Software pro návrh integrovaných obvodů

Studijní materiál k předmětu A4M34SIS – ČVUT FEL – katedra mikroelektroniky

Abstrakt: - Materiál pojednává o nástrojích používaných při návrhu integrovaných obvodů. Zaměřuje se na HDL jazyky, na jejich praktický význam, použití v jednotlivých krocích návrhového postupu. Shrnuje historii jednotlivých jazyků, rozdíly mezi nimi, popis návrhového postupu a praktické příklady návrhu.

1 Mikroelektronika

Integrované obvody (IO) jsou v dnešní době základem každého elektronického zařízení. Stěžejním oborem, zabývajícím se problematikou funkce, použití a návrhu integrovaných obvodů, je mikroelektronika. Hlavním cílem tohoto rychle se rozvíjejícího oboru je snaha o miniaturizaci elektronických obvodů, samozřejmě při zachování kvality jejich funkce popř. její zlepšování. Proces miniaturizace umožňuje snižovat rozměry obvodů a jejich hmotnost, snižovat výrobní náklady a v důsledku toho i cenu, zvyšovat spolehlivost součástek a celých obvodů. Mikroelektronika nám dovoluje vyrábět nejrůznější druhy specializovaných integrovaných obvodů (dnešní integrované obvody neplní zdaleka jen funkci elektronickou, jejich součástí mohou být rozsáhlé mechanické struktury či nejrůznější typy elektronických senzorů, optické, magnetické a chemické struktury), elektronických obvodů a funkčních bloků miniaturních rozměrů neoddělitelně spojených na povrchu nebo v objemu polovodičového materiálu. Obor komplexně řeší fyzikální, chemické, technologické a obvodové otázky, které v sobě zahrnují problémy funkčních celků, zhotovených na jedné nebo několika základních podložkách, např. integrované obvody, obvody v pevné fázi atd.
Rozměry obvodových struktur se velice rychle blíží rozměrům nanometrovým, rozvíjí se tak další obory, např. nanoelektronika, molekulární elektronika, kvantová elektronika atd.
Návrh IO a jejich testování vyžaduje specifické postupy a programové prostředky. Návrh integrovaného obvodu je velice komplexním procesem a správná funkce výsledného obvodu či systému velkou měrou závisí na kvalitní práci návrháře a samozřejmě na CAD prostředcích, kterými disponuje.

2 Návrh integrovaných obvodů

Dnešní mikroelektronika se nejčastěji pohybuje v oblasti VLSI návrhu. Zkratka VLSI znamená Very Large Scale Integration, tedy velmi vysoký stupeň integrace. Pojem byl zaveden na počátku 80. let a vyjadřuje hustoty vyšší než desítky tisíc tranzistorů na jediném chipu. Nejmenší rozměry (délka hradla MOSFETu) se v dnešní době (2006) pohybují na úrovni 90 nm.
VLSI se objevuje v mikroprocesorech (osobní počítače, mikrokontroléry), paměti RAM, procesory pro speciální účely, DSP moduly atd.
Návrh integrovaného obvodu začíná specifikací systému (funkce, náklady atd.) a prochází jednotlivými návrhovými kroky: výběr architektury (velké bloky), logický design (hradla + registry), obvodový design (tranzistory navržené pro vysoké rychlosti, výkonové obvody), layout (určuje parazitní jevy) a testování (testovací vektory). Důležitým požadavkem na návrh je rychlost uvedení na trh. Obvod s výbornými parametry se pravděpodobně neprosadí, pokud se na trhu objeví později než konkurenční.
Návrh VLSI obvodů není možné provádět na úrovni hradel pro celý systém najednou, je proto nutné přistoupit k určitému stupni abstrakce. To je důvod, proč seskupujeme jednotlivé dílčí „stavební kameny“ do větších celků. Tranzistory tvoří hradla, hradla tvoří funkční jednotky a ty pak subsystémy pro zpracování dat, přenos, zapamatování atd.
Správný návrh by měl využívat jak top-down, tak bottom-up postup. Top-down návrh přidává postupně funkční detaily, z vyšších úrovní abstrakce vytváří nižší úrovně abstrakce. Bottom-up návrh vychází z chování na nejnižší úrovni, opakovaně používá bloky se známými charakteristikami chování.
Na konci návrhového procesu je soubor výrobních podkladů, na základě kterých je pak možno obvod vyrobit. Formální jazyk nám umožňuje definovat a zaznamenávat veškeré dílčí návrhové kroky a razantně snižuje riziko

