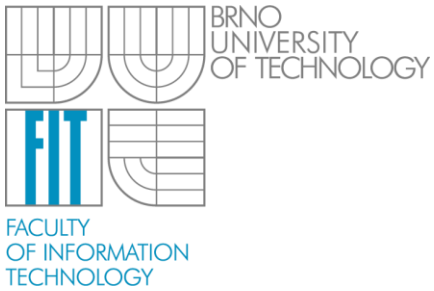
Verifikace číslicových obvodů

Michal Kajan

Fakulta informačních technologií, Vysoké učení technické v Brně,
Božetěchova 2, 612 66 Brno



Tento materiál vznikl za podpory Fondu rozvoje vysokých škol (projekt 1798/2012).

Pokročilé číslicové systémy

Osnova

- verifikace počítačových systémů
- návrhový proces
- SystemVerilog
- Metodika OVM
- 0-In Formal Verification

Motivace

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

-- John Barton (Intel vice-president in charge of product testing)

To ensure that its products are as error-free as possible, Intel, based in Santa Clara, Calif., now spends a half-billion dollars annually in its factories around the world, testing the chips for more than a year before selling them.

Even the slightest design error in the chip could end up being a billion-dollar mistake.

- ⇒ 1994, floating-point calculation error in Pentium, costs: $420 million
- ⇒ 2007, AMD's Barcelona processor – series of bugs, severe impact on revenue

chip complexity: Intel 8088 (1981): 29 000 transistors, Nehalem (2008): 731 million transistors!

New York Times, 16.8.2008

Verifikace

Verifikace – proces ověřování, zda daný systém splňuje zadané požadavky
- výstup je v souladu se specifikací

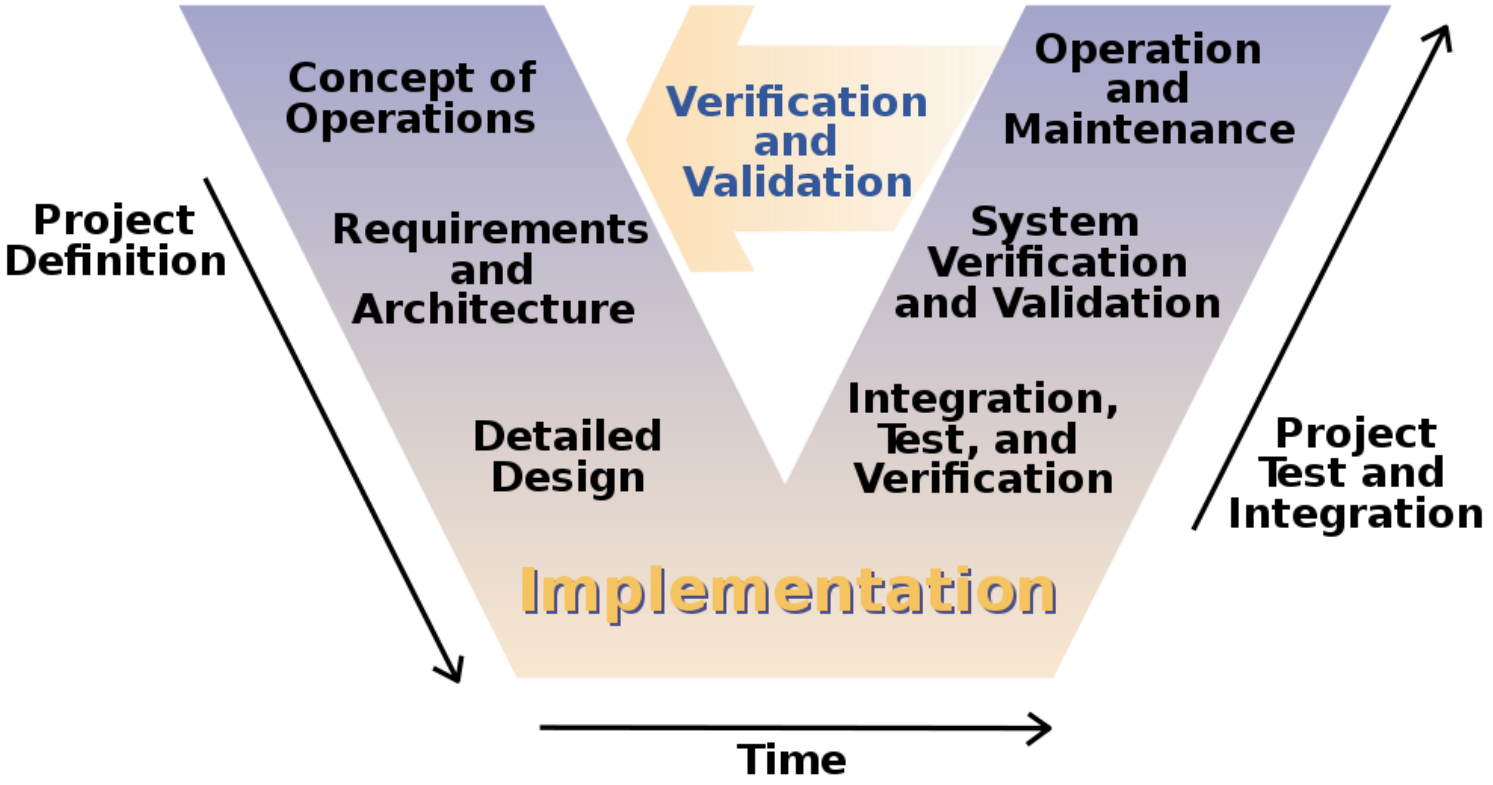
Validace – ověřuje faktickou správnost výsledku
- výstup odpovídá požadavkům uživatelů

Verifikace v oblasti návrhu počítačového systému:

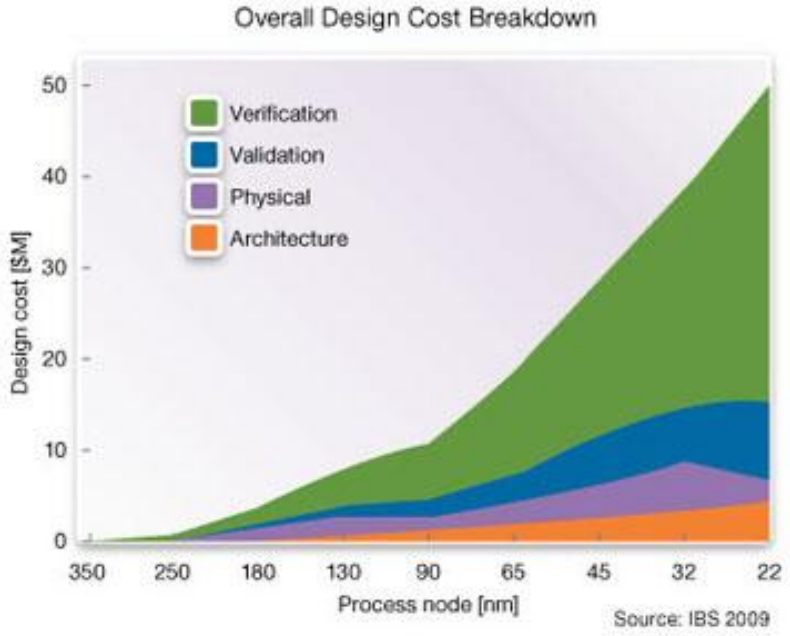
Funkční verifikace – ověřování, zda model obvodu (nebo syntetizovaná struktura) plní funkci dle specifikace

Formální verifikace – využívá matematických metod k formálnímu popisu specifikace nebo sledované vlastnosti k ověřování, zda funkčnost systému je v souladu se zadaným formálním popisem (výsledkem je **důkaz** správnosti, angl. **proof** nebo **protipříklad, angl. *fire, counter-example***)

Vývojový proces



Verifikace v procesu vývoje systému

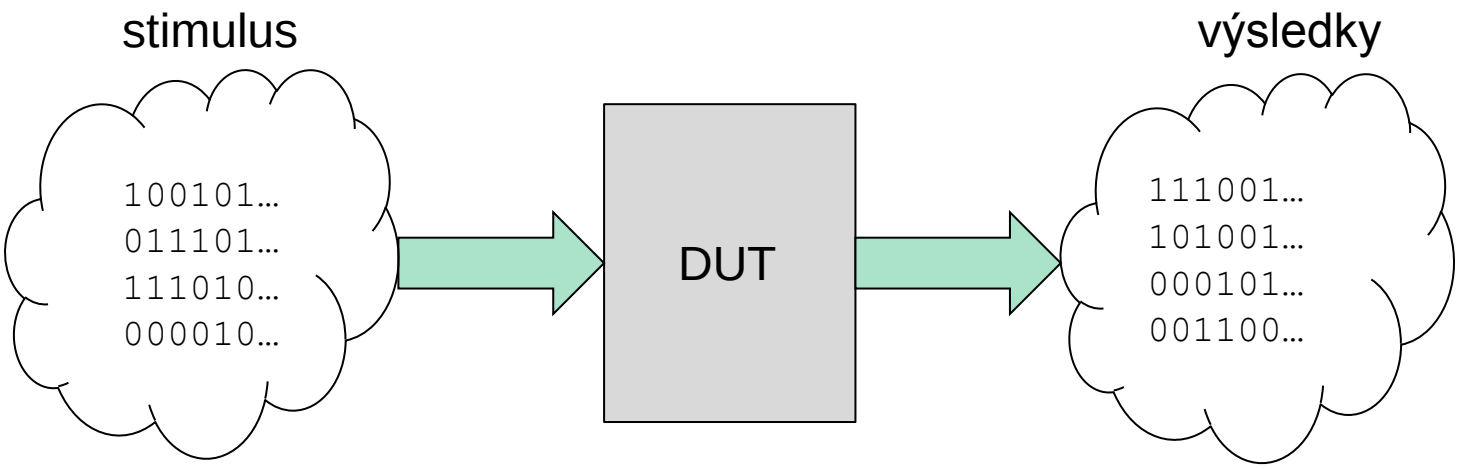


- funkční verifikace se stala nejkritičtější a nejtěžší fází v procesu vývoje
- současné systémy dosahují vysoké složitosti – exponenciální nárůst prostoru pro verifikaci → je nemožné ověřit každou funkci

