VEŽBA 22

U okviru ove vežbe upoznaćemo se sa kreiranjem agenata kao ključnih UVM klasa koje se koriste u automatizaciji složenih ispitnih scenarija.

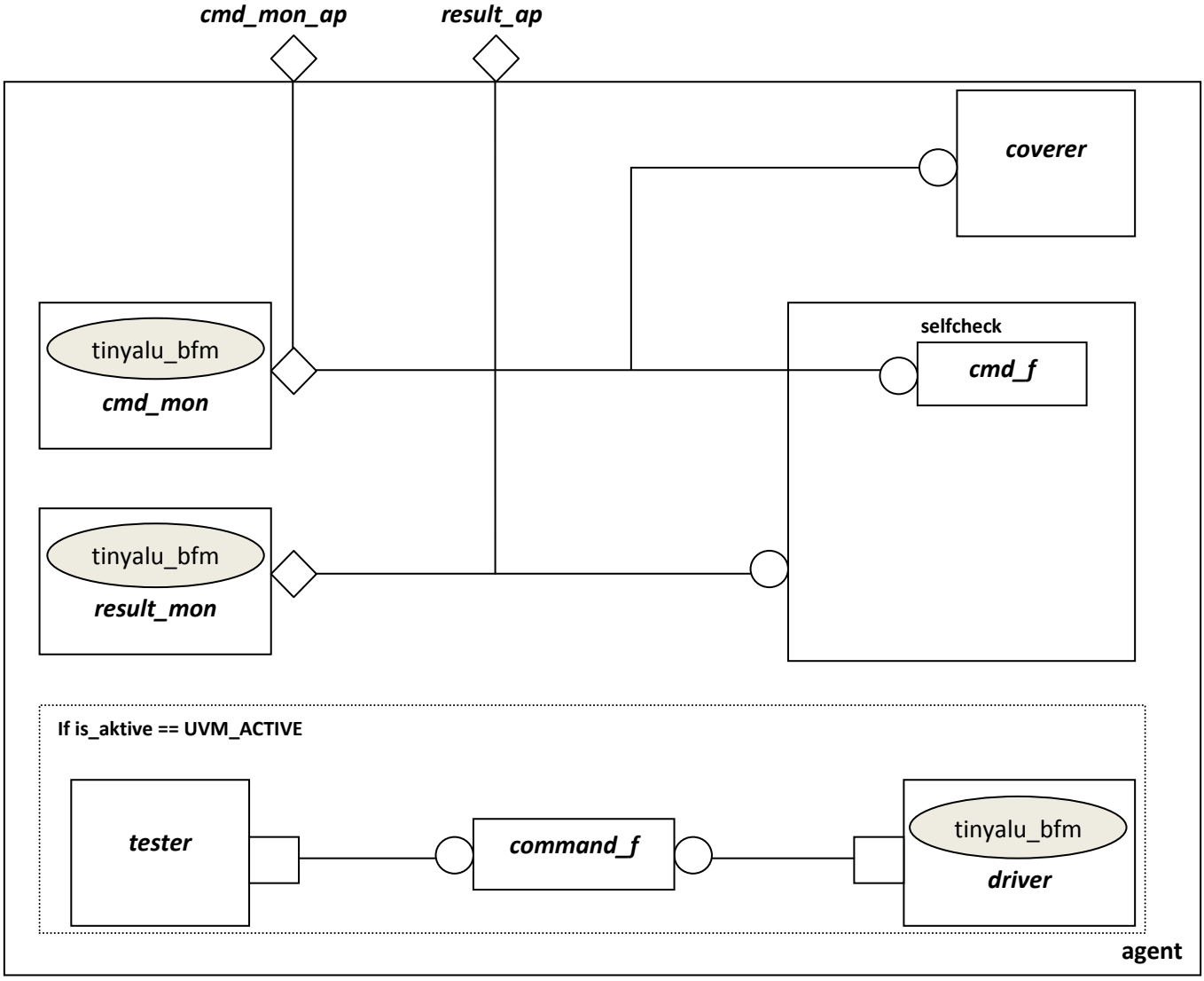
U direktorijumima:

22_UVM_Agents,

nalaze se fajlovi: ***tinyalu_bfm.sv, tinyalu_macros.svh, tinyalu_pkg.sv, tinyalu_tester_module.sv, top.sv***

22_UVM_Agents\tb_classes

nalaze se fajlovi: ***add_transaction.svh, command_monitor.svh, command_transaction.svh, coverage.svh, driver.svh, dual_test.svh, env.svh, env_config.svh, result_monitor.svh, result_transaction.svh, scoreboard.svh, tester.svh, tinyalu_agent.svh, tinyalu_agent_config.svh.***



Slika 1 tinyalu agent

Agent-i predstavljaju kolekciju klasa, odnsno UVM komponenti povezanih sa specifičnim protokolom ili specifičnim bus-om ili delom ***testbench***-a. Agenti nam omogućavaju da kreiramo hiarahijske ***testbench***-eve, gde možemo da kombinujemo delove ***testbench***-a, na višem nivou. Ako pogledamo *sliku 1* vidimo da kada grupišemo sve komponente iz našeg ***testbench***-a, koje su povezane sa tinyalu blokom kreirali samo ***tinyalu_agent***.

