

Verifikace číslicových obvodů

Marcela Šimková, Michal Kajan

Fakulta informačních technologií
Vysoké učení technické v Brně



6.12.2012

Motivace

„The design and testing of an advanced microprocessor chip is among the most complex of all human endeavors.“

-- John Barton (Intel vice-president)

„It has been observed that verification becomes a major bottleneck in hardware design development, up to 80% of the overall development cost and time.“

-- R. Drechsler et al.: Advanced Formal Verification

Stále platí **Moorův zákon**: složitost integrovaných obvodů se zdvojnásobuje každých 24 měsíců.

Se složitostí obvodu roste až exponenciálně **složitost verifikace !!!**

→ více logiky na čipu, složitější funkce a chování systému, časové závislosti, atd.

Verifikace

Snaha odhalit co nejvíce chyb již v počátečních fázích návrhu hardwarových systémů

- selhání bezpečnostních systému,
- ztráty na životech (automobilový a letecký průmysl),
- cena produktu (1994 – chyba ve floating-point aritmetice procesoru Intel Pentium, ztráty 420 mil. dolarů, 2007 – série chyb v procesoru AMD Barcelona)

Verifikace – proces ověřování, zda je implementovaný systém (v HDL) korektní, neobjevují se nesrovnalosti a chyby.

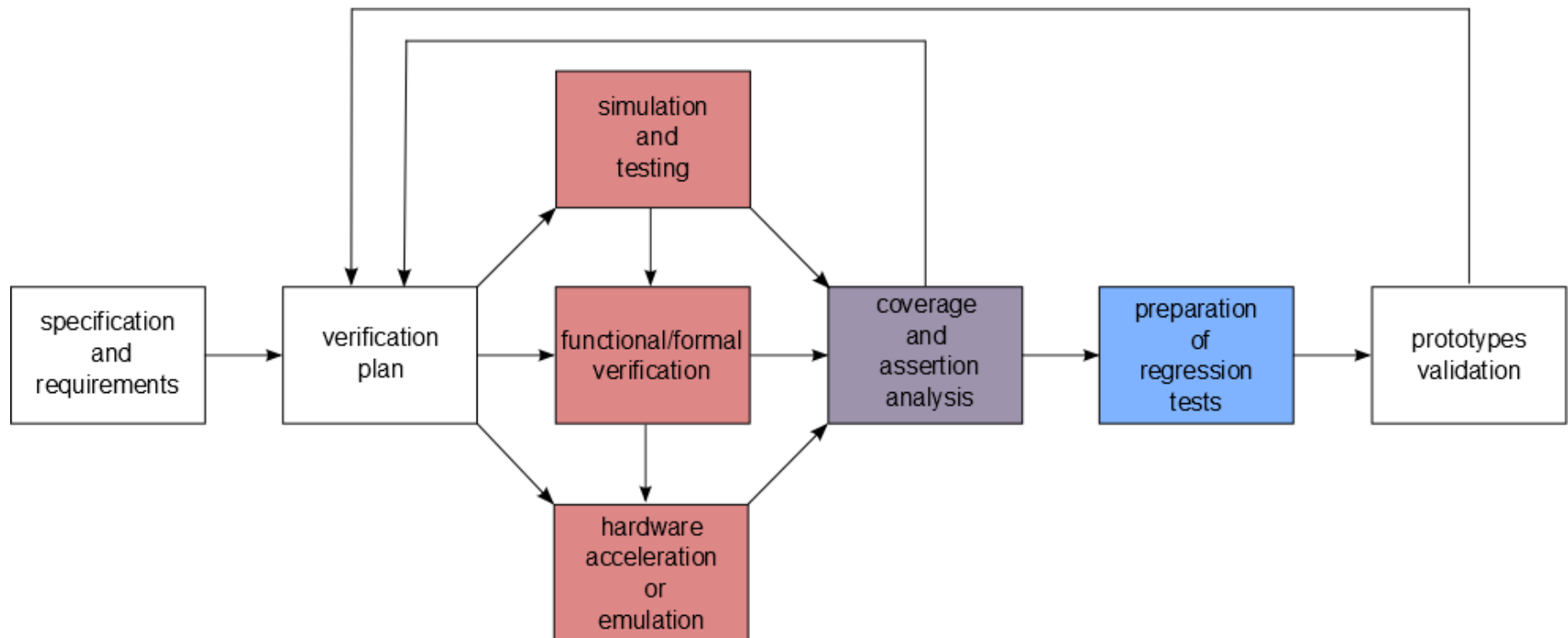
- Výstup je v souladu s jistou interpretací specifikace.
- Označuje se i pojmem *pre-silicon testing*.