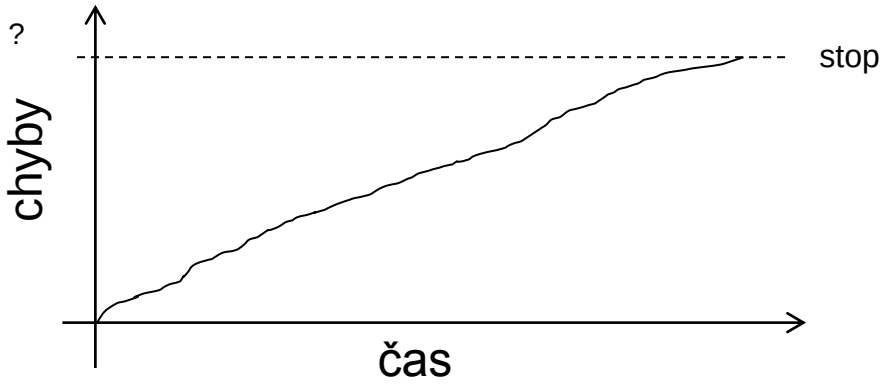
- verifikace klade požadavky na výpočetní infrastrukturu a lidské zdroje (pro velké projekty až tisíce serverů, terabajty verifikačních dat, …)
- verifikace se stává (stala) nejnákladnější a nejdelší částí v procesu vývoje

- verifikaci nikdy nemožno považovat za ukončenou – je potřeba stanovit dostačující úroveň pokrytí (požadovaná úroveň spolehlivosti ve funkčnosti)

Simulace a verifikace



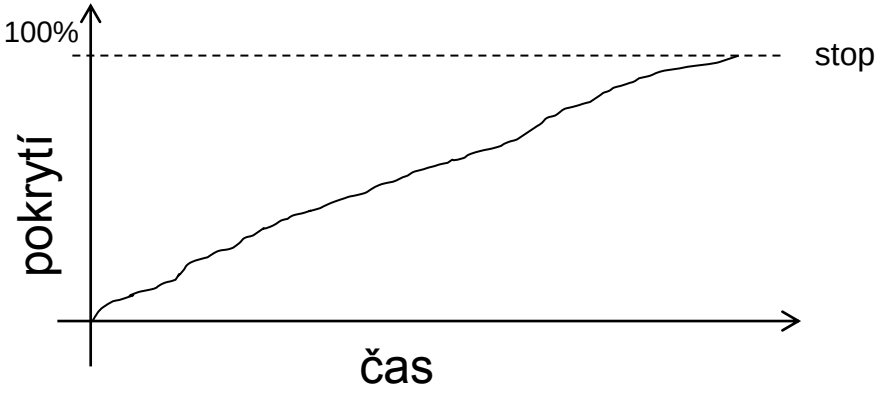
- řízený test – ruční příprava vektoru vstupů, simulace a kontrola výstupu (výstupní hodnoty a průběh signálů)



- iterativní proces
- zastavení v okamihu, kdy už se neobjevují chyby
- jednoduchý přístup
- nízký počet odhalených chyb

Verifikace

- coverage-driven verification

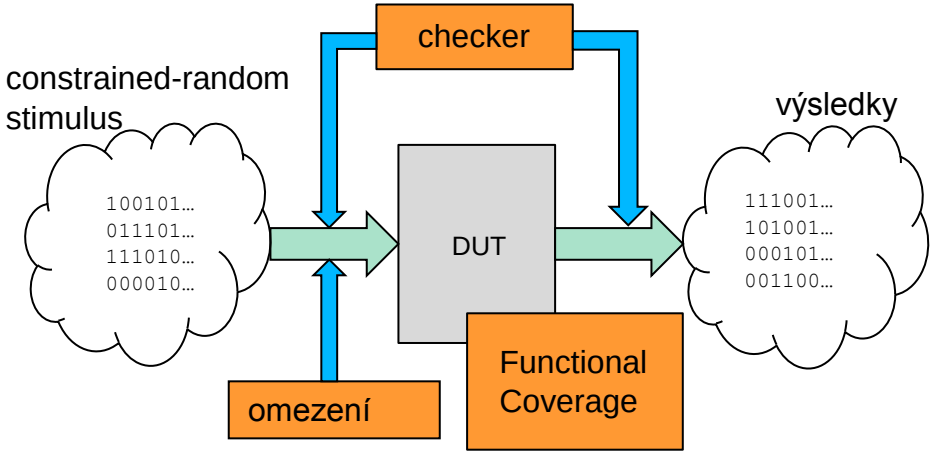


- stanovení cílů pokrytí – *coverage goals* – **CO** je potřeba verifikovat
- a **JAK** určit 100% pokrytí cíle

- v době návrhu se tvoří model (**coverage model**), který svým popisem zachycuje stanovené cíle verifikace

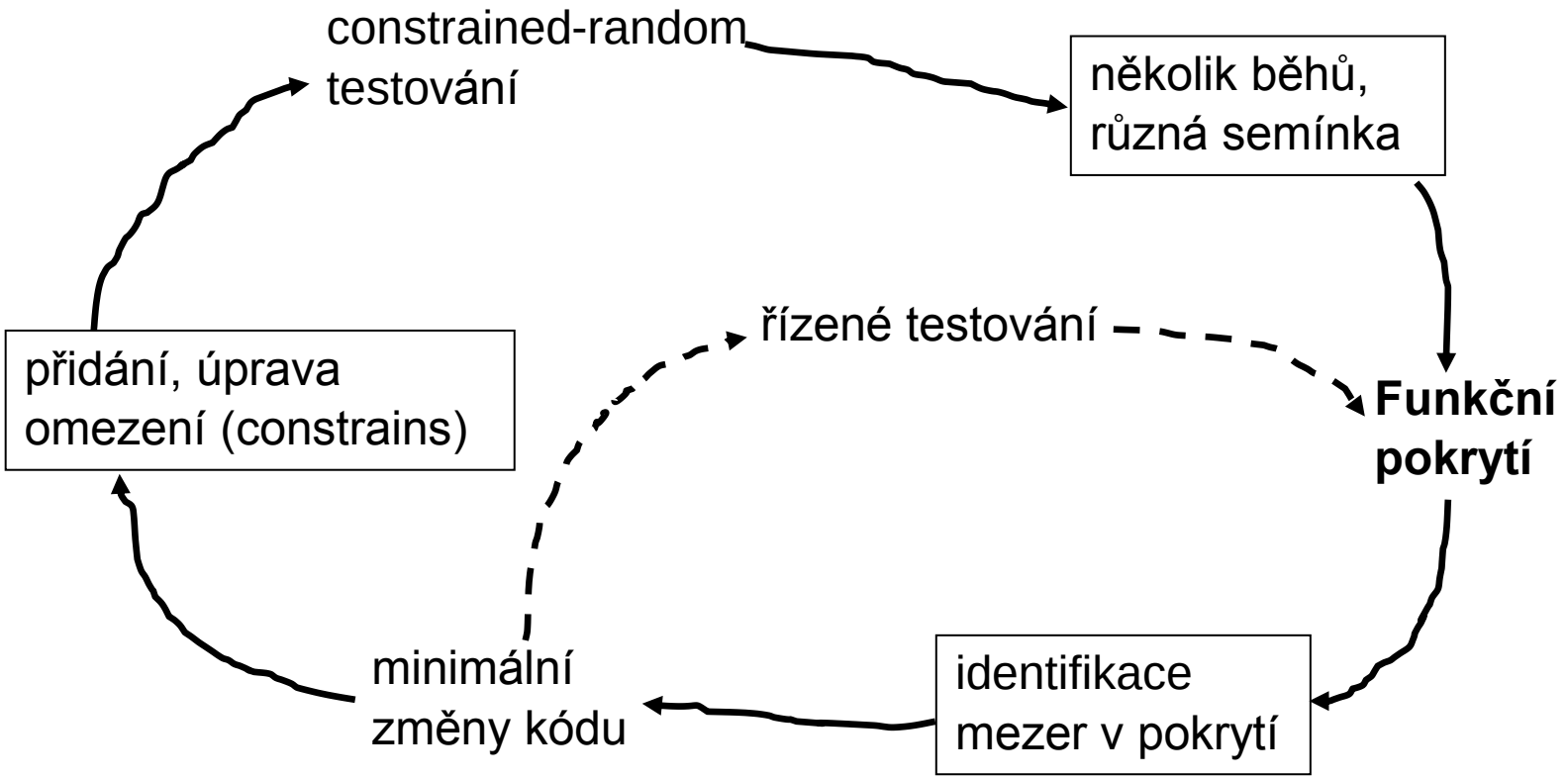
- generování testů:
  - řízený (*directed*)
  - náhodný (*random*)
  - náhodný s omezeními (*constrained-random*)

Constrained-random verification (1)



- automatizování procesu verifikace
- tvorba verifikačního prostředí
- je třeba vzít do úvahy: konfiguraci zařízení a prostředí, vstupní data, chyby a porušení kom. protokolů, časové zpoždění, …
- 3C's model:
    - constrained-random verification
    - checking
    - coverage & constraints

