Tento článek je původním rukopisem textu publikovaného v časopise DPS Elektronika A-Z: J. Šťastný. Verifikace pomocí assertions: seznámení, DPS Elektronika od A do Z, no 6, pp. 4-8, 2012.

Verifikace pomocí assertions: seznámení

Jakub Šťastný
ASICentrum, s.r.o.
FPGA Laboratoř, Katedra teorie obvodů FEL ČVUT Praha

1 Úvod

S vlivem Mooreova zákona na návrhový cyklus integrovaných obvodů byl čtenář seznámen už v předchozím článku [1]; dozvěděli jsme se, že neustále se zvyšující složitost navrhovaných číslicových systému – jakkoliv pozitivní pro koncového zákazníka – komplikuje návrhový proces integrovaných obvodů. Čtenáři byla představena jedna technika zrychlující číslicový návrh, syntéza na vyšší úrovni. Tento příspěvek na [1] volně navazuje a jeho cílem je seznámit čtenáře s technikou umožňující urychlení ladění a ověření správné funkce číslicového systému: s verifikací pomocí assertions.

Cílem verifikace číslicového obvodu je ověření jeho správné funkce. Pomocí sady testů je simulacemi a dalšími prostředky prokázáno, že jednotlivé funkce navrhovaného obvodu pracují jak mají. Roli důkladných a důsledných simulací v návrhovém procesu nelze podceňovat. V každém systému jsou chyby a praxe ukazuje, že nesimulované bloky budou s jistotou chybně navržené, viz např. [2]. Ani v případě dokonalého RTL návrhu naprosto nelze vyloučit chyby v návrhových nástrojích, jež mohou mít na finální produkt devastující dopad.

Verifikace je významnou součástí návrhového cyklu číslicového obvodu, protože umožňuje najít chyby v návrhu co nejdříve a tedy s co nejnižšími náklady na odstranění. Obecně platí, že náklady na odstranění chyby během tvorby specifikace, psaní RTL kódu, verifikace a validace vyrobeného číslicového systému jsou zhruba v poměru 1:10:100:1000. Vezmeme-li v potaz cenu chyby, která pronikne do vyrobeného systému, vyjádřenou ne jen nemalými náklady na její opravu, ale i potenciálním dopadem na společnost dnes už zcela závislou na fungujících elektronických zařízeních, vidíme, že verifikaci nelze podceňovat [3]. Více o obecných principech verifikace se lze dozvědět v knize [4].

Množství času stráveného verifikací závisí na složitosti simulovaného obvodu. Je to proto, že stavový prostor číslicového obvodu – tedy například počet kombinací, kterými lze zkombinovat jeho jednotlivé funkční módy – narůstá exponenciálně s velikostí obvodu. S tím, jak roste množství funkcí integrovaných do číslicových obvodů a narůstá velikost navrhovaného systému je třeba napsat větší množství testů, které také typicky déle běží. V důsledku toho je stále složitější chyby v obvodu najít, a změny v projektu se stávají dražší na čas i prostředky. Rostoucí složitost číslicových systémů také vede k častějšímu znovupoužívání již existujících bloků [5]. Ty, ačkoliv odladěny v předchozích návrzích, mohou v novém návrhu selhat vinou špatné integrace, nebo změn ve znovupoužitém bloku, které nejsou slučitelné s jeho vnitřní implementací.

2 Verifikace pomocí assertions

Assertion[1] je úsek kódu vložený buď do vlastního návrhu číslicového systému nebo do simulačního prostředí obvykle proto, aby kontroloval požadované chování systému. Kromě kontroly, zda se vše v návrhu děje jak má, může assertion sloužit i pro monitorování výskytů očekávaného chování obvodu – například mezních podmínek. Je-li určen ke kontrole, oznamuje porušení očekávaného chování, je-li užit k monitorování, počítá výskyty hledáného chování, které pak jsou na konci simulace reportovány.

Assertions představují snadnou cestu, jak zrychlit verifikaci návrhu. Může je využít jak verifikátor, tak návrhář číslicového obvodu. Ačkoli jejich použití vyžaduje naučit se nové techniky, představují jen inkrementální vylepšení používaných přístupů. Proto je lze téměř vždy snadno integrovat do návrhu a jejich použití nevyžaduje velké úsilí. V současné době je použití assertions běžnou součástí verifikace všech větších číslicových systémů (viz např. [3,6]), nejedná se tedy o nijak exotickou techniku.