Validace – proces ověřování, zda je interpretace systému a její implementace správně svolená, zda se jedná skutečně o systém popsany v specifikaci.

- Jsou implementovány všechny požadované funkce, systém odpovídá požadavkům uživatelů.
- Označuje se i pojmem *post-silicon testing*.

Pre-silicon verifikace

Typicky běží v simulačním prostředí.



Specifikace a požadavky

Specifikace a požadavky jsou typicky ve formě dokumentu, kde jsou popsány:

- Rozhraní (vstupní, výstupní, sběrnice, ...).
- Funkce obvodu na vyšší úrovni abstrakce (aritmetické operace, zpracování instrukci, výpočet kontrolního součtu, ...).
- Formát vstupů a výstupů (formát paketu, paměťové operace, ...).
- Bloková schéma obvodu na nižší úrovni abstrakce.
- Časování a frekvence obvodu.
- Požadavky na paměť.
- A jiné.

Verifikační plán

Verifikační plán řeší dvě základní otázky:

1. **Co verifikujeme?**
2. **Jak to budeme verifikovat?**

Odpovědi zahrnují především:

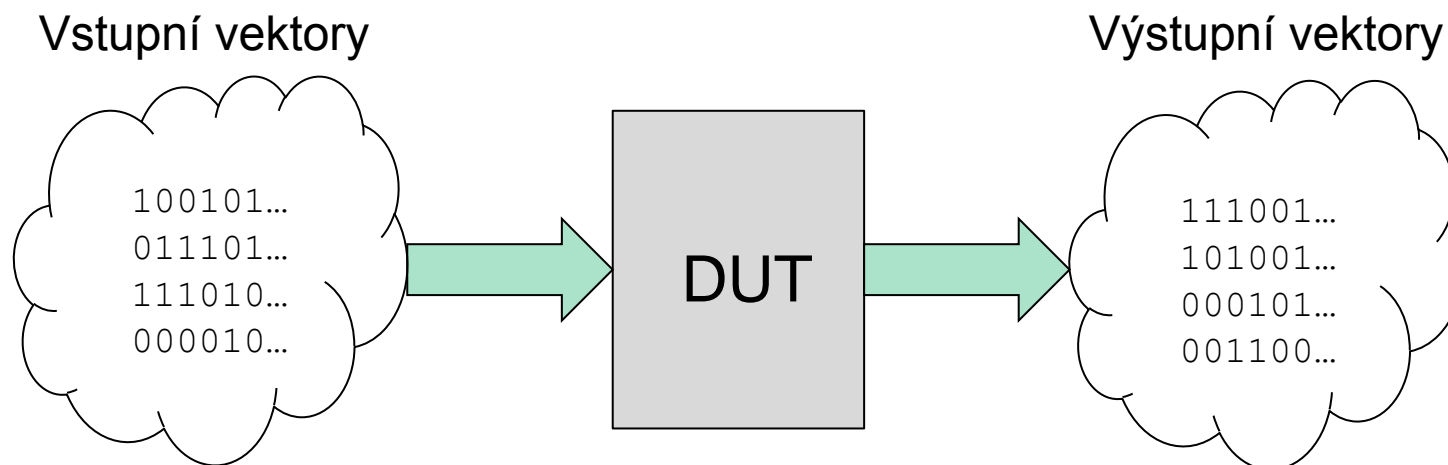
- Specifické testy, verifikační přístupy a nástroje (SW, HW), které se použijí.
- Jak se pozná, kdy je verifikace ukončena.
- Zdroje – lidské, výpočetní, odhad ceny.
- Seznam ověřovaných funkcí na jednotlivých úrovních abstrakce obvodu.
- A jiné.

