Verifikace digitálních obvodů

BPC-NDI 2022

Autor: Vojtěch Dvořák

26. 10. 2022

Osnova

- Funkční verifikace
  - Jak generovat testovací vektory?
  - Jak kontrolovat správnost výstupů?
  - Kdy ukončit verifikaci?
- Verifikační plán
- Verifikační prostředí
- Verifikace AAU

Co je verifikace?

- = Ověření správnosti (porovnání s požadavky, vzorem, referenčním modelem)
- V průběhu návrhu obvodu (analogové i digitální) může probíhat několik různých verifikačních procesů
  - Funkční verifikace
  - Formální verifikace (LEC po syntéze a vložení DFT, LVS)
  - Fyzická verifikace (layout – DRC)
- Verifikace vs. Validace:
  - „Dělá obvod to, co se od něj očekává?“

    vs.
  - „Jsou požadované funkce obvodu užitečné?“

Proč verifikovat?

- Bug = chyba v návrhu, kdy funkce obvodu neodpovídá specifikaci
- Nalezení bugů před výrobou -> je levnější simulovat než vyrábět
- Snadnější odstranění bugů pomocí simulací
- Ověření funkčnosti celého systému
- Pochopení funkčnosti celého systému
- Předvídatelnost průběhu vývoje obvodu

Funkční verifikace

- U moderních systémů může být počet možných testů (resp. vstupních kombinací) větší než 10^{150}, není možné testovat všechny kombinace -> Funkční verifikace
- Ověřuje se funkce obvodu, vstupní data jsou pouze takové kombinace, které jsou pro chování obvodu „zajímavé“ - snaha simulovat skutečnou činnost obvodu
- Jazyky – SystemVerilog, Vera, Specman, ale i VHDL

Funkční verifikace

- *Jak generovat vstupní data?*
  - Jak dostat DUT do všech možných stavů?
  - Jak ověřit všechny požadované funkce systému?
- *Jak kontrolovat správnost výstupů?*
  - Odkryla generovaná data bug v návrhu?
- *Kdy ukončit verifikaci?*
  - Je celý systém bez bugů?
  - Byly ověřeny všechny požadované funkce systému?

Ukončení verifikace - pokrytí

- = Coverage
- Informuje, jak velká část DUT byla testována
- Měřeno simulátorem (např. Questa)
- Pokrytí kódu (*Code Coverage*)
  - ukazuje, které části RTL byly testovány
  - Line Coverage (pokrytí řádků)
    - Uvádí, které řádky kódu byly vykonány (a kolikrát)
    - Po verifikaci musí být 100%
  - Branch Coverage (pokrytí větví)
    - Uvádí, zda byly provedeny obě větve příkazů *if-else* a *when-else*
    - Po verifikaci musí být 100%

Ukončení verifikace - pokrytí

- Pokrytí kódu (*Code Coverage*)
  - Expression Coverage (pokrytí výrazů)
    - Uvádí, zda bylo u všech výrazů použito všech možných pravdivostních hodnot
    - Po verifikaci nemusí být 100% (mnohdy nemožné dosáhnout)
  - FSM Coverage (pokrytí stavových automatů)
    - Uvádí, zda byly navštíveny všechny stavy FSM a realizovaný všechny možné přechody mezi stavy
    - Po verifikaci musí být 100%
  - Pokud není dosaženo 100% - nedostatečné testy nebo mrtvý kód

Ukončení verifikace - pokrytí

- Funkční pokrytí (*Functional Coverage*)
  - Informuje, kolik funkčních požadavků (tzv. body pokrytí, coverpoint) bylo úspěšně ověřeno
  - Každý bod pokrytí musí být splněn alespoň jednou
  - Měření funkčního pokrytí ručně (např. dokumentace) nebo pomocí speciálních nástrojů (např. Questa Verification Management)