nesprávného výkladu řešené problematiky. Je možné využít známé programovací jazyky, což je v některých případech stále používaný způsob návrhu. Vhodnější alternativou pro návrh hardwaru (tedy vnitřní struktury integrovaných systémů) je využití tzv. hardware description language (HDL).
Je důležité si uvědomit hlavní rozdíl mezi programováním a simulací. Výpočty v simulacích jsou závislé na čase. Pro správnou simulaci funkce hardwaru je třeba vědět, kdy se jednotlivé signály mění. Druhým podstatným rozdílem je paralelní činnost. Mnoho událostí probíhá současně, tedy ve stejném časovém okamžiku.
Každý programovací i HDL jazyk má své silné i slabé stránky. Velice zřetelnou slabinou HDL jazyků vytvořených v minulosti bylo to, že většinou nebyly navzájem kompatibilní, nebyly standardizovány. Návrh integrovaných obvodů ještě nebyl tak komplexní, a tak tajemství tohoto oboru zůstávalo většinou ukryto v univerzitních a výzkumných laboratořích. Situace se však změnila. V roce 1980, Ministerstvo obrany Spojených Států Amerických dalo podnět k vytvoření HDL jazyka pro vojenské účely. Tento jazyk byl nazván VHDL, což je zkratka VHSIC Hardware description language, tedy Very high speed integrated circuits HDL. Jazyk se brzy stal oblíbený i mimo vojenskou oblast, obzvláště v Evropě. V USA měl větší úspěch tzv. Verilog, který je dodnes hojně používán pro civilní aplikace. Oba tyto jazyky, VHDL i Verilog, byly přijaty za standard organizace IEEE.

2.1 Programovatelné obvody

Pokud uvažujeme o návrhu struktury pomocí programovatelných obvodů (FPGA), je chování obvodu popsané HDL jazyky obecně technologicky nezávislé a při jeho tvorbě se můžeme oprostit od znalosti architektury konkrétního IO. Pro jednodušší obvodové návrhy programovatelných IO (pro architektury PLD a CPLD) se nejčastěji používá jazyk Abel HDL. V tomto jazyce v podstatě zadáváme požadovaný systém jedním ze tří možností – logickými rovnicemi, pravdivostními tabulkami nebo stavovým diagramem. Takovýto zápis je ale nemožný pro složité systémy realizované např. obvody FPGA. Proto se k tomuto účelu používá jazyků vyšší úrovně (VHDL či Verilog), jejichž struktura se blíží vyšším programovacím jazykům.

2.2 Simulační programy

Nejznámějším simulačním programovým prostředím je PSpice A/D, které umožňuje popis smíšených (analogově-digitálních) soustav. Simulátor spojitých analogových bloků integruje nelineární maticovou diferenciální rovnici stavové reprezentace a ovládá hodnotami – událostmi – spouštěný číslicový simulátor. Ten je popsán Boolovskými sekvenčními a kombinačními předpisy ve formě tabulek a funkcí.
Jazyky HDL pracují s vyšší úrovní abstrakce, dochází tak mimo jiné ke zrychlení simulace a návrhu. Tyto jazyky je možno rozdělit na analogové, číslicové a smíšené systémy. Číslicové jazyky, jakými jsou VHDL či Verilog jsou založené na událostmi podmíněných konstrukcích a časově diskrétním popisu soustav. Podporují modelování číslicových soustav různé úrovně abstrakce až na úrovni jednotlivých hradel. Analogové jazyky HDL-A, Spectre HDL podporují popis systému diferenciálními algebraickými maticovými rovnicemi, kde řešení má spojitý časový průběh.