Simulace a testování

Označují se i pod pojmem *bug hunting* metody.

Využívají prostředí simulátoru (např. *ModelSim*), které umožňuje reprodukovat chybové scénáře a ladit chyby.

- Jednoduchý přístup, ale **nedokazuje** že systém již neobsahuje další chyby!
- Neprověří se všechny stavy a každé možné chování systému.



DUT = *Device Under Test*, ověřovaný systém.

Formální verifikace

Využívá matematických metod k formálnímu popisu systému nebo jeho vlastností a ověřuje, zda funkčnost systému je v souladu se specifikací.

Výsledkem je:

Důkaz správnosti (angl. *proof*) – neexistuje žádný vstup, který by způsobil porušení sledované podmínky.

nebo

Protipříklad (angl. *fire, counter-example*) – určitý běh systému, nebo vstup, který vede k porušení sledované podmínky.

Formální verifikace - techniky

Nejznámější techniky formální verifikace:

- **Model Checking** – ověřuje vlastnosti systému (*properties*) úplným prozkoumáním jeho stavového prostoru. Verifikovaný systém je typicky reprezentován konečným automatem nebo jeho variantami.
- **Statická analýza** – automatická analýza zdrojového kódu, nevyžaduje model systému, používá se i pro optimalizaci a generování kódu.
- **Theorem Proving** – deduktivní metoda, podobná matematickému dokazování.
- **Equivalence Checking** – formálně dokazuje, že dvě reprezentace obvodu mají stejné chování (např. na různých úrovních abstrakce).
- **SAT Solving** – SAT (*satisfiability*) reprezentuje NP-úplný problém splnitelnosti Booleovských formulí, tj. zda existuje také přiřazení hodnot proměnným v Booleovské formuli, že formule je ohodnocena jako TRUE.

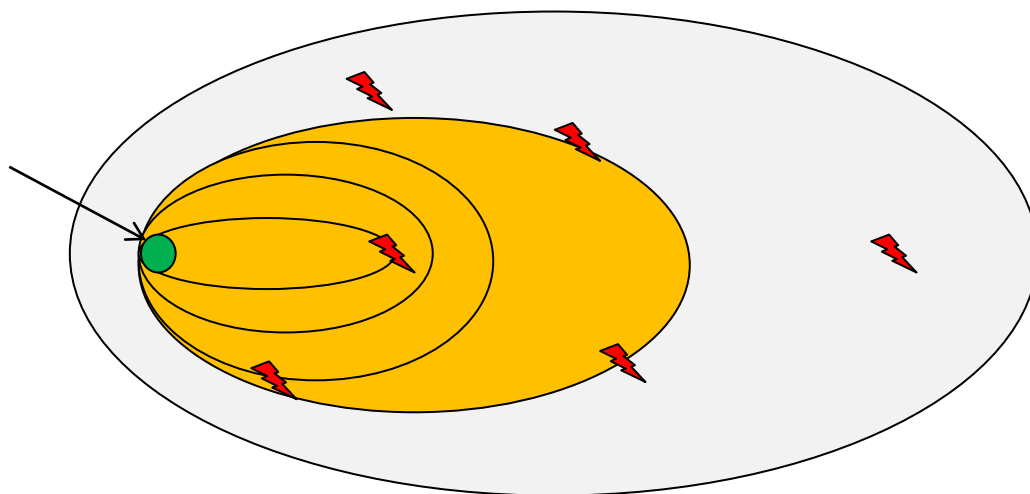
Formální verifikace - problémy

Oblast formální verifikace zaznamenává prudký vývoj, ale má stále jisté nedostatky:

- často negarantuje konečnost → použití aproximace způsobuje falešné alarmy,
 - problém stavové exploze (Model Checking),
 - vyžaduje interakci s člověkem,
 - odpovědi typu don't know.
- I když celkový běh formální verifikace selže, může objevit chyby!

