarového prototypu. Dříve bylo obvyklé, že pro každý nový procesor byl implementován speciální model, na kterém se testovaly vlastnosti výrobku. Většina výrobců si vytváří vlastní knihovny a nástroje, které takovou implementaci urychlují. Tento přístup může být výhodný v případě, že se nepředpokládají větší úpravy návrhu, ale pokud k takové úpravě dojde, je obvykle nutné vše přepracovat, což povede ke zpoždění a zvýšení ceny.

Z toho důvodu v poslední době vznikají integrovaná prostředí zahrnující prostředky pro rychlý návrh zdrojů a instrukční sady procesoru a testování jejich funkčnosti a efektivnosti ještě na úrovni, kdy neexistuje žádný fyzický prototyp. K tomu se používají speciálně navržené programovací (modelovací) jazyky popisující jak zdroje, které architektura využívá, tak instrukční sadu pracující nad definovanými zdroji. Na základě takto vytvořeného modelu mohou být automaticky generovány nástroje pro manipulaci s ním. S modelem a s generovanými nástroji pak pracuje simulátor, který napodobuje činnost modelovaného systému [2].

Toto čistě softwarové řešení je levné a rychlé z hlediska možnosti změn architektury a opravy chyb, avšak limitujícím faktorem je samotná rychlost simulace. I když se v posledních letech rychlost a schopnosti počítačů velmi zlepšily, je stále simulace komplexnějších (víceprocesorových) systémů značně časově náročná.

Cílem této práce je navrhnout a implementovat řešení, které by umožnilo běh simulace paralelně na několika počítačích. To samozřejmě znamená vyřešit celou řadu problémů, které s paralelním zpracováním souvisí. Tyto problémy jsou postupně popsány v jednotlivých a kapitolách a současně je předkládáno jejich řešení.

Kapitola 2 obsahuje obecný úvod do problematiky tvorby a využití modelů k simulaci reálných systémů. Popisuje různé metody analýzy modelovaných systémů a zároveň poskytuje přehled základních funkcí projektu Lissom jakožto modelovacího nástroje. Kapitola 3 se blíže zabývá projektem Lissom. Záměrem této kapitoly je seznámit čtenáře s celým projektem a objasnit cíle této práce. V kapitole 4 je popsán návrh možných způsobů synchronizace simulátorů během simulace. Obsahuje popis různých metod komunikace mezi procesy a důvody, které vedly ke zvolení implementovaného řešení. Kapitola 5 se zabývá strukturou zpráv používaných pro komunikaci mezi simulátory a kapitola 6 popisuje funkce, které poskytuje navrhovaný protokol. V kapitole 7 se řeší návrh mechanismu pro sdílení dat během simulace. 8. kapitola popisuje vlastní implementaci a následující kapitola se zabývá testováním a hodnocením vlastností implementovaného řešení. V závěrečné kapitole je shrnutí výsledků a přínosu této práce pro projekt Lissom a jsou zde nastíněny možnosti vývoje projektu navazující na tuto práci.

2 Modelování a simulace

Pod pojmem modelování rozumíme cílevědomou činnost, která slouží k získávání informací o jednom systému prostřednictvím jiného systému — modelu. Systémem je definován jako množina prvků a jejich vazeb účelově definovaných pro vykonávání určitých funkcí. Při modelování pak využíváme toho, že model je také systém a cílem modelování je nalezení takového systému (modelu), který se modelovanému systému podobá. Význam modelování spočívá v tom, že umožňuje ekonomické studium systémů. Je výhodnější, rychlejší a často jedině možné získávat informace o systémech experimentováním na jejich modelech, než na originálech. V tomto smyslu náleží do modelování výstavba všech modelů, fyzikálních i matematických, statických i dynamických.

Je-li modelovaný systém jednoduchý, nebo můžeme-li formulovat tak zjednodušující předpoklady, aby byl model řešitelný analyticky, popíšeme chování systému matematickými vztahy a hledané veličiny stanovujeme matematickými prostředky. Výsledky získáváme ve formě funkčních vztahů, ve kterých se jako proměnné vyskytují parametry modelu. Řešení konkrétního modelu získáme dosazením konkrétních hodnot do funkčních vztahů. Výsledné hodnoty jsou tedy funkcí jednoho, či více obecných parametrů. Hlavní předností analytického řešení je menší časová náročnost řešení matematického modelu. Jde však o modely jednoduché nebo podstatně zjednodušené.

V současné době je však stále aktuálnějším problémem analýza složitých systémů. Hlubší analýzu těchto systémů umožnil teprve rozvoj počítačů, které nabízí velmi perspektivní prostředky pro zkoumání složitých systémů na základě jejich modelů.

Metody modelování systémů na počítačích využíváme zvláště v těchto případech:

- neexistuje-li úplná matematická formulace problému nebo nejsou-li známé analytické metody řešení matematického modelu
- vyžadují-li analytické metody tak zjednodušující předpoklady, že je nelze pro daný model přijmout
- jsou-li analytické metody dostupné pouze teoreticky a praktické řešení by bylo tak obtížné a dlouhé, že je metoda modelování problému na počítači jednodušší cestou jeho řešení

- je-li žádoucí modelovat historii procesu v určitém časovém intervalu za účelem odhadu některých parametrů
- je-li modelování na počítači jedinou možností získání výsledků v důsledku obtížnosti provádění experimentů ve skutečném prostředí
- potřebujeme-li pro pozorování systému měnit časové měřítko – modelování systému na počítači umožňuje urychlování nebo zpomalování příslušných dějů

Proces modelování systémů na počítačích můžeme velmi zjednodušeně rozdělit do tří základních etap:

1. Formování účelového a zjednodušeného popisu zkoumaného systému – vytvoření abstraktního modelu.
2. Zápis abstraktního modelu formou programu – vytvoření simulačního modelu.
3. Experimentování s reprezentací simulačního modelu na počítači – simulace.

2.1 Vytvoření abstraktního modelu

V případě, že vytváříme návrh systému, který se bude teprve realizovat a nemá tedy skutečnou předlohu, vycházíme ze znalostí obdobných systémů, které slouží jako základ navrhovaného systému. Budováním abstraktního modelu rozumíme formulaci zjednodušeného popisu systému abstrahujícího od všech nedůležitých detailů vzhledem k účelu modelu. Jádrem tohoto procesu je identifikace jeho vhodných složek, které mají vliv na efektivnost, či neefektivnost systému a dále rozhodnutí, zda tyto složky budou součástí systému nebo jeho okolí. Různé specifické cíle a účely modelů můžeme zahrnout do několika základních tříd[1]:

a) *Vyhodnocení* - určení, jak je navržený systém vhodný v absolutním smyslu; jeho chování studujeme pro určitá specifická kritéria.
b) *Srovnávání* - porovnávání funkcí systémů vzhledem k jejich určitým alternativním složkám nebo operačním strategiím.

c) *Predikce* - vyhodnocení chování systému za určitých, v reálném systému potenciálních, podmínek.

d) *Analýza citlivosti* - určení těch faktorů (parametrů), jež jsou pro činnost celého systému nejvýznamnější.

e) *Optimalizace* - nalezení takové kombinace parametrů, která vede k nejlepší odezvě systému.

f) *Funkcionální vztahy* - objevení povahy závislostí mezi nejvýznamnějšími parametry a odezvou systému.

Tento výčet účelů a cílů modelu není jistě vyčerpávající a mohou existovat i jiné důvody, pro které se rozhodneme metody modelování na počítačích využít. Explicitní vymezení účelu modelu má však významný dopad na celý proces budování abstraktního modelu i na vlastní experimentování se simulačním modelem. Je-li například cílem modelu absolutní vyhodnocení navrženého nebo existujícího systému, pak musí být model velmi přesný. Je-li však cílem modelu relativní srovnávání dvou nebo více systémů, pak může být přesnost modelu pouze relativní.

Z toho vyplývá, že pro účely návrhu a simulace procesorů je nezbytné, aby model procesoru velmi přesně popisoval modelovaný procesor. Mezi mezi modelem a modelovaným procesorem tedy musí být homomorfní vztah, který vyžaduje korespondenci prvků abstraktního modelu s prvky modelovaného procesoru a korespondenci jejich struktur.

2.2 Vytvoření simulačního modelu

Pod pojmem simulační model rozumíme abstraktní model zapsaný formou programu v programovacím jazyce. Na rozdíl od dvojice *modelovaný systém - abstraktní systém*, kde předpokládáme homomorfní vztah, vyžadujeme mezi dvojicí *abstraktní systém - simulační model* vztah izomorfní, jenž představuje silnější vztah ekvivalence mezi abstraktními systémy - shodnost struktur a chování prvků uvažovaných systémů.

Pro popis abstraktního modelu a jeho následný převod na model simulační se používá různých jazyků a metod. Například projekt Lissom, jehož je tato práce součástí, používá pro popis abstraktního modelu procesoru jazyk ISAC a XML, a pro zápis simulačního modelu C++.

2.3 Simulace

Simulací označujeme etapu experimentování s reprezentací simulačního modelu. Jejím cílem je analýza chování systému v závislosti na vstupních veličinách a na hodnotách parametrů. Vlastní etapě simulace předchází verifikace simulačního modelu, kdy ověřujeme korespondenci simulačního a abstraktního modelu, zpravidla tedy izomorfní vztah mezi těmito dvěma modely. Analogicky s programy v běžných programovacích jazycích představuje verifikace simulačního modelu jeho ladění jak po stránce syntaktické, tak, a to zvláště, po stránce sémantické.

Proces simulace spočívá v opakovaném řešení modelu, v provádění simulačních běhů, které jsou charakterizovány určitými hodnotami parametrů modelu a určitými podněty z okolí. S každým simulačním během je spojeno vyhodnocení výstupních dat simulačního modelu, které představuje informaci o chování systému, tj. o jeho reakcích na podněty z okolí. Simulační běhy, jako základní jednotky simulace, opakujeme tak dlouho, dokud nezískáme dostatečnou informaci o chování systému nebo pokud nenalezneme takové hodnoty parametrů, pro něž má systém žádané chování. Důležitou složkou simulace je neustálá konfrontace informací, které o modelovaném systému máme a které simulací získáváme. Tato konfrontace nám pomáhá rozhodnout jeden z nejobtížnějších problémů modelování – problém validity (platnosti) modelu. Ověřování validity modelu je proces, v němž se snažíme dokázat, že skutečně pracujeme s modelem adekvátním modelovanému systému. V případě, že chování modelu neodpovídá předpokládanému chování originálu, musíme model modifikovat s přihlédnutím k informacím, které jsme získali předcházející simulací.

Pro efektivní realizaci těchto etap modelování potřebujeme jednak prostředky pro práci s abstraktními systémy a prostředky pro tvorbu simulačních modelů a experimentování se simulačními modely. A právě jedním z takovým prostředků je i projekt Lissom.[1]

1 Teorie modelování a simulací převzata z [1]

3 Projekt Lissom

Projekt Lissom se zabývá tvorbou univerzálního nástroje pro vývoj procesorových architektur založených na instrukčních sadách. Jeho cílem je poskytnout kompletní sadu nástrojů pro návrh těchto architektur spolu s nástroji potřebnými pro tvorbu softwaru pro vyvíjenou architekturu. Jako hlavní implementační prostředí byl zvolen jazyk C/C++, protože umožňuje efektivní multiplatformnost[4].

3.1 Stav projektu

V době, kdy vznikala tato práce bylo možné modelovat jednoprocesorové systémy. Cílem této práce je rozšířit stávající možnosti o simulaci víceprocesorových systémů tzv. System on chip (SoC).