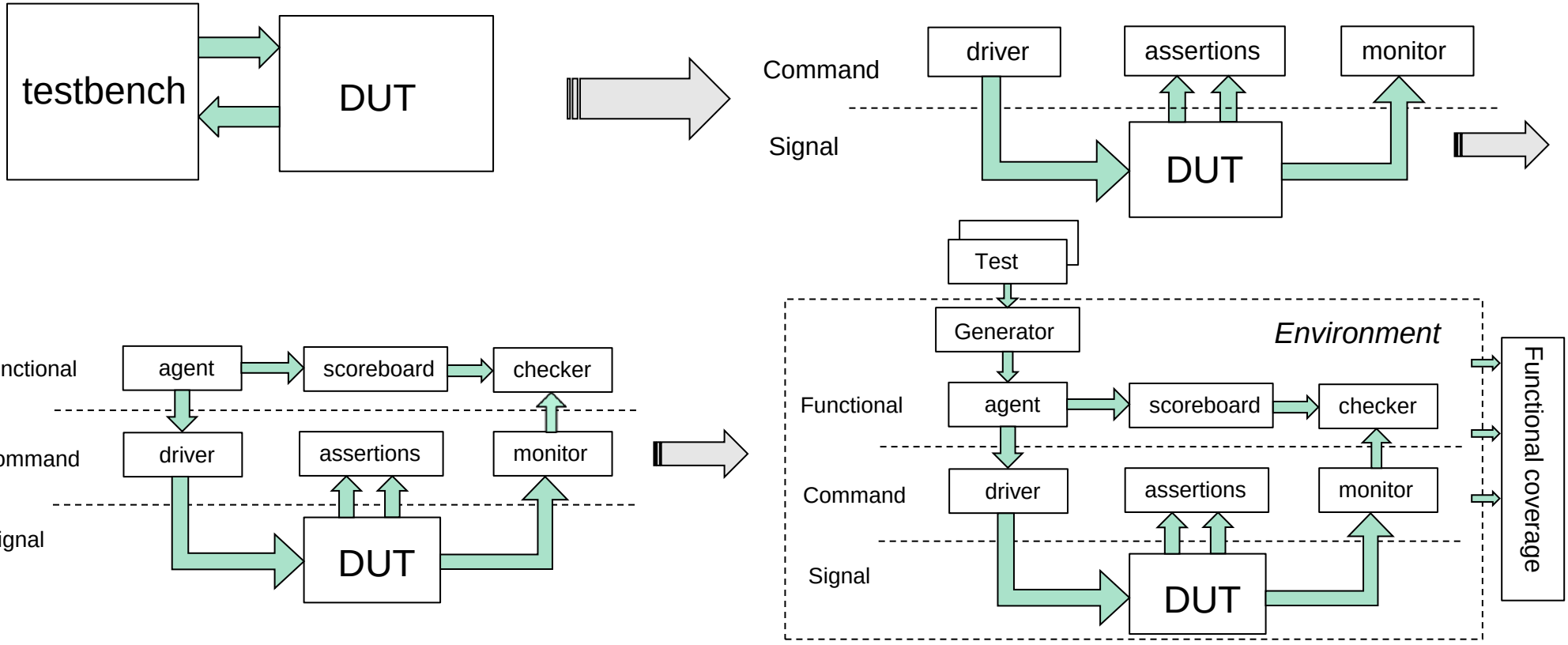
Constrained-random verification (2)



- konvergence k danému pokrytí – iterativní proces
- testovací plán – ze specifikace a implementace je možné stanovit výčet vlastností, které je potřeba testovat
- coverage-model sbírá data a vyhodnocuje pokrytí

Verifikační prostředí – testbench

- v procesu vývoje systému je velké úsilí věnováno tvorbě testbench (ver. prostředí)
- snaha o co nejvyšší znovupoužitelnost nejen pro další testy, ale i projekty
- cílem co nejmenší zásahy do kódu, pouze změna konfigurace
- tvorba testbench pomocí rozdělení do několika vrstev



OVM metodika

- metodologie – vědná disciplína zabývající se metodami, jejich tvorbou a aplikací
- metodika (v comp. sc.) – souhrn praktik a postupů

- doporučení pro vytváření verifikačních prostředí pro číslicové systémy
- objektově orientovaný přístup
- knihovna vytvořena v SystemVerilogu
- OpenSource licence (Apache)
- vytvořena ve spolupráci firem Cadence a Mentor Graphics (AVM & URM kompatibilní)
- podpora více jazyků v jednom prostředí (SystemVerilog, SystemC, VHDL, C/C++)

- standardizována v rámci organizace Accelera – základ pro UVM (v1.0 vydána 17.5.2010)

OVM metodika

- standardizované a konzistentní prostředí pro verifikace
- konfigurovatelnost a flexibilita – co nejmenší zásahy do zdrojového kódu
    - pouze změna konfigurace

- oddělení testů od verifikačního prostředí
- transaction-level abstrakce komunikace
- víceúrovňová komunikace – protocol stacking
- standardizované zasílaní zpráv (chyby, varování, hlášky)

- komponenty verifikačního prostředí mají také standardizovanou strukturu
    - driver, monitor, sequencer, …

Transakční vrstva - abstrakce komunikace

- událost (např. změna úrovně signálu)
  - vzniká v časovém okamžiku
  - je svázána s daty, kontrolní informací
- v simulátoru
  - událost se propaguje na všechny vstupy
  - jednotlivé bloky jsou vyhodnoceny
  - naplánuje se časový okamžik výstupu
- množství událostí ovlivňuje výkon simulátoru
  - simulace sběrnice pomocí jedné události je efektivnější než skupiny samostatných signálů s jednotlivými událostmi
- na transakční úrovni je transakce modelována jako jedna událost
  - s ní jsou svázána data a kontrolní informace a může (nemusí) být s ní být spojena časová informace
  - skrytí nízkoúrovňové implementace komunikace

SystemVerilog

- původně se pro návrh a simulaci používali HDL jazyky
  - mají však jenom omezené možnosti pro návrh komplexních prostředí pro testování
- složitost obvodů vedla k potřebě vzniku jazyků umožňujících jejich testování
  - OpenVera a *e* (komerční jazyky)
- OpenVera se stala základem pro návrh nového jazyka
  - Accelera konsorcium
  - 2005 – vznik HVL jazyka SystemVerilog
  - podpora ze strany IEEE – standard jazyka P1800-2005

SystemVerilog

- založený na prvcích jazyka Verilog
- objektově orientovaný jazyk
- použitelný ve fázích implementace, vytváření testbenchů a assertion konstrukcí

- umožňuje vytvářet testy na vysoké úrovni abstrakce

- přináší mnoho nových datových typů pro lepší návrh verifikačního prostředí a návrhu číslicového systému

- syntax podobná jazyku C

SystemVerilog – 2-stavové dat. typy

- pro zlepšení výkonu simulací a redukce spotřeby paměti

```
bit b;            // 2-state, single bit
bit [31:0] b32;   // 2-state, 32b unsigned integer
int unsigned ui;  // 2-state, 32b unsigned integer
int i;            // 2-state, 32b signed integer
byte b8;          // 2-state, 8b signed integer
shortint s;       // 2-state, 16b signed integer
longint l;        // 2-state, 64b signed integer
integer i4;       // 4-state, 32b signed integer
time t;           // 4-state, 64b unsigned integer
real r;           // 2-state, double precision fp
```

- připojování 2-stavových proměnných k DUT (problém s X,Z)
  - použití operátoru `$isunknown()` – vrací 1, jestliže je některý bit výrazu X nebo Z

SystemVerilog – dat. typ pole

```
int lo_hi[0:15];    // 16 ints [0]..[15]
int c_style[16];    // 16 ints [0]..[15]
```

- vícerozměrná pole

```
int array2 [0:7][0:3];
int array[8][4];          // kompaktnější zápis
```

- čtení mimo rozsah vede k vrácení implicitní hodnoty pro daný datový typ
  - pole 4-stav. prvků typu logic – X
  - pole 2-stav. prvků typu bit – 0
  - platí i pro dynamická a asociativní pole a fronty

- inicializace pole

```
int ascend[4] = '{0,1,2,3};
ascend[0:2] = '{5,6,7};          // first 3 elements
ascend = '{4{8}};                   // 4 values of 8
ascend = '{9,8, default:-1};  // {9,8,-1,-1}
```

SystemVerilog – operace s poli

- procházení polem

```
for (int i; i < $size(array); i++)
    …
foreach (array[j])
    …
foreach(multidimensional[i,j])
    …
```

- podpora přímého porovnání a kopírování, třídění, obrácení pořadí, zamíchání
- dynamická pole – `new[]` konstruktor

```
int dyn[], dyn2[];          // declaration of dynamic arrays

initial begin
      dyn = new[5];                    // allocation of 5 elements
      foreach (dyn[j]) dyn[j] = j;     // initialization
      d2 = dyn;                        // copy of dynamic array
      d2[0] = 5;                       // modification of element
      dyn = new[20](dyn);              // allocation of 20 elements and copy
      dyn = new[100];                  // allocation of 100 elements, old values lost
      dyn.delete();                    // delete all elements
end
```

SystemVerilog – dat. typy

- podpora front – kombinace seznamů a polí
- podpora asociativních polí
- struktury, uniony, výčtové typy
- vytváření nových datových typů – `typedef`
- statické a dynamické přetypování

SystemVerilog – tasks, functions

- task – spotřebovává čas
- funkce – nulový čas, nemůže mít časovou prodlevu, příkaz wait, blokující příkaz související s časem, může volat task pouze ve vláknu ohraničeném fork…join_none příkazmi