Formální verifikace - problémy

Pokryvání stavového prostoru verifikovaného systému za běhu Model Checkingu:



Problém: stavová exploze při verifikaci komplexních systémů a z toho plynoucí nedostatek paměti.

Funkční verifikace

Ověřuje, zda **model obvodu** (nebo syntetizovaná struktura) plní specifikaci, a to sledováním jeho vstupů a výstupů **v simulaci**.

Využívá přídatné techniky, čímž výrazně zvyšuje efektivitu samotné simulace:

- **generování náhodných vstupních vektorů** (*constrained-random stimulus generation*),
- **verifikace řízená pokrytím** (*coverage-driven verification*),
- **verifikace založená na formálních tvrzeních** (*assertion-based verification*),
- **samokontrolní mechanismy** (self-checking mechanisms).

Implementace v jazyku SystemVerilog (principy OOP).

SystemVerilog

Objektově orientovaný jazyk, speciálně určen pro tvorbu verifikačních prostředí (*hardware verification language* - HVL) i implementaci hardwarových systémů.

Standard IEEE 1800-2009.

Založen na prvcích jazyka Verilog, ale zahrnuje v sobě výhody různých programovacích jazyků s důrazem na simulaci a verifikaci.

Umožňuje vytvářet testy na vysoké úrovni abstrakce.

Přináší mnoho nových datových typů.

Poskytuje specializované rozhraní **DPI** (*Direct Programming Interface*) pro volání funkcí zapsaných v jiných programovacích jazycích (v současnosti je podporován pouze jazyk C) z prostředí jazyka SystemVerilog a naopak.

Výhodou je zakomponování již existujících částí testbenchů napsaných v jazyce C do verifikačního prostředí v jazyce SystemVerilog.

Generování náhodných vstupů

Otestování systému na velkém množství náhodných vstupů.

Korektnost formátu těchto vstupů (ve funkční verifikaci označovaných jako **transakce**) je zajištěna pomocí **omezujících podmínek** (*constraints*):

- vyžadováno, když transakce musí dodržovat formát protokolu vstupního rozhraní,
- vyžadováno, když má transakce pokrýt určitou hraniční situaci nebo konkrétní stav.

Příklad:

```
// Randomization parameters
int packetCount      = 3;
int packetSizeMax[] = '{32,32,32}';
int packetSizeMin[] = '{8,8,8}';

// Randomized transaction data [packet][byte]
rand byte unsigned data[][];

// -- Constraints --
constraint c1 {
    data.size == packetCount;
    foreach (data[i]) data[i].size inside
        {[packetSizeMin[i]:packetSizeMax[i]}};
};
```

Verifikace řízená pokrytím

Statistika o pokrytí reflektuje, které vlastnosti, stavy systému byly během verifikace řádně prověřeny.

Průběžně měří pokrok a produktivitu verifikačního procesu.

Typy sledovaného pokrytí:

- **funkční pokrytí** (*functional coverage*) - pokrytí funkcí a chování systému,
- **pokrytí kódu** (*code coverage*) - provedené kódové konstrukce,
- **pokrytí cest** (*path coverage*) - provedené cesty v systému,
- **FSM pokrytí** (*FSM coverage*) - navštívené stavy konečného automatu,
- a jiné.

Techniky pro dosažení 95% až 100% pokrytí:

- a) ručně, pomocí cílených přímých testů (*directed tests*),
- b) automaticky, kdy inteligentní program kontroluje statistiku o pokrytí a na základě toho řídí generování vhodných vstupů (výběr parametrů, počáteční hodnoty generátoru).

Verifikace založená na formálních tvrzeních

Formální tvrzení (*assertion*) – tvrzení (výraz v temporální logice), které musí v daném systému vždy platit.

Formální vyjádření vlastností systému, očekávaných operací, vnitřní synchronizace.

Kontrola dodržení protokolů vstupních a výstupních rozhraní, křížení hodinových domén, stavových automatů.

Při porušení assertionu je verifikace přerušena a je reportována chyba → rychlá lokalizace zdroje problému.