3.1.1 Architektura projektu

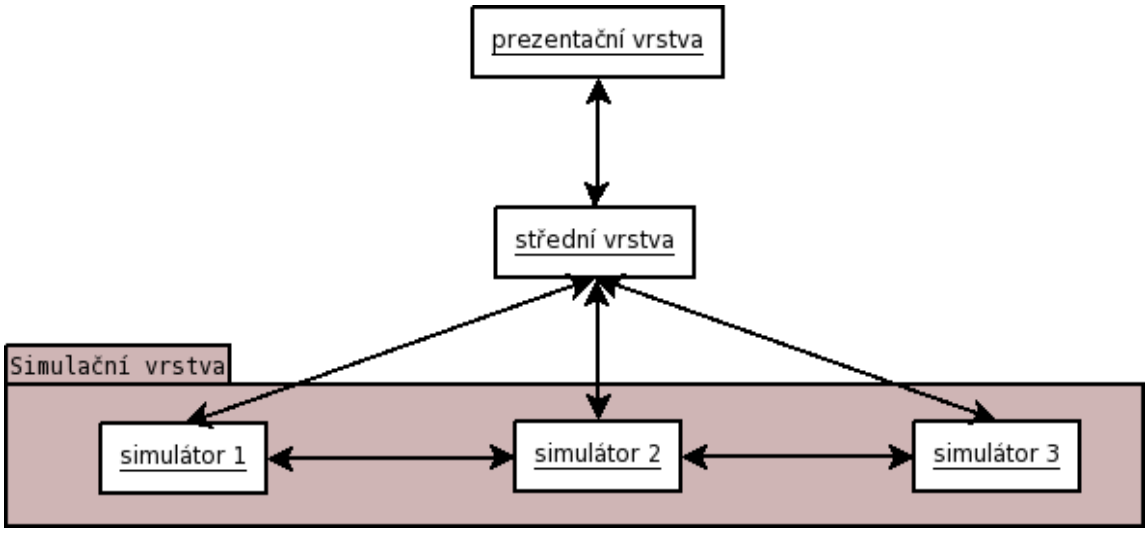
Celé vývojové prostředí vyvíjené v rámci projektu Lissom je koncipováno jako třívrstvá architektura. První vrstvou, která by se dala nazvat „tenký klient“ je prezentační vrstva. Prezentační vrstva je uživatelské rozhraní, které pouze zprostředkovává interakci s uživatelem. Poskytuje ovládací prvky, kterými může uživatel ovlivňovat simulaci a zároveň zobrazuje různé statistiky týkající se simulace.

Další je střední vrstva, která zpracovává požadavky od prezentační vrstvy a zajišťuje komunikaci mezi simulátory a prezentační vrstvou. Požadavky od prezenční vrstvy zpracovává na základě definic z konfiguračního souboru. Definice akcí jsou strukturovány ve formátu XML a je tedy velmi snadné přidávat akce nové. Zprostředkováním komunikace se simulátory je myšleno především ovládaní simulátorů a přenos nejrůznějších statistik, které jsou potřeba pro ladění simulovaného programu.

Poslední – simulační vrstva je tvořena samotnými simulátory. Počet simulátorů, které se podílejí na simulaci je teoreticky neomezený. Simulátory musí mít možnost komunikovat jak mezi sebou, tak se střední vrstvou.

3.1.2 Cíle práce

Jak bylo uvedeno v předchozím textu, je cílem této práce rozšířit stávající model projektu o možnost víceprocesorové simulace. Z pohledu zmiňované třívrstvé architektury to znamená navrhnout a implementovat podporu pro komunikaci jednotlivých simulátorů v simulační vrstvě – konkrétně podporu pro sdílení dat během simulace a modifikovat stávající implementaci střední vrstvy tak, aby byla schopna obsluhovat více simulátorů.

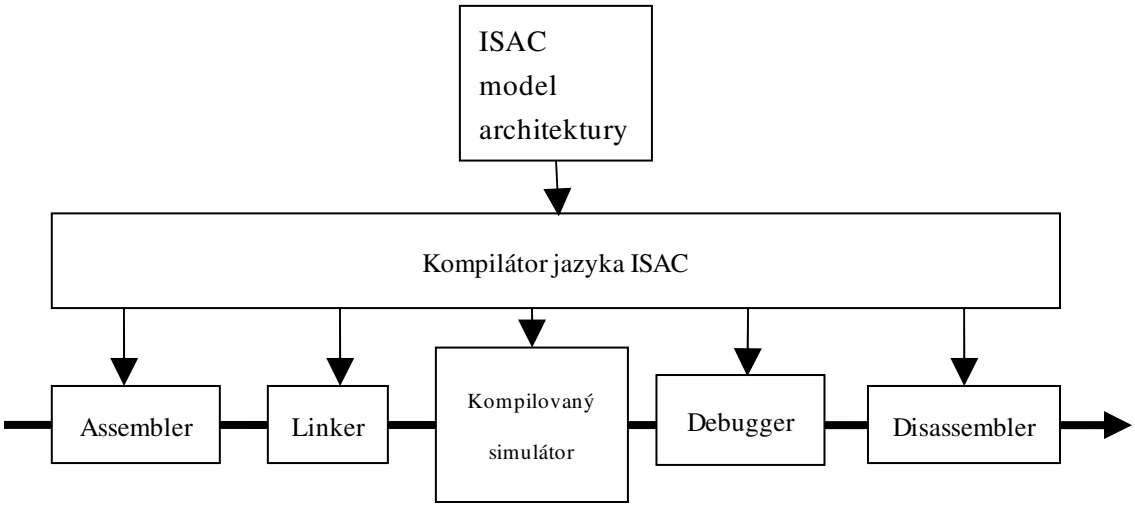


Ilustrace 1: Architektura projektu

3.2 Jazyk ISAC

Základním prvkem, o který se opírají ostatní nástroje, je jazyk pro popis zdrojů a instrukční sady modelovaného procesoru s názvem ISAC. Překladač tohoto jazyka generuje XML dokument popisující základní architekturu procesoru. Jde především o popis instrukční sady, zdrojů a chování procesoru.

Podle tohoto XML popisu se následně generují jednotlivé komponenty pro podporu vývoje. V současnosti se jedná o assembler, linker, disassembler, debugger a simulátor. Program který se má na této architektuře testovat prochází jednotlivými vývojovými stupni jako při běžném programování pro fyzický procesor. Schéma tohoto mechanizmu je patrné z následujícího obrázku[4]:



Ilustrace 2: Vývoj softwaru s pomocí nástroje pro návrh architektury

3.3 Víceprocesorový model SoC v jazyce ISAC

Jazyk ISAC v současné verzi neposkytuje konstrukce pro popis víceprocesorového modelu SoC. Je tedy nutné vytvořit konstrukce, které umožňují mikroprocesoru pracovat se zdroji jiného mikroprocesoru. Pokud nechceme definovat vlastní strukturální propojení mikroprocesorů, ale pouze pracovat s externími zdroji jiného mikroprocesoru, vytvoříme pro tento účel pro sekce chování (BEHAVIOR) operací mechanismus, který nám umožní číst a zapisovat do zdrojů externího mikroprocesoru.

Navrhovaná konvence pro pro práci s externími zdroji bude vypadat následovně.

„`název mikroprocesoru.název zdroje`“, kde název mikroprocesoru odpovídá názvu ISAC souboru s popisem mikroprocesoru.

Při generování simulátoru se do simulátoru, který pracuje s externím zdrojem, vygeneruje obraz těchto zdrojů (vytvoří se v simulátoru kopie), a stavy skutečného zdroje a obrazu se aktualizují (synchronizují) pouze pokud je potřeba.

příklad práce s externím zdrojem zapsané pomocí navrhované konvence:

```
BEHAVIOR{
      write(dsp.ax,data)
};
```

- instrukce s tímto kódem zapíše do *ax* procesoru *dsp* hodnotu proměnné *data*

4 Návrh vlastní synchronizace simulačních vrstev

Existuje mnoho různých způsobů komunikace (synchronizace) mezi procesy. Způsob komunikace se může lišit jde-li o komunikaci při paralelních výpočtech, nebo jen o získávání dat ze vzdálených zdrojů (webové stránky). Protokol navrhovaný v této práci má za úkol zajišťovat jak výměnu (sdílení) dat při simulaci, tak volání funkcí, či jen získávání informací o průběhu simulace.

Protože nechceme uživatele omezovat určitou platformou a zároveň chceme udržet jednoduchou přizpůsobivost protokolu různým komunikačním médiím, měl by být navrhovaný protokol na těchto parametrech nezávislý. Tyto vlastnosti splňuje protokol, který používá ke komunikaci mezi jednotlivými účastníky techniku „zasílání zpráv“.

4.1 Komunikace zasíláním zpráv

Komunikace zasíláním zpráv (tzv. message passing) je velmi rozšířenou formou komunikace, která se hojně používá v paralelním programování, objektově orientovaném programování a obecně při meziprocesové komunikaci. Komunikace je tvořena zasíláním zpráv příjemcům (procesům, objektům,...), kteří podle obsahu přijaté zprávy provedou příslušnou akci. Obsahem zpráv bývá obvykle volání nějaké funkce, signalizování změny stavu nebo prostá data.

Komunikace zasíláním zpráv je definovaná jako asynchronní zasíláni dat k příjemci (většinou se posílá přímo hodnota, ale existují i výjimky, kde se používají odkazy). Tento systém komunikace se používá například u volání webových služeb přes SOAP. Komunikace zasíláním zpráv je podobná komunikaci přes datagramy s tím rozdílem, že zprávy mohou být delší a komunikace spolehlivá nebo zabezpečená[5].

4.1.1 Message Passing Interface

Speciální programovací jazyky pro „message passing“ neměly valný úspěch vzhledem k nechuti programátorů učit se nový jazyk. Syntaxe sekvenčního programovacího jazyka a vyhrazená slova se dají ale rozšířit pro zasílání zpráv. Velmi rozšířený standard pro zasílání zpráv je Message Passing Interface (MPI)[3].

MPI je jazykově nezávislé API pro předávání zpráv mezi aplikacemi při paralelních výpočtech. MPI má definovanou pouze sémantiku, ale není vázáno na žádný konkrétní komunikační protokol, který by musel být pro implementaci použit. Nezávislost MPI na protokolu, nad kterým je implementováno je velkou výhodou, protože je možné implementovat MPI jak nad klasickými sockety s TCP/IP tak nad sdílenou pamětí, případně nad jakýmkoliv jiným médiem, které je vhodné pro konkrétní problém.

Nejvýznačnějšími vlastnostmi MPI jsou výkonnost, rozšiřitelnost a přenositelnost. Bohužel mezi tyto vlastnosti nepatří pohodlné programátorské rozhraní, které by poskytovalo jakoukoliv abstrakci, ale veškerý paralelismus je nutné vyjádřit explicitně – proto je také MPI považováno za tzv. low-level protokol[5].

4.1.2 Implementace MPI

Vzhledem k tomu, ze MPI je pouze specifikace knihovních funkcí, existuje mnoho různých implementací, které sice poskytují stejná rozhraní a funkcionalitu, ale také se od sebe v mnoha ohledech liší.

Pokud bychom se rozhodli použít pro implementaci navrhovaného protokolu některou z existujících implementací MPI, musela by kromě výše zmíněných vlastností mít také vhodnou licenční politiku. Těmto požadavkům vyhověly knihovny Open MPI a MPICH2.

4.1.2.1 Open MPI

Open MPI [6] je projekt, který v sobě spojuje technologie z několika jiných projektů (FT-MPI, LA-MPI, LAM/MPI, a PACX-MPI) a klade si za cíl vytvořit implementaci, která poskytuje plnou kompatibilitu s nejnovějším standardem MPI-2 a navíc má mnoho dalších vlastností, ze kterých asi nejzajímavější jsou :

- multiplatformnost (různé OS, 64/32bit)
- odolnost vůči chybám v síti
- funkčnost na heterogenních sítích

4.1.2.2 MPICH2

Stejně jako LAM/MPI, tak i projekt MPICH2 [7] poskytuje knihovnu funkcí podle standardu MPI a lze jej provozovat na různých platformách (testován na OS Linux, Windows, Solaris, Mac OS/X). MPICH2 obsahuje také podporu MPI pro clustery i symetrické multiprocesorové systémy.

