Tenhle předmět je mi záhadou, nic nás nechtěli naučit, písemky se psaly se skriptama, obsah, který je zajímavý by se dal shrnout do dvou přednášek a slajdy jsou jeden velký **bordel**. Snažil jsem se vypsat, co dávalo smysl, ale nevím nevím jak to na státnicích dopadne.

1 Úvod

HW/SW codesign je souběžný a kooperující návrh HW a SW. Typické pro vestavěné systémy, často CASE.

Snaha o co nejnižší cenu, nejlepší výkon a co nejrychlejší uvedení na trh.

Tradiční cesta navrhovala HW a SW od počátku zvlášť a integrace přišla v úvahu až na konci vývoje. Při nekompatibilitě byl třeba re-design (drahé, časové náročné).

HSC tvoří systémový návrh jako celek (chyby se zjistí již v počátku) a z něj se jednotlivé části "přidělí" HW nebo SW.

SW řešení je pomalejší, ale levnější a rychleji dokončeno, HW naopak rychlejší a dražší. Hledá se kompromis.

Vestavěný (embedded) systém je implementován jako soubor programovatelných bloků a hardwarových bloků (paměť, ASIC, procesor, časovače a další čipy), které interagují s okolím pomocí senzorů a akčních členů. Realizují specializovanou funkci, bývají součástí větších systémů. Musejí být spolehlivé, někdy i realtime.

System on a Chip (SoC) je moderní platforma pro vývoj HW. Na jednom čipu je několik propojených subsystémů.

IP cores (intellectual property) jsou znovupoužitelné komponenty ES

Při specifikaci se používá mnoho modelů, je potřebná globální verifikace a cosimulace.

Dále je ES – embedded systém a RTS – realtime systém.

ASIC = Application-specific Integrated Circuit

ASIP = Application-specific Instruction Processor (např. DSP – digital signal processor), vysoký výkon, ale specializované, dražší vývoj

FPGA = Field of Programmable Gate Arrays

Amdahlův zákon popisuje výpočet, jehož část je vždy sekvenční a část paralelní. Pak možné zrychlení paralelizací je konečné, jelikož se zrychluje pouze určitá část práce.

2 Specifikace systému

Specifikace – popis chování systému (formální a neformální), požadavky na systém, formální modely

Implementace – realizace systému

Konceptuální pohled je nejvyšší možný pohled na systém (výpočet je program, řadič je konečný automat). Specifikační jazyk má za úkol reprezentovat konceptuální pohled co nejmenším úsilím.

Důležité prvky specifikace: paralelní běh (concurrency), hierarchie, synchronizace, speciální požadavky na ES, RTS.

Data-driven concurrency: Operace se provádějí ihned, jakmile jsou přístupné vstupy. Pořadí vykonávání je dáno závislostmi na datech (typické pro ALU).

Control-driven concurrency: Rozdělení procesu na vlákna, která běží paralelně, původní proces rozdělil explicitně a vlákna se explicitně spojí.

Systémy založené na stavech, nejčastěji řadiče, lze výborně modelovat konečnými automaty.

Hierarchie: Důležitá součást návrhu složitějších systémů, rozdělení na podsystémy usnadní práci (rozdělení mezi více typů, lze se soustředit na podproblém). Hierarchický model má úrovně, dá se tak lépe pochopit.

Strukturální hierarchie popisuje architekturu systému. Systém jsou propojené komponenty (pomocí drátů na HW úrovni) a zavádí vrstvy (např. systém→čipy→hradla).

Behaviorální hierarchie popisuje chování systému. Je možné definovat "pod-chování" (například výstup→přesuny v paměti). Paralelní a sekvenční dekompozice chování.

Výjimky: Je třeba ošetřit chyby a nečekané situace. Řízení se nastaví do daného stavu (přerušení, reset).

Časování je problém u systémů, které jsou přímo vázané na reálný svět (RTOS, řízení strojů, ...). Je zapotřebí timeoutu, hlídání apod.