Nejpoužívanější jazyky pro definici assertionů:

- *PSL (Property Specification Language)*,
- *SVA (SystemVerilog Assertions)*.

Samokontrolní mechanismy

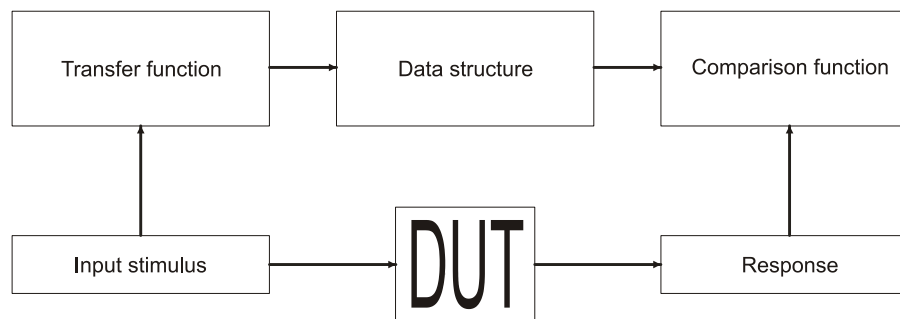
Predikce výstupů systému → transformace vstupních vektorů na výstupní nezávisle od provedené implementace, podle specifikace systému.

Automatická kontrola skutečných výstupů (typicky z výstupních rozhraní) s očekávanými.

Technika scoreboardingu:

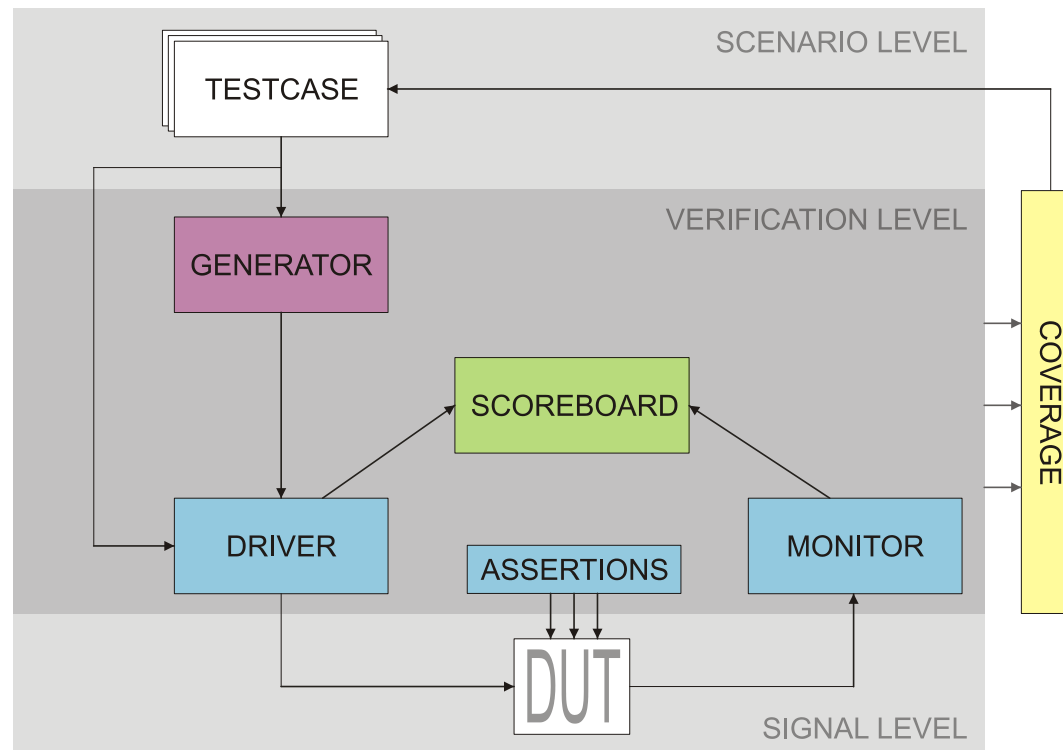
Převodová funkce (*transfer function*) provádí všechny transformace vstupů na výstupy dle specifikace, výsledek ukládá v paměti v podobě **datové struktury** a vytváří tak očekávaný výstup.