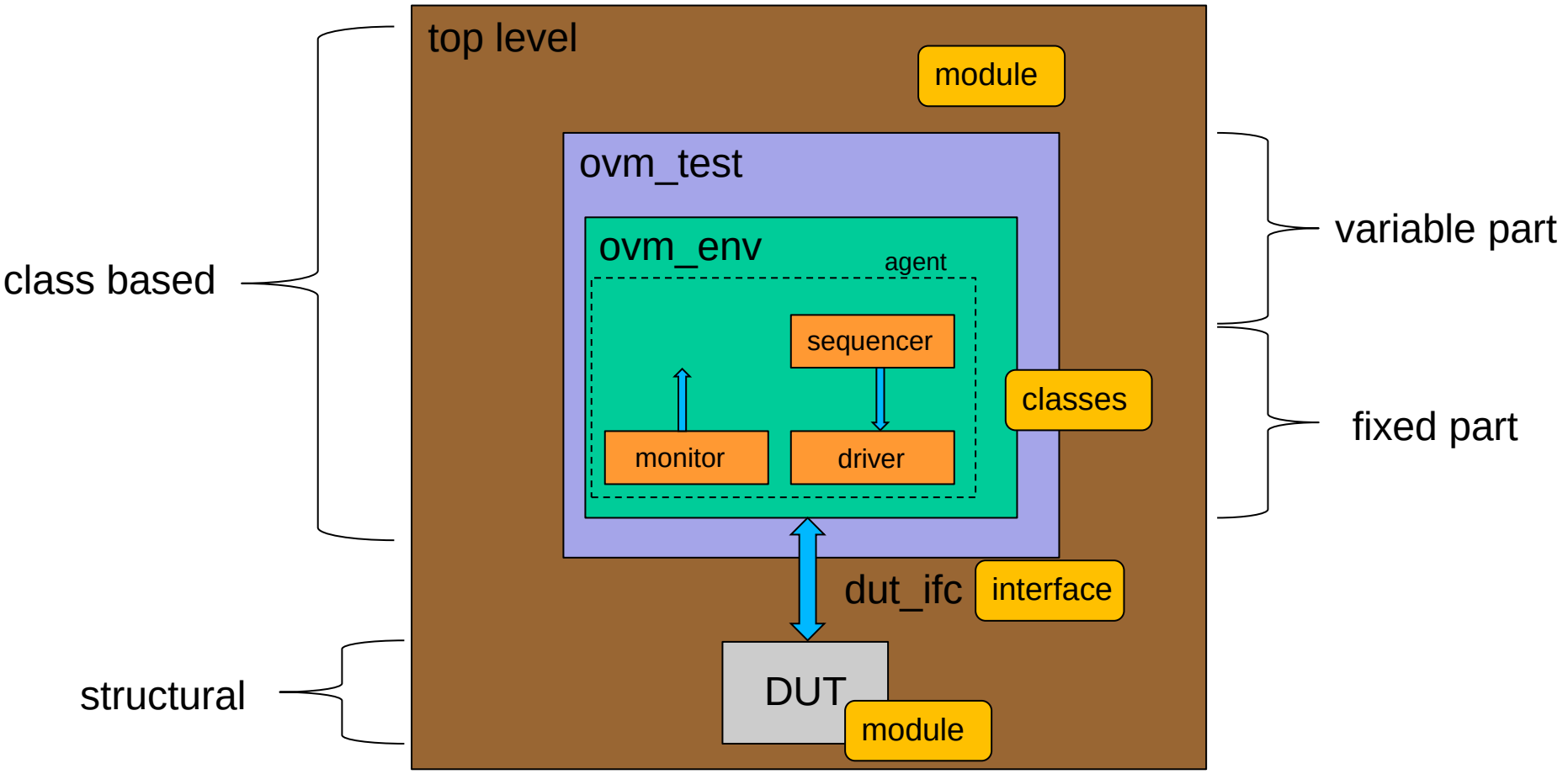
- ignorování návratové hodnoty funkce
  - `void '($fscanf(file, "%d"), i);`

- zápis funkcí a tasků velmi blízký jazyku C
- odstraněna klíčová slova `begin…end`
- směr parametrů funkcí a tasků – `input`, `output`,`inout` – hodnotou
- podpora předávaní odkazem – `ref`, vhodné pro pole (nemusí se kopírovat celé), změna proměnné předané odkazem je viditelná hned volající funkci (vhodné pro vlákna)
- podpora implicitní hodnoty argumentů (pomocí přirazení)

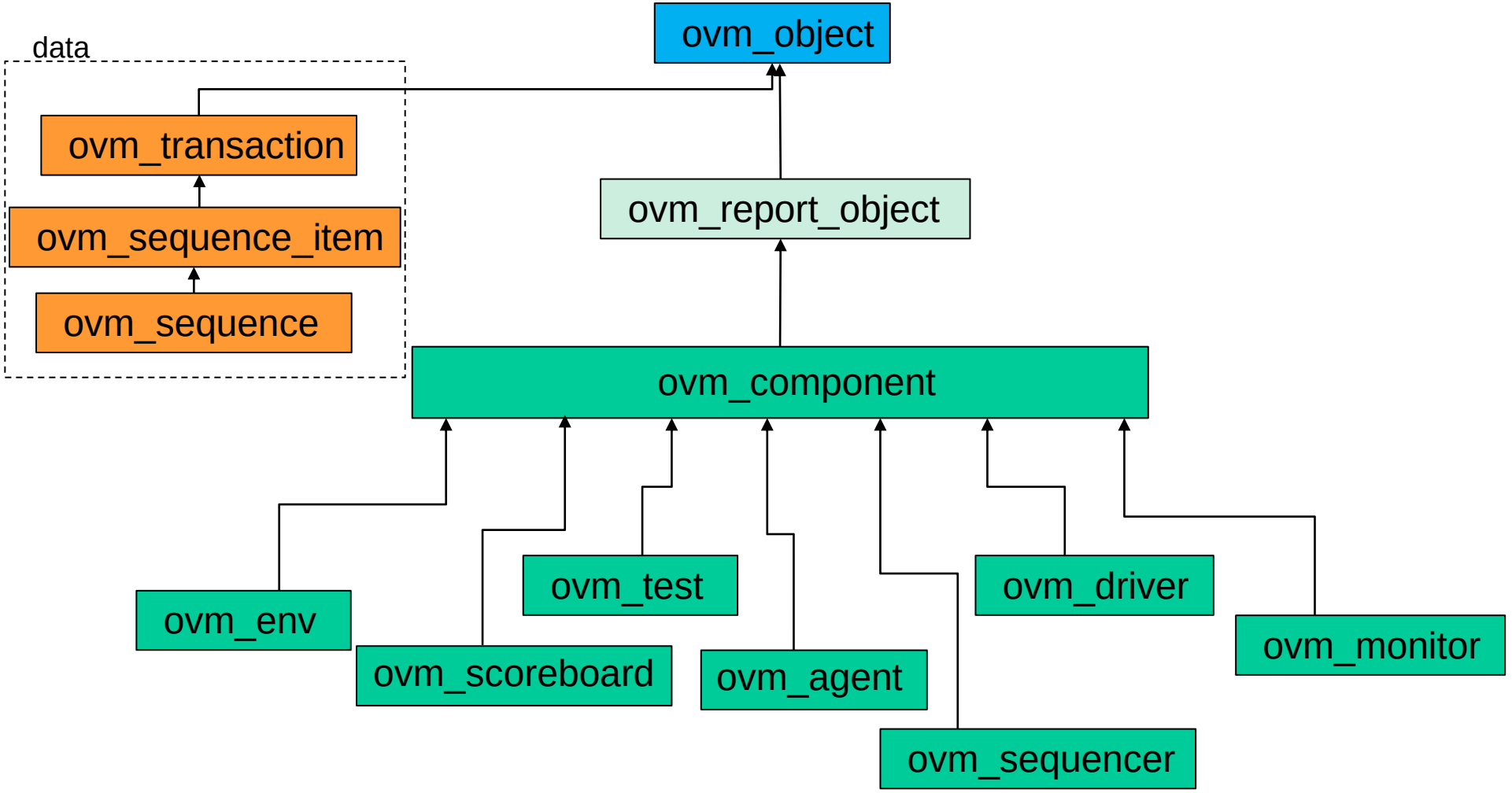
SystemVerilog – práce s časem

- direktiva překladače `timescale – nutno překládat soubory ve správném pořadí (kvůli míre a přesnosti)
- pro každý modul je však možné určit tyto hodnoty pomocí `timeunit` a `timeprecision`, musí však být v každém modulu, který pracuje s časem

OVM – příklad


top level
module
ovm_test
ovm_env
agent
sequencer
monitor
driver
classes
dut_ifc
interface
DUT
module
class based
structural
variable part
fixed part