Nástroje pro specifikaci:

- *VHDL* – IEEE standard, nepodporuje výjimky a stavové přechody, implementuje sdílenou paměť, podmínky "wait" a "after"
- *Verilog a Esterel*
- *SDL* (Specification and Description Language)
- *CSP* (Communicating Sequential Processes) – vytvořeny pro multiprocesorové stroje
- *SpecCharts* – vyvinuty pro ES, mají stavové přechody

3 Výpočetní (komunikační) modely

CT – Continuous time – závislost na čase, reaktivní, věrný pro fyzikální systémy, drahý a specializovaný

PN – Process networks – paralelní aktivní procesy, které spojují vstupy a výstupy (FIFO, roury, apod.)

DF – Dataflow – varianta PN, procesy jsou aktéři a "vystřelují" události, vhodné pro zpracování signálů, deterministické a paralelní, nereaktivní, dobrá verifikace

FSM – Finite state machines – dekompozice na stavy a přechody, synchronní i asynchronní, NFSM a DFSM

- Moore – nereaktivní (odpověď až po přechodu), lehce komponované a implementovatelné
- Mealy – reaktivní (odpověď při přechodu), hůře komponované a implementovatelné

CFSM – Codesign FSM – FSM rozšířené o posílání dat a asynchronní zprávy, lokálně synchronní/globálně asynchronní

DT – Discrete time – diferenciální rovnice, lehce simulovatelné a implementovatelné, vhodné pro DSP, hodně specifické

DE – Discrete events – asynchronní, událost vyvolaná kdykoli, výstup je okamžitý, nedeterministické, globální fronta událostí (VHDL), těžko implementovatelné

SR – Synchronous/reactive – řada automatů komunikujících přes události, události jsou globální pro všechny, synchronní

CSP – Concurrent threads with rendezvous – události jsou setkáním odesilatele a příjemce, dobré sdílení prostředků, těžká synchronizace

4 Rozdělování

Funkce systému jsou implementovány na různých komponentách. (paměť, procesor, sběrnice, ...).

Je potřeba tyto komponenty vybrat a *rozdělit* systém tak, aby byl na nich implementovatelný.

Sktrukturální rozdělování – implementuje se struktura systému a pak se rozdělí funkce.

Behaviorální rozdělování – rozdělí se funkce systému a ty se pak zvlášť implementují a složí se systém (je to obtížnější ale s lepšími výsledky)

Bloky jsou části HW; *Objekty* jsou části modelu; Jde o mapování těchto entit.

Obecné algoritmy:

- *Integer linear programming* – exaktní, NP-úplný problém, cena se počítá z rovnic, ale ty exponenciálně rostou
- *Hierarchical clustering* – heuristická, konstruktivní, sdružuje blízké objekty, dokud není splněn požadavek (např. velikost)
- *Kernighan-Lin (min-cut)* – heuristická, iterativní, sdruží objekty, ale pro dosažení dobré ceny objekty migruje (omezení komunikace apod.)
- *Simulated annealing* – heuristická, iterativní, založena na fyz. jevu dosažení minimálního energetického stavu (termodynamická rovnováha)
- *Genetic evolution* – založena na inspiraci přírodou, použití genetických algoritmů

Algoritmy pro rozdělení HW a SW:

- *Greedy* – Implementace plně v HW/SW a pokud je příliš drahá/pomalá, migrace částí na SW/HW.
- *Hill climbing* – Náhodné rozdělení, postupně se migruje HW a SW tak, aby klesala cena a rostl výkon, ukončen v bodě, kdy již nelze zlepšit.

Programy pro funkční dělení: Cosyma, Vulcan

5 Hodnocení

Jedná se o zjištění parametrů produktu bez implementace. Využívá se modelování, analýza, simulace, prototypování.

Zaměřuje se na výkon, cenu, spolehlivost, čas vydání, atd.

Čím přesnější model, tím lepší odhad, ale také je model dražší a náročnější, déle trvá výpočet.