Srovnávací funkce (*comparison function*) ověřuje, zda se skutečný výstup systému v podobě transakce shoduje s některým z očekávaných výstupů uložených v datové struktuře.



Verifikační prostředí

Základní vrstvy a komponenty verifikačních prostředí (testbenchů):



Verifikační metodiky

Metodologie – vědecká disciplína zabývající se metodami, jejich tvorbou a aplikací.

Metodika – souhrn praktik a postupů.

Verifikační metodiky specifikují, jak vytvářet **znovupoužitelné** a **snadno rozšířitelné** verifikační prostředí v jazyce SystemVerilog.

Preferují tzv. *vrstvený testbench* (*layered testbench*) a verifikaci řízenou pokrytím.

Umožňují použít předpřipravené komponenty verifikačních prostředí definované v podobě **knihoven**.

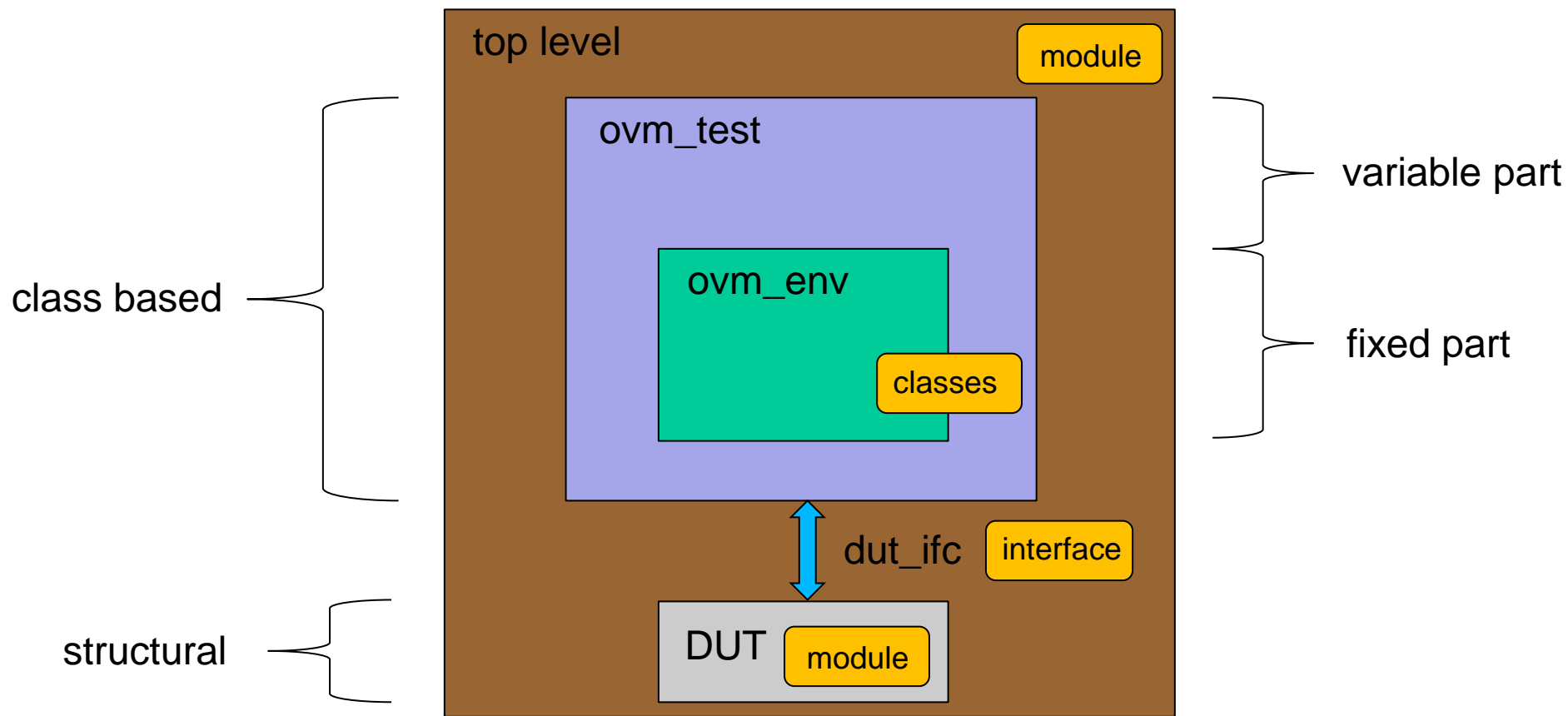
OpenSource licence.