V softwarovém průmyslu jsou assertions úspěšně používány už několik desítek let. Typický příklad assertion v softwarovém kódu je následující:

1Autorovi článku není známý žádný český ekvivalent termínu *assertion*, a proto bude v textu používán nepřeložený (v množném čísle tedy ve tvaru *assertions*).

Situace: Aplikace napsaná v C++ alokuje paměť pro dynamické proměnné na haldě.

Co se může kontrolovat: Alokace paměti může selhat z důvodu nedostatku místa na haldě.

Příklad jednoduchého kódu v C++, který tuto situaci korektně ošetří:

```
a = new int;
if (a==NULL) {
  printf ("Not enough memory");
  … ukonči aplikaci, atd. ….
}
```

Aplikace v případě nedostatku paměti vypíše chybové hlášení a je ukončena. Podobný příklad assertion v číslicovém návrhu by mohl například vypadat takto:

Situace: Číslicový obvod obsahuje blok realizující hardwarový zásobník pro 16 hodnot. Zásobník indikuje stav „plný" (signálem *full* v log. 1). Zápis do zásobníku se provádí tehdy, pokud je signál *write* v log. 1, zápis je synchronní s náběžnou hranou hodin.

Co se může kontrolovat: Nikdy nesmí dojít k tomu, že by se číslicový systém používající zásobník pokusil zapsat do plného zásobníku.

Jak bude vypadat velmi jednoduchý assertion v jazyce VHDL:

```
ASSERT NOT(full='1' AND write='1')
REPORT "Write to the full stack detected."
SEVERITY ERROR;
```

3 Práce s assertions

Primárním účelem použití assertions je zkrátit fázi ladění návrhu číslicového obvodu. Podívejme se, jak vypadá běžný postup práce při návrhu a verifikaci číslicového obvodu:

1. Blok je navržen na RTL úrovni. Návrhář k jeho simulaci používá jen jednoduché scénáře pro hrubé ověření správné funkce. Tu ověřuje obvykle pohledem do časových průběhů v simulátoru, nepoužívá automatické kontroly správné funkce bloku.
2. Verifikátor sestaví simulační prostředí a začne psát testy.
3. Uvnitř návrhu je vždy někde chyba [2]; ta se projeví při vhodném buzení příslušného bloku a v důsledku ní dojde k nesprávné funkci obvodu. Verifikátor o chybě zatím neví.
4. Následují-li v simulaci vhodné stimuly, je chyba zpropagována na primární výstupy obvodu.
5. Na primárních výstupech je chyba objevena prostřednictvím rozporu mezi očekávaným a reálným chováním.
6. Verifikátor zpětně odtrasuje rozpor do nitra obvodu a nahlásí návrháři zjištěný problém.
7. Návrhář celý postup zreprodukuje, chybu opraví a zkontroluje, že byl problém opravdu vyřešen. Verifikátor pokračuje v práci.

Popsaný postup funguje celkem dobře, u větších a složitějších obvodů ale má následující nevýhody:

- Propagace chybného chování na primární výstupy trvá dlouho a často k ní ani nemusí dojít.
- Verifikace chování, které se neprojevuje chybnou funkcí obvodu je velmi obtížná. Takovým případem může být například chybná funkce logických obvodů, které slouží pro snížení spotřeby elektrické energie číslicovým obvodem. Systém je funkčně v pořádku, spotřebu v číslicovém simulátoru ale jednoduše sledovat nelze.
- Krok trasování problému zpět do nitra obvodu je časově náročný, navíc s sebou pro verifikátora nese nutnost pochopit funkci obvodu. To celou operaci ladění dělá ještě více časově náročnou a navíc může být i stresující pro verifikátora, který musí být schopen absorbovat mnohem více informací.

Představme si ale, že by už během návrhu byl do bloku vložen assertion, který hlídá jeho správné chování a kontroluje vnitřní integritu implementace. Výše popsaný scénář bude teď vypadat následovně:

1. Blok je navržen na RTL úrovni a do bloku jsou vloženy assertions. Návrhář k simulaci bloku používá jen jednoduché scénáře pro hrubé ověření správné funkce. Tu obvykle ověřuje pohledem do časových průběhů v simulátoru, nepoužívá automatické kontroly správné funkce bloku.
2. Verifikátor sestaví simulační prostředí a začne psát testy.
3. Uvnitř návrhu je vždy někde chyba; ta se projeví při vhodném buzení příslušného bloku a v jejím důsledku dojde k nesprávné funkci obvodu. Chyba je zachycena assertions a do logu simulátoru je vypsáno chybové hlášení, které obsahuje mimo jiné lokalizaci chyby v bloku ve struktuře obvodu.
4. Verifikátor nahlásí návrháři přítomnost rozporu reálného a očekávaného chování obvodu.