4.1.2.3 Vlastní řešení

Přesto, že obě zmíněné implementace nabízí velký potenciál, tak bohužel nesplňují jednu, zatím sice nezmíněnou, ale přesto velmi důležitou vlastnost - snadnou instalaci na straně uživatelů. Obě tyto implementace vyžadují instalaci různých skriptů včetně zdlouhavé konfigurace hostitelského systému a to na každé ze stanic, na které by měla běžet některá z vrstev. Navíc projekt Lissom v aktuální a pravděpodobně ani v žádné další verzi převážnou část funkcí z těchto knihoven nevyužije, a proto vychází jako nejvhodnější varianta vytvoření vlastní knihovny. Tato knihovna bude obsahovat jen základní množinu funkcí potřebných pro implementaci protokolu, který se bude používat při synchronizaci simulátorů.

4.2 Synchronizační protokol

Synchronizační protokol slouží k udržovaní koherence jednotlivých sdílených zdrojů. Protokol musí zajišťovat, že simulátor při čtení sdíleného zdroje dostane vždy aktuální hodnotu a při zápisu do sdíleného zdroje musí zajistit, aby kopie tohoto zdroje, které mají ostatní simulátory, byly označeny jako neaktuální, případně musí zajistit přímo distribuci nové hodnoty.

4.3 Dvoustavová synchronizace

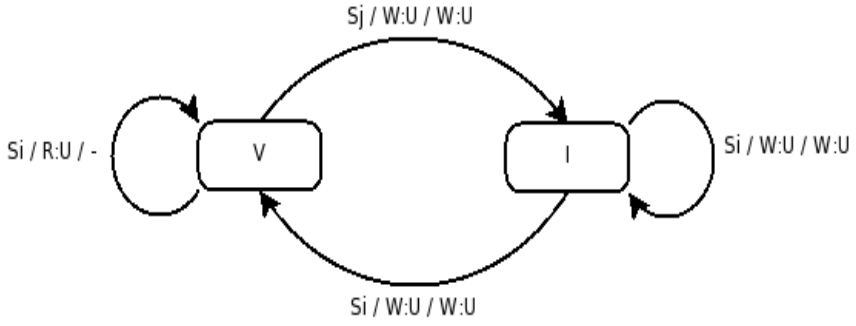
Dvoustavová synchronizace je nejjednodušší variantou synchronizačního protokolu. Sdílené zdroje mohou být ve dvou různých stavech a to V – valid a I – invalid.

Proces synchronizace probíhá následovně. Pří zápisu do sdíleného zdroje se ohlásí zápis všem simulátorům, které tento zdroj sdílí a ty změní stav zdroje na „invalid“. Při čtení sdíleného zdroje dojde v závislosti na jeho stavu nejdříve k aktualizaci hodnoty (pokud byl zdroj ve stavu I) nebo se přímo načte hodnota lokální (zdroj byl ve stavu V) [3].

Nevýhoda tohoto protokolu spočívá v tom, že se rozhlašuje každý zápis do sdílené proměnné i když je simulátor, který chce zapisovat výlučným vlastníkem daného zdroje(všechny ostatní simulátory mají zdroj ve stavu I). To je způsobeno tím, že výlučné vlastnictví není možné nijak zjistit. Oznamování každého zápisu pak zbytečně zatěžuje komunikační kanál a také simulátory, které by se tímto zápisem vůbec nemusely zabývat (stav ani hodnota lokální kopie zdroje se nezmění I -> I).

Stavový diagram čtení a zápisu sdíleného zdroje U pro simulátor Si [3]:

– popis přechodů je ve tvaru *simulátor provádějící akci / akce / broadcast*



Ilustrace 3: Stavový diagram dvoustavového protokolu

–

4.4 Třístavová synchronizace protokolem MSI

Řešení problému se zbytečným ohlašováním zápisů, kterým trpí dvoustavový protokol, se nabízí celkem jednoduché – rozšířit protokol o stav, který signalizuje, že daný zdroj je ve výlučném vlastnictví přistupujícího simulátoru. Tento rozšířený protokol se podle názvů jednotlivých stavů nazývá MSI.

Místo dvou stavů I a V má protokol tři stavy I – invalid, S – shared a M – modified. Stav *invalid* má stejný význam jako u dvoustavového protokolu, stav *shared* má podobný význam jako původní stav *valid*, jen s tím rozdílem, že zároveň značí skutečnost, že simulátor není výlučným vlastníkem zdroje a pokud bude chtít měnit hodnotu tohoto zdroje, musí to ohlásit. Poslední, avšak neméně důležitý, je stav *modified*, který značí, že zdroj byl byl naposledy modifikován simulátorem, který k němu přistupuje(je tedy jeho výlučným vlastníkem), a proto není nutné ohlašovat jakoukoliv změnu tohoto zdroje.

4.4.1 Modifikace MSI pro synchronizaci simulátorů

MSI protokol je původně navržen pro synchronizaci cache v multiprocesorových systémech se sdílenou pamětí a do synchronizace vstupuje navíc řadič sběrnice a hlavní sdílená paměť. V návrhu synchronizace simulátorů se používá upravená verze MSI tak, aby pokrývala požadavky simulace. Roli sběrnice zastává v návrhu komunikační médium a arbitráž můžeme řešit následujícími způsoby.

První způsob náhrady arbitra je úplné vynechání jakékoliv explicitní arbitráže na aplikační úrovni a nechat vše řešit síť případně operační systém. Druhou možností je zavedení další vrstvy do navržené architektury, která by byla v roli arbitra a zároveň by emulovala sdílenou paměť.

4.4.2 Modifikace s arbitrem

Jeden z uzlů je ve funkci arbitra. Arbitr, stejně jako simulátory, udržuje seznam kopií sdílených zdrojů a s informací o tom, který simulátor má aktuální hodnotu. Při synchronizaci by simulátory komunikovaly pouze s arbitrem, který by mimo arbitráže fungoval také jako proxy server pro požadavky simulátorů.

Na žádost o čtení zdroje reaguje arbitr takto: v seznamu zdrojů zkontroluje stav zdroje a pokud má aktuální hodnotu, odpoví simulátoru sám. Pokud arbitr nemá aktuální hodnotu, vyžádá si ji od simulátoru, který tuto hodnotu měnil poslední a má ji tedy aktuální. Odpoví simulátoru, který zaslal požadavek na aktualizaci zdroje a zároveň si aktualizuje hodnotu a stav zdroje.

Při žádosti o zápis arbitr rozešle všem simulátorům, které daný zdroj sdílí, zprávu o jeho změně a jak arbitr, tak simulátory sdílející zdroj si tento zdroj zneplatní. Změna hodnoty zdroje probíhá jen lokálně – nová hodnota zdroje se propisuje až ve chvíli, kdy o její čtení požádá simulátor, který aktuální hodnotu nemá.

Výhoda modelu s arbitrem spočívá v tom, že se omezí komunikace mezi jednotlivými procesory – arbitr by fungoval jako proxy server s cache (pokud má arbitr aktuální hodnotu, uspokojí požadavek na čtení sám) a simulátory by nepotřebovaly vědět, kde běží ostatní. Navíc získáme kontrolu nad pořadím přístupu simulátorů ke zdrojům. Naopak nevýhodou tohoto modelu je zvýšení nároků na střední vrstvu, která by plnila úlohu arbitra.

Příklad synchronizace zápisu a čtení [3]:

Aktuální data má simulátor 1.

	Stav před	Stav po	Signál arbitrovi	Stav v S1	Stav v S2	Stav u arbitra(po)	broadcast	Data od:
S2 a čte U	I	V	R:U	-	-	V	-	Arbitra
S2 zapisuje U	V	M	W:U	I	-	I	I:U	-
S1 zapisuje U	I	M	W:U	-	V	I	I:U	-
S2 čte U	I	V	R:U	V	-	V	-	P1

Tabulka 1: Synchronizace s arbitrem

4.4.3 Modifikace bez arbitra

Stejně jako u synchronizace s arbitrem má každý sdílený zdroj mimo hodnoty také stav. Pokud chce některý z simulátorů zapisovat do sdíleného zdroje, rozhlásí to všem ostatním simulátorům, se kterými tento zdroj sdílí. Získá tak výlučné vlastnictví, zapíše novou hodnotu zdroje a zdroj přejde do stavu *modified* a ve všech ostatních simulátorech se změní stav tohoto zdroje na *invalid*.

Při čtení hodnoty zdroje může v závislosti na stavu zdroje dojít ke dvěma situacím. Je-li zdroj ve stavu *shared* nebo *modified*, znamená to, že tento simulátor byl buď poslední, který zapisoval do tohoto zdroje(stav modified) nebo žádný jiný simulátor od poslední aktualizace zdroj neměnil a hodnota zdroje je tedy aktuální(stav shared). V tomto případě není třeba žádat o aktualizaci a lze použít hodnotu z lokální paměti. Je-li zdroj ve stavu *invalid,* musí simulátor zaslat žádost o aktuální hodnotu zdroje simulátoru, který měnil zdroj jako poslední. Po aktualizaci bude zdroj v obou simulátorech ve stavu *shared.*

Přestože synchronizace s další vrstvou v roli arbitra více odpovídá původnímu protokolu MSI, dostala nakonec přednost verze bez arbitra. Při simulaci víceprocesorového systému totiž explicitní synchronizace na úrovni na které pracuje navrhovaný protokol nemá smysl. Algoritmus synchronizace přístupů ke sdíleným zdrojům (ve smyslu serializace zápisů a čtení) je totiž modelován v samotném popisu architektury a MSI je pouze prostředek pro její implementaci.

Příklad synchronizace zápisu a čtení [3]:

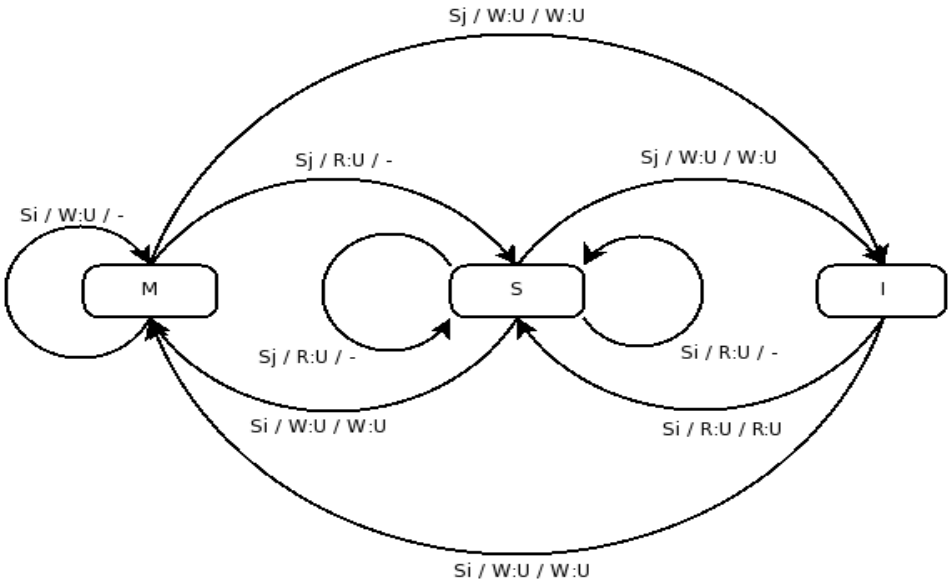
Aktuální data má simulátor 1.