  - Verifikace končí, pokud je funkční pokrytí 100%

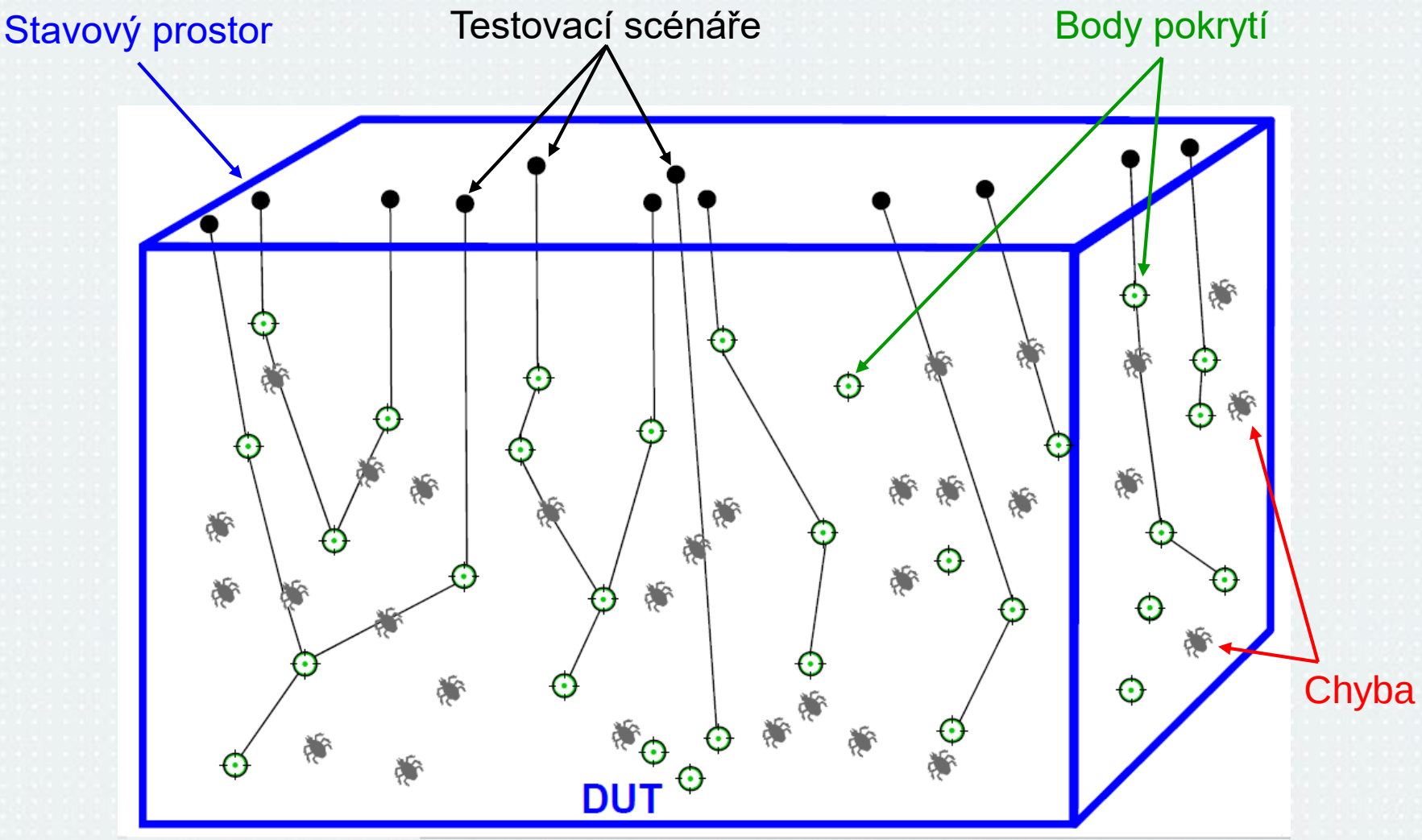
Generování vstupních dat

- Různé přístupy ke generování vstupních dat
  - Přímé testy *(Direct stimuli test)*
  - Testy s náhodnými daty *(Constraint-random stimuli test*)
  - Verifikace řízená pokrytím *(Coverage-driven verification, Metric driven verification)*
- Každý přístup má své výhody a nevýhody
- Dnešní trend je používat verifikaci řízenou pokrytím