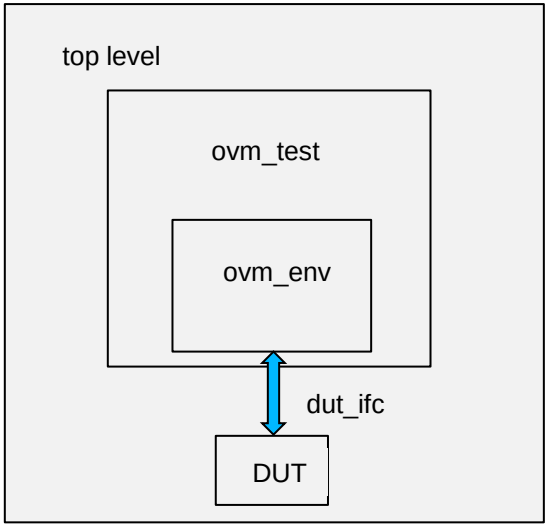
OVM – hierarchie tříd



OVM – DUT interface

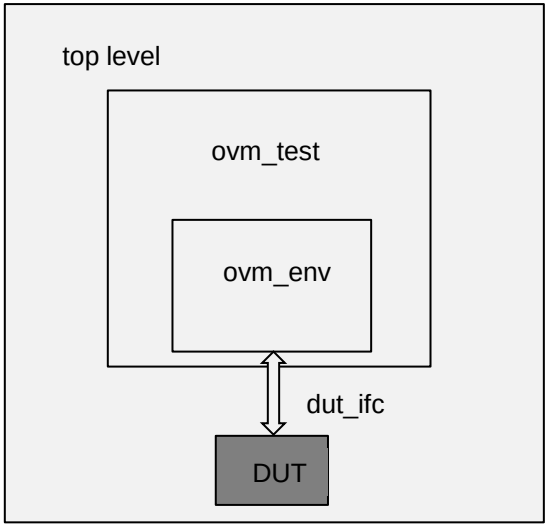
```
interface dut_if(dut_if _if);
  logic clock, reset, cmd;
  logic [7:0] addr;
  logic [7:0] data;
endinterface: dut_if
```

```
class dut_if_wrapper extends ovm_object;
  virtual dut_if dut_vi;

  function new(string name,
               virtual dut_if arg);
    super.new(name);
    dut_vi = arg;
  endfunction: new

endclass: dut_if_wrapper
```



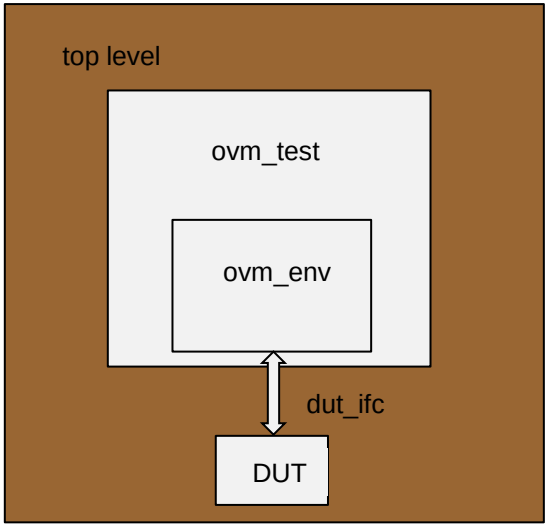
OVM – DUT

```
module dut(dut_if _if);
  always @(posedge _if.clock)
  begin
    ovm_pkg::ovm_top.ovm_report_info(“PCS", $psprintf("DUT received
          cmd=%b, addr=%d, data=%d", _if.cmd, _if.addr, _if.data));
  end
endmodule: dut
```



OVM – top_level

```
module top;
  ...

  dut_if dut_if1();
  dut     dut_1 ( ._if(dut_if1) );

  ...

  initial
  begin: blk
    dut_if_wrapper if_wrapper = new(
                      "if_wrapper", dut_if1);

    set_config_object("*", "dut_if_wrapper",
                       if_wrapper, 0);

  ...
endmodule: top
```

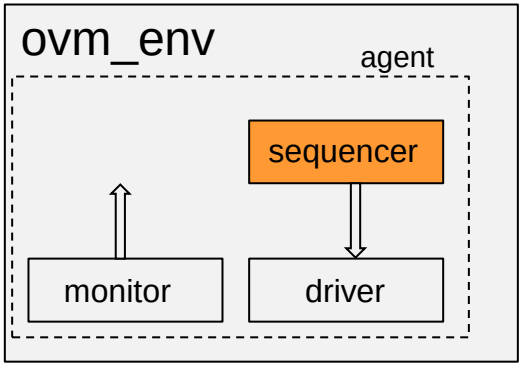


OVM – transakce

```
class my_transaction extends ovm_sequence_item;

  `ovm_object_utils(my_transaction)

  rand bit cmd;
  rand int addr;
  rand int data;

  constraint c_addr { addr >= 0; addr < 256; }
  constraint c_data { data >= 0; data < 256; }

  function new (string name = "");
    super.new(name);
  endfunction: new

  ...

endclass: my_transaction
```

OVM – sequencer

```
class my_sequencer extends ovm_sequencer #(my_transaction);

  `ovm_component_utils(my_sequencer)

  function new(string name, ovm_component parent);
    super.new(name, parent);
  endfunction: new

endclass: my_sequencer
```



OVM – sequence

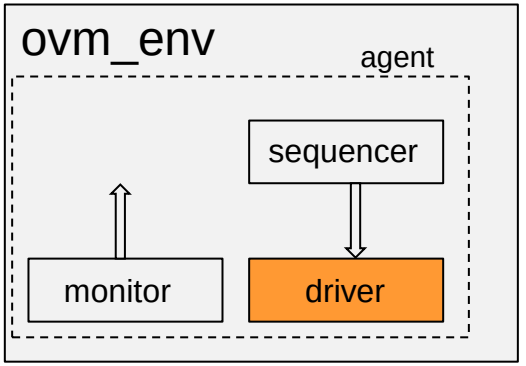
```
class read_modify_write extends ovm_sequence #(my_transaction);

  `ovm_object_utils(read_modify_write)

  function new(string name = "");
    super.new(name, parent);
  endfunction: new

  task body;
    my_transaction tx;
    int a;
    int d;

    tx = my_transaction::type_id::create("tx");
    start_item(tx);
    assert( tx.randomize() with { cmd == 0; } );
    finish_item(tx);

    a = tx.addr;
    d = tx.data;
    ++d;

    tx = my_transaction::type_id::create("tx");
    start_item(tx);
    assert( tx.randomize() with { cmd == 1; addr == a; data == d; } );
    finish_item(tx);

endclass: my_sequence
```

OVM – sequences

```
class seq_of_commands extends ovm_sequence #(my_transaction);

  `ovm_object_utils(seq_of_commands)

  rand int n;        // nob - controls operation of the sequence

  constraint how_many { n inside {[4:6]}; }

  function new (string name = "");
    super.new(name);
  endfunction: new

  task body;
    repeat(n)
    begin
      read_modify_write seq;
      seq = read_modify_write::type_id::create("seq");
      start_item(seq);
      finish_item(seq);
    end
  endtask: body

endclass: seq_of_commands
```

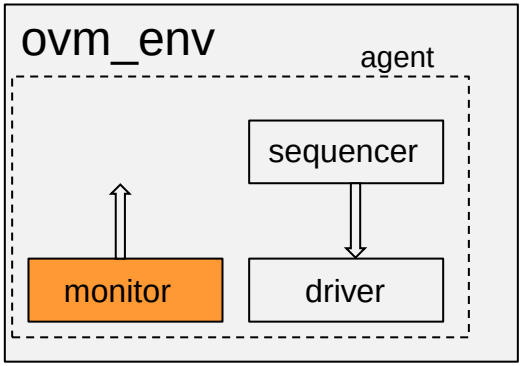
OVM – driver

```
class my_driver extends ovm_driver #(my_transaction);

  `ovm_component_utils(my_driver)

  virtual dut_if dut_vi;

  function new(string name, ovm_component parent);
    ...

  function void build;
    super.build();
    ovm_object obj;
    dut_if_wrapper if_wrapper;
    get_config_object("dut_if_wrapper", obj, 0);
    assert( $cast(if_wrapper, obj) );
    dut_vi = if_wrapper.dut_vi;
  endfunction: build

  task run;
    forever
    begin
      my_transaction tx;

      @(posedge dut_vi.clock);
      seq_item_port.get(tx);

      // Wiggle pins of DUT
      dut_vi.cmd  = tx.cmd;
      dut_vi.addr = tx.addr;
      dut_vi.data = tx.data;
    end
endclass: my_driver
```



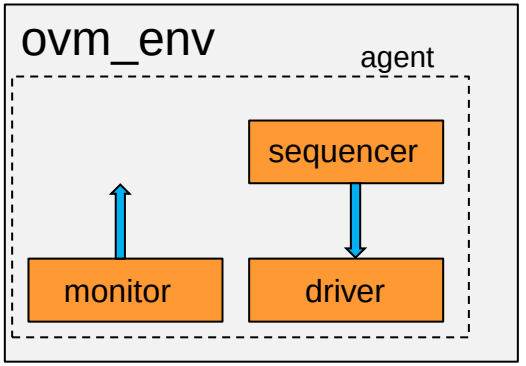
OVM – monitor

```
class my_monitor extends ovm_monitor;

  `ovm_component_utils(my_monitor)

  ovm_analysis_port #(my_transaction) aport;

  virtual dut_if dut_vi;

  function new ...

  function void build;
    super.build();
    aport = new("aport", this);
    ...

  task run;
    forever
    begin
      my_transaction tx;

      @(posedge dut_vi.clock);
      tx = my_transaction::type_id::create("tx");
      tx.cmd  = dut_vi.cmd;
      tx.addr = dut_vi.addr;
      tx.data = dut_vi.data;

      aport.write(tx);
```



OVM – agent

```
class my_agent extends ovm_agent;

  `ovm_component_utils(my_agent)

  ovm_analysis_port #(my_transaction) aport;

  my_sequencer my_sequencer_h;
  my_driver    my_driver_h;
  my_monitor   my_monitor_h;

  function new(string name, ovm_component parent);
    super.new(name, parent);
  endfunction: new

  function void build;
    super.build();
    aport = new("aport", this);
    my_sequencer_h = my_sequencer::type_id::create("my_sequencer_h", this);
    my_driver_h    = my_driver::type_id::create("my_driver_h", this);
    my_monitor_h   = my_monitor::type_id::create("my_monitor_h", this);
  endfunction: build

  function void connect;
    my_driver_h.seq_item_port.connect( my_sequencer_h.seq_item_export );
    my_monitor_h.aport.connect( aport );
  endfunction: connect

endclass: my_agent
```