3 Historie HDL jazyků

Existují analogie programovacích jazyků (C, C++) určené pro návrh hardwaru, například ART-C od firmy Adelante, HANDEL-C od firmy Celoxica, SystemC atd.
První používané jazyky pro návrh hardwaru (např. ABEL, OHDL) byly závislé na architektuře obvodů, a tedy nepřenositelné a pro složité návrhy nejsou vhodné. Dnes máme k dispozici dva stejně mocné přenositelné, na architektuře nezávislé jazyky, kterými jsou Verilog HDL a VHDL.

3.1 Verilog

Verilog byl původně soukromým jazykem, vlastníkem byla firma Gateway Design Automation, která se později stala součástí firmy Cadence. V roce 1990 byl jazyk uvolněn pro veřejnost a v roce 1995 byl přijat za standard organizace IEEE. V roce 2001 byl přijat nový standard pro Verilog s mnoha rozšířeními, tento standard je ovšem zpětně kompatibilní s předchozím. Během vývoje byl jazyk prakticky testován a vylepšován podle požadavků a představ návrhářů.

3.1 VHDL

VHDL byl vyvinut s cílem dokumentovat číslicové systémy na úrovni chování (funkce) na základě programu VHSIC v roce 1981. Roku 1983 byla sepsána smlouva o používání jazyka s několika firmami, mezi nimi i IBM. Produkt byl označován VHDL 7.2 a byl dokončen v roce 1985. Snaha o zavedení jazyka do praxe byla doprovázena mnoha vylepšeními a v roce 1987 byl organizací IEEE vytvořen standard 1076. Další mezníkem ve vývoji byl VITAL (VHDL Initiative Towards ASIC Libraries) s cílem zvýšit možnosti praktické aplikace jazyka. V roce 1992 byl organizací IEEE přijat standard 1164. Obsahuje doporučení pro práce v různých simulačních prostředích.

4 Verilog HDL versus VHDL

Je těžké rozhodnout, který z HDL jazyků je lepší, stejně jako rozhodnout, který automobil je dokonalejší ve všech ohledech. Důvodem pro implementaci HDL jazyka v návrhovém procesu je především zvýšení produktivity návrhu. Zvýšení produktivity je však znatelné až po určité, poměrně dlouhé době. Rozhodnutí, který jazyk si vybrat, je zpravidla ovlivněno mnoha různými faktory. Těmi základními, které je třeba zvážit, jsou: Jednoduchost nastudování jazyka, jednoduchost použití a budoucí využití jazyka. Pomocí HDL jazyků, analogickým vyšším programovacím jazykům, jsme schopni převézt návrhový proces na vyšší úroveň abstrakce, na úroveň popisu chování syntetizovaného integrovaného obvodu.

Každý jazyk je tak dobrý, jak dobré jsou nástroje, které jej podporují. Tento fakt platí i pro HDL jazyky. Téměř každá fáze návrhu integrovaného obvodu je silně ovlivněna výběrem HDL jazyka. HDL jazyk používá editor schémat, simulační prostředí, nástroje pro syntézu, emulační a zobrazovací nástroje. Mnoho EDA (Electronic Design Automation) nástrojů i přes to, že obsahovaly vlastní HDL jazyky, implementovaly VHDL popis. Důvodem, proč se tak nestalo i v případě Verilogu, byl fakt, že VHDL byl volně přístupný, zatímco Verilog byl intelektuálním vlastnictvím firmy Gateway Design Automation, a později byl zakoupen firmou Cadence.

Verilog HDL na rozdíl od VHDL vzešel z obchodní oblasti a byl vyvinut jako součást kompletního simulačního systému. Byl také vyvinut za účelem popisu digitálně zaměřeného hardwaru. Jazyk umožňuje reprezentovat návrh důvěrně známou formou (tedy na úrovni popisu hradel a spínačů), stejně tak jako formou abstraktního popisu či behaviorálních modelů. Jinými slovy, podporuje jak top down, tak botom up postup.