5. Návrhář u sebe celý postup zreprodukuje, chybu opraví a zkontroluje, že byl problém opravdu vyřešen. Verifikátor pokračuje v práci.

Celý proces ladění byl zkrácen o fázi propagace chybného chování na primární výstupy a zpětného trasování problému do nitra obvodu za cenu počáteční investice při psaní bloku – vložení assertions. Uplatnil se tu nejvíce patrný pozitivní dopad používání assertions – assertions představuje nový virtuální výstup obvodu [7] s automatickou kontrolou správného chování. Tím je dosaženo lepší pozorovatelnosti vnitřních stavů návrhu.

Jsou-li assertions správně napsané, je nicméně možný i jiný, ještě příjemnější scénář:

1. Blok je navržen na RTL úrovni a do bloku jsou vloženy assertions. Návrhář k simulaci bloku používá jen jednoduché scénáře pro hrubé ověření správné funkce. Během těchto jednoduchých simulací, typicky kontrolovaných jen očima, integrované assertions detekují chybné chování v bloku. Návrhář tedy chybu hned opraví, aniž by byl do verifikace zatím zapojen i verifikátor.
2. Dále pokračuje ladění jako výše.

Verifikátor nyní dostává lépe odladěný blok a nemusí trávit tolik času reportováním problémů a dalším případným odlaďováním. Narazili jsme tu na druhý důležitý pozitivní efekt assertions – jsou aktivní ve všech testech a jsou tedy schopné najít například zákeřné chyby, které se v bloku objeví jako důsledek změny chování např. jiného bloku v okolí a to i v testech, kdy by se chování bloku s assertions vůbec nepropagovalo na primární výstupy systému.

Autor tohoto článku – z vlastní praxe – zná nicméně ještě jeden scénář:

1. Blok je navržen na RTL úrovni a do bloku jsou průběžně vkládány assertions. Návrhář si během psaní assertions uvědomí, že to, co navrhl, nemůže fungovat jak to je, a rovnou návrh přepracuje bez nutnosti pouštět jedinou simulaci.
2. Dále pokračuje ladění jako výše.

V tomto případě se uplatnil jiný příjemný efekt verifikace pomocí assertions – jejich používání nutí návrháře více přemýšlet nad tím, co vlastně dělají.

4 Přínosy assertions

Hlavní přínos použití assertions pro číslicový návrh je tedy zrychlení nacházení chyb v časných fázích návrhu. Kromě rychlejší identifikace chyby assertions nepřímo umožňují také její rychlejší odladění, protože v časných fázích návrhu má autor návrhu v hlavě ještě vše potřebné a na bloku často ještě pracuje; odstraní se tak zdržení způsobené přemýšlením nad tím „co jsem to tu vlastně před tím měsícem dělal...“.

Samozřejmě, nic není zcela zdarma. Abychom mohli přínosu assertions využít v plné síle, musíme něco investovat:

1. **Náš vlastní čas na zvládnutí příslušných nástrojů a technik.** Ze zkušenosti autora – základy psaní assertions v jazyce PSL, nebo pomocí knihovny OVL, lze zvládnout za cca týden. Není to tedy nijak zvlášť časově náročné. Čas strávený učením se metodologii a nástrojům je dobře investovaný čas jak pro návrháře, tak pro celou firmu, ve které pracuje. Navíc, jak zvládnete základy, zjistíte, že je jejich psaní jednoduché a přirozené. Asi nejtěžší je se naučit při své práci promýšlet, co se všechno může pokazit.
2. **Čas v počátečních fázích návrhu.** Assertions je nezbytné do RTL kódu návrhu vkládat už během jeho psaní. Taktika „dneska si napíšu RTL kód a zítra, když budu mít čas kolem oběda, tak tam vrazím ty assertions“ nefunguje. Pokud nejsou assertions v bloku již v okamžicích prvních simulací na stole návrháře, jejich přínos rapidně klesá. Ani zde se ale nejedná o velkou investici, vložení assertions do kódu může – podle zkušeností autora článku i podle [8] – prodloužit dobu strávenou návrhem odhadem o jednotky procent. Pravda je, že identifikace a odladění chyb bez assertions je časově výrazně náročnější. Abychom dosáhli zrychlení fáze ladění návrhu, musíme trochu času investovat jinde a akceptovat prodloužení fáze RTL návrhu.
3. **Simulační čas.** Běh assertions v číslicovém simulátoru prodlužuje simulační čas, nicméně u dobře napsaných assertions je pozorovatelné zpomalení v řádu maximálně 15-30 procent [8]. Místo vzácného času návrháře se tak spotřebovává mnohem levnější čas stroje a návrhář může dělat něco zajímavějšího, případně důležitějšího.