Akce	Stav před	Stav po	Broadcast	Stav v S1	Stav v S2	Data od:
S2 a čte U	I	V	R:U	V		S1
S2 zapisuje U	V	M	W:U	I		-
S1 zapisuje U	I	M	W:U		I	-
S2 čte U	I	V	R:U	M		S1

Tabulka 2: Synchronizace bez arbitra

Stavový diagram čtení a zápisu sdíleného zdroje U pro simulátor Si [3]:

- popis přechodů je ve tvaru *simulátor provádějící akci / akce / broadcast*

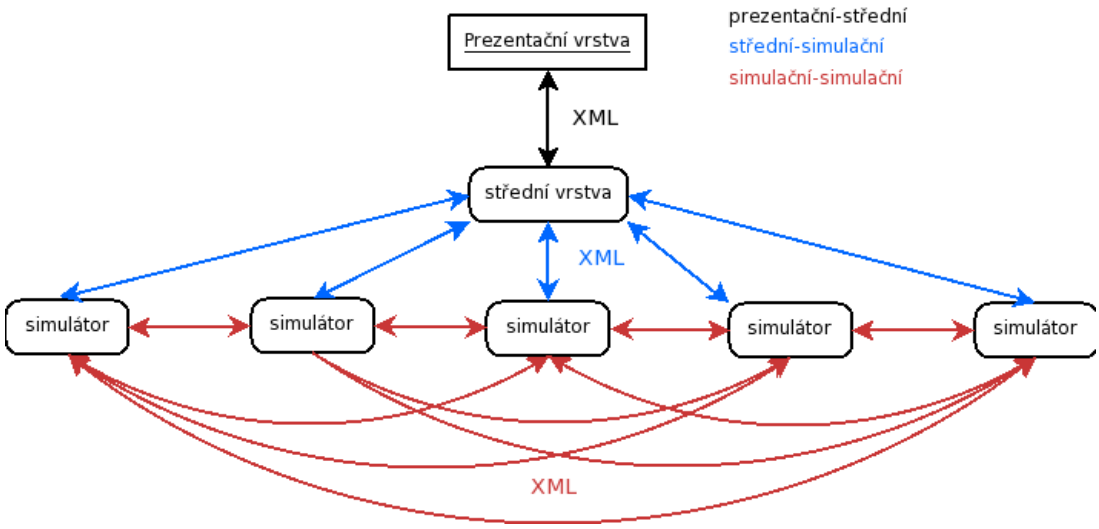


Ilustrace 4: Stavový diagram protokolu MSI

5 Návrh struktury protokolu

Při návrhu struktury protokolu bylo nutné brát v úvahu dvě hlediska transparentnost protokolu a propustnost komunikačního média.

Transparentnost protokolu se řeší z hlediska komunikace mezi vrstvami „prezentační - střední“, „střední - simulační“ a „simulační - simulační“. Struktura protokolu „prezentační-střední“ vrstva je definovaná ve formátu XML. Aby nemusely být jednotlivé fragmenty protokolu „prezentační-střední“ vrstva transformovány na strukturu protokolu „střední-simulační“ vrstva případně na strukturu „simulační-simulační“ vrstva, je struktura protokolu „simulační-střední“ vrstva a„simulační – simulační“ vrstva definována rovněž ve formátu XML. Oba protokoly obsahují společné struktury (pro přenos ladících informací, statistik a stavů zdrojů), které se pomocí střední vrstvy pouze vhodně přesměrovávají.



Ilustrace 5: Struktura protokolu

Pro lepší čitelnost(alespoň při fázi ladění) a při použití XML pro definici struktury protokolu se jako nejlepší řešení nabízí použít klasické textové reprezentace.

5.1 Struktura zpráv

Při návrhu byl kladen důraz na jednoduchost zpracování zprávy. I když mají dnešní počítače dostatek výkonu, je zbytečné plýtvat prostředky na parsování XML a ubírat tak na rychlosti simulace. Z toho důvodu je struktura zpráv navržena co možná nejjednodušší, aby bylo umožněno rychlé zpracovaní informací obsažených ve zprávě.

Jelikož je protokol definován pomocí XML, skládá se z elementů. Hlavním elementem, který obaluje celé tělo každé zprávy je *command.* Element určující význam zprávy je *id* a za ním následuje část *params*, která obsahuje různý počet parametrů *items*. Význam atributů se liší podle typu zprávy.

Struktura zprávy v XML:

```
<COMMAND>
      <ID>COMMAND_ID</ID>
            <PARAMS>
                  <ITEM>item1</ITEM>
                  <ITEM>item2</ITEM>
                  ....
            </PARAMS>
</COMMAND>
```

Z pohledu funkcionality existují v protokolu 2 druhy zpráv:

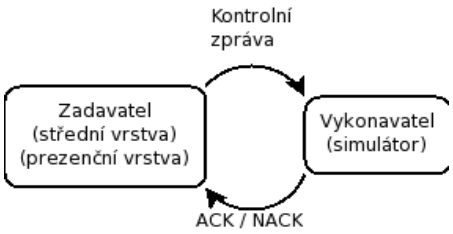
1. Kontrolní zprávy
2. Zprávy pro přenos dat

5.1.1 Kontrolní zprávy

Kontrolní zprávy jsou zprávy, které ovlivňují běh simulátoru – například spuštění nebo zastavení simulátorů, vložení bodu přerušení. Kontrolní zprávy samozřejmě také obsahují v těle zprávy data a to například jméno odesílatele nebo místo vložení breakpointu. Odpovědí na kontrolní zprávy je buď ACK (potvrzení vykonání) nebo NACK (zamítnutí).

Příklad kontrolní zprávy (zastavení simulace):

```
<COMMAND>
        <ID>SML_STOP</ID>
                <PARAMS>
                        <ITEM>simulator1</ITEM>
                </PARAMS>
</COMMAND>
```



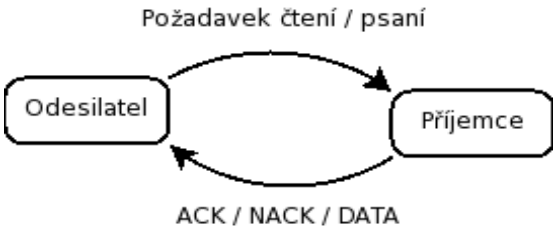
Ilustrace 6: Kontrolní zprávy

5.1.2 Zprávy pro přenos dat

Jak již napovídá název, slouží tyto zprávy pro přenášení dat mezi simulátory – konkrétně hodnot zdrojů. Na rozdíl od kontrolních zpráv slouží tyto zprávy k zápisu nebo čtení hodnot sdílených zdrojů. Tělo těchto zpráv obsahuje vždy seznam jmen požadovaných zdrojů (při čtení), případně seznam jmen zdrojů společně s hodnotou (při zápisu). Odpovědí může být ACK, NACK nebo zpráva s požadovanými daty.

Příklad přenosu dat (zaslání hodnoty zdroje):

```
<COMMAND>
        <ID>VALUE</ID>
        <PARAMS>
                <ITEM>ax</ITEM>
                <ITEM>1</ITEM>
        </PARAMS>
</COMMAND>
```



Ilustrace 7: Zprávy pro přenos dat

6 Specifikace funkcionality protokolu

6.1 Inicializace simulátorů

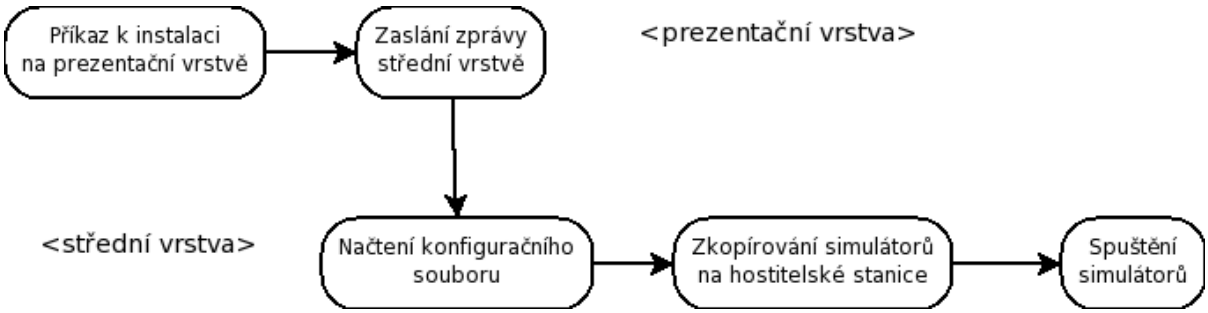
Před inicializací simulátorů se nejdříve musí spustit střední vrstva, která má za úkol celou inicializaci včetně instalace jednotlivých simulátorů na jejich hostitelské stanice. Instalace i inicializace probíhá přes ssh (nahrávání souborů přes scp).

Použít k inicializaci ssh je vhodné hned z několika důvodů – má jednoduchou obsluhu, existují implementace pro všechny rozšířenější operační systémy a samotné připojování ke vzdálenému počítači se dá plně automatizovat a spouštění příkazů je pak transparentní.

Bezprostředně po startu zná simulátor pouze identifikaci vlastníka každého sdíleného zdroje. Konkrétní adresy ostatních simulátorů získá simulátor od střední vrstvy při spouštění simulace. Střední vrstva načte seznam jmen simulátorů a odpovídajících adres při instalaci simulátorů a tento seznam uchovává po celou dobu simulace. Jakmile simulátor dostane požadovaný seznam adres, přejde do stavu, ve kterém čeká na příkaz ke spuštění simulace. Simulace začne jakmile všechny simulátory tento seznam získají.

6.1.1 Instalace simulátorů

Instalace a spuštění simulátorů provede střední vrstva na žádost prezentační vrstvy (resp. na žádost uživatele). Jakmile obdrží střední vrstva příkaz k instalaci simulátorů, načte odpovídající konfigurační soubor, ve kterém jsou uloženy jména simulátorů společně s jejich adresami. Pomocí scp pak nahraje jednotlivé simulátory na odpovídající stanice. Podle stejného konfiguračního souboru se ve fázi instalace vytvoří ve střední vrstvě tabulka simulátorů, která bude obsahovat informace potřebné pro následnou inicializaci všech simulátorů. Průběh instalace je znázorněn na ilustraci 8.



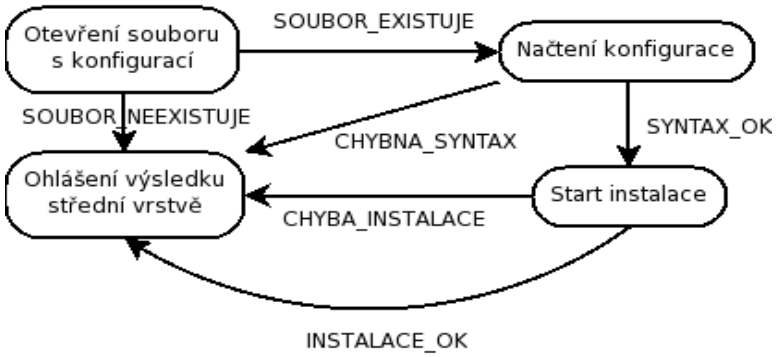
Ilustrace 8: Instalace simulátorů

6.1.1.1 Chybové stavy

Při instalaci simulátorů může dojít k několika chybovým stavům:

1. neexistuje konfigurační soubor,
2. konfigurační soubor má špatnou syntax,
3. konfigurační soubor obsahuje neplatné informace(adresy, jména simulátorů),
4. vzdálený počítač neodpovídá.

Po ukončení instalace zašle střední vrstva zprávu prezentační vrstvě, která v závislosti na výsledku instalace informuje uživatele nebo ho vyzve k nápravě.