Každý z těchto jazyků má své přednosti, každý je zaměřen jiným směrem: Předností Verilogu je jeho schopnost reprezentace digitálního hardwaru (a analogového v prostředí Verilog-A) způsobem umožňujícím návrháři si obvod představit a realizovat. Verilog má definované obvodové prvky (wire, wor, wand, tri atd.), jejich použití však není tak striktní jako v případě VHDL. Jak už bylo řečeno, ve Verilogu jsme schopni modelovat obvody na hradlové úrovni, jazyk tedy ASIC výrobcům umožňuje popis knihoven a jednotlivých buněk.

Některé znaky Verilogu (jako jsou globální proměnné) dovolují modelování systémového prostředí. Není tedy problém vytvořit si „testovací sadu“, což je možnost, která velmi chybí v jazyce VHDL. Verilog zároveň poskytuje možnost použití monitorovacího kódu uvnitř modelu. To zaručuje odstranění chyb v raném stadiu návrhu. V jazyce VHDL analogie tohoto kódu bohužel také neexistuje.

Jazyk Verilog umožňuje modelování zpoždění pomocí bloků. Tato možnost je velice významná, protože ani soustředěné ani koncentrované časové modely nejsou samy o sobě kompletní. Standardní formát zpoždění SDF je součást Verilogu a umožňuje zpětně dopočítat zpoždění. Tuto možnost ve VHDL postrádáme.

Pokud uvážíme výše zmíněné možnosti jazyka, jistě nepřekvapí, že si většina návrhářským ASIC firem vybrala za spolupracovníka právě Verilog.

Další významnou vlastností Verilogu je možnost „sáhnout si“ na signál i uvnitř modulu. To je nezbytná vlastnost, která nám dává možnost libovolného přístupu k proměnné i uvnitř modelu. Spojení na proměnnou je možno provést bez nutnosti deklarace další strukturních jednotek či portů. Verilog dále podporuje „pojmenované události“, které jsou důležité při vyšších úrovních abstrakce. Tato funkce je velice výhodná v případě vytváření „testovací sady“. Ve VHDL probíhá komunikace pomocí hodnot signálů. Jazyk Verilog podporuje systémové úlohy a funkce, návrhář tak může vkládat kontrolní příkazy a učinit tak vývoj hardwaru a jeho odlaďování efektivnější a produktivnější. VHDL tento princip jednoduše nepodporuje.

Mnoho příkazů jazyka Verilog je možno jednoduše syntetizovat, odpadá tak nutnost používat rozsáhlé úrovně parametrizace. Naproti tomu u VHDL, v případě vytváření modelů, je složitá parametrizace nutností. Tato nepříjemná vlastnost omezuje jinak „výjimečné programovací vlastnosti“ jazyka VHDL.
Významné přednosti Verilogu oproti VHDL a jeho vyšší praktická využitelnost v oblasti návrhu ASIC obvodů je důvodem, proč se v dalším textu budeme zabývat významem, funkcí a strukturou jazyka Verilog HDL.

5 Proč Verilog HDL při návrhu IO?

Hlavním přínosem HDL jazyků při návrhu integrovaných obvodů je možnost popisu chování struktury na úrovni chování obvodu. To je zpravidla popis, který návrhář obdrží od zákazníka. Ten toho o mikroelektronice nemusí příliš vědět, v každém případě však ví, co od svého nového obvodu očekává. Specifikace požadavků na funkci obvodu není jednorázová, spousta problémů, které je nutno dořešit, se objevuje během vlastního návrhu, ať už se jedná o jakoukoliv jeho fázi. Tento způsob popisu obvodu se nazývá „behavioral“, tedy popis obvodu tak, jak se nám jeví „z venku“.
Je jistě velice zřejmé, že tento popis je nejjednodušší. O to složitější jsou však nástroje, které k návrhu obvodu používáme. Při tomto způsobu popisu se totiž pohybujeme na úrovni, která má málo společného s elektronikou, jedná se o podobný princip, jako při programování ve vyšších programovacích jazycích (C, C+, C++, C#, Java, Delphi, Pascal atd.).
Jednou z možností je popsat navrhovanou strukturu pomocí HDL (Verilogu) a poté pomocí nástrojů (Synopsis, EDA) provést syntézu elektrického zapojení, vygenerování layoutu (technologického postupu), případně zapouzdření atd. Tento způsob se jeví jako velice jednoduchý, je třeba si však uvědomit, že popisem ve Verilogu návrh zdaleka nekončí. Podstatnou částí celé procedury je totiž testování a odlaďování, o kterém se zmíníme na konci této části.

