Vysoké učení technické v Brně
Fakulta informačních technologií

Ročníkový projekt

Souběžný návrh vestavěných systémů

autor: Jan Vlach, login: xvlach06
akademický rok: 2005/2006, semestr: letní

Květen 2006 Jan Vlach

Poděkování

Děkuji doc. Ing. Josefu Schwarzovi, CSc a Dr. Ing. Petru Peringerovi za pomoc při vypracovávání tohoto projektu.

Abstrakt

Tento ročníkový projekt se zabývá souběžným návrhem technického a programového vybavení počítačových systémů. Popisuje jak obecnou teorii, jak postupovat při takovém návrhu, tak i specifikaci systému v systému Esterel a praktickou realizaci v návrhovém prostředí Polis s využitím simulačního prostředí Ptolemy.

Klíčová slova

Souběžný návrh technického a programového vybavení (hardware/software codesign), alternativa technického/programového vybavení (hardware/software alternative), dělení technického a programového vybavení (hardware/software partition), virtuální stroj, Esterel, Polis, Ptolemy.

Obsah

1 Úvod

2 Pozadí hardwaru/softwaru

3 Zkoumání hardware/software codesignu

4 Koncepty codesignu

5 Metodologie pro codesign

6 Jednotná reprezentace pro hardware a software

7 Abstraktní hardware/software model

8 Specifikace návrhu vestavěných systémů

9 Souběžný návrh vestavěných systémů v návrhovém prostředí Polis

10 Návrh jednoduché aritmeticko-logické jednotky v prostředí Polis

11 Literatura

1 Úvod

Souběžný návrh počítačových systémů dává dohromady myšlenky a koncepty tří základních disciplín z oblasti návrhu systémů: modelování na úrovni systémů, návrh hardwaru a návrh softwaru.

Běžný postup při vývoji systémů po provedení analýzy požadavků systému probíhá ve dvou navzájem nezávislých větvích, které se navzájem ovlivňují velmi málo. První větev představuje vývoj hardwaru, druhá vývoj softwaru. Obě tyto větve se spojí až ke konci procesu návrhu při tzv. integraci systému (system integration), během níž je hardware a software dán dohromady a celý systém je testován jako celek. Nevýhodou tohoto postupu je, že chyby, na které se narazí až v této fázi, mohou vyžadovat dodatečnou změnu v hardwaru nebo softwaru (nebo v obojím), což má za následek zvyšování ceny a nedodržení plánu.

Proto byl vyvinut nový přístup k vývoji počítačových systémů, ve kterém je hardware navrhován společně se softwarem. Tento přístup se nazývá souběžný návrh hardwaru a softwaru (hardware/software codesign, dále jen codesign). Pomocí codesignu lze dosáhnout vyššího celkového výkonu systému, vyšší spolehlivosti. Další výhodou je možnost dříve odhalit možné problémy a tedy i změny vedoucí k jejich odstranění lze provést mnohem dříve. Díky tomu lze docílit nižších nákladů na vývoj systémů.

Codesign se zvláště hodí pro vývoj vestavěných systémů (embedded systems), což jsou systémy, které jsou navrženy pro nějakou konkrétní aplikaci.

Při hardware/software codesign je třeba řešit mnoho problémů. K nejtěžším z nich patří **hardware/software dělení** (hardware/software partitioning), které určuje, které funkce budou implementovány v hardwaru a které budou implementovány v softwaru.

2 Pozadí hardwaru/softwaru

Codesign se dá s výhodou použít při vývoji vestavěných systémů. Vestavěné systémy jsou aplikačně specifické systémy, které obsahují hardware a software navržený speciálně pro určitou úlohu. Obecně bývají součástí větších systémů. Vestavěné systémy patří do kategorie reaktivních systémů (reactive systems).

Vestavěné systémy se dnes hojně vyskytují v našem každodenním životě. Je možné je najít např. v mobilních telefonech, domácích spotřebičích, průmyslových robotech nebo v automobilech.

Typický počítačový systém může být popsán pomocí **hierarchie virtuálního stroje** (virtual machine hierarchy). Ta je tvořena pěti (někdy jen čtyřmi) úrovněmi (level 0 až level 4).

úroveň 4 .... aplikace (applications)
úroveň 3 .... operační systém (operating system)
úroveň 2 .... strojový jazyk (machine language)
úroveň 1 .... mikroprogramování (microprogramming)
úroveň 0 .... fyzická zařízení (physical devices)

Nejnižší úroveň (úroveň 0) je tvořena fyzickými zařízeními (physical devices). Úroveň 1 tvoří primitivní software, který bývá uložen v paměti ROM. Tento software přímo řídí fyzická zařízení. Je také označován jako firmware. Tato úroveň u mnoha počítačových systémů neexistuje, řídicí funkce jsou potom poskytovány hardwarem. Úroveň 2 sestává z programů zapsaných v assembleru. Úroveň 3 (operační systém) poskytuje uživateli množinu služeb vyšší úrovně (higher set of services). Příkladem je například poskytování abstrakce souboru, který může být otvírán a zavírán, a složitost detailních operací na disku je uživateli skryta. Nejvyšší úroveň - úroveň 5 - je tvořena aplikacemi, které jsou spouštěny na počítačovém systému.

Každá z uvedených úrovní, s výjimkou úrovně 0, představuje abstrakci virtuálního stroje. Virtuální stroj poskytuje množinu zařízení (facilities), kterými jsou operace (operations) a zdroje (resources), a skrývá před uživatelem implementační detaily. Pro každou úroveň je možné definovat architekturu, obsahující množinu datových typů, operací a dalších vlastností, které jsou uživateli na této úrovni viditelné.

3 Zkoumání hardware/software codesignu

Pro codesign je obzvlášť důležité **křížové oplodnění** (cross fertilization). Jde o aplikaci konceptů softwarového inženýrství na hardwarové inženýrství a naopak. Tato oblast se zabývá takovými pojmy jako podobnost a odlišnost mezi hardwarem a softwarem a kromě toho také sjednocením vývojových cest hardwaru a softwaru, což zahrnuje zkoumání jednotných reprezentací hardwaru a softwaru.