Přímé testy

- Testovací data jsou vytvořena verifikačním inženýrem, snaha co nejrychleji přivést DUT do požadovaného stavu
- Výhody
  - Nejrychleji vytvořený test
  - Lze s ním otestovat každou funkci (i ty, které nebyly ověřeny testem s náhodnými čísly)
- Nevýhody
  - Otestováno pouze chování obvodu, které je očekávané
  - Každá změna specifikace vyžaduje pracně upravit simulační scénář

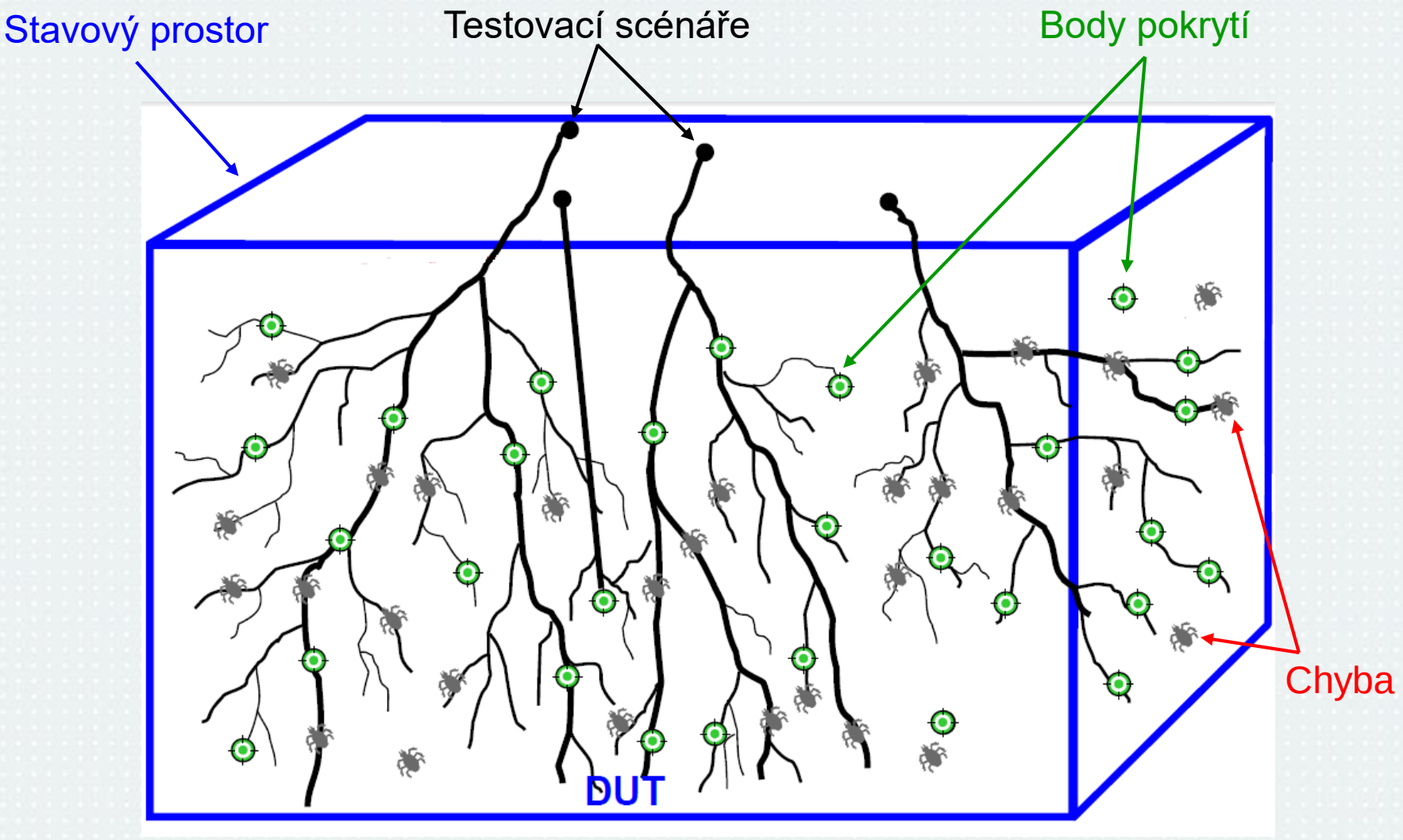
Přímé testy



Testy s náhodnými daty

- Testovací data jsou generována pomocí funkce *random*, je však nastaven interval hodnot nebo pravděpodobnost výskytu jednotlivých hodnot či intervalů
- Funkce *random* není náhodná – algoritmus, který z jedné hodnoty (*seed*, semínko) vytvoří jinou
- Výhody
  - Pokrytí i nepředpokládaných chyb
  - Lze rychleji dosáhnout funkčního pokrytí 100%
- Nevýhody
  - Další a pracnější vývoj verifikačního prostředí
  - Některé funkce obvodu se nepodaří otestovat

Testy s náhodnými daty



Kontrola výstupů

- Ručně
  - Možné jen u malých obvodů a přímých testů (v podstatě jen simulace)
- Automaticky
  - Monitor
    - Sleduje výstupy jednoho nebo více funkčních bloků