5.1 Verilog při návrhu

Význam Verilogu jako hlavního návrhového prostředku je patrný z následujícího příkladu.
Představme si, že máme navrhnout analogový filtr,který má odfiltrovat daný signál tak, abychom pokud možno eliminovali nepodstatné rysy a naopak zvýraznili charakteristiky signálu, které nás zajímají. Pokud pro návrh použijeme Verilog, výrazně si usnadníme práci a ušetříme čas. Obvodové struktura filtru může být totiž značně komplexní a jednotlivé realizace se mohou značně lišit v závislosti na použitých prvcích, vlastnostech dané technologie, nebo na tom, kterého řádu filtr je. V první fázi návrhu známe pouze požadavky na funkci obvodu, pravděpodobně známe i technologii, ve které se obvod bude vyrábět (v závislosti na ceně, složitosti obvodu atd.). Pravděpodobně ale nemáme dobrou představu o tom, jakého řádu filtru bude muset být, aby splňoval nároky na něj kladené.
Pokud bychom přistoupili k návrhu na úrovni tranzistorové, po dlouhých dnech, či v lepším případě hodinách práce bychom během simulace obvodového chování zjistili, že námi navržený filtr ani zdaleka nesplňuje požadavky a usoudili bychom, že je potřeba použít filtr vyššího řádu. Modifikace struktury na filtr vyššího řádu nám může zabrat další hodiny práce, a výsledek může být podobně nepříznivý, jako při prvním pokusu.
Pokud se však pustíme do návrhu pomocí Verilogu, musíme napsat kód popisující chování obvodu, ve kterém popíšeme algoritmy filtrace. V popisu bude pomocí několika řádek specifikován stupeň filtru. Pokud při následující simulaci zjistíme, že stupeň filtru nevyhovuje požadavkům, stačí pouze změnit několik údajů ve zdrojovém kódu. Tímto způsobem značně zkrátíme časovou smyčku, ve které se chtě nechtě pohybujeme. Zbavíme se také zbytečných problému např. s parazitními jevy, které nám mohou obvod degradovat, a stálým dolaďováním obvodu pro stále vyšší stupeň filtru.
Poté, co se nám pomocí Verilogu podaří zvolit správný stupeň filtru, můžeme přistoupit k návrhu obvodového schématu, a to buď pomocí dalších softwarových nástrojů, nebo ručně. V každém případě jsme si tímto způsobem ušetřili několik zbytečných obvodových návrhů a problém se stupněm filtru se nám podařilo vyřešit na nejvyšší úrovni obvodového popisu.

5.2 Verilog při testování

Druhou možností návrhu je nepoužívat Hardware Description Language, ale pustit se přímo do návrhu struktury obvodu, tzn. návrhu na úrovni tranzistorové, popř. u digitálního obvodu na úrovni hradel (o úroveň výš). Tento