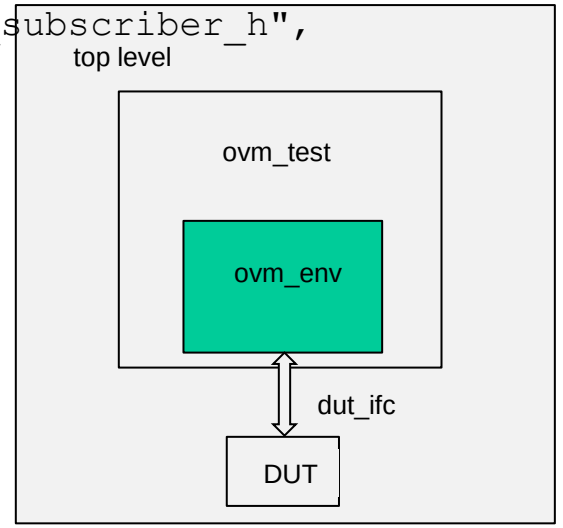


OVM – subscriber

```
class my_subscriber extends ovm_subscriber #(my_transaction);

  `ovm_component_utils(my_subscriber)

  bit cmd;
  int addr;              // coverage registers
  int data;


  function void write(my_transaction t);
    ovm_report_info("PCS", $psprintf("Subscriber received tx %s",
                        t.convert2string()));
    cmd  = t.cmd;
    addr = t.addr;        // transaction content extraction
    data = t.data;
```

OVM – environment

```
class my_env extends ovm_env;
  `ovm_component_utils(my_env)

  my_agent my_agent_h;
  my_subscriber my_subscriber_h;

  ...

  function void build;
    super.build();
    my_agent_h = my_agent::type_id::create("my_agent_h", this);
    my_subscriber_h = my_subscriber::type_id::create("my_subscriber_h",
                      this);
  endfunction: build

  function void connect;
    my_agent_h.aport.connect(
                        my_subscriber_h.analysis_export);
  endfunction: connect

endclass: my_env
```

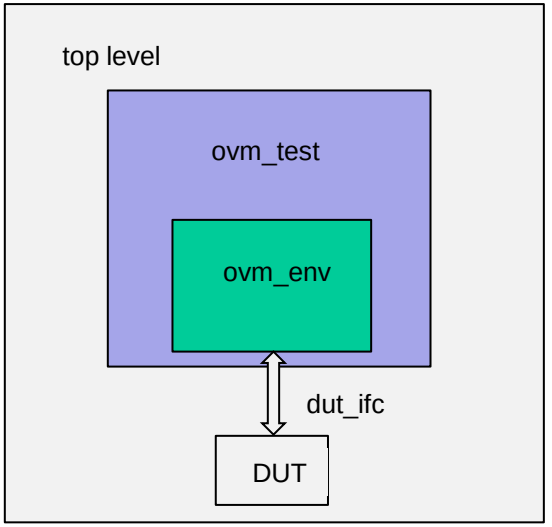


OVM – package

```
package my_seq_library;
  import ovm_pkg::*;

  class read_modify_write extends ovm_sequence #(my_transaction);

  ...

  class seq_of_commands extends ovm_sequence #(my_transaction);

  ...

endpackage
```

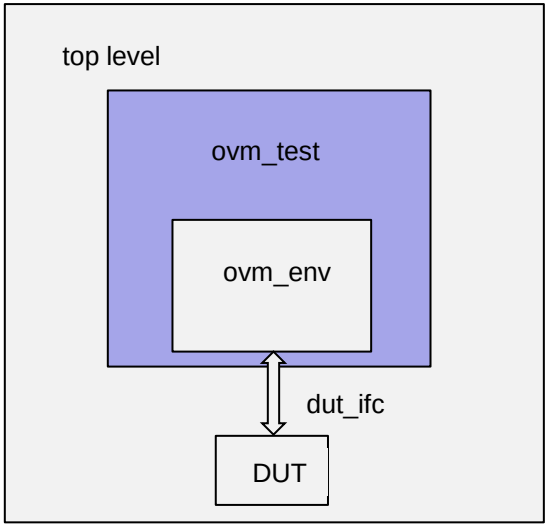
OVM – package (2)

```
`include "ovm_macros.svh"

package my_pkg;

  import ovm_pkg::*;
  import my_seq_library::*;

  class my_env extends ovm_env;
    ...
  class my_subscriber extends ovm_subscriber #(my_transaction);
    ...

  class my_agent extends ovm_agent;
    ...

  class my_monitor extends ovm_monitor;
    ...

  ...

  class my_test extends ovm_test;
    ...
endpackage: my_pkg
```



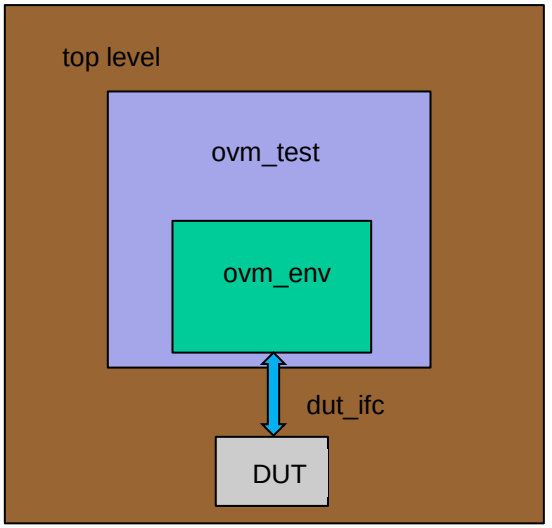
OVM – test

```
class my_test extends ovm_test;

 `ovm_component_utils(my_test)

  my_env my_env_h;

  function new(string name, ovm_component parent);
    ...

  function void build;
    ...

  task run;
    seq_of_commands seq;
    seq = seq_of_commands::type_id::create("seq");
    assert(seq.randomize());
    seq.start(my_env_h.my_agent_h.my_sequencer_h);
    ovm_top.stop_request();
  endtask: run
endclass: my_test
```



OVM – instancování testu

```
module top;

  import ovm_pkg::*;
  import my_pkg::*;

  dut_if dut_if1 ();

  dut    dut1 ( ._if(dut_if1) );

  initial
  begin: blk
    ...
    run_test("my_test");
  end

endmodule: top
```



OVM – průběh simulace

```
...
# OVM-2.1.1
# (C) 2007-2009 Mentor Graphics Corporation
# (C) 2007-2009 Cadence Design Systems, Inc.
# ----------------------------------------------------------------
# OVM_INFO @ 0: reporter [RNTST] Running test my_test...
# OVM_INFO @ 0: reporter [OVMTOP] OVM testbench topology:
# ----------------------------------------------------------------------
# Name                      Type                Size                Value
# ----------------------------------------------------------------------
# ovm_test_top              my_test             -           ovm_test_top@4
#   my_env_h                my_env              -               my_env_h@6
#     my_agent_h            my_agent            -             my_agent_h@8
#       aport               ovm_analysis_port   -                 aport@14
#       my_driver_h         my_driver           -           my_driver_h@46
#         rsp_port          ovm_analysis_port   -             rsp_port@50
#         sqr_pull_port     ovm_seq_item_pull_+ -         sqr_pull_port@48
#       my_monitor_h        my_monitor          -          my_monitor_h@52
#         aport             ovm_analysis_port   -                 aport@54
#       my_sequencer_h      my_sequencer        -        my_sequencer_h@16
#         rsp_export        ovm_analysis_export -            rsp_export@18
#         seq_item_export   ovm_seq_item_pull_+ -       seq_item_export@42
#         num_last_reqs     integral            32                    'd1
#         num_last_rsps     integral            32                    'd1
#     my_subscriber_h       my_subscriber       -       my_subscriber_h@10
#       analysis_imp        ovm_analysis_imp    -          analysis_imp@12
# ----------------------------------------------------------------------
...
```