Assertions mohou při návrhu pomáhat mnoha rozličnými způsoby:

1. **Kontrolovat očekávané chování na vstupech a výstupech bloku.** Assertions umístěné v rozhraní bloku monitorují správné používání bloku a dodržování požadovaného protokolu na jeho vstupních a výstupních signálech. Primární účel assertions je zde hlídat, zda je blok správně integrován do svého okolí. Navrhovaný blok je většinou poměrně dobře odladěný sám o sobě již návrhářem, ale během integrace je propojen svými rozhraními s bloky jiných návrhářů a právě rozhraní a příslušná infrastruktura pro propojení je svým způsobem „země nikoho“. Je časté, že v této šedé zóně vznikají nepříjemné chyby. Proto jsou assertions vkládány do rozhraní znovupoužitelných bloků, aby kontrolovaly jejich správné použití. K opětovnému použití obvykle dochází s časovým odstupem a často u něj není přítomen původní autor bloku, takže automatické kontroly správné integrace jsou velmi žádoucí. Charakteristickým příkladem může být například assertion, který monitoruje správnou funkci

korespondenčního protokolu (viz [4], kapitola 11). Takovým assertions se někdy říká předpoklady (*assumptions*).

2. **Kontrolovat očekávané chování logických struktur uvnitř obvodu.** Assertions umístěné uvnitř bloku hlídají, zda je udržena jeho vnitřní integrita. Primární účel takových assertions je pomoci návrháři odladit chování bloku co nejdříve. Další použití mohou tyto assertions nalézt při znovupoužití bloku, pokud je blok modifikován z důvodu odlišných požadavků nového systému, do kterého je blok vložen. Pak vložené assertions automaticky hlídají, zda změnami (prováděnými typicky jiným návrhářem s jen zprostředkovanou znalostí původního systému) nebyla do funkce bloku zanesena nová chyba. Typickým příkladem je assertion monitorující sběrnici, která má být kódovaná v kódu 1 z N; pokud se na sběrnici objeví více, nebo méně než jedna jednička, assertion vyhlásí chybu.
3. **Monitorovat chování bloku a počítat výskyty definovaných podmínek, měřit funkční pokrytí**. Jedním z nejtypičtějších příkladů je monitorování chování FIFO paměti. Obvykle během verifikace obvodu chceme alespoň jednou dosáhnout stavu, že je FIFO zcela prázdné a zcela plné (někdy také třeba chceme, aby určitě nastala situace, kdy se obvod pokouší zapsat do plného FIFA, nebo číst z FIFA prázdného). Pomocí vhodně napsaného assertion můžeme prokázat, že příslušná situace nastala a protože víme, že se obvod v simulacích vždy choval jak měl, můžeme říci, že je logika implementující okrajové podmínky správně implementována. Tyto assertions se označují jako assertions měřicí pokrytí (*cover*).
4. **Sloužit jako komentář k funkci bloku a pomoci jeho pochopení.** Dobře napsané assertions zachycují velmi přehlednou a srozumitelnou formou správnou funkci obvodu. Mohou tak pomoci návrháři, který se blok bude po několika letech snažit znovu použít, v pochopení jeho funkce. Obecně lze říci, že kdykoliv chce návrhář napsat do kódu komentář typu „zde se vždy musí stát, že...“, případně „zde se nikdy nesmí stát, že...“, je to vhodný okamžik k tomu do kódu místo toho napsat assertion.
5. **Sloužit jako komentář k funkci bloku, který upozorňuje na potřebu ještě něco dokončit.** Autor článku si během své návrhářské praxe osvojil zvyk psát do RTL kódu důležité poznámky ve formě assertions, které vypíší komentář typu „.... zde ještě není dokončeno generování hodin ….“ jako varování při každém spuštění simulace. Je to velmi praktické, protože skutečnost, že je třeba ještě něco dodělat, se připomíná na začátku každé siumulace v logu simulátoru a je velmi těžké ji přehlédnout.
6. **Poskytnout informace nástroji pro formální verifikaci.** Nástroje pro formální verifikaci potřebují pro správnou funkci definovat, jaké stimuly jsou přípustné na rozhraní verifikovaného bloku. Implementované assertions jsou použity proto, aby nástroj lépe uchopil funkci analyzovaného bloku a ušetřil si například prohledávání části stavového prostoru.
7. **Pomoci nalézt chyby v systému v prototypu.** Některé nástroje pro emulaci číslicových obvodů a jejich prototypování jsou schopny assertions zkompilovat do podoby kontrolních bloků hlídajících správnou funkci příslušných logických obvodů přímo v FPGA prototypu.