Ilustrace 9: Instalace simulátorů

6.1.2 Start simulátorů

Po dokončení instalace simulátorů na hostitelské stanice dá střední vrstva příkaz přes ssh k jejich spuštění. Pro zajištění určité míry volnosti nemají simulátory předem určenou ani IP adresu, ani port, na kterém mají běžet. Proto, jak bylo zmíněno v úvodu kapitoly, je povinností střední vrstvy simulátorům tyto parametry předat. Informace o tom, na kterých portech mají jednotlivé simulátory běžet je součástí konfiguračního souboru, podle kterého prováděla střední vrstva instalaci. Pro předání informace o přiděleném portu se nabízí velmi jednoduché řešení – předat číslo portu jako parametr při spouštění.

6.1.2.1 Tabulka simulátorů udržovaná v střední vrstvě

Střední vrstva při inicializaci musí vytvořit a udržovat seznam simulátorů. Tento seznam obsahuje informace o všech simulátorech, které se účastní simulace. Seznam je reprezentován tabulkou, ve které je pro každý simulátor jeden záznam. Každý z těchto záznamů obsahuje jméno simulátoru, adresu a port pro daný simulátor.

Název simulátoru	adresa	port
dsp	192.168.1.1	4000
mp3dec	192.168.1.1	5000
gpu	192.168.1.2	7000

Tabulka 3: Tabulka zdrojů udržovaná ve střední vrstvě

Ilustrace 10: Spuštění simulátorů

6.1.3 Spojení mezi každým simulátorem a střední vrstvou

Po spuštění simulátorů, před započetím samotné simulace, se musí ustavit spojení mezi každým simulátorem a střední vrstvou. Toto spojení iniciuje střední vrstva, která po instalaci simulátorů na jejich hostitelské stanice zná potřebné IP adresy a porty, na kterých simulátory poslouchají. Po přijetí spojení na straně simulátoru pošle simulátor zprávu střední vrstvě o úspěšném přijetí a inicializaci spojení.

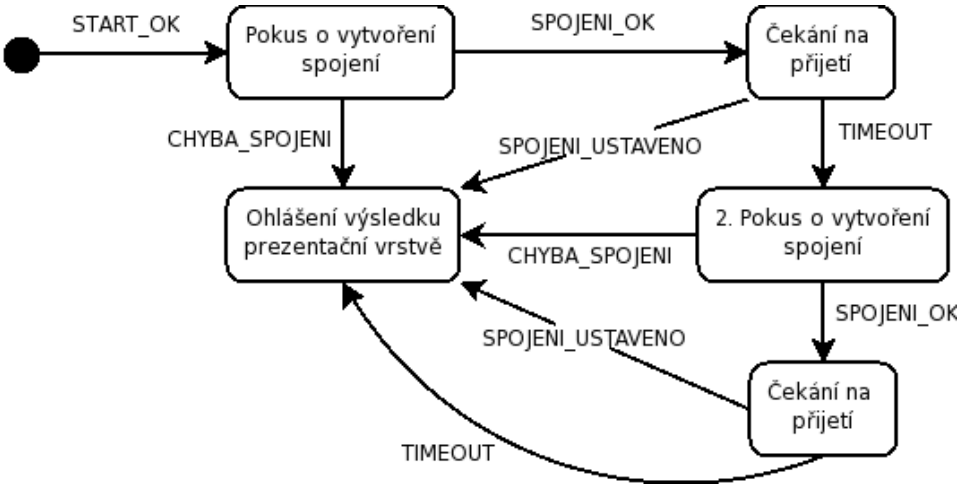
Toto řešení je odlišné od původního návrhu, ve kterém byl v roli iniciátora spojení simulátor. V tomto původním návrhu bylo třeba, aby se při spouštění předávala simulátorům také adresa a port střední vrstvy. Výhodou proti původnímu řešení je jednodušší a menší zásah do implementace střední vrstvy.

6.1.3.1 Chybové stavy

Při ustavování spojení mezi simulátory a střední vrstvou mohou nastat 2 chybové stavy:

1. timeout
2. zamítnutí připojení

Pokud dojde k zamítnutí připojení, ohlásí střední vrstva chybu prezenční vrstvě, která informuje uživatele. Dojde-li k timeoutu, odpověď nedojde v požadovaném časovém intervalu, opakuje se žádost o spojení. Pokud se spojení neustaví ani při opakovaném pokusu, zašle opět střední vrstva zprávu o chybě do vrstvy prezentační.



Ilustrace 11: Ustavování spojení mezi střední vrstvou a simulátory

6.1.4 Získávaní adres simulátorů

Po spuštění simulátoru a inicializaci serverové část simulátoru je simulátor ve stavu, ve kterém pouze čeká na příkaz ke spuštění simulace. Protože každý simulátor potřebuje komunikovat s ostatními simulátory musí znát jejich síťovou adresu. Stejně jako porty, tak ani adresy nejsou pevně určené, a proto není jiná možnost, než získat je až za běhu, před započetím samotné simulace. Seznam adres všech simulátorů proto zasílá střední vrstva již spuštěnému simulátoru jako parametr příkazu pro spuštění simulace. Simulátor pak při zpracování tohoto příkazu nejdříve vytvoří připojení s ostatními simulátory a následně spustí simulaci.

Příklad zprávy ke spuštění simulace společně se seznamem adres:

```
<COMMAND>
 <ID>SIMULATOR_START<ID>
       <PARAMS>
             <ITEM>dsp</ITEM> -- identifikace simulátoru
             <ITEM>192.168.1.1</ITEM>  -- IP adresa
             <ITEM>4000</ITEM> -- port
             <ITEM>mp3dec</ITEM>
             <ITEM> 192.168.1.1</ITEM>
             <ITEM>5000</ITEM>
       </PARAMS>
</COMAND>
```

6.1.4.1 Chybové stavy

Při zpracování seznamu adres mohou nastat tyto chybové stavy:

1. adresa některého simulátoru není k dispozici.

K chybě, kdy některý ze simulátorů nezíská adresy ke všem simulátorům, jejichž zdroje podle popisu používá dojde tehdy, když je chyba v popisu simulátoru nebo v konfiguračním souboru, který střední vrstva použila při instalaci simulátorů.

Ilustrace 12: Získávání adres simulátorů

6.1.5 Spojení mezi jednotlivými simulátory

Každý simulátor uchovává seznam simulátorů s adresami v tabulce podobně jako střední vrstva. Pro dohledání adresy pro konkrétní sdílený zdroj musí simulátor využít dvě tabulky – tabulku, která spojuje zdroj se jménem simulátoru a tabulku, která spojuje jméno simulátoru s adresou.

Spojení mezi všemi simulátory se vytvoří před začátkem simulace a existují pak až do jejího konce (existence spojení není velká zátěž pro systém - naproti tomu neustálá tvorba spojení by zpomalovala běh simulace).

Přímé spojení mezi simulátory se využívá pro výměnu simulačních dat mezi simulátory. Protože se hodnoty zdrojů ve střední vrstvě aktualizují pouze při žádosti prezentační vrstvy, probíhá výměna dat mezi simulátory bez vědomí střední vrstvy.

Název zdroje	adresa	port
mem.dsp	192.168.1.1	4000
acc.dsp	192.168.1.1	5000
mem.mp3dec	192.168.1.2	6000

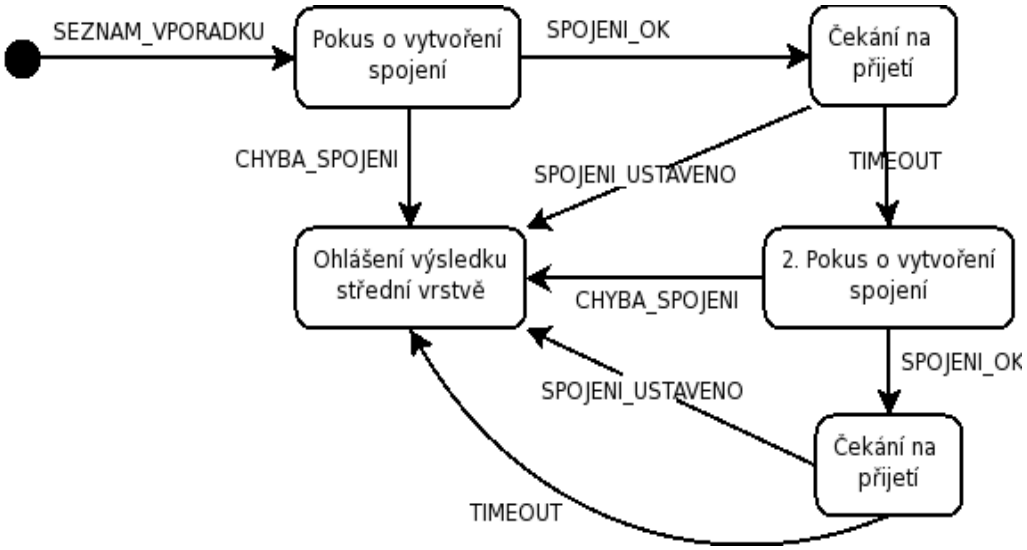
Tabulka 4: Tabulka externích zdrojů v simulátorech

6.1.5.1 Chybové stavy

Při ustavování spojení mezi simulátory mohou nastat 2 chybové stavy:

1. zamítnutí připojení,
2. timeout.

Pokud dojde k zamítnutí připojení, ohlásí se chyba střední vrstvě a simulace končí. Dojde-li k timeoutu – odpověď nedojde v požadovaném časovém intervalu, opakuje se žádost o spojení. Pokud se spojení nepovede ustavit ani při opakovaném pokusu, ohlásí se střední vrstvě chyba, která se propaguje do prezentační vrstvy.



Ilustrace 13: Ustavování spojení mezi simulátory

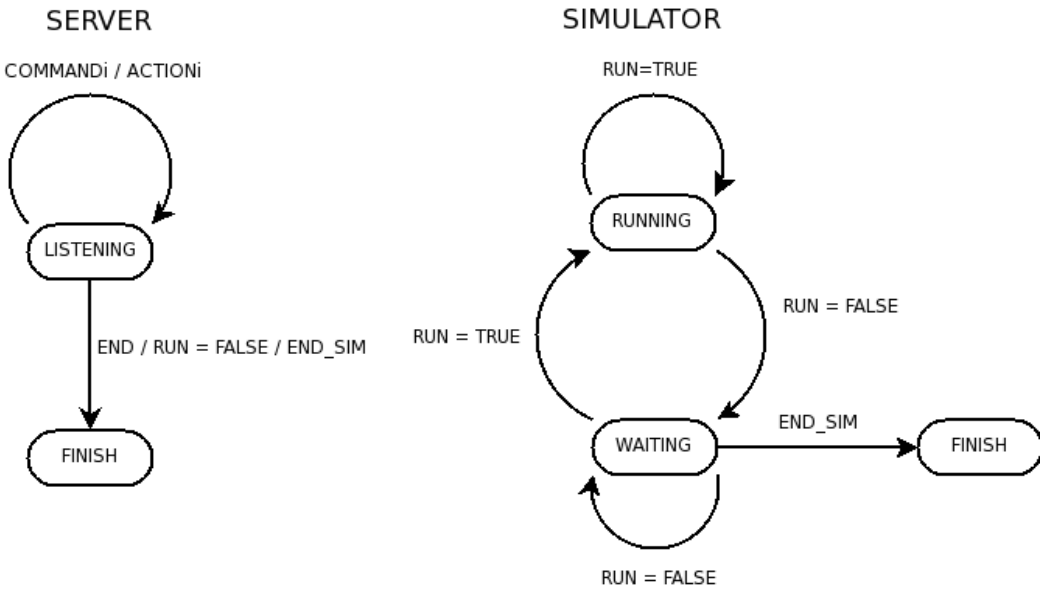
6.2 Ovládání simulátoru

Aby simulátor mohl provádět simulaci a zároveň přijímat zprávy od ostatních simulátorů a střední vrstvy, běží ve dvou vláknech. Jedno vlákno zajišťuje komunikaci (serverová část) a druhé vlákno provádí samotnou simulaci.