způsob je poměrně náročnější, především na znalosti návrháře v oblasti elektronických struktur, jejich chování a funkce.
Ať v našem výrobním procesu přistoupíme k návrhu s pomocí nebo bez HDL jazyka, čeká náš dříve či později testování a odlaďování návrhu, jak již bylo zmíněno výše. V této fázi může hrát Verilog velice významnou úlohu, jak je popsáno v následujícím textu.
Máme za úkol navrhnout například ASIC obvodovou strukturu fázového závěsu. Navrhneme obvod jedním z výše zmíněných způsobů. Je zřejmé, že při testování obvodu by bylo nejlepší odsimulovat kompletní zapojení, tedy integrovaný obvod zapojený tak, jak se v praxi bude pravděpodobně používat. Použití obvodů může být předem dáno, nebo může jít o universální strukturu.
Obvod fázového závěsu (PLL – Phase Lock Loop) se používá např. jako FM demodulátor. Mnohem širší oblast však zaujímá využití obvodu PLL v oblasti nepřímé kmitočtové syntézy, tedy generování signálů s programovatelnými, přesnými a stabilními hodnotami kmitočtu z referenčního kmitočtu. Nepřímá kmitočtová syntéza využívá smyčky fázového závěsu, doplněné programovatelnými děliči kmitočtu pro dělení a násobení kmitočtu celými čísly, případně směšovači pro realizaci součtu nebo rozdílu kmitočtů.
Při simulaci budeme chtít odsimulovat námi navrhovaný obvod v zapojení jako násobič kmitočtu. Bylo by však nesmyslné navrhovat kvůli tomu čítač, strukturu oscilátoru, případně další součásti. Mnohem jednodušším a efektivnějším řešením je návrh těchto komponent pomocí HDL popisu. Je zřejmé, že čas strávený při návrhu fázového závěsu bude několikanásobně kratší, než kdybychom museli každou z použitých komponent navrhovat na úrovni obvodové.