Významnou vlastností assertions je i jejich znovupoužitelnost [9]; dobře navržené jednou implementované assertions lze snadno využít i v jiném bloku ve stejném kontextu (například assertion kontrolující správné chování korespondenčního protokolu).

Největší přínos má použití assertions v blocích určených pro opětovné použití. Zde se čas investovaný do jejich implementace a použití mnohonásobně zúročí při každém dalším použití bloku.

Doporučené je použití assertions i v nově navrhovaných blocích, přestože nejsou určeny pro opětovné použití. I zde se silně uplatní pozitivní efekt integrovaných kontrol; zkušenosti autora ukazují, že v pozdních fázích návrhu, kdy už si návrháři tak dobře nepamatují všechny detaily, jsou kontroly pomocí assertions neocenitelné.

Implementace assertions do existujících bloků již tak velký přínos nemá. Není sice neobvyklé, že pomocí nově dopsaných assertions jsou nalezeny chyby, o kterých jste netušili, ale většinou jsou již existující bloky dobře odladěné. U existujících návrhů má smysl doplňovat assertions zejména do bloků, které jsou často v návrhu využívány na mnoha místech.

Používáte-li při návrhu knihovnu vlastních základních stavebních bloků (například různé elementární buňky jako synchronizátory, buňky určené pro hradlování hodin, čítače, atd., ale i složitější bloky jako je FIFO, I2C, nebo SPI rozhraní), jsou tyto horkými kandidáty pro implementaci assertions kontrolujících jejich správné použití.

5 Implementace assertions, potřebné nástroje

Assertions lze implementovat jak v běžných jazycích pro RTL návrh, tak i ve specializovaných jazycích. Assertions mohou být do návrhu vloženy dvěma způsoby. Jedna možnost je, že jsou implementovány v samostatných blocích (ve VHDL entitách, ve Verilogu v modulech) a nainstancovány do příslušného RTL kódu bloku. Takto se například používá knihovna OVL (*Open Verification Library*) [10]. Druhou možností je vkládání assertions přímo do psaného RTL kódu; tak je tomu při použití jazyka PSL (*Property Specification Language*), případně SVA (*System Verilog for Assertions*). Assertions napsané v PSL se přitom vkládají do komentářů v kódu, zatímco SVA assertions jsou přímo prvkem jazyka. Následující tabulka shrnuje základní vlastnosti, výhody a nevýhody popsaných přístupů.

Jazyk	VHDL/Verilog	OVL	PSL/SVA
Způsob zápisu	Běžný behaviorální kód typicky obklopený direktivami *pragma synthesis_off* *pragma synthesis_on* aby se zabránilo syntezátoru v pokusu o jeho interpretaci	Nainstancované bloky v RTL kódu.	PSL assertions jsou uváděny v komentářích v RTL kódu. SVA assertions se píší přímo do kódu jako jazykové konstrukce System Verilogu.
Podpora nástroji pro simulaci	Každý číslicový simulátor	Každý číslicový simulátor	Potřeba simulátor s podporou PSL/SVA, například ModelSim DE [11] nebo Questa.
Podpora dalšími nástroji	Žádná	Podpora dalšími nástroji – formální verifikace.	V simulátoru funkční pokrytí a monitorování spouštění assertions, monitorování jejich vykonávání a možnost komfortního ladění. Podpora dalšími nástroji – formální verifikace.
Časová náročnost implementace	Vyšší, obvykle u jednoho assertion až ekvivalentní implementaci stavového automatu na RTL úrovni.	Snadno se lze naučit používat, pro jednodušší assertions rychlé, u složitějších časově náročnější, než použití PSL/SVA, ale rychlejší, než vlastní implementace v HDL jazyce.	Je potřeba se naučit používat nový nástroj, pak je ale možné assertions psát velmi rychle.
Časová náročnost simulace	Vyšší	Vyšší	Nižší, assertions mohou být interpretovány a optimalizovány simulátorem.