Po spuštění čekají všechny simulátory na příkaz ke startu simulace. V tomto stavu je aktivní pouze serverová část simulátoru. Část provádějící vlastní simulaci se spustí až ve chvíli, kdy k tomu dojde příkaz ke od střední vrstvy. Spuštění a zastavení simulace je signalizováno vlajkou, kterou nastavuje serverová část na základě příkazů od střední vrstvy. Vlákno provádějící simulaci kontroluje vlajku mezi prováděním jednotlivých instrukcí a pokud je signalizováno zastavení, simulace se přeruší a simulátor pouze čeká další příkazy (např. vkládání breakpointů).

Zpráva pro spuštění simulace:

```
<COMMAND>
<ID>SIMULATOR_START</ID>
        <PARAMS>          -- seznam simulátorů s adresami – viz. Získávání adres
                <ITEM></ITEM>
        </PARAMS>
</COMMAND>
```



Ilustrace 14: Ovládání simulátoru

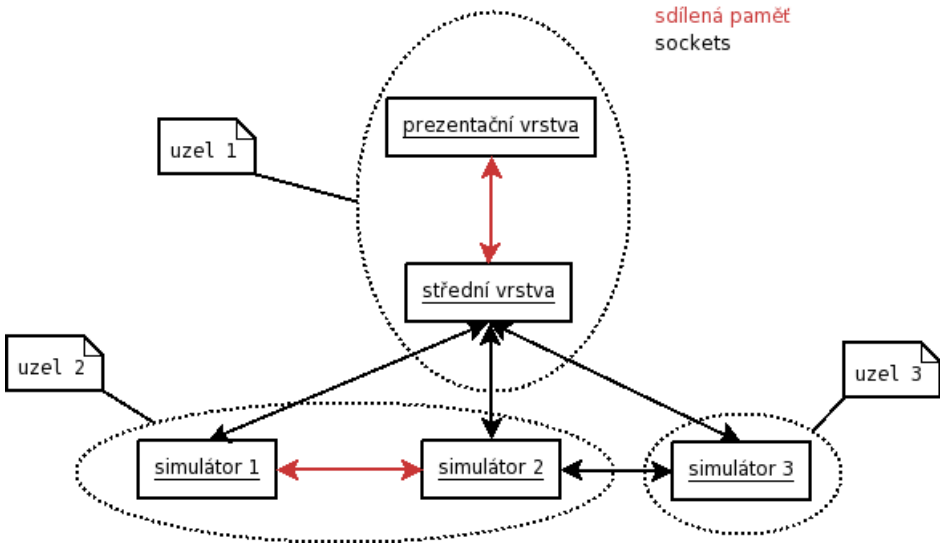
6.3 Běh více vrstev na stejném uzlu

Při paralelní simulaci může nastat situace, kdy bude méně fyzických uzlů, na kterých má simulace běžet, než vrstev, které se účastní simulace. V takovém případě budou některé vrstvy sdílet jeden společný uzel.

V případě běhu více vrstev na stejném uzlu se nabízí několik kombinací vrstev, které mohou běžet na stejném uzlu - „simulátor – simulátor“, „simulátor – střední vrstva“, „střední vrstva – prezentační vrstva“. Poslední kombinaci „prezentační vrstva – simulační vrstva“ nemá smysl uvažovat, protože mezi těmito dvěma vrstvami neexistuje přímá komunikace. O konkrétní kombinaci se rozhoduje při instalaci jednotlivých vrstev s ohledem na požadavky na rychlost komunikace mezi konkrétními vrstvami a s ohledem na výkonnost uzlů, na kterých simulace poběží.

Vrstvy, které běží na stejném uzlu mohou využívat rychlejšího spojení pomocí sdílené paměti. Pro komunikaci mezi vrstvami na různých uzlech se bude používat klasického síťového spojení (přes sockety). Jako komunikační médium mezi jednotlivými vrstvami je tedy možné používat sockety, sdílenou paměť nebo jejich vhodnou kombinaci.

Například máme-li k dispozici cluster ze 3 počítačů a 3 simulátory, může vypadat rozdělení takto:



Ilustrace 15: Běh více vrstev na jednom uzlu

7 Návrh mechanismu pro sdílení simulačních dat

Během ladění aplikace simulované simulátorem je nutné zobrazovat statistiky a stavy zdrojů v klientech uživatelů (prezenční vrstvě), a zároveň zabezpečit předání simulačních dat (zejména stavy zdrojů) mezi definovanými simulátory během simulace. Pro splnění těchto požadavků musí být navržena podpora v simulační knihovně a v simulačním protokolu.

Pokud simulátory běží ve dvou vláknech, tak, jak bylo navrženo v kapitole 6.2, tak jedno vlákno zpracovává požadavky ostatních simulátorů a střední vrstvy a druhé vlákno provádí simulaci. Pokud tedy některý ze simulátorů požádá o čtení nebo zápis, není nutné simulaci přerušovat.

Při ladění zasílá prezentační vrstva požadavky střední vrstvě, která pak tyto požadavky směruje na konkrétní simulátor. Prezentační vrstva nezná fyzické umístění jednotlivých simulátorů – zná pouze jméno sledovaného zdroje a vše řeší střední vrstva, která má tabulku zdrojů, podle které určí adresu simulátoru, na který má požadavek směrovat. Pro přenos stavů vybraných mikroprocesorů mezi simulátory se bude používat stejných stejných volání jako mezi střední vrstvou a simulátory.

Ilustrace 16: Sdílení simulačních dat

Požadavek na aktualizaci hodnoty může vypadat například takto:

(žádost střední vrstvy o hodnotu registru **ax**)

```
<COMMAND>
        <ID>READ</ID>
                <PARAMS>
                        <ITEM>ax</ITEM>  -- název požadovaného zdroje
                </PARAMS>
</COMMAND>
```

Požadavek, který pošle střední vrstva simulátoru bude mít stejný tvar, jako požadavek prezentační vrstvy. Odpověď posílá simulátor střední vrstvě, která výsledek předá prezentační vrstvě.

Odpověď simulátoru může vypadat například takto:

```
<COMMAND>
        <ID>VALUE<ID>
                <PARAMS>
                        <ITEM>ax</ITEM>  -- název požadovaného zdroje
                        <ITEM>150</ITEM> -- hodnota
                </PARAMS>
</COMMAND>
```

8 Implementace

Současná verze projektu je schopná generovat dva druhy simulátorů – kompilovaný a interpretovaný. Obě verze simulátoru mají své výhody i nevýhody. Interpretovaný simulátor je nezávislý na programu, který chceme odsimulovat a doba pro jeho vygenerování je výrazně kratší, než je tomu u simulátoru kompilovaného. Naproti tomu kompilovaný simulátor je závislý na simulovaném programu, takže se při změně programu musí vytvořit nový simulátor, ale nabízí mnohem větší rychlost při simulaci.

V rámci této práce byla implementována podpora pro víceprocesorovou simulaci do interpretovaného simulátoru. Dále byly obohaceny příkazy prezentační vrstvy a rozšířena střední vrstva o možnost spravovat více simulátorů.

8.1 Zdroje interpretovaného simulátoru

Interpretovaný simulátor je implementován s využitím objektového modelu. Každý typ zdroje je reprezentován jednou třídou a operace nad zdroji jsou definovány jako metody těchto tříd a přetížené operátory.

Pro podporu multiprocesorové simulace bylo nezbytné rozšířit původní třídy o atributy uchovávající jméno zdroje a jeho stav. Tyto atributy jsou potřebné pouze pro sdílené zdroje, a proto byl pro sdílené zdroje vytvořen i nový konstruktor, který požadované atributy nastavuje. Dále bylo zapotřebí upravit i původní metody a operátory tak, aby ve svém chování tyto nové vlastnosti odrážely. Při implementaci potřebných úprav byl brán ohled na to, aby se neměnilo rozhraní upravovaných metod. Zachování původního rozhraní bylo výhodné z toho důvodu, aby se změny uchovaly lokálně a nebylo tedy nutné upravovat ostatní nástroje, které s definicemi zdrojů pracují.

Příklad třídy představující registr procesoru - TRegister:

```
class TRegister
{
public:
 int size;                //bitová šířka
 int data;                //prostor pro uložení dat
 mutable int wstat;      //počítadlo zápisů
 mutable int rstat;      //počítadlo čtení
 bool shared;            //označení jestli je zdroj sdílený
 char *name;             //jméno zdroje
 tResState flag;         //vlajka, která udržuje stav zdroje M,S,I
 pthread_mutex_t guard; //mutex, který hlídá zápis/čtení hodnoty
 pthread_mutex_t flag_mutex;//mutex, který hlídá změny stavu
 read();
 write();
 //konstruktor sdíleného zdroje
 TRegister::TRegister(const char* rname, MSIL* msi)
```

Příklad vytvoření objektu, který reprezentuje sdílený zdroj:

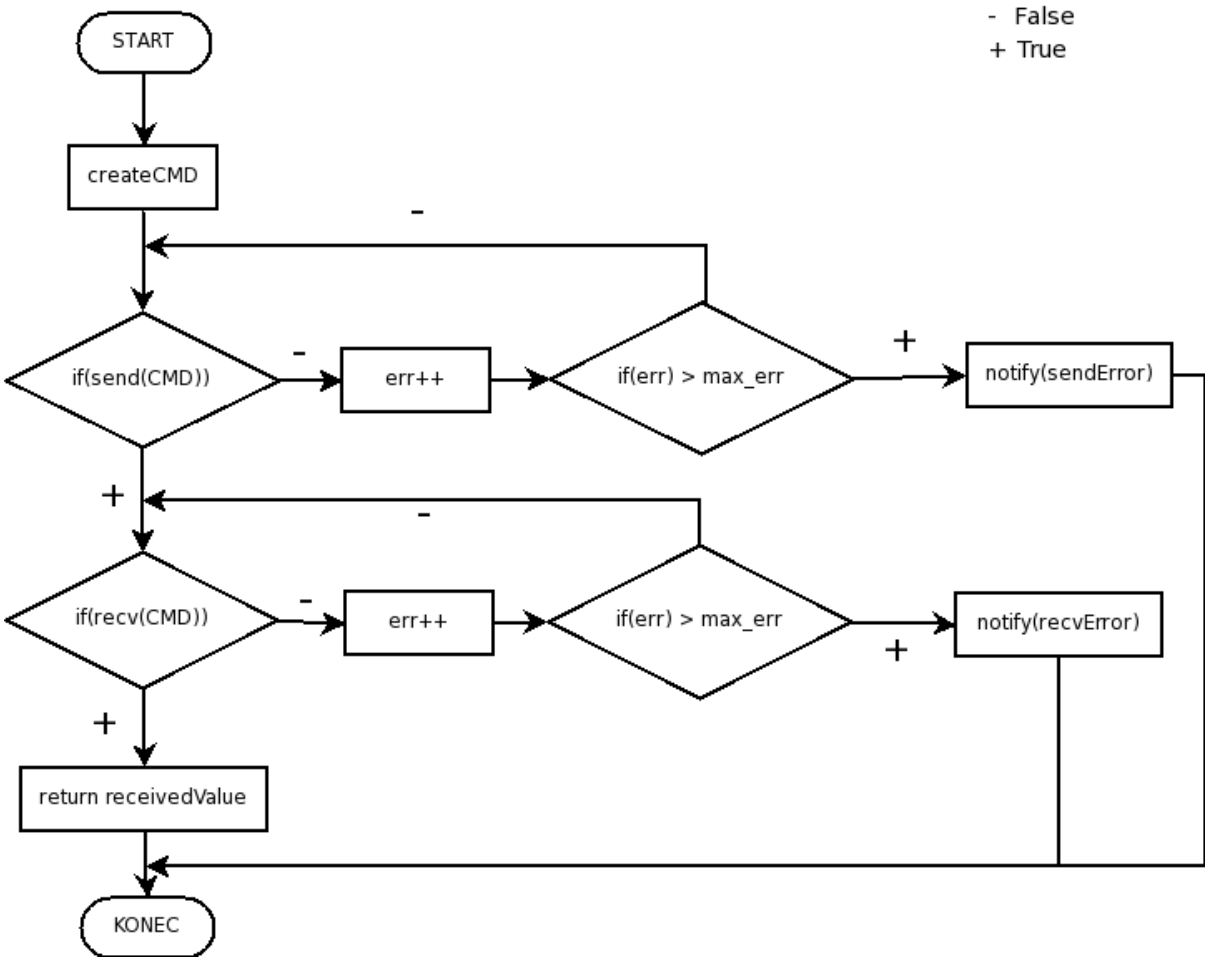
```
TRegister ax = new Tregister("ax", msi)
```