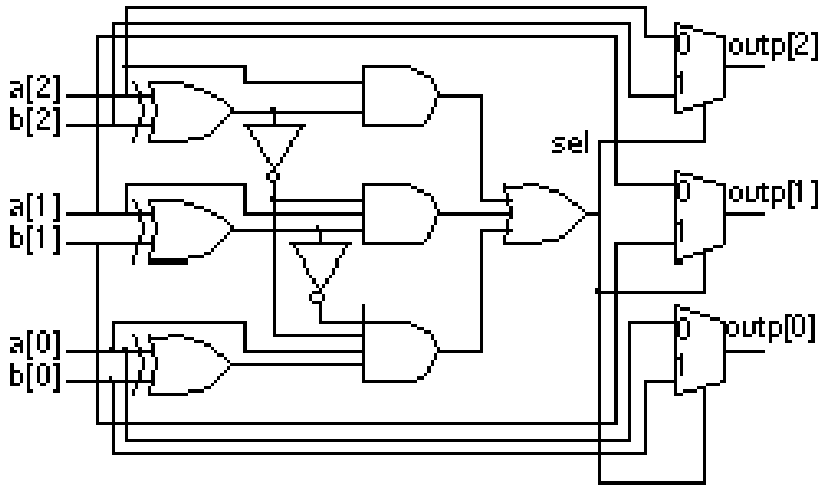
6 Návrhový postup

Filozofie návrhu integrovaného obvodu je taková, že se nový obvod vytváří jako nová knihovna - library - (v předem zvolené technologii). Pro jednotlivé části obvodů v knihovně vytváříme určitý počet bloků - cell. Jejich počet může být libovolný. Každá cell může obsahovat několik způsobů popisu - viewname, např. schematic, VerilogA, symbol, layout atd.
Prvním úkolem návrháře je běžně popis chování obvodu v některém z HDL jazyků, který se nijak neodvolává na logické buňky - cells . Popis ale může obsahovat také stavové diagramy, popis signálových cest, pravdivostní tabulky, šablony RAM/ROM Jak už jsme se zmínili, může jít o Verilog, VHDL či jiný standard. Zdrojem může být libovolný textový editor. Důležité však je, převézt nestrukturovaný a technologicky nezávislý popis na popis vázaný na zvolenou knihovnu, a tedy realizovatelný v rámci dodaných standardních buněk (standard_cell), v dané technologii. K tomuto kroku je často využíván program synopsys . Pro převod je vhodné vytvořit nový adresář (nebo pro tento účel mít jeden vyhrazený), neboť některé části Cadence přepisují nastavení důležitá pro převod (obvykle se spouští dávkou synopsys).
Poté je potřeba spustit analyzer, který umí zmíněný převod. V analyzeru načteme převáděný soubor. Pokud neobsahuje žádné chyby, a je-li prostředí Synopsisu správně nastaveno, provede se import souboru do prostředí Synopsis. V okně se vykreslí syntetizované schéma zapojení, a to v různých úrovních abstrakce, podle volby uživatele. V další fázi je potřeba provést optimalizaci. Ta se provede pomocí příslušné funkce. Výsledek syntézy je možno uložit v některém HDL jazyce (Verilog, VHDL, EDIF). Tento popis je však již na úrovni schematické.
Pro vlastní návrh obvodu je nutné vytvořit schéma zapojení, které pomocí základních stavebních prvků zvolené technologické knihovny (standard cell) realizuje námi zvolenou funkci. To můžeme kreslit přímo v editoru schémat (jsme-li toho schopni), nebo importem dříve popsaného HDL souboru. V každém případě je třeba spustit Cadence, přesněji jeho ciw (common interface window).
Nyní můžeme načíst kód Verilogu. Jestliže je vše v pořádku (viz log file), soubory jsou na svých místech, je pravděpodobné, že námi vytvořená knihovna bude bohatší o několik cell. Výsledek je možno otevřít buď pomocí File_manageru, nebo přímoz ciw. Vytváření schematu ručně se provádí touto posloupností příkazů: File -- New -- Library (zadáme nové jméno např. My_Cell a následně výchozí tech. knihovnu), pak File -- New -- Cell (zadáme jméno cell, viewname a po vytvoření se již otevře okno schematického editoru). Máme-li schéma hotové, provedeme uložení a kontrolu návrhu (Design -- Check & save) a v případě, že počet chyb nás k tomu opravňuje, můžeme přejít k simulaci.
V dalším kroku se provede simulace syntetizovaného obvodu a provede se srovnání s dřívější simulací chování.

Následuje vytvoření layoutu, tedy technologického postupu. Je možné jej nakreslit ručně, nebo jej nechat automaticky vygenerovat. Zde i v předchozím kroku, kde jsme automaticky generovali obvodovou strukturu, však platí, že výsledek bude tak dokonalý, jak dokonalý je program, který jej vytváří. To jinými slovy znamená, že schéma, i po optimalizaci, může být redundantní. Zkušený návrhář tak v některých případech dokáže navrhnout strukturu obvodu jednodušší, než automaticky generovanou. Při automatické generaci layoutu nemusí např. dojít ke sloučení terminálů tranzistorů, i když jsou ve skutečnosti společné. Je zřejmé, že ručním nakreslením lze dosáhnout úspory místa (ovšem za cenu delší doby návrhu). Jak ale uvidíme v následujícím příkladu, nemusí tomu tak být vždy.

6.1 Příklad návrhu

V tomto příkladu je navrhován komparátor a multiplexer – MUX – používaný ve Viterbiho dekodéru.
Na obrázku 1 je obvod navržen ručně:



Obr.1: Ručně navržený obvod komparátoru a multiplexeru

V následující textu je popis chování ve Verilogu:

```
// comp_mux.v
module comp_mux(a, b, outp);
input [2:0] a, b;
output [2:0] outp;
function [2:0] compare;
input [2:0] ina, inb;
begin
if (ina <= inb) compare = ina;
else compare = inb;
end
endfunction
assign outp = compare(a, b);
endmodule
```

Popis je analogický obvodové struktuře, jeho vytvoření však o poznání jednodušší.

Tab.1: Přehled parametrů jednotlivých struktur

	zpoždění /ns	počet standardních buněk	počet tranzistorů	plocha /mm^2
ručně	4.3	12	116	68.68
aut.	2.9	15	66	46.43

Z tohoto příkladu vidíme, že automatický návrhový systém nám vygeneroval více standardních buněk, které však celkově obsahují menší počet tranzistorů a zabírají menší plochu.