Věrnost udává, ne jak přesný je model, ale jak dobře modeluje skutečnost. Věrný model *vždy* odhadne cenu *stejně špatně/dobře* (tedy vždy stejná odchylka v %). Naopak, pokud odhadne vždy v jiném poměru, pak věrný není.

Výkonové metriky: počet cyklů, instrukcí (MIPS, FLOPS), doba běhu, rychlost komunikace, ...

Cenové metriky: cena za plochu čipu, cena balení, ...; velikost programu, zabraná paměť, ...

Jiné metriky: doba návrhu, příkon, čas do vydání, ...

Je snaha minimalizovat dobu nečinnosti jednotek HW, např. paralelizací.

Nejčastěji je modelován nejhorší případ.

6 Zjemňování

Po mapování objektů na komponenty je potřeba změny zanést do specifikace a zpřesnit (zjemnit) specifikaci, aby byla konzistentní a bylo možné dobře simulovat/verifikovat systém.

Proměnné: umístění proměnných na pevné místo v paměti, přidělení adres a začlenění odkazů na ně.

Kanály: Komunikační cesty jsou implementovány sběrnicí. Je potřeba určit propustnost (počet drátů) a protokol komunikace.

Pro specifikaci protokolu se často používají FSM nebo časové diagramy.

Protokol charakterizuje: délka zpráv, přístupová doba, maximální a průměrná propustnost protokolu, implementace sběrnice (datová, řídící část, nebo jen jedna) ...

Sběrnici charakterizuje: propustnost, zpoždění protokolu, průměrná a maximální propustnost sběrnice, paralelní/sériová ...

Řídící členy (arbiters) jsou potřeba při paralelním běhu v HW (přístup na sběrnici a do paměti)

7 Optimalizace modelu

Spotřeba – energie je drahá → ekonomické a ekologické důvody omezení spotřeby

Dynamické techniky omezují příkon, pokud není potřeba (dynamické napětí apod.).

Snížení příkonu se dosahuje nižším napětím čipu, popř. různými úrovněmi, také dynamické změny časování.

Je také možné použít různé algoritmy pro různé úrovně spotřeby, omezit přístupy do paměti, optimalizované protokoly komunikace.

Spotřeba sběrnice se dá omezit správným rozdělením na části.

Optimalizace paralelizací má limitu (viz Amdahlův zákon).

Je důležité optimalizovat největší část i kdyby nepatrně, lepší než silně optimalizovat malou část (na celkové rychlosti se to neprojeví).

8 Komunikace

Komunikace mezi paralelními komponentami: sdílená paměť nebo zasílání zpráv.

Sdílená paměť: odesilatel zapisuje do společného média, může být trvalé nebo dočasné, problém synchronizace přístupu.

Zasílání zpráv: odesilatel využije abstraktní kanál (volání, přerušení) a zašle zprávu příjemci – asynchronní. Kanály mouhou být i obousměrné, vícecestné, blokující/neblokující, ...).

Synchronizace: Paralelní procesy neběží stejnou rychlostí, je potřeba je synchronizovat pro výměnu dat nebo pokud je zapotřebí najednou provést akci.

Synchronizace pomocí dat: společnou událostí, detekcí stavu, společnou proměnnou

Synchronizace pomocí řízení: explicitní fork nebo join, popř. reset.

9 Verifikace, ko-simulace, rychlé prototypování, testování

Verifikace = oveřování, zda produkt splňuje specifikaci (lze automatizovat).

Testování = ověřování, zda produkt splňuje funkčnost (ověřují se případy použití).

Prototypování = vytváření "polotovarů" (ne úplně funkčních produktů), které ukazují, jak bude produkt vypadat. Tvoří se zdola nahoru nebo shora dolů, mohou se zahazovat nebo použít při další iteraci vývoje.

Ko-simulace = simulace HW (VHDL kódu) a SW (C kódu) najednou, jako by produkt byl hotov.

10 Překlad a syntéza

Samotný model systému většinou nelze zapsat v běžném programovacím jazyce, který by byl schopen jej pak přímo implementovat.