8.2 Podpora protokolu MSI

Jak je uvedeno v předchozím textu, používá se pro komunikaci mezi jednotlivými vrstvami tzv. zasílání zpráv. Tímto způsobem je tedy provedena i implementace samotného MSI protokolu, který se používá pro synchronizaci hodnot mezi simulátory. Pro jednoduchost použití byla vytvořena knihovna poskytující jednoduché rozhraní pro čtení a zápis sdílených zdrojů s využitím MSI protokolu. Knihovna zapouzdřuje jak tvorbu posílaných zpráv, tak i samotnou komunikaci. Hlavními dvěma funkcemi této knihovny jsou funkce *read_res* – používaná při čtení sdíleného zdroje a funkce *write_res* používaná při zápisu sdíleného zdroje. Ostatní funkce a jejich implementace není z hlediska implementace protokolu podstatná.

8.2.1 Funkce read_res

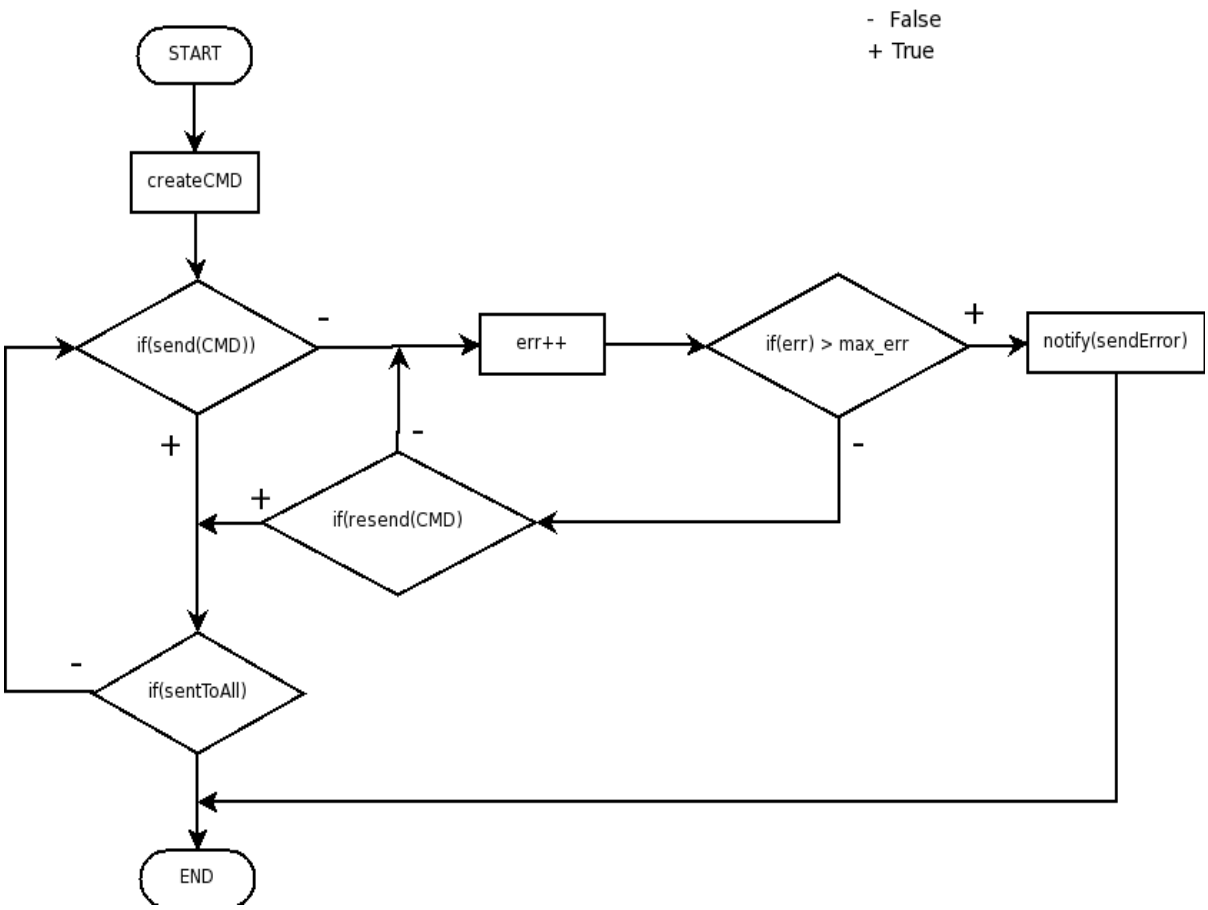
Funkce *read_res* slouží k získání aktuální hodnoty zdroje. Jako parametr má pouze jméno zdroje, jehož hodnotu je potřeba aktualizovat. Funkce na základě tabulky *lastWrite,* ve které je ke každému zdroji uloženo jméno simulátoru, který jako poslední měnil hodnotu zdroje, zjistí, který simulátor má aktuální hodnotu zdroje a zašle mu zprávu obsahující požadavek o zaslání této hodnoty. Návratovou hodnotou funkce je aktuální hodnota zdroje získaná z odpovědi od dotazovaného simulátoru. Detail implementace je zachycen na následujícím obrázku.



Ilustrace 17: Vývojový diagram funkce read_res

8.2.2 Funkce write_res

Již z názvu je zřejmé, že se jedná o funkci která má za úkol zprostředkovat zápis hodnoty sdíleného zdroje v rámci protokolu MSI. Funkce rozešle zprávu o zápisu všem simulátorům (funkce nepřenáší zapisovanou hodnotu).



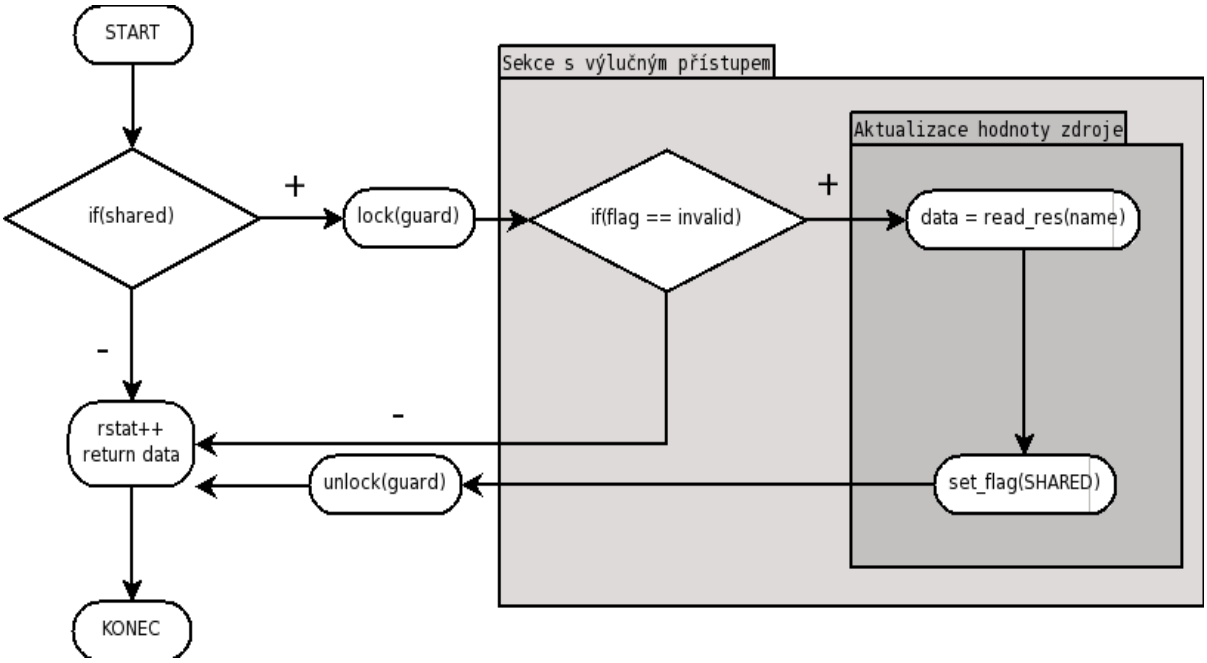
Ilustrace 18: Vývojový diagram funkce write_res

8.2.3 Čtení sdíleného zdroje

Metody pro čtení zdrojů byly upraveny tak, aby se jejich chování měnilo v závislosti na tom, jde-li o zdroj sdílený nebo lokální.

Metoda nejprve zkontroluje, je-li zdroj sdílený. Pokud se nejedná o sdílený zdroj, zvýší se počítadlo čtení a metoda vrátí lokání hodnotu zdroje. Pro sdílený zdroj se nejprve podle atributu *flag* zjišťuje, stav zdroje. Stavy MODIFIED a SHARED znamenají, že hodnota lokální kopie je aktuální. Pokud je ale zdroj ve stavu INVALID, musí se hodnota zdroje aktualizovat.

K aktualizaci hodnoty se využívá knihovní funkce *read_res* implementovaná v rámci protokolu MSI. Ta zašle požadavek na aktualizaci hodnoty simulátoru, který má aktuální hodnotu. Jakmile získá odpověď, aktualizuje se hodnota lokální kopie, zvýší se počítadlo čtení a metoda vrátí aktualizovanou lokální hodnotu.

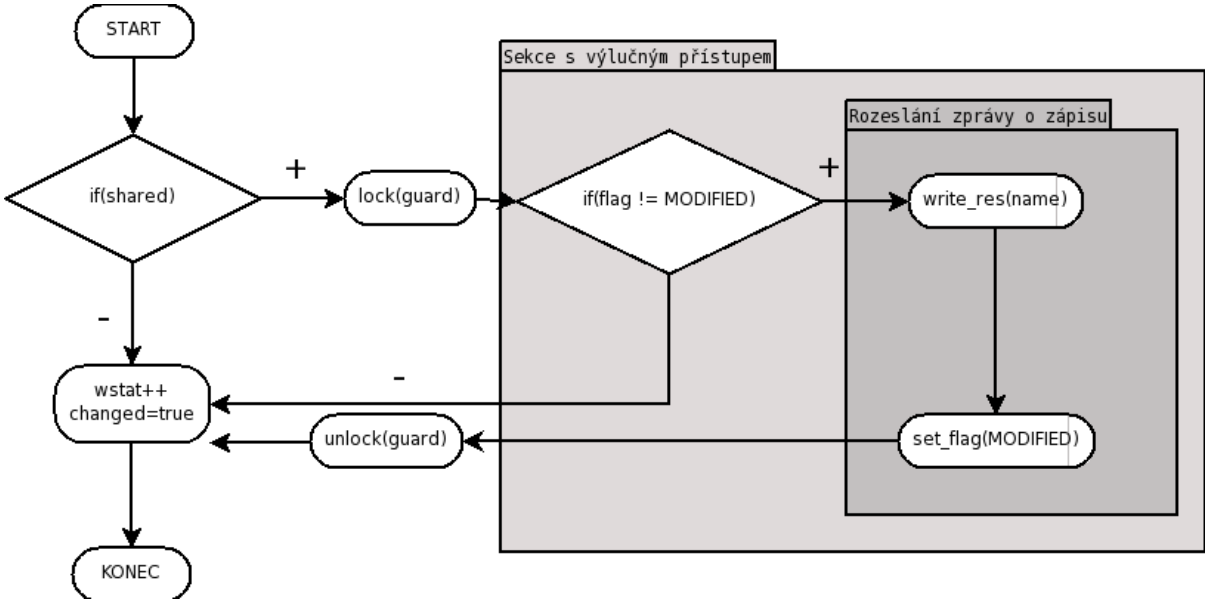


Ilustrace 19: Čtení sdíleného zdroje

8.2.4 Zápis do sdíleného zdroje

Stejně jako u metod pro čtení musí i metody pro zápis odrážet jde-li o zdroj sdílený nebo lokální a sdíleného zdroje reagovat podle jeho stavu.