Nejznámější metodiky:

- *Verification Methodology Manual (VMM)* – ARM, Synopsys.
- *Open Verification Methodology (OVM)* – Cadence, Mentor Graphics.
- *Universal Verification Methodology (UVM)* – Acclera.

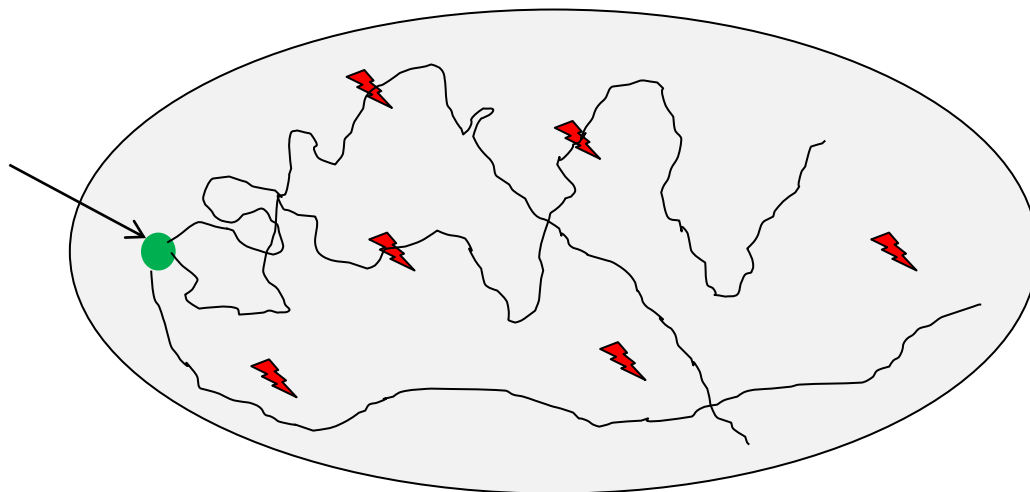
OVM metodika

Vytvořena ve spolupráci firem Cadence a Mentor Graphics.
Knihovna tříd v SystemVerilogu.



Funkční verifikace - problémy

Pokrývání stavového prostoru verifikovaného systému za běhu funkční verifikace:

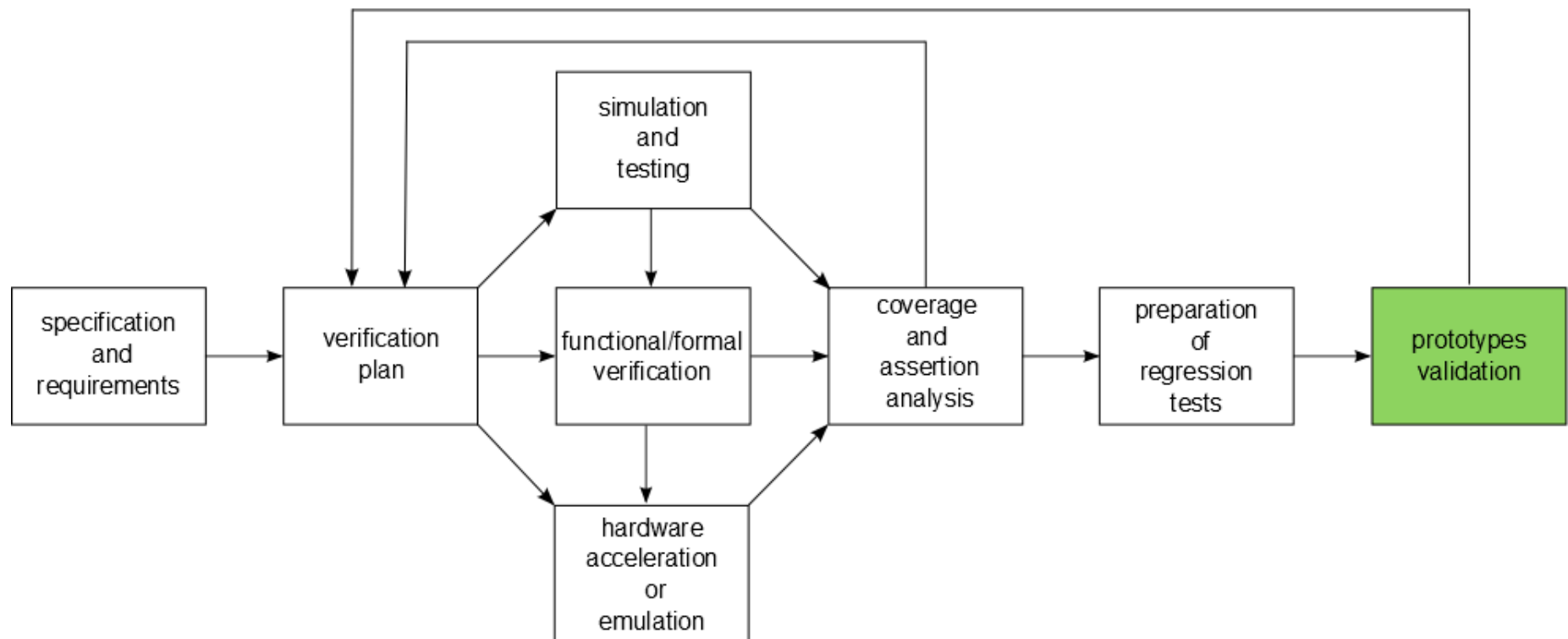


Problém: vstupní transakce typicky nepokryjí celý stavový prostor verifikované jednotky.

Verifikaci považujeme za ukončenou, když je dosažena jistá úroveň pokrytí (námi požadovaná úroveň spolehlivosti ve funkčnosti).

Post-silicon validate

Validace fyzické reprezentace (prototypu) číslicového obvodu.



Post-silicon validace

Ve fyzickém obvodu mohou být přítomny další chyby!

- Chyby, které vznikly ve výrobě.
- Chyby zaneseny v čase syntézy, place&route.
- Funkční chyby, které se projeví až při běhu v hardware, např. související s vysokou rychlostí obvodu.

Aplikace testů by měla odpovídat **pracovní rychlosti obvodu** (*at-speed testing*) a neměla by vyžadovat žádné změny v architektuře obvodu → vliv na časování, plochu, frekvenci...

Tohle je však značně náročné a především drahé.

Dalším problémem je **ladění chyb** – není k dispozici pohodlné ladící prostředí simulátoru, kde vidíme hodnoty vnitřních signálů a kde můžeme jednoduše řídit test!

Užitečné zdroje a odkazy

- IEEE Computer Society. IEEE Std 1800-2009: *IEEE Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language*, 2009, ISBN: 978-0-7381-6129-7.
- Spear, Chris: *SystemVerilog for Verification*, 2nd Edition, Springer, New York, 2008, ISBN 978-0-387-76529-7.
- Glasser, Mark: *Open Verification Methodology Cookbook*, Springer, New York, 2009, ISBN 978-1-4419-0967-1.
- Verification Academy [online]. <<http://verification-academy.mentor.com/>>
- Questa Formal Verification [online]. <http://www.mentor.com/products/fv/0-in_fv/>
- UVM World [online]. <<http://www.uvmworld.org/>>
- HAVEN - Hardware-Accelerated Verification ENvironment [online]. <<http://www.fit.vutbr.cz/~isimkova/haven/>>

Děkuji za pozornost!