OVM – průběh simulace

```
# OVM_INFO @ 5: ovm_test_top.my_env_h.my_subscriber_h [PCS] Subscriber received tx cmd=0, addr=0, data=0
# OVM_INFO @ 5: reporter [PCS] DUT received cmd=x, addr=  x, data=  x
# OVM_INFO @ 15: reporter [PCS] DUT received cmd=0, addr= 59, data= 70
# OVM_INFO @ 15: ovm_test_top.my_env_h.my_subscriber_h [PCS] Subscriber received tx cmd=0, addr=59, data=70
# OVM_INFO @ 25: ovm_test_top.my_env_h.my_subscriber_h [PCS] Subscriber received tx cmd=1, addr=59, data=71
# OVM_INFO @ 25: reporter [PCS] DUT received cmd=1, addr= 59, data= 71
# OVM_INFO @ 35: reporter [PCS] DUT received cmd=0, addr= 87, data=143
# OVM_INFO @ 35: ovm_test_top.my_env_h.my_subscriber_h [PCS] Subscriber received tx cmd=0, addr=87, data=143
# OVM_INFO @ 45: ovm_test_top.my_env_h.my_subscriber_h [PCS] Subscriber received tx cmd=1, addr=87, data=144
# OVM_INFO @ 45: reporter [PCS] DUT received cmd=1, addr= 87, data=144
# OVM_INFO @ 55: reporter [PCS] DUT received cmd=0, addr=251, data=214
# OVM_INFO @ 55: ovm_test_top.my_env_h.my_subscriber_h [PCS] Subscriber received tx cmd=0, addr=251, data=214
# OVM_INFO @ 65: ovm_test_top.my_env_h.my_subscriber_h [PCS] Subscriber received tx cmd=1, addr=251, data=215
# OVM_INFO @ 65: reporter [PCS] DUT received cmd=1, addr=251, data=215
# OVM_INFO @ 75: reporter [PCS] DUT received cmd=0, addr= 39, data=252
# OVM_INFO @ 75: ovm_test_top.my_env_h.my_subscriber_h [PCS] Subscriber received tx cmd=0, addr=39, data=252
# OVM_INFO @ 85: ovm_test_top.my_env_h.my_subscriber_h [PCS] Subscriber received tx cmd=1, addr=39, data=253
# OVM_INFO @ 85: reporter [PCS] DUT received cmd=1, addr= 39, data=253
# OVM_INFO @ 95: reporter [PCS] DUT received cmd=0, addr= 17, data=134
# OVM_INFO @ 95: ovm_test_top.my_env_h.my_subscriber_h [PCS] Subscriber received tx cmd=0, addr=17, data=134
# OVM_INFO @ 105: ovm_test_top.my_env_h.my_subscriber_h [PCS] Subscriber received tx cmd=1, addr=17, data=135
# OVM_INFO @ 105: reporter [PCS] DUT received cmd=1, addr= 17, data=135
# OVM_INFO @ 115: reporter [PCS] DUT received cmd=0, addr=112, data=  9
# OVM_INFO @ 115: ovm_test_top.my_env_h.my_subscriber_h [PCS] Subscriber received tx cmd=0, addr=112, data=9
#
# --- OVM Report Summary ---
#
# ** Report counts by severity
# OVM_INFO :   26
# OVM_WARNING :    0
# OVM_ERROR :    0
# OVM_FATAL :    0
# ** Report counts by id
# [OVMTOP]     1
# [PCS]    24
# [RNTST]     1
# ** Note: $finish    : ../../src/base/ovm_root.svh(507)
#    Time: 115 ps  Iteration: 24  Instance: /ovm_pkg::ovm_root::run_test
```

0-In Formal Verification

- Mentor Graphics
- statická formální verifikace
- assertion-based verifikace

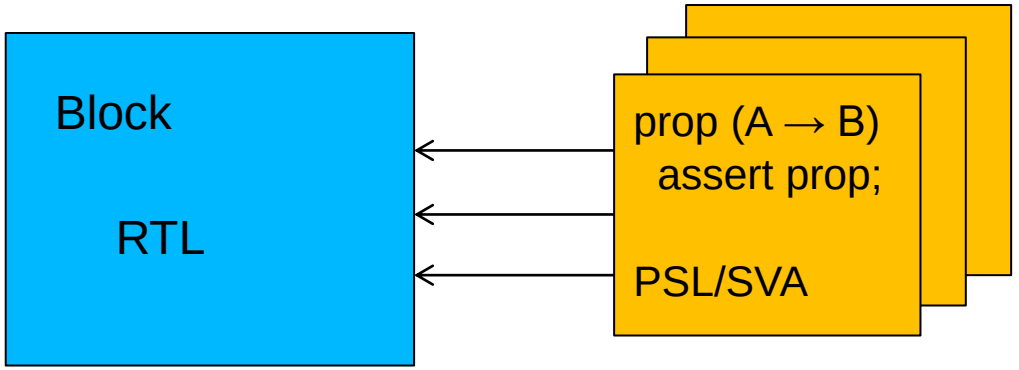
- model-checking – algoritmický přístup ověřující, zda daný systém splňuje zadanou vlastnost systematickým zkoumáním stavového prostoru daného systému

- vysoký stupeň automatizace
- poměrně jednoduše použitelné
- není potřeba testbench

- funkční verifikací není možné pokrýt všechny možné vstupy a kontrolovat výstupy
    - 32b komparátor – více 500 000 let na kontrolu chování (při dnes dostupných technologiích)!

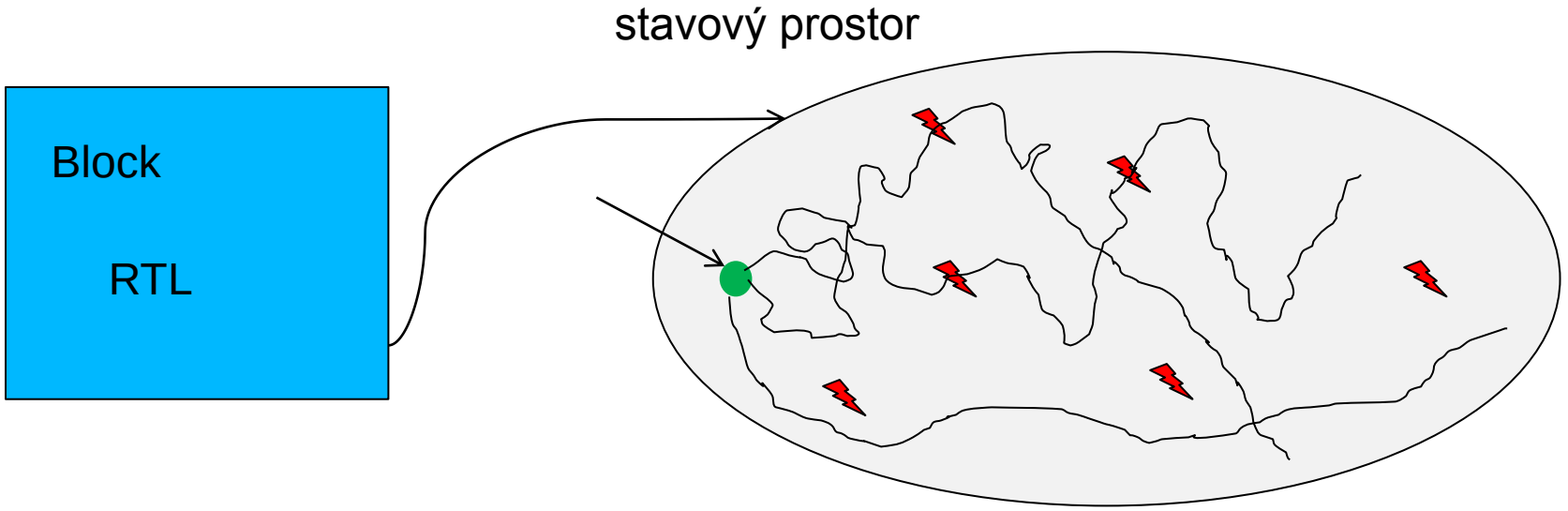
Formální verifikace – model checking

- model checking ~ property checking
- porovnání 2 modelů
  - model obvodu
  - property, assertion – sledované vlastnosti, popis chování
  - při shodě – důkaz správnosti



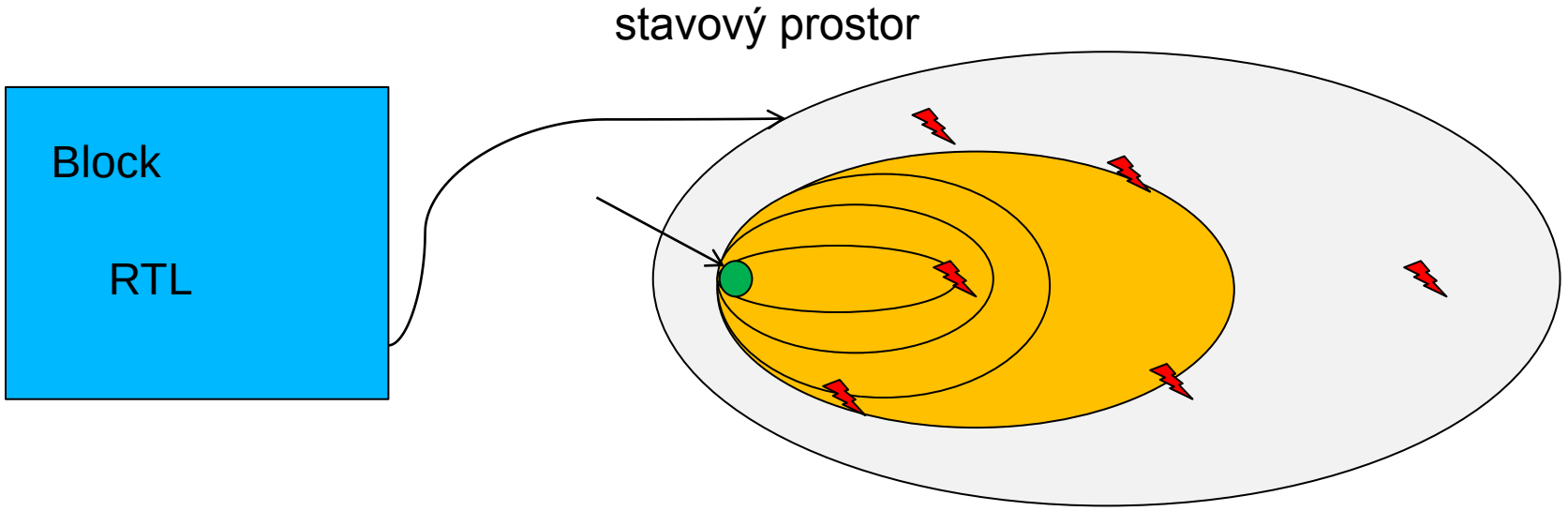
- důkaz znamená, že neexistuje žádný vstup, který by způsobil porušení sledované podmínky