Podobně jako je tomu při čtení, se nejdříve podle vlajky *shared* rozhodne, o jaký zdroj se jedná. Pokud se nejedná o sdílený zdroj, provede se zápis hodnoty a zvýší se počítadlo zápisů. Při zápisu do sdíleného zdroje se nejprve zkontroluje jeho stav. Je-li stav MODIFIED, je možné bez dalších opatření provést zápis nové hodnoty. V opačném případě (zdroj je ve stavu SHARED nebo INVALID) je nutné oznámit zápis ostatním simulátorům, aby si své lokální kopie zneplatnili. K oznamování zápisu se používá knihovní funkce *write_res,* která všem simulátorům rozešle zprávu o zápisu, obsahující jméno zapisovaného zdroje a jméno simulátoru, který zápis provedl. Ostatní simulátory si musí aktualizovat tabulku *lastWrite*, která ke každému sdílenému zdroji udržuje informaci, který ze simulátorů jej poslední měnil. Podle této tabulky se při aktualizaci hodnoty zdroje zjistí jméno simulátoru, který provedl poslední změnu a může tedy poskytnout aktuální hodnotu.



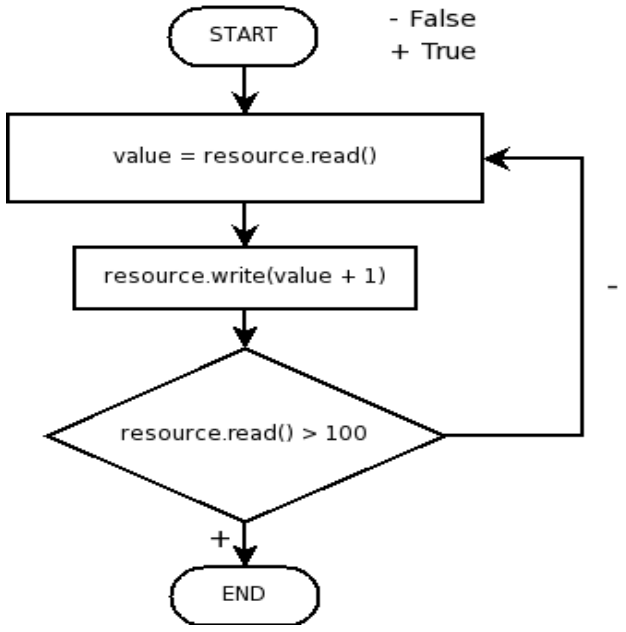
Ilustrace 20: Zápis sdíleného zdroje

9 Testování, hodnocení výsledků

V době kdy vznikala tato práce nebylo ještě hotové rozšíření jazyka ISAC o struktury pro modelování víceprocesorového systému SoC z čehož plyne, že nebylo možné použít při testování reálný model vytvořený pomocí nástrojů, které Lissom poskytuje. Proto bylo nezbytné vytvořit pro tyto účely jednoduchý testovací simulátor ručně – s trochou nadsázky by se dalo říct, že bylo potřeba vytvořit simulátor simulátoru.

Podle návrhu je tedy testovací simulátor implementován jako dvouvláknová aplikace, ve které jedno z vláken provádí simulaci a druhé vlákno zprostředkovává komunikaci s okolím. Simulační vlákno pracuje s jedním sdíleným zdrojem typu TRegister. Celá simulace spočívá pouze v počítání vzestupné sekvence čísel v rozsahu 10 – 100.

Cílem testování bylo ověřit funkčnost implementovaného protokolu a navrhovaných úprav simulátoru.



Ilustrace 21: Vývojový diagram testovací úlohy

9.1 Průběh testování

Funkčnost celého návrhu byla průběžně testována v jednotlivých fázích vývoje. V počátečních stádiích šlo pouze o testování funkčnosti rozšíření střední a prezentační vrstvy. Další fáze testování spočívala v ověření funkčnosti protokolu MSI a komunikace simulátorů se střední vrstvou.

Jednoduchou modifikací testovacího simulátoru bylo vytvořeno několik dalších simulátorů, které byly postupně připojovány do simulace. Postupné přidávání simulátorů mělo ukázat schopnost implementovaného protokolu pracovat s různým počtem simulátorů. Společně s přidáváním simulátorů se také měnilo jejich umístění. Simulátory byly spouštěny jak na společném hostitelském počítači, tak i na oddělených uzlech, a v různých kombinacích.

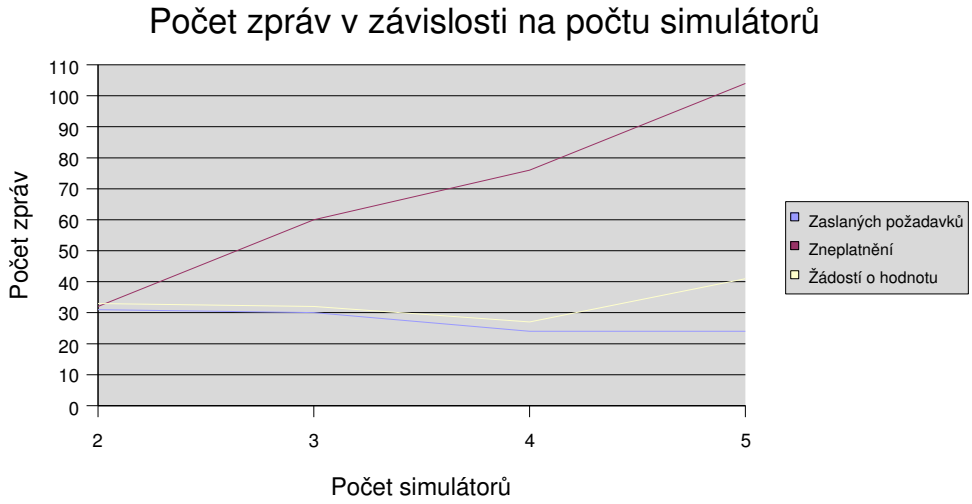
9.2 Výsledky testování

Výsledky testů jsou uspokojivé. Ukázalo se, že implementovaný protokol MSI funguje s obecným počtem simulátorů a střední vrstva je schopna spravovat simulátory běžící na oddělených i společných uzlech.

Testování ale také odhalilo chybu v implementaci ustavování spojení mezi střední vrstvou a simulátory. V případě, kdy spuštění simulátoru na hostitelské stanici bylo z nějakého důvodu pomalé (např. málo výkonný stroj) docházelo na střední vrstvě k timeoutu při připojování a start simulace se nezdařil. Řešení této situace se však nabízí celkem jednoduché. Stačí prodloužit dobu čekání mezi jednotlivými pokusy o připojení, případně zvýšit počet pokusů. Jelikož je doba spouštění simulátorů závislá na výkonnosti počítačů, na kterých se simulace provozuje je také vhodné, aby bylo možno dobu čekání a počet pokusů nastavovat jako parametr v některém z konfiguračních souborů.

Součástí testování bylo také posouzení efektivity paralelní simulace. Jak je vidět z následujícího grafu počet předaných zpráv během testovací simulace roste úměrně s počtem simulátorů. To je v tomto modelovém případě způsobeno charakterem simulované úlohy a skutečností, že simulátory pracují navzájem asynchronně. V optimálním případě by počítaná posloupnost měla být souvislá, ale asynchronní zpracovaní úlohy způsobuje redundanci zápisů (generovaná posloupnost je např. 1-2-2-3-4-5-5). Toto chování ovšem není chybou protokolu ale jeho

vlastností, a jak je uvedeno v předchozím textu, řešení serializace čtení a zápisu je záležitostí modelované architektury a nikoliv protokolu.



10 Závěr

Seznámil jsem se problematikou paralelního programování pomocí zasílání zpráv a s jazykem ISAC pro popis architektury mikroprocesorů. Na základě získaných znalostí a s využitím současných síťových technologií jsem navrhl a implementovat upravenou variantu komunikačního protokolu MSI, která umožňuje sdílení zdrojů generických simulátorů procesorů a tím efektivní simulaci víceprocesorových systémů SoC.

Mimo implementace samotného protokolu pro podporu simulace SoC systémů jsem rozšířil některé stávající komponenty projektu tak, aby umožňovaly práci s obecným množstvím simulátorů. Byl automatizován proces instalace simulátorů a prezentační vrstva poskytuje ovládání jak jednoprocesorové, tak i víceprocesorové simulace. Implementované řešení jsem úspěšně otestoval nad simulací jednoduchých programů.

V rámci této práce jsem rozšíření pro možnost simulace SoC implementoval pouze do interpretovaného simulátoru. Jelikož projekt Lissom nabízí možnost vytvořit také kompilovanou verzi simulátoru, která je rychlejší, nabízí se tedy jako další logický krok rozšíření kompilovaného simulátoru. Navíc řešení použité v interpretovaném simulátoru lze celkem snadno převést do simulátoru kompilovaného, čímž se proces rozšiřování značně zjednodušuje.

Bohužel v době vzniku této práce nebyla v projektu ještě plně zahrnuta podpora pro modelování SoC systémů a neměl jsem tedy možnost vyzkoušet implementované řešení společně s ostatními nástroji. Kompilátor jazyka ISAC ani další nástroje včetně generátoru simulátorů, které jsou potřebné pro vytvoření funkčního simulátoru, neumožňovaly práci se sdílenými zdroji. I když jsem se při návrhu a následné implementaci snažil vytvořit řešení, které by vyžadovalo co nejmenší zásah do existujících komponent nebylo možné tento cíl vždy úplně splnit a je předmětem dalšího vývoje obohatit potřebné nástroje o schopnosti nezbytné pro modelování SoC.

Literatura

[1] RÁBOVÁ, Z.; ČEŠKA, M.; ZENDULKA, J.;PERINGER, P.; JANOUŠEK, V. Modelování a simulace: Brno 2005

[2] HRUŠKA, T. Návrh jazyka ISAC 0.0. Brno: Interní materiál FIT VUT Brno, 2004.

[3] DVOŘÁK, V. Architektura a programování paralelních systémů:VUTIUM Brno 2004

[4] KŘÍŽ, S. Diplomová práce – Generický instrukční simulátor, FIT VUT Brno 2006

[5] Wikipedia, Message passing: [cit 2007-05-12] Dostupný z WWW: http://en.wikipedia.org/wiki/Message_passing

[6] Open MPI Team: Open Source High Performance Computing [cit. 2007-05-17]. Dostupný z WWW: http://www.open-mpi.org/

[7] MPICH2 Introduction [cit. 2007-05-17]. Dostupný z WWW: http://www-unix.mcs.anl.gov/mpi/mpich/

Seznam příloh

Příloha 1. CD se zdrojovými kódy a elektronickou verzí této zprávy.