    - Kontrola výstupů na signálové úrovni a časování signálů
    - Součást BFM (Bus Functional Model)

Kontrola výstupů

- Automaticky
  - Referenční model
    - Abstraktní popis obvodu nebo části jeho funkce
    - Obvykle popsán ve vyšším programovacím jazyce (Matlab, C++, SystemC, ...)
    - Generuje stejné výstupy jako DUT v daném stavu
    - Používá se pro kontrolu obsahu registrů, matematických výpočtů, složitých algoritmů
  - Assertions (*„tvrzení“*)
    - Popis platné sekvence událostí či signálů
    - Součástí VHDL kódu (PSL) nebo samostatně (SVA)
    - Při porušení „tvrzení“ simulátor vypíše chybu

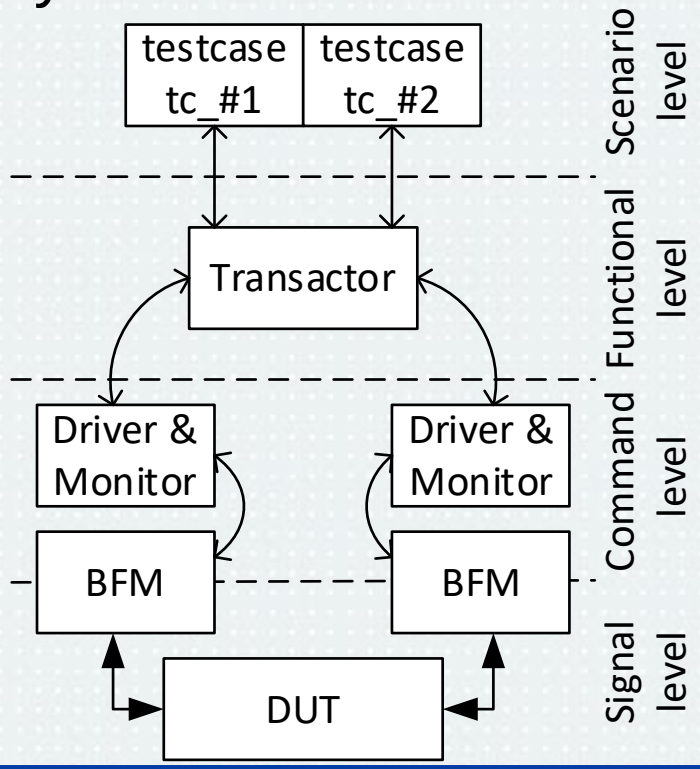
Verifikační plán

- Obsahuje informace o tom, co (bod pokrytí, coverpoint) a jak má být verifikováno
- Vytvořen verifikačním týmem na základě specifikace
- Nezávislý na obvodové implementaci
- Struktura dokumentu dána interními předpisy
  - Referované dokumenty (minimálně specifikace)
  - Seznam verifikačních nástrojů
  - Verifikační matice se seznam bodů pokrytí
  - Verifikační metodika (přímé testy, UVM, VVM, …)
  - Stručný popis testů