Slično kao u vežbi 16, povezujemo ***result_mon*** na ***selfcheck*** (bivši ***scoreboard***), ***cmd_mon*** na komandni fifio po imenu ***cmd_f*** i preko njega ponovo na ***selfcheck***, imamo direktnu vezu sa ***cmd_mon*** na ***coverer*** (bivši ***coverage*** blok).

Takođe ima još posebno grupisane blokove ***tester, command_f*** i ***driver*** uokvirene tačkastim pravouganikom na slici. Ova tri bloka realizuju generisanje stimulusa našeg tiny_alu DUT-a. Za svrhu aktivacije, odnosno deaktivacije ovih blokova koristi se kontrolni bit nasleđen od ***UVM agent*** klase po imenu ***is_active***, pomoću njega jednostavno aktiviramo ove blokove za generisanje stimulusa (***is_active=UVM_ACTIVE***). Ukoliko želimo da ih deaktiviramo postavljamo ***is_active=UVM_PASSIVE,*** deaktiviran agent više nema ulogu generatora stimulusa, ali i dalje radi monitoring transakcija na ***bfm***-u i prati rezultate simulacije.

Podsećamo se da se ***driver*** vezuje na ***tinyalu_bfm***, time se ostvaruje veza na signalnom vremenski egzaktnom nivou, ovu vezu ostvaruju svi blokovi koju su vezani na ***tinyalu_bfm***, a to su još ***cmd_mon*** i ***result_mon***.

U našem primeru realizujemo situaciju, gde želimo da testiramo naš ***tinyalu*** na dva različita načina, jedan način gde **pobuda** dolazi iz ***agent***-a, a drugi gde **pobuda** dolazi iz spoljašnjeg modula. Clij je u ovoj vežbi da vidimo na koji način koristiti agent za obe uloge, kao prvo generisanja pobude i praćenja transakcija, kao drugo samo za prećenje transakcija pri čemu će generisanje pobude raditi spoljašnji modul. Primetimo ovde na slici da sa gornje strane imamo rezervisana dva dodatna ***uvm_analysis_port***-a upravo za svrhu praćenja bfm transakcija iz spoljašnjeg modula. Ovakav scenario je vrlo tipičan u inženjerskoj praksi i omogućava nam jednostavno korišćenje postojeće ispitne infrastrukture, koju nam daje naš agent, pri čemu prepuštamo generisanje stimulusa drugom bloku. Povezivanje blokova da bismo dobili ovakav scenario vidimo u kodu ***top.sv*** izlistanom niže:

```
module top;

  import uvm_pkg::*;
  import   tinyalu_pkg::*;
`include "tinyalu_macros.svh"
`include "uvm_macros.svh"

 tinyalu_bfm      class_bfm();

 tinyalu class_dut (.A(class_bfm.A), .B(class_bfm.B), .op(class_bfm.op),
              .clk(class_bfm.clk), .reset_n(class_bfm.reset_n),
              .start(class_bfm.start), .done(class_bfm.done),
              .result(class_bfm.result));

 tinyalu_bfm      module_bfm();

 tinyalu module_dut (.A(module_bfm.A), .B(module_bfm.B), .op(module_bfm.op),
               .clk(module_bfm.clk), .reset_n(module_bfm.reset_n),
               .start(module_bfm.start), .done(module_bfm.done),
               .result(module_bfm.result));

 tinyalu_tester_module stim_module(module_bfm);

initial begin
  uvm_config_db #(virtual tinyalu_bfm)::set(null, "*", "class_bfm", class_bfm);
  uvm_config_db #(virtual tinyalu_bfm)::set(null, "*", "module_bfm", module_bfm);
```

```
 run_test("dual_test");
end

endmodule : top
```

Primeticemo da smo u ovom slučaju instancirali dva ***tinyalu_bfm-a: module_bfm*** i ***class_bfm***, instancirali smo i dva DUT-a: ***module_dut*** i ***class_dut***. Vidimo pri instanciranju da je ***class_dut*** povezan na ***class_bfm (***stimulus će u ovom slučaju generisati ***agent),*** dok je ***module_dut*** povezan na ***module_bfm (***stimulus će generisati namenski ***tinyalu_tester_module).***
U ***initial begin*** ubacujemo oba ***bfm***-a u ***config_db***, i pozivamo pokretanje testa ***dual_test.***

Pogledajmo ***dual_test*** (iz fajla ***dual_test.svh***), on izgleda kao i ostali ***uvm_test-ovi*** od ranije, ima ***environment***, koji opisuje vezu između blokova, zatim iz ***config_db***–a preuzima ***bfm***-ove i kreira objekat klase ***env_config_h***, ovo je bitna novina. Ovaj objekat