Dodejme, že OVL knihovna obsahuje cca 30 standardních assertions implementovaných v jazycích VHDL, Verilog, VHDL-PSL a Verilog-PSL; ve všech je k dispozici stejná sada kontrolních bloků. OVL tedy není univerzální jazyk pro psaní assertions, ale knihovna předpřipravených assertions, které lze buď přímo použít, nebo dále upravovat pro specifické účely. Knihovna je k dispozici zdarma na internetových stránkách [10].

6 Shrnutí

Verifikace s pomocí assertions je dnes již široce používánou a vyspělou technikou. Její použití zrychluje fázi ladění číslicového systému a omezuje nejistotu při plánování projektu, která vyplývá z toho, že čas strávený laděním chyb je velmi obtížné předvídat. Navíc jsou assertions slučitelné s libovolnou verifikační metodikou a jejich podpora je v nástrojích pro simulaci a verifikaci vyspělá a dobře odladěná. Konečně, jejich pozitivní dopad na kvalitu produktu je možné pocítit ihned po jejich integraci do navrhovaného obvodu.

Problematika verifikace pomocí assertions je příliš rozsáhlá na to, aby mohla být zachycena v jednom příspěvku. Zatímco tento text byl zaměřen na poněkud teoretičtější popis koncepce, výhod a nevýhod assertions, v navazujících textech se budeme věnovat praktičtějším stránkám věci – jak základům jazyka PSL, tak několika případovým studiím použití assertions. Závěrem potom budou shrnuty zásady správné praxe užívání assertions spolu s návodem, jak začít i bez specializovaných nástrojů. Čtenář s hlubším zájmem o verifikaci pomocí assertions může také nalézt některé zajímavé volně dostupné články na stránce [12].

7 Použitá a doporučená literatura

[1] Jakub Šťastný. Od algoritmu k číslicovému obvodu. DPS Plošné spoje od A do Z, číslo 1, pp. 14 – 18, 2012.

[2] Arthur Bloch. Murphyho zákon, Argo, 1999.

[3] Bob Bentley. Validating the Intel Pentium 4 Microprocessor. Proceedings of the 38th annual Design Automation Conference DAC '01, pp 244 – 248, 2001.

[4] Jakub Šťastný. FPGA prakticky, BEN Praha 2011.

[5] Erich Marschner, Bernard Deadman, Grant Martin. IP Reuse Hardening Via Embedded Sugar Assertions.

Proceedings of IP Based SoC Design 2002, pp. 1 – 4.

[6] S. A. Taylor, M. Quinn, D. Brown, N. Dohm, S. Hildebrandt, J. Huggins a C. Ramey. Functional verification of a multiple-issue, out-of-order, superscalar alpha processor - the DEC alpha 21264 microprocessor. In Proceedings of the Design Automation Conference, pp. 638 – 643, 1998.

[7] Brian Bailey. Assertions: Standards Make Proven Technology Portable and Universal, Mentor Graphics white paper, 2004.

[8] Harry Foster, Adam Kolnik, David Lacey. Assertion Based Design, 2nd edition. 2004 Kluwer Academic Publisher.

[9] Charu Aggarwal, Genadi Osowiecki, Shobha Subramanian. A practical guide for deploying assertions in RTL. CDN Live, 2007.

[10] Open Verification Library. http://www.eda.org/ovl/ [kontrolováno 10.9.2012]

[11] ModelSim DE. http://model.com/content/modelsim-de-simulation-and-verification [kontrolováno 10.9.2012]

[12] FPGA Laboratoř. http://amber.feld.cvut.cz/fpga/prednasky/FPGA_navrh/fpga_navrh.html [kontrolováno 10.9.2012]

[13] Don Mills, Stuart Sutherland. System Verilog Assertions are for design engineers, too! In SNUG San Jose 2006.

[14] Yunfeng Tao. An Introduction to Assertion-Based Verification. Proceedings of the ASICON '09. IEEE 8th International Conference on ASIC. pp. 1318 – 1323, 2009.