- *Ukázka – AAU závěrečná zpráva*

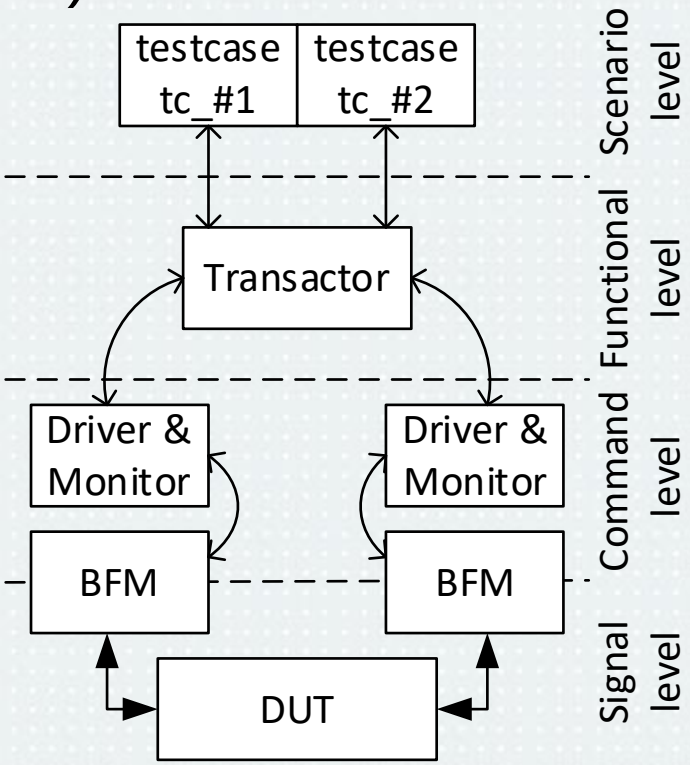
Verifikační prostředí

- Rozdělení na úrovně (Layered testbench)
  - Snadnější úprava při změně specifikace
  - Snadnější oprava chyb
  - Znovupoužití (*reuse*) mezi projekty a mezi různými testy
- Použití ve všech moderních verifikačních metodikách (VMM, UVM), liší se v názvech bloků
- Struktura dle VMM (zjednodušena)



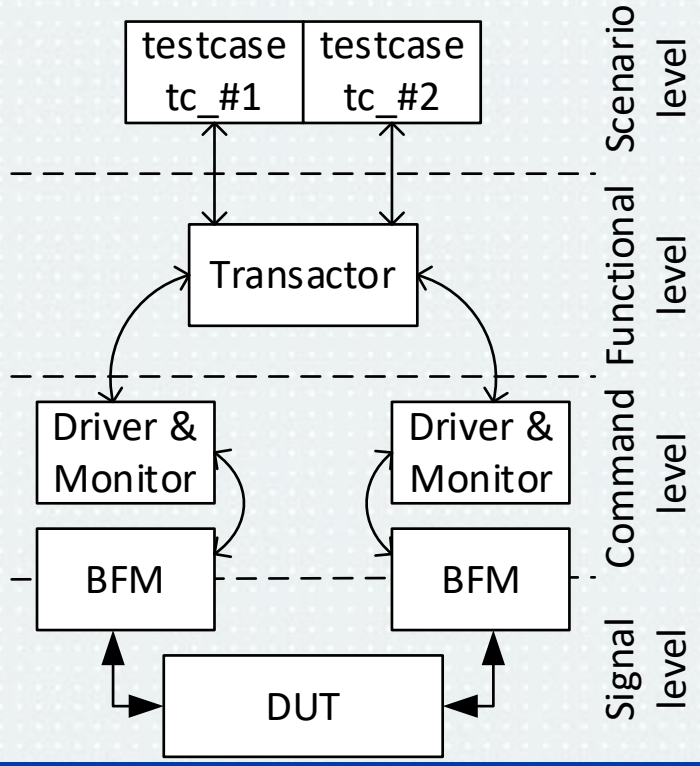
Verifikační prostředí

- Signálová úroveň *(Signal Level)*
  - Událost – změna signálů
  - Datové typy – std_logic, std_logic_vector
  - Obsahuje DUT (*Design Under Test*) a BFM (*Bus Functional Model*)
  - BFM – řízení vstupů DUT a sledování výstupů
  - BFM ovládáno z vyšší úrovně (*BFM Driver*)
    Definice časování (příkaz *wait*)