Doména hardwaru i doména softwaru mají mnoho společných konceptů. Mezi ně například patří moduly a znovupoužitelné prvky. Nejpatrnější rozdíl mezi hardwarem a softwarem je, že hardware má fyzikální charakteristiky, jako je zpoždění nebo spotřeba energie, zatímco u softwaru se nic takového nevyskytuje. Protože hardware má fyzikální omezení, mohlo by se zdát, že návrh hardwaru je jednodušší než návrh softwaru, jelikož množství voleb návrhu je omezeno. Ale právě fyzikální charakteristiky činí návrh hardwaru složitým. Časování, hluk, odvod tepla a elektromagnetické účinky komplikují návrh hardwaru a v mnoha případech brání jeho správné funkci.

Proces codesignu začíná funkčním popisem systému, který je nezávislý na hardwaru nebo softwaru. Tento popis může být realizován například konečnými automaty (FSM) nebo souběžnými procesy (concurrent processes). Systém může také být popsán pomocí programovacího jazyka, který je potom přeložen do vnitřní reprezentace (internal representation), kterou může představovat popis toku dat a popis toku řízení (data/control flow description). Takové popisy systému mohou sloužit jako **jednotná reprezentace** (unified representation), která může reprezentovat software nebo hardware.

Jednotná reprezentace vstupuje do procesu hardware/software dělení, ve kterém se rozhoduje, které funkce budou implementovány v hardwaru a které v softwaru. Hardware/software dělení může být provedeno ručně nebo automaticky. Jeho výsledkem je **hardware/software rozdělení** (hardware/software partition), které specifikuje funkce, které mají být implementovány v hardware, a funkce, které mají být implementovány v softwaru. Z tohoto popisu je generován hardware i software systému a také všechna rozhraní, která jsou potřeba pro komunikaci. Výsledky syntézy hardwaru (hardware synthesis), syntézy softwaru (software synthesis) a syntézy rozhraní (interface synthesis) jsou vstupem procesu integrace systému (system integration), který z nich sestaví výsledný systém. Výhodnost určitého hardware/software rozdělení může být vyhodnocena buď ihned po jeho vytvoření nebo po provedení integrace systému. Pokud je výsledek vyhodnocení neuspokojivý, je vytvořeno nové hardware/software rozdělení. Tyto kroky se opakují, dokud není vyvinuta implementace, která splňuje požadavky na systém.

Největším rozdílem mezi procesem codesignu a běžným procesem návrhu systémů je použití jednotné reprezentace pro popis hardwaru a softwaru a schopnost provádět hardware/software dělení iterativně.

Jednotné reprezentace pro hardware nebo software poskytují grafově založené reprezentace, vnitřní reprezentace v překladačích konečné automaty a souběžné procesy. Mezi grafově založené reprezentace patří Petriho sítě nebo grafy toku dat. Například uzly v grafech toku dat mohou reprezentovat operace v softwaru nebo funkční jednotky v hardwaru. Grafově založené reprezentace mají přirozenou schopnost modelovat tok dat/řízení a souběžnost. Vnitřní reprezentace v překladačích, jako jsou abstraktní syntaktické stromy a orientované acyklické grafy, jsou použity pro podporu automatické syntézy.

Existují dva běžné přístupy k hardware/software dělení: hardwarově orientovaný a softwarově orientovaný. V hardwarově orientovaném přístupu je vybráno počáteční hardware/software rozdělení, ve kterém je převážná část funkcí určena pro implementaci v hardwaru. Funkce implementované v hardwaru jsou postupně přesouvány do softwaru. Tento proces pokračuje, dokud nejsou narušena výkonnostní omezení. Softwarově orientovaný přístup začíná s takovým hardware/software rozdělením, ve kterém jsou všechny funkce určeny pro implementaci v softwaru a postupně se části kritické z hlediska výkonu přesouvají do hardwaru. V těchto obou přístupech je často aplikováno pravidlo 90/10, které říká, že 90 procent času běhu programu představuje běh 10 procent části kódu. Například často se opakující smyčky (loops) jsou často vhodnými kandidáty pro přesun do hardwaru.

4 Koncepty codesignu

Funkční dekompozice je proces vývoje implementace pro funkci, vyjádřený jako algoritmus, který kombinuje množinu jednodušších funkcí. Jde o zjemňovací techniku, která používá funkční abstrakce pro zvládnutí složitosti ve strukturovaných přístupech shora dolů (top-down). Funkční dekompozice vytváří stromovou strukturu, ve které každý uzel odpovídá funkci.

Dekompozice D funkce f v úrovni dekompozice p je reprezentována jako $D_p = \{U, V\}$, kde U je množina funkcí, které mají být dekomponovány, a V je množina listových funkcí, které již dále dekomponovány nebudou. Před zahájením procesu dekompozice obsahuje množina U funkci f a množina V je prázdná. Aby bylo možné provést dekompozici, jsou definovány dvě operace - expanze (expansion) a selekce (selection). Při expanzi je nějaká funkce f ve stupni p nahrazena algoritmem, který je tvořen w funkcemi. Tyto funkce se nacházejí v úrovni o 1 nižší než ve které se vyskytuje expandovaná funkce, tedy v úrovni p+1. Selekce vybírá funkce, které nebudou dále dekomponovány. Tyto funkce se přidají do množiny V a zároveň se tyto funkce musí odstranit z množiny U. Proces dekompozice končí v okamžiku, kdy je U prázdná množina. V této chvíli množina V obsahuje listové funkce pro daný dekompoziční strom, $V = \{f_1, f_2, ... , f_y\}$.

Po provedení funkční dekompozice může být funkce f vyjádřena jako orientovaný graf $\gamma = (\gamma_N, \gamma_E)$, kde $\gamma_N = \{n_1, n_2, ... , n_z\}$, n_j je prvkem V pro j = 1 .... z je množina uzlů a γ_E množina hran. a je podmnožinou kartézského součinu γ_N x γ_N. Každý uzel n_j z množiny γ_N odpovídá jedné listové funkci v množině V. Hrana e_{ij} z množiny γ_E reprezentuje tok dat a/nebo tok řízení z uzlu n_i do uzlu n_j. Graf γ může být použit pro hardware/software dělení.

Virtuální stroj je abstraktní výpočetní prvek, který má virtuální sadu instrukcí, která definuje operace podporované tímto výpočetním prvkem. Je to abstrakce, která poskytuje množinu zařízení, jako jsou operace a zdroje, která může být použita pro definování zařízení abstraktnějšího virtuálního stroje vyšší úrovně. Virtuální stroj má charakteristiky programovatelného počítače a koncept virtuální sady instrukcí je podobný tradiční představě instrukční sady. Virtuální stroj tedy může provádět program specifikovaný posloupností virtuálních instrukcí.

Hardware/software dělení je proces, který určí, které funkce mají být implementovány v hardwaru a které v softwaru. Nejjednodušeji může být reprezentováno jako binární rozhodovací proces, kdy se rozhodujeme například na základě požadovaného výkonu, pružnosti nebo faktoru tvaru výsledné implementace. Hardware/software dělení odráží rozhodnutí návrhu, ale neříká nic o tom, jak budou funkce implementovány. Pokud se např. rozhodneme implementovat funkci v softwaru, neplyne z toho, jaký konkrétní algoritmus se nakonec použije ani jaký procesor bude použit pro běh tohoto algoritmu. Přesto takové rozhodnutí ve fázi návrhu zavádí jistá omezení pro konečnou hardware/software implementaci. Určité rozhodnutí učiněné při hardware/software dělení se nazývá **vazba** (binding). Vazba může být buď časná (early) nebo zpožděná (late), což označuje, v jakém stupni procesu návrhu byla vazba vytvořena. Časná vazba umožňuje lepší podporu pro plánování procesu vývoje hardware/software. Obecně je obtížné učinit rozhodnutí o hardware/software dělení v raných stádiích procesu návrhu, neboť je k dispozici malé množství dostupných informací. Pokud se použije časná vazba, je návrhové rozhodnutí o hardware/software dělení provedeno okamžitě. V případě zpožděné vazby jsou funkce nejprve dekomponovány na úroveň dekompozice, která je dána stupněm "důvěry" (degree of "confidence"), a až poté jsou provedena rozhodnutí návrhu o hardware/software dělení.

Hardware/software rozdělení je popis, který specifikuje funkce, které mají být implementovány v hardwaru a funkce, které mají být implementovány v softwaru. Je to výsledek procesu hardware/software dělení.

Hardware/software alternativa je jedna z možných hardware/software implementací funkce f. Hardware/software alternativa A je tvořena množinou softwarových jednotek S, množinou hardwarových jednotek H a množinou jejich komunikací C, tedy A = (S, H, C). Softwarová jednotka reprezentuje dvojici software/procesor, někdy obsahuje i plánovač (scheduler). Důležitou součástí softwarové jednotky je programový interpreter. Softwarová jednotka také zahrnuje zařízení pro speciální účely (special purpose devices), jako jsou např. floating point koprocesory, které jsou použity jako část fetch/execute cyklu interpreteru. Hardwarová jednotka je zařízení pro speciální účely, které není součástí fetch/execute cyklu. Tato zařízení obecně přijímají data a produkují data.

Hardware/software kompromis (hardware/software trade-off) je rozhodnutí týkající se alokace funkcí do hardwaru a softwaru, které se pokouší splnit množinu kritérií (cílů), z nichž mnoho je ve vzájemném konfliktu, a nejsou tedy dosažitelná ve stejném okamžiku. Hardware/software kompromisy se tedy pokoušejí splnit určité požadavky a omezení. Některé typické kompromisy zahrnují rozhodnutí týkající se implementace funkce v hardwaru nebo softwaru a také rozhodnutí týkající se migrace softwarových funkcí do hardwaru a naopak.

Codesign je spolupracující vývoj softwarových a hardwarových částí systému, při kterém se klade důraz na interakci mezi návrhem softwaru a návrhem hardwaru. Návrhové interakce mezi hardwarem a softwarem se vyskytují v několika ohledech. Jedním z nich je proces provádění hardware/software kompromisů. Obecně může být provedeno mnoho hardware/software kompromisů. Zkoumání hardware/software kompromisů je proces vytváření a ohodnocování hardware/software alternativ a výběr implementace pro určitou funkci na základě nějakého kritéria. Tyto alternativy jsou ohodnocovány s ohledem na různé metriky.

Vyhodnocení alternativy vyžaduje dva kroky. První krok se zabývá porovnáváním jednotlivých alternativ. Druhý krok se týká odhadu dopadu alternativy na celkové chování systému. Pro provedení prvního kroku byl vyvinut **model kvantitativního ohodnocení** (quantitative evaluation model), s jehož pomocí se porovnávají alternativy na základě různých metrik. Používá se zde množina vah (weights), z nichž každá odpovídá určité metrice. Váha je reprezentována číslem od 0 do 1, které určuje důležitost odpovídající metriky. Model kvantitativního ohodnocení tedy určuje kvalitu alternativy, která je použita pro porovnání s jinými alternativami. Pro různé funkce systému mohou být důležité různé metriky. Např. jedna funkce systému může vyžadovat implementaci s minimální dobou provedení, zatímco jiná funkce může vyžadovat vysokou spolehlivost. Váhy mohou zvýšit důležitost určitých metrik, což má vliv na kvalitu alternativ a tedy i na jejich výběr pro implementaci. Model kvantitativního ohodnocení může být vyjádřen jako pětice $Q = (\rho(A), \lambda, G, W, \sigma(A, G, W))$, kde A je množina alternativ, G je množina jakostních hodnot pro různé metriky a W je množina vah. Model počítá množinu hodnot kvality κ, která obsahuje hodnotu kvality pro každou alternativu z množiny A. Na model kvantitativního ohodnocení je tedy možné se dívat jako na funkci, která alternativě z množiny A přiřazuje hodnotu kvality. První komponentou modelu je $\rho(A)$, což je množina metrických funkcí (metric functions). Pro každou metriku, která nás zajímá, je zde jedna metrická funkce. Tyto metrické funkce přijímají jako parametr množinu alternativ A. Každá metrická funkce mapuje alternativy na kvantitu m, která reprezentuje množinu metrických hodnot. Pro každou uvažovanou alternativu se zde vyskytuje jedna metrická hodnota, tj. pro $A = \{A_1, A_2, \ldots, A_d\}$ platí, že $m = \rho(A) = \{mval_1, mval_2, \ldots, mval_d\}$. Druhou komponentou modelu kvantitativního ohodnocení je množina jakostních funkcí (goodness functions) λ, která obsahuje pro každou metriku jednu jakostní funkci. Každá jakostní funkce z množiny λ přijímá jako svůj vstup kvantitu m a jejím výstupem je kvantita g, což je množina hodnot z intervalu od 0 do 1, která obsahuje jednu hodnotu pro každou alternativu z množiny A, tedy $g = \{gval_1, gval_2, \ldots, gval_d\}$, kde $gval_i$ se nazývají jakostní hodnoty. Každá hodnota v kvantitě g určuje "výhodnost" odpovídající alternativy z hlediska metriky, kterou určuje použitá jakostní funkce. V tomto okamžiku se jakostní hodnoty pro všechny alternativy setřídí podle použité jakostní funkce, tedy podle různých metrik, a tyto skupiny se umístí do množiny G. Pokud například bude existovat 5 alternativ a 2 metriky (doba výpočtu (T), cena (C)), měla by množina G obsahovat 2 prvky g_T a g_C, z nichž každý by měl obsahovat 5 jakostních hodnot (jednu pro každou alternativu). Váhy W určují důležitost jednotlivých metrik. Součet jednotlivých vah z množiny W je 1.

Nakonec ohodnocovací funkce (evaluation function) σ, jejímž vstupem jsou množiny A, G a W, vypočítá množinu hodnot kvality κ pro jednotlivé alternativy z množiny A, $\kappa = \sigma(A, G, W)$. Model kvantitativního ohodnocení může být použit pro ohodnocení různých alternativ na různých úrovních detailu.

5 Metodologie pro codesign

Rozlišující charakteristikou mezi různými přístupy k návrhu hardware/software je **množství unifikace** (amount of unification), která se vyskytuje mezi procesem návrhu hardwaru a procesem návrhu softwaru. Této unifikace je dosaženo použitím společných reprezentací. Jednotné reprezentace (unified representations) hrají sjednocující roli při podpoře návrhových interakcí mezi doménou hardwaru a doménou softwaru. Stupeň, do kterého jsou používány jednotné reprezentace, určuje množství unifikace.

Metodologie pro návrh systémů podporující codesign je tvořena několika po sobě jdoucími fázemi. První z nich je reprezentace systému (system representation), další je dělení systému (system partitioning), následující fází je codesign, který můžeme dále rozdělit na několik podkroků (dekompozice, hardware/software partitioning, zjemnění (refinement) a ohodnocení alternativy (alternative evaluation)). Další fází je syntéza (synthesis) a poslední fáze je integrace systému (system integration).

Reprezentace systému

Systém ψ je popsán orientovaným grafem, který se skládá z jedné nebo více funkcí systému, které mají být provedeny, nezávislých na hardwaru nebo softwaru. Funkční popis systému $\psi^F = (N, E)$, kde $N = \{f_1, f_2, \ldots, f_v\}$ a E je podmnožinou nebo rovno N x N, $e_{ij} = (f_i, f_j)$. Hrany e_{ij} grafu reprezentují tok dat. Datový tok vstupující do uzlu představuje operandy, které mají být zpracovány funkcí, která je přiřazena tomuto uzlu, a datový tok vystupující z uzlu odpovídá výsledku generovanému touto funkcí. Funkční prvky (uzly) pracují, pokud jsou data přítomna na jejich vstupech. Na uzly je možné pohlížet také jako na množinu virtuálních instrukcí.

Analýza výkonu a spolehlivosti může být použita pro stanovení omezení pro pozdější fáze návrhu. Omezení systému mohou být např. vyjádřena propustností systému, dobou provedení nebo spolehlivostí. Hardwarová omezení, jako jsou plocha a spotřeba energie, mohou být použita, aby pomohla řídit pozdější návrhová rozhodnutí. Softwarová omezení mohou udávat složitost nebo počet řádků kódu. Navíc by měly být použity funkční specifikace, které by zjemňovaly funkce systému do detailnějších popisů. Tato omezení a specifikace mohou být zachyceny v **specifikační šabloně** (specification template) pro každou funkci systému a pro systém jako celek. Specifikační šablona může být chápána jako "atribut" uzlu v reprezentaci systému.

Dělení systému

V této fázi jsou funkce systému mapovány na oddělené fyzické jednotky jako jsou čipy nebo desky (boards). Výsledkem dělení systému je vytvoření vzájemně disjunktních bloků Bj systému. Sjednocení všech těchto bloků systému tvoří množinu funkcí N a průnik všech bloků systému je prázdná množina. Popisem systému v této fázi je $\psi^B = (B, E)$, kde $B = \{B_1, B_2, ... , B_t\}$, B_j je prvkem N. Kritériem, podle kterého se provede dělení systému - přiřazení funkcí do bloků systému - může být např. minimalizace komunikace mezi funkcemi systému. Tento krok metodologie odpovídá **funkčnímu dělení systému**. Při dělení systému nejde o hardware/software dělení.

Codesign

Codesign se skládá z iterativního provádění dekompozice, hardware/software dělení, zjemnění (refinement) a ohodnocení hardware/software alternativy pro každou funkci systému. Na závěr tohoto procesu je odvozena implementace I systému, která je tvořena implementacemi pro každou funkci systému (pro každou funkci jedna implementace), $I = \{I_1, I_2, ... , I_v\}$. Systém je potom možné popsat jako $\psi^I = (I, E)$.

Dekompozice znamená odvození virtuální sady instrukcí pro každou funkci systému a vyjádření každé funkce jako graf γ. Hardware/software dělení určuje, které z virtuálních instrukcí určených grafem γ mají být implementovány v hardwaru a které v softwaru. Zjemnění vytváří abstraktní hardware/software model, který reprezentuje alternativu návrhu na nějaké úrovni detailu. Ohodnocení alternativy používá model kvantitativního ohodnocení pro odhadnutí výhodnosti jednotlivých alternativ.

Syntéza a integrace systému

Během syntézy jsou z popisů získaných ve fázi codesignu generovány hardwarové realizace. Pro generování softwarového systému jsou použity techniky syntézy softwaru. Integrace systému vyžaduje běh syntetizovaného softwaru na syntetizovaném hardwaru. V tomto okamžiku je systém ohodnocen jako celek.

6 Jednotná reprezentace pro hardware a software

Abychom bylo možné se detailněji zabývat jednotnou reprezentací, je nejprve nutné uvést některé důležité koncepty modelování (modeling concepts). Jde o definici modelu, abstrakce (abstraction) a interpretace (interpretation) a také sem patří diskuse o rozdílu mezi abstrakcí a interpretací.

Model je popis entity na nějaké úrovni detailu, který zachycuje důležité vlastnosti, které jsou předmětem našeho zájmu. Modely jsou důležité pro zvládnutí složitosti. V některých případech nemusí být daná úroveň detailu dostatečná na to, aby bylo možné zkoumat vlastnosti, které nás zajímají, a tak mohou být v modelu vyžadovány další detaily. Obecně jsou modely aproximací modelované entity. Je důležité model ohodnotit, ale ne tak, že bychom o něm prohlásili, že je buď správný nebo špatný, ale zda je dostatečný na to, abychom pomocí něj odhalili určité, zatím pro nás neznámé, vlastnosti zkoumané entity.

Abstrakce je proces ignorování (skrývání) nepodstatných detailů a zaměření se na důležité vlastnosti, které nás zajímají. Abstrakcí také nazýváme popis, ke kterému dojdeme tímto procesem.

Úroveň abstrakce je množství informací, které jsou jak skryté, tak vystavené v popisu (vlastnosti, které se při vytváření popisu ignorovaly, zde nejsou).

Model je abstrakcí entity. Modely existují na různých úrovních abstrakce l^a. Model M je funkcí l^a, píšeme $M(l^a)$. Na nižší úroveň abstrakce se dostaneme přidáním detailů do modelu. Při vývoji modelu mohou být použity funkční nebo datové abstrakce. Existují dvě metody abstrakce. Jde o abstrakci pomocí parametrizace (abstraction by parametrization) a abstrakci pomocí specifikace (abstraction by specification). Obě metody lze aplikovat na funkční i na datové abstrakce. Abstrakce pomocí parametrizace umožňuje, aby byla reprezentována množina výpočtů. Termín výpočet je zde použit v obecném smyslu. Například v softwaru může jít o vyhledávací funkci, která má jako parametry pole a prvek, který se má hledat, a tak je umožněno popsat několik různých hledání. V hardwaru může být model aritmeticko-logické jednotky (ALU) parametrizován vstupními operandy a operací, která má být provedena, což umožňuje aby mohlo být reprezentováno několik operací s různými operandy. Abstrakce pomocí specifikace umožňuje, aby byla ignorována implementace a podporuje skrývání informací. Například v případě výše zmíněné vyhledávací funkce nás může zajímat pouze vstupně-výstupní chování a ne algoritmus, který je použit k provedení této funkce.

Interpretace je proces přiřazování sémantiky, zejména hodnot a funkčních informací, různým prvkům modelu.

Úroveň interpretace je rozsah přiřazení sémantiky prvkům modelu. Extrémní úrovně se nazývají neinterpretovaná a interpretovaná. Neinterpretované modely, jako jsou např. Petriho sítě nebo frontové modely, se používají pro analýzu výkonu a spolehlivosti na úrovni systému. Interpretované modely se používají pro detailní návrh a obsahují funkční informace. Hybridní model se skládá jak z neinterpretovaných, tak z interpretovaných modelů.

Jednotná reprezentace je reprezentace, která může být použita pro popis hardwaru nebo softwaru. Při vývoji jednotné reprezentace mohou být použity funkční i datové abstrakce.

Dekompoziční graf (decomposition graph) DG je hierarchický, orientovaný graf, který se skládá z uzlů N a hran E, DG = (N, E). Uzly mohou odpovídat buď funkčním abstrakcím nebo datovým abstrakcím. Hrany reprezentují vztah "sestává z" mezi uzly, což odráží skutečnost, že uzel může být vyjádřen pomocí primitivnějších uzlů. Dekompoziční graf, který používá funkční abstrakce, zobrazuje funkční dekompozici a dekompoziční graf, vyjádřený pomocí datových abstrakcí, popisuje datovou dekompozici, tj. dekompozici založenou na abstraktních datových typech. Listové uzly $L = \{n_j \mid (n_i, n_j)$ je prvkem E a (n_j, n_k) není prvkem E pro $k = 1, 2, \ldots, |N|\}$ jsou použity k definování virtuálního (abstraktního) stroje a odpovídající virtuální sady instrukcí. Při datové dekompozici reprezentují listové uzly nejprimitivnější abstraktní datové typy v grafu a operace, které se mají provádět nad těmito datovými typy, tvoří virtuální sadu instrukcí. Virtuální sada instrukcí tedy může být odvozena jak pomocí funkční dekompozice, tak pomocí datové dekompozice.

Dekompoziční graf je jednotnou reprezentací v několika ohledech. Za prvé, uzly mohou reprezentovat buď funkční abstrakce nebo datové abstrakce. Za druhé, graf slouží jako reprezentace, která může být použita jak pro software, tak pro hardware.

7 Abstraktní hardware/software model

Abstraktní hardware/software model používá jednotnou reprezentaci.

Na abstraktní hardware/software model jsou kladeny určité požadavky. Musí být proveditelný (executable), nezávislý na jakémkoli konkrétním jazyku (language-independent) a integrovaný (integrated). Jazyková nezávislost eliminuje jakékoliv vlivy jazyků používaných pro vývoj hardwaru a softwaru. Integrovaným modelem rozumíme model, ve kterém jsou popisy hardwaru a softwaru systému začleněny do společného simulačního prostředí. Abstraktní hardware/software model může být použit pro obecné ohodnocení výkonu (performance evaluation), nalezení úzkých hrdel v návrhu softwaru (identification of software bottlenecks), ohodnocení hardware/software kompromisů (evaluation of hardware/software trade-offs) a ohodnocení alternativ návrhu (evaluation of design alternatives).

Interpretační systémy (interpretive systems)

Stroj (machine) je tvořen třemi komponentami: **sadou instrukcí** (instruction set), **pamětí** (storage) a **interpretačním mechanismem** (interpretive mechanism). Sada instrukcí definuje množinu primitivních operací podporovaných strojem. Paměť se skládá z adresovatelných míst, které obsahují hodnoty. **Stav** (state) je dán určitou konfigurací paměti. Interpretační mechanismus provádí program, který se skládá z **instrukcí** (instructions) nebo **příkazů** (commands), a tím způsobuje změny stavu stroje. Interpretační mechanismus může být sám strojem, potom se mu říká **hostitelský stroj** (host machine) nebo také **základní stroj** (base machine). Vytváří se tak hierarchie strojů.

Stroj vyšší úrovně se nazývá **obrazový stroj** (image machine) (také **cílový stroj** (target machine) nebo **virtuální stroj** (virtual machine)). Program, který je prováděn na hostitelském stroji, se nazývá **program interpretu** (interpreter program) nebo **emulátor** (emulator). Taková strojová hierarchie představuje jednoúrovňový interpret. Avšak obrazový stroj může také být hostitelským strojem pro obrazový stroj vyšší úrovně. Potom jde o dvouúrovňový interpret. Tuto myšlenku můžeme rozšířit na N úrovní. Důležitou vlastností interpretu je jeho fetch/execute chování, bez ohledu na to, zda je interpret hardwarový prvek, mikroprogram nebo program na úrovni stroje.

V Hoffmanově klasifikaci interpretačních systémů se rozlišují dva typy interpretů: **interpret příkazů** (command interpreter) a **interpret programu** (program interpreter). V interpretu příkazů, příkazy, což jsou instrukce, které mají být interpretovány, nejsou uloženy v paměti programu. V interpretu programu jsou instrukce, které mají být interpretovány, částí uloženého programu.

Abstraktní hardware/software model HSM se skládá z softwarového modelu SM a hardwarového modelu HM, z nichž každý může být popsán za použití jednotné reprezentace. Každý model může být vyjádřen jako dekompoziční graf, který používá funkční abstrakce. $HSM(l^s, l^h) = (SM(l^s), HM(l^h))$, kde l^s a l^h jsou dvojice (l^a, l^i) (úroveň abstrakce a úroveň interpretace) pro software (l^s) a pro hardware (l^h). Aby bylo možné realizovat HSM, musí být úrovně abstrakce softwarového modelu a hardwarového modelu konzistentní, tj. každé primitivní operaci softwarového modelu musí "rozumět" hardwarový model. Softwarový model a hardwarový model mohou mít různé úrovně interpretace. Pokud hardwarový model neobsahuje časové informace (temporal information), není jej možné použít pro analýzu výkonu. Aby bylo možné použít abstraktní hardware/software model i pro funkční verifikaci (functional verification), musí být softwarový model interpretovaný.

8 Specifikace návrhu vestavěných systémů

Vestavěné systémy patří většinou mezi reaktivní systémy, tj. systémy, které reagují na vstupní signály výstupními signály a navíc na stejnou kombinaci vstupních signálů reagují vždy stejnou kombinací výstupních signálů. Mezi jazyky, které lze použít pro specifikaci reaktivních systémů, patří například Esterel, Lustre nebo Signal. Následující text se zaměří na jazyk Esterel. Esterel je synchronní jazyk. To znamená, že výpočet trvá nulovou dobu. Není tím myšlen skutečný čas, ale modelovací čas. Čas plyne pouze tehdy, když tok řízení dosáhne příkazu halt nebo await.

Některé konstrukty jazyka Esterel:

Programy v jazyce Esterel se skládá z kolekce modulů. Tyto moduly popisují pouze určitou funkčnost a mohou být později implementovány jak v hardwaru, tak v softwaru. Zejména zde jde o definování rozhraní modulu (jeho vstupy a výstupy) a o vnitřní stav modulu (může být reprezentován jeho lokálními proměnnými). Vstupy a výstupy modulu mohou být buď čisté signály (pure signals) nebo ohodnocené signály (valued signals). Čisté signály mohou nést pouze informaci o tom, zda jsou aktivní či nikoliv (tj. present nebo absent), zatímco ohodnocené signály navíc nesou i určitou hodnotu, kterou může být např. číslo typu integer. Příkladem definice modulu, který má definované pouze vstupy A, B a výstup O je:

```
module Example:
input A, B;
output O
end module
```

Vysílání výstupního signálu O s hodnotou exp je, za předpokladu, že O je ohodnocený signál (tj. signál, který může hodnotu), možné provést příkazem emit O(exp). Pro získání hodnoty určitého signálu je třeba použít operátor ?, např. ?O.

Pokud chceme například inkrementovat hodnotu vysílaného signálu , je třeba si uvědomit, že v určitém okamžiku může určitý signál nést pouze jednu hodnotu. Nelze tedy napsat emit O(?O + 1), ale je třeba místo toho napsat emit O(pre(?O) + 1). Zápis pre(?O) vrací hodnotu signálu O v předchozím okamžiku.

Příkaz loop .... end představuje nekonečnou smyčku, která provádí své tělo (příkazy mezi klíčovými slovy loop a end) a jakmile se řízení programu dostane na konec smyčky (ke klíčovému slovu end), je celá smyčka restartována a její tělo se provádí opět od začátku.

Příkaz loop p .... each R má podobnou sémantiku. Když smyčka začne, začne se ve stejném okamžiku provádět její tělo p. V okamžiku, kdy se vyskytne signál R, provádění těla se přeruší a smyčka se začne provádět znovu od začátku. Jestliže provádění těla smyčky skončí dříve než se objeví signál R, potom se čeká na výskyt signálu R a tělo smyčky se opět začne provádět od začátku.

Příkaz await A čeká na výskyt signálu A. Od doby spuštění příkazu await A do doby výskytu signálu A je příkaz await A aktivní. Příkaz await A není okamžitý. Jestliže je spuštěn v okamžiku t0, skončí v okamžiku t1, kde t1 > t0.

Klíčové slovo var se používá k deklaraci proměnné. Jméno proměnné se uvádí za klíčovým slovem var. Za jménem proměnné následuje přiřazení počáteční hodnoty a potom dvojtečka a jméno typu proměnné a nakonec klíčové slovo in. Proměnná platí v bloku programu od deklarace proměnné po klíčová slova end var. Příklad deklarace proměnné typu integer s počáteční hodnotou 4:

var Promenna := 4 : integer in

..... příkazy .....

end var

Příkaz every p do S provede příkaz p pokaždé, když je přítomen signál S. V prvním okamžiku provádění příkazu every p do S se však na signál S nijak nereaguje. Reaguje se na něj počínaje následujícím okamžikem. Pokud se však za klíčové slovo every přidá klíčové slovo immeditate, potom toto omezení neplatí.

V případě, že chceme zabránit, aby v jednom okamžiku přešlo současně několik vstupních signálů ze stavu absent do stavu present, je třeba v deklarační části modulu zapsat příkaz relation S1 # S2 # .... # Sn, kde Si jsou vstupní signály modulu.

Kromě signálů a proměnných se v jazyce Esterel vyskytují ještě **senzory**. Senzory jsou takovým typem signálů, které nesou pouze hodnotu, ale ne informaci o své přítomnosti. Jejich stav je vždy považován za přítomný (present). Senzory jsou výstupem fyzických senzorů, jako je například teploměr. Senzory se vyskytují pouze na vstupech modulů.

Překladač jazyka Esterel:

Zdrojový program napsaný v jazyce Esterel je třeba uložit do souboru s příponou strl. Překladač jazyka Esterel - spustitelný soubor esterel - překládá zdrojový soubor Esterelu do jazyka C, pokud je jeho jediným parametrem jméno zdrojového souboru. Překlad z jazyka Esterel do jazyka C však neprobíhá přímo, ale v několika etapách. Každá z těchto etap, kromě poslední, dá na svůj výstup intermediární kód překladu (intermediate Esterel code). Poslední etapa přeloží poslední z posloupnosti intermediárních kódů do jazyka C. Pro každý z těchto intermediárních kódů poskytuje distribuce Esterelu překladač, který provede překlad pouze mezi sousedními intermediárními kódy v posloupnosti intermediárních kódů. Pokud potřebujeme získat některý z intermediárních kódů místo programu v jazyce C, není nutné volat jednotlivé dílčí překladače, ale je možné požadovaný intermediární kód určit pomocí parametru, se kterým spustíme hlavní překladač esterel. Pokud potřebuje získat například kód, který představuje automat, tzv. automatový kód (automaton code) pro zdrojový soubor zdroj.strl, zadáme do příkazové řádky příkaz esterel -oc zdroj.strl. Získáme tak soubor zdroj.oc, který obsahuje stavů/přechodů automatu, který modeluje modul v souboru zdroj.strl.

9 Souběžný návrh vestavěných systémů v návrhovém prostředí Polis

Počáteční specifikace vestavěného systému se zapíše v jazyce Esterel. Soubor s touto specifikací je třeba přeložit do intermediárního automatového kódu, který bude obsažen v souboru *.oc. Vstupem programu POLIS jsou soubory *.shift obsahující specifikaci vestavěného systému ve formátu SHIFT (Software Hardware Interface formaT). K překladu souboru *.oc na soubor *.shift slouží program oc2shift, který je součástí distribuce Polisu. Je také možné použít program strl2shift, který volá jak překladač Esterelu, tak nástroj oc2shift. Soubor ve formátu SHIFT specifikuje množinu interagujících rozšířených konečných stavových strojů (CFSM, Codesign Finite State Machines). Pro každý specifikovaný modul se zde nachází jeden CFSM. Jakmile je soubor *.shift vytvořen, spustíme program Polis. Objeví se příkazový řádek programu Polis, ve kterém zadáme příkaz read_shift jmeno.strl, kde jmeno představuje jméno načítaného souboru. Nyní je náš návrh načten do prostředí Polis, v kterém je třeba vytvořit vnitřní grafovou reprezentaci, tento graf zoptimalizovat a vybrat cílový mikroprocesor, který bude použit pro běh softwaru ve výsledné implementaci. Potom spustíme odhadové nástroje (estimation tools), díky kterým získáme představu o velikosti a rychlosti výsledného softwaru. Následuje generování kódu (*.c) v jazyce C pro každý softwarový modul a generování Ptolemy kódu (*.pl) pro každý modul. Poté ukončíme běh Polisu.

Ptolemy je prostředí pro simulaci a syntézu. Systém Polis používá prostředí Ptolemy pro kosimulaci (co-simulation) rozšířených konečných stavových strojů. V systému Ptolemy je možné provádět jak funkční, tak vysokoúrovňovou časovou simulaci. Během vysokoúrovňové časové simulace může návrhář vybírat dynamicky (tj. bez nutnosti znovu kompilovat simulační modely) a hierarchicky (tj. s výchozím šířením na nižší hierarchické úrovně):

- hardwarovou nebo softwarovou implementaci každého CFSM
- typ a kmitočet zdroje, na kterém dané CFSM běží
- typ plánovače a prioritu každého CFSM

Soubor *.pl vygenerovaný systémem Polis popisuje:

a) vstupy, výstupy a senzory
b) stavové proměnné
c) konstanty, které mohou být měněny během simulace (tj. ty, které nejsou definovány pomocí #define)
d) rozhraní mezi simulátorem Ptolemy a kódem v jazyce C generovaným systémem Polis

Soubor *.c vygenerovaný systémem Polis popisuje chování CFSM a hodinový cyklus, který bude vyžadován při výběru procesoru, na kterém se bude provádět daný kód.

Rozdíl mezi simulací a vlastním prováděním je, že při simulaci jsou vstupy a výstupy ovládány systémem Ptolemy, zatímco v případě provádění kódu operačním systémem v reálném čase (Real-Time Operating System). Během simulace komponenty, která byla vybraná pro implementaci v softwaru, vždy když je proveden segment kódu, je čítač akumulující hodinové cykly inkrementován o hodnotu odhadnutého počtu cyklů vyžadovaných pro provedení tohoto kódu na aktuálně vybraném procesoru.

V systému Ptolemy se zkontroluje vzájemné propojení jednotlivých CFSM, nastaví se parametry, spustí se simulace, abychom ověřili, zda se návrh chová tak, jak se očekává a případně se změní hardware/software rozdělení, aby se zvýšil výkon. Když je návrh verifikován touto smíšenou hardware/software simulací, spustíme opět systém Polis, abychom vytvořili reálnou implementaci návrhu.

V prostředí systému Polis načteme návrh. Zde bude třeba načíst také pomocný (auxiliary) soubor *.aux, ve kterém jsou uloženy informace o vzájemném propojení jednotlivých modulů. Potom označíme každý modul buď jako hardware nebo jako software.

Pro hardwarovou část:

a) přeložíme vnitřní formát do hardwarového formátu
b) optimalizujeme tuto reprezentaci
c) zapíšeme hardware do formátu netlistu

Pro softwarovou část:

a) vytvoříme vnitřní graf reprezentující software
b) optimalizujeme tento graf
c) vybereme cílový mikroprocesor (ten, který bude použit pro běh softwaru ve výsledné implementaci)
d) spustíme odhadovací nástroje (abychom získali představu o velikosti a rychlosti výsledného softwaru na vybraném mikroprocesoru)
e) vygenerujeme kód v jazyce C pro každý softwarový modul
f) vygenerujeme operační systém (také v jazyce C)

Ukončíme systém Polis. Zbývá provést několik posledních kroků:

1) Vybereme hardware (například Xilinx FPGA).
2) Přeložíme soubory ve formátu netlistu do hardwaru (například za použití softwarových nástrojů Xilinx namapujeme vygenerovaný netlist na netlist vhodný pro danou architekturu a potom jej nahrajeme do FPGA).
3) Zapojíme FPGA do desky.
4) Použijeme softwarové a hardwarové nástroje dodané s mikroprocesorem (nebo spíše s vestavěným kontrolérem, který obsahuje mikroprocesor a paměť) k uložení modulů vygenerovaných v jazyce C a operačního systému vygenerovaného v jazyce C na mikrokontrolér.
5) Zapojíme tento mikrokontrolér do desky.

10 Návrh jednoduché aritmeticko-logické jednotky v prostředí Polis

Jde o jednoduchou aritmeticko-logickou jednotku, která má dva logické (D1L, D2L) a dva aritmetické (D3A, D4A) datové vstupy, dva řídicí vstupy (OP1, OP2), které určují, která operace se bude provádět a dva výstupy OL (pro výsledek logických operací) a OA (pro výsledek aritmetických operací). Nejprve je třeba, aby byly oba datové vstupy příslušející dané operaci ve stavu present. Poté je emitován výstupní signál OL nebo OA (podle větve příkazu case, která zrovna provádí příkaz emit.

Operace, podporované touto aritmeticko-logickou jednotkou jsou:

1) logický součin (and), OP1 = 0, OP2 = 0
2) logický součet (or), OP1 = 0, OP2 = 1
3) aritmetický součet, OP1 = 1, OP2 = 0
4) aritmetický rozdíl, OP1 = 1, OP2 = 1

Navíc je zde ještě přítomen vstupní signál R, který, pokud je aktivní, přeruší provádění cyklu loop.

Zdrojový program je obsažen v souboru alu.strl, alu.oc je výstupem příkazu esterel –oc alu.strl a alu.shift jsem získal pro provedení příkazu oc2shift –o alu.shift alu.oc. Následovalo načtení souboru alu.shift do prostředí Polis (příkaz read_shift), nastavení implementace na software (příkaz set_impl –s) (protože v simulačním prostředí Ptolemy se pro simulaci hardwaru i softwaru používá softwarový kód), vybudování vnitřních datových struktur pro software (příkaz partition), vytvoření grafu toku řízení/dat (příkaz build_sg) a vytvoření souboru DEalu.pl (příkaz write_pl). Tento soubor je vstupem pro simulační prostředí Ptolemy.

11 Literatura

[1] Sanjaya Kumar (Honeywell Technology Center), James H. Aylor, Barry W. Johnson, WM. Wulf (University of Virginia): The codesign of embedded systems: A unified hardware/software representation. Kluwer Academic Publishers, 1996, ISBN 0-7923-9636-7

[2] Manuály v elektronické formě obsažené v distribuci systému Esterel

[3] Manuály v elektronické formě obsažené v distribuci systému Polis