Funkční a formální verifikace



- Funkční verifikace:
  - nedokáže pokrýt všechny stavy – nedostatek času a výpočetního výkonu

Funkční a formální verifikace



- Formální verifikace:
  - prohledává stavový prostor
  - nedokáže pokrýt celý stavový prostor – nedostatek paměti

0-In formální verifikace - property

- jazyky pro podporu zápisu vlastností – *properties, assertions*
  - SVA, PSL

SVA:

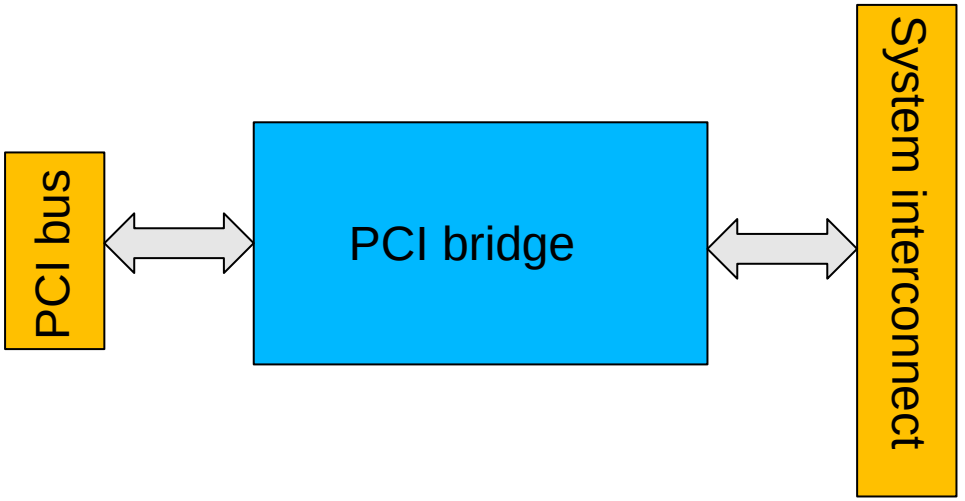
```
always @(posedge clk)
assert property (
  req && ack |=>
  (!req within gnt[->1])
);
```

PSL:

```
assert always (
  {req && ack} |=>
  {!req within gnt[->1]}
)@(posedge clk);
```

- podpora SVA, PSL (IEEE standardy)
- využití assertion knihoven – jednoduché zakomponování do testu
  - QVL – Questa Verification Library
  - OVL – Open Verification Library
- podpora automatických kontrol – bez potřeby psaní assertions
  - kontrola mrtvého kódu
  - analýza FSM (deadlock, dosažitelnost stavů)
  - …

0-In formální verifikace

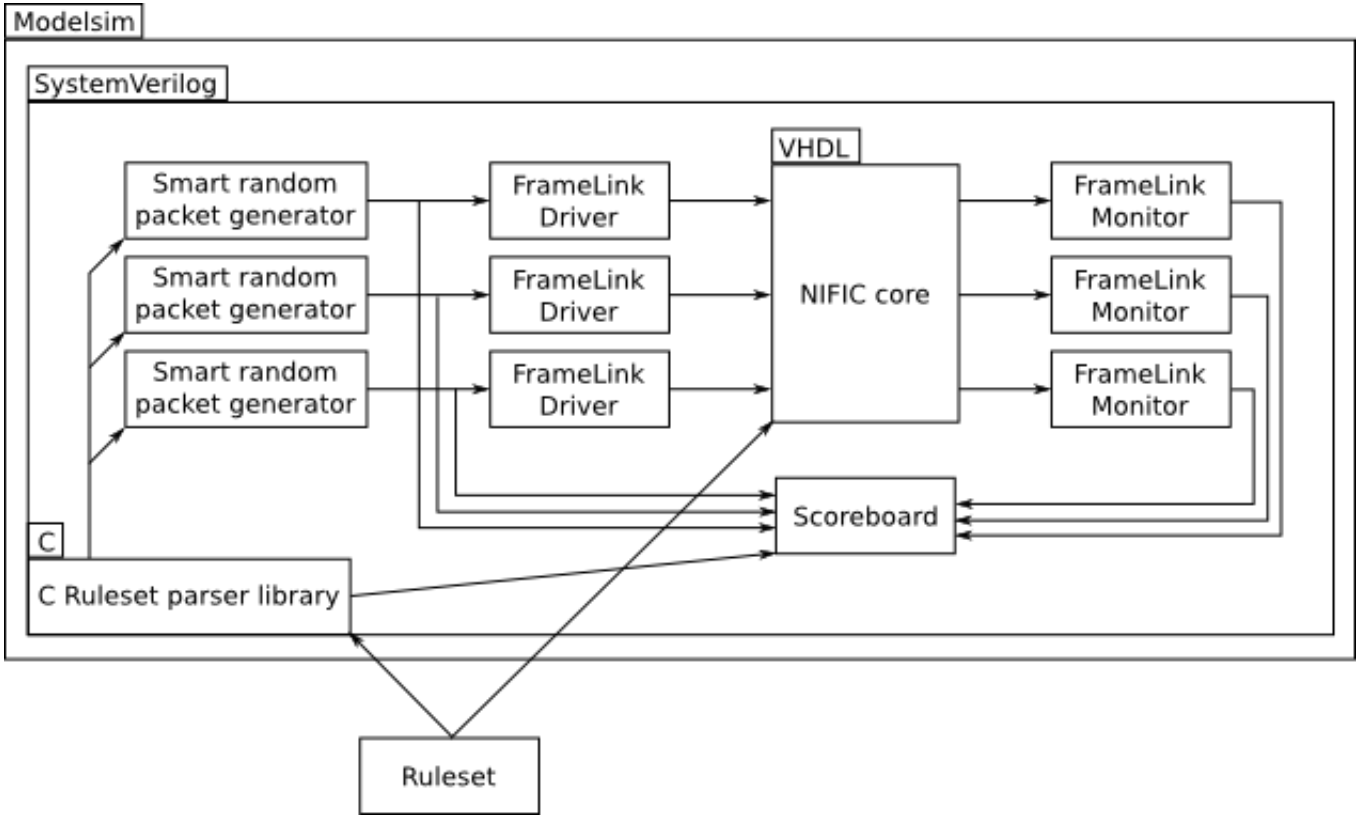


- videoukázka

Příklady z praxe

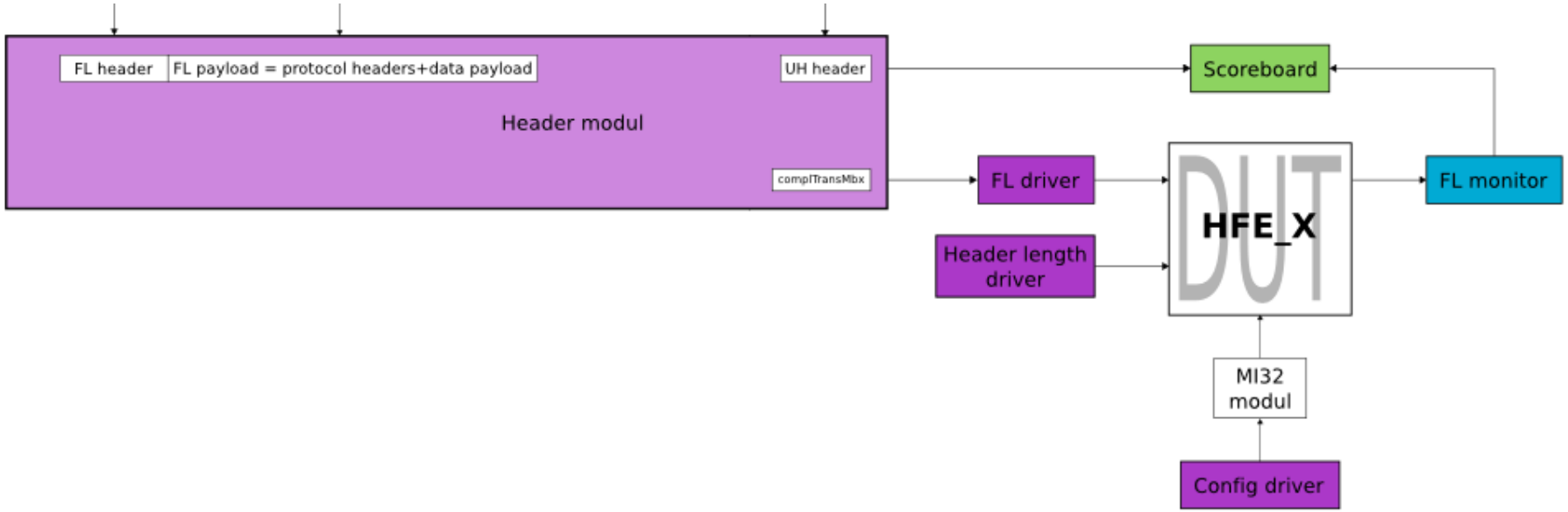
NIFIC

- wirespeed packet filtering



Flowmon

- Flow-based (NetFlow/IPFIX) monitoring platform
- verifikace subkomponent
    - příklad: verifikace jednotky HFE-X – extrakce položek z hlaviček paketů



Konec přednášky

Děkuji za pozornost!

Použité zdroje a odkazy

- 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]. [cit. 21.11.2010]. <http://verification-academy.mentor.com/>
- 0-In Formal Verification [online]. [cit. 21.11.2010]. <http://www.mentor.com/products/fv/0-in_fv/>
- OVM World [online]. [cit. 21.11.2010]. <http://www.ovmworld.org>
- Fundamentals of Accelerated Functional Verification [online]. [cit. 21.11.2010]. <http://www.eetimes.com/Content/Courses/4209369/player.html/>