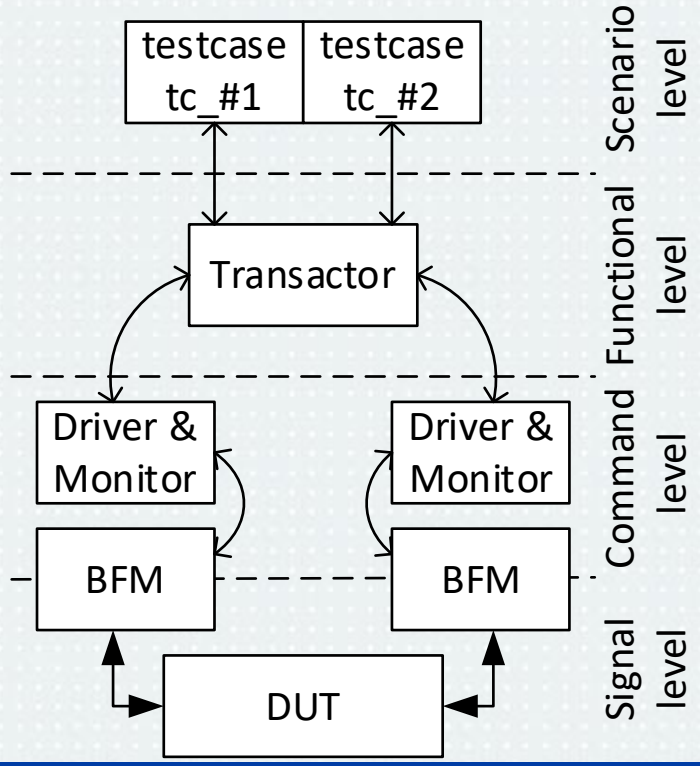
Verifikační prostředí

- Příkazová úroveň (*Command Level*)
  - Událost – operace, vykonání procedury
  - Datové typy – integer → std_logic_vector
  - Obsahuje jeden či více *Driver* a *Monitor* pro řízení BFM
  - Příklad: procedure *send_frame*
    - Spustí BFM s požadovanou konfigurací (daty)
    - Čeká na odpověď od BFM, že operace byla vykonána
  - *Ve VHDL bude BFM řízeno signály !!!*