7 Struktura jazyka Verilog HDL

Jazyk Verilog je tzv. case sensitive. To znamená, že je důležité rozlišovat velká a malá písmena.
Důležitým prvkem jazyka je tzv. identifikátor – identifier (např. názvy proměnných) může obsahovat posloupnost písmen, číslic, značku dolaru $, či podtržítko _.
Jako v každém jazyce (programovacím, formátovacím, HDL) i zde existují tzv. komentáře, které se oddělují znaky „/*“ a „*/“ pro víceřádkový komentář, popř. dvěma lomítky „//“ pro jednořádkový komentář.
Logické hodnoty jsou čtyři: „0“, „1“, „x“ a „z“. Hodnota „x“ značí neznámou, hodnota „z“ stav vysoké impedance.
Verilog má samozřejmě implementovány aritmetické operace (i na vektorech – rozdíl oproti VHDL) a logické operace (and, nand, or, nor, xor, xnor, negace) a další operátory (podmínka, logická rovnost, menší než, posun vlevo atd.). Příkazy jsou prakticky shodné s těmi v jazyce C.
Hlavní strukturní jednotkou jazyka je tzv. modul – module. Není nutné explicitně definovat skalární vodiče. Proměnné se totiž objeví v popisu modulu a jsou tedy implicitně definovány jako vstupy a výstupy (inputs a outputs). Rozhraní modulů umožňuje jednotlivé moduly navzájem spojovat pomocí portů. Každý z portů musí být explicitně definován jako vstupní (input), výstupní (output) nebo vstupně-výstupní (inout).
Procedury – procedures – jsou příkazy následující po podmínkových výrazech jako always, initial, task nebo function. Příkazy uvnitř sekvenčního bloku (příkazy objevující se mezi begin a end), který je součástí procedury, jsou prováděny sekvenčně, tzn. v pořadí, ve kterém se objevují, ale jednotlivé procedury jsou vykonávány paralelně. Toto je základní rozdíl oproti počítačovým programovacím jazykům.
Příkazy uvnitř sekvenčního bloku (část kódu mezi begin a end) jsou prováděny v pořadí, ale bez jakéhokoliv zpoždění, tedy následují po sobě vzdáleny vždy o časový krok – time step.Ve skutečnosti však v obvodu existují zpoždění, která je možno modelovat pomocí časování – timing control. Tímto způsobem můžeme nastavit určitě zpoždění signálu – delay control – pomocí příkazu timescale. Druhou možností je vyčkání určité události – event control.
Ve Verilogu existují příkazy provádějící různé cykly, podobně jako v programovacích jazycích: if, case, while, loop, atd.
Ve Verilogu existuje několik různých příkazů pro výpis na obrazovku (podobně jako v C).

8 Závěr

HDL jazyky hrají velice důležitou úlohu v oblasti návrhu mikroelektronických struktur. Dva nejpoužívanější a standardizované jazyky jsou Verilog HDL a VHDL. Jejich struktura se značně liší. Jazyk Verilog, který byl vytvořen pro účely návrhu elektronických struktur, je vhodnějším nástrojem. Možnosti jazyky jsou široké, umožňuje popis na různých úrovních abstrakce, rychlý a účelný návrh a vhodné nástroje pro simulaci a testování navržených struktur.

Použitá literatura

[1] http://www.fm.vslib.cz/%7Ekes/asic/
[2] http://utelnt.el.utwente.nl/links/gerez/soc/index.html
[3] http://www-ee.eng.hawaii.edu/~msmith/ASICs/HTML/ASICs.htm
[4] http://www.cs.bilkent.edu.tr/~baray/cs224/
[5] http://ece.colorado.edu/~ecen5007/
[6]manuál VerilogA od firmy Cadence
www.cadence.com
[7] Alfred K. Wong, „Some Thoughts on the IC Design-Manufacture Interface“, *Design&Test of Computers,* Vol.22, p.206-213, May/June 2005