Je možné použít standardní jazyk, k němu je i mnoho nástrojů, ale model je zachycen nepřirozeně.

Při použití speciálního jazyka je model zachycen přirozeně, ale není jich mnoho a nástrojů na simulaci apod. je málo.

Nejlépe je použít frontend nad standardním jazykem, který pomůže model zachytit přirozeně a pak jej přeloží do standardního jazyka.

Příklady: SpecCharts→VHDL; FSM→VHDL; překlady výjimek nebo paralelních prostředí

11 Výpočetní platformy

Platformy jsou novým řešením SoC, ale není přesná definice pojmu "platforma" a v praxi nejsou tolik používané.

Myšlenka: Nenavrhovat čip od počátku, ale použít předdefinovanou architekturu pro určitý účel (platformu) – procesor s RTOS, sběrnice apod. Možno dodat další HW na čip, přeprogramovat FPGA, atd. → platforma PC.

Platformy jsou levnější na vývoj a rychleji je mžné dát do oběhu výrobek, ale nejsou tak flexibilní jako nový čip.

Platforma PC – levná, mnoho možností, ale moc se nemění v čase a i když je levná tak není vhodná pro specializované věci, protože má příliš mnoho nepotřebných částí pro takové užití.

Nemusí se navrhovat rozložení čipu (velmi složité), ale musí se navrhovat komunikace jednotlivých částí, které se dodaly na čip.

12 Architektury

Aplikačně specifické architektury: řadičová architektura, arch. datových cest, FSMD (konečný automat s datovými cestami)

Obecné architektury: CISC (complex instruction set computer), RISC (reduced...), vektorový stroj, VLIW (very long instruction word)

Paralelní architektury

Řadičová architektura: Obsahuje stavový registr a funkci, která mění stav (na základě vstupu a současného stavu), podle stavu se generuje výstup.

Architektura datových cest: Podle vstupů se přes jednotlivé kanály (pipelines) předávají dílčí výsledky na výstup (sčítačka apod.)

FSMD: Konečný automat (implementovaný řadičovou architekturou) jehož vstupy a výstupy jsou spojeny s datovými cestami.

CISC: Obsahuje kompletní sadu instrukcí pro jednotlivé úkony, intrukce jsou tvořeny mikroprogramem.

RISC: Obsahuje jen několik základních instrukcí (přístup do paměti pouze `LOAD` a `STOR`), pomocí kterých se ostatní instrukce simulují, obsahuje mnohem více registrů než CISC. Instrukce většinou jeden cyklus a stejná délka (dnes již plně neplatí)

Vektorový stroj: Pracuje nad vektory dat a ne nad skaláry (ale má i skalární část).

VLIW: Instrukce obsahuje více dat, nad různou části se provádí různá operace při pouze jedné instrukci (není plně paralelní).

Pipeline stroj: Na vstup dáváme data a po jednom taktu můžeme vkládat další, mezitím se počítá a za čas na konci bude výsledek (pajpy v UNIXu, moderní PC CPU).

Paralelní procesory: Používají SIMD/MIMD, takže jednotlivé části operandů jsou zpracovány paralelně.

13 Metodologie návrhu

V minulosti se design zaměřoval na nižší vrstvy abstrakce a bylo tak vytvořeno mnoho podpůrných nástrojů

Se současnými velkými systémy je však potřeba vyšších vrstev abstrakce a tedy nové metodologie a nástroje.

Metodologie musí popisovat:

- Syntaxi a sémantiku vstupů a výstupů

- Algoritmy transformace vstupů na výstupy
- Komponenty použité při implementaci
- Definice podmínek a omezení
- Mechanismy pro výběr architektury
- Řídící strategie

Požadavky na metodologii:

- Kompletní – všechny úrovně designu, implementační styly
- Rozšiřitelná – možnost přidat nástroje a algoritmy
- Kontrolovatelná – řízení kvality
- Interaktivita – nejen design ale i modifikace
- Udržovatelnost – možnost zlepšování