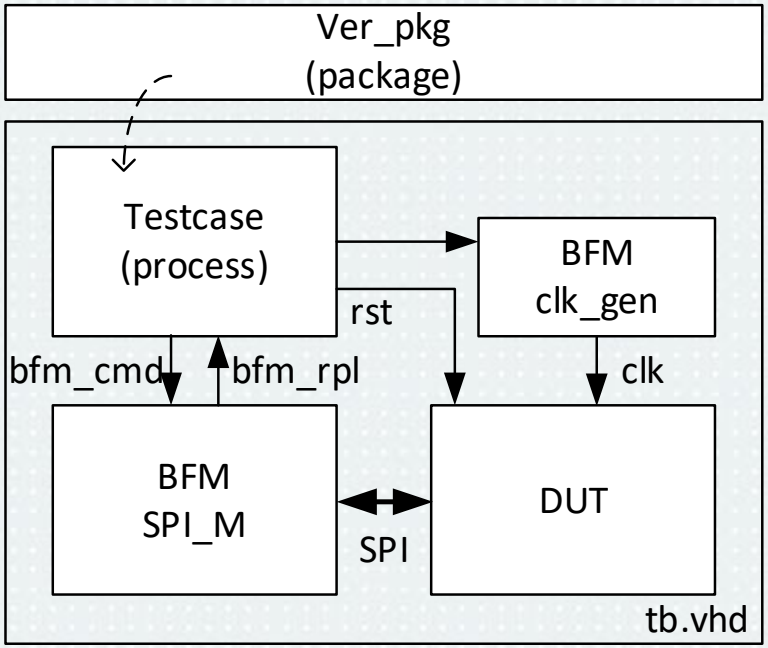
Verifikační prostředí

- Funkční úroveň (*Functional Level*), Simulační scénář (*Scenario level*)
  - Událost – vykonání transakce
  - Transactor (Sequencer) – posloupnost operací
  - Příklad: procedure *send_packet*
    - Požádá Driver o postupné odeslání rámců (→ paket) dle nastavení od scénáře
    - Potvrzení od Driveru, že operace byla vykonána
  - Scénáře simulace
    - Konfigurace (data) pro ověření splnění požadavku



Verifikační prostředí - projekt

- Pro verifikaci AAU lze zjednodušit verifikační prostředí
  - BFM SPI_M – SPI Master generuje SPI signály pro DUT, ovládán přímo z procesu scénáře
  - BFM clk_gen – proces pro generování hodinového signálu
  - Testovací scénář – proces, využívá procedury ze slohy pro řízení BFM (Driver) – odeslání jednoho rámce, odeslání paketu



Verifikační prostředí - projekt

- BFM SPI_M
  - Vstupy (*record*): data k odeslání (jeden rámec), spuštění odeslání, nastavení generované chyby
  - Výstupy (*record*): přijatá data (jeden rámec), konec rámce

```
use work.aau_ver_pkg.all;

entity bfm_spi_m is
   port (
      -- SPI signals

      -- CMD & RPL IF
      bfm_cmd : in  t_BFM_CMD;
      bfm_rpl : out t_BFM_RPL
   );
end bfm_spi_m;
```

Ukázkový kód je neúplný a pravděpodobně nesprávný

Verifikační prostředí - projekt

- BFM SPI_M
  - Generování průběhů SPI signálů a načítání dat - jeden rámec

```
process begin
   wait until rising_edge(bfm_cmd.start);
   CS_b <= '0';
   for idx in 0 to 15 loop
      SCLK <= '0';
      MOSI <= bfm_cmd.data(idx);
      wait for 1 us;
      SCLK <= '1';
      bfm_rpl.data(idx) <= MISO;
      wait for 1 us;
   end loop;
   CS_b <= '1';
   bfm_rpl.done <= '1';
end process;
```

Ukázkový kód je neúplný a pravděpodobně nesprávný

Verifikační prostředí - projekt

- Verifikační sloha
  - Definice datových typů pro porty BFM SPI_M
  - Definice procedury pro odeslání jednoho rámce (a následně i dalších procedur

```
procedure task_send_frame (variable frame_cmd : in  <type>;
                           variable frame_rpl : out <type>;
                           signal bfm_rpl : in  t_BFM_RPL;
                           signal bfm_cmd : out t_BFM_CMD) is
  begin
       bfm_cmd.data    <= frame_cmd;
       bfm_cmd.start   <= '1';
       wait until rising_edge(bfm_rpl.done);
       frame_rpl := bfm_rpl.data;
  end procedure;
```

Ukázkový kód je neúplný a pravděpodobně nesprávný

Verifikační prostředí - projekt

- Scénář simulace
  - Sekvence operací pomocí volání procedur ze slohy ovládající BFM

```
p_tc_1 : process
   variable data1, data2 : <type>;
   begin
      -- perform reset
      -- generate data
      data1 := 1;
      -- send frame
      task_send_frame(data1, data2, bfm_cmd, bfm_rpl);
      -- send packet
      task_send_packet(pkt_send, pkt_rcv, bfm_cmd, bfm_rpl);
      -- stop simulation
      clk_gen_en <= '0';
      wait;
   end process;
```

Ukázkový kód je neúplný a pravděpodobně nesprávný

děkuji za pozornost

(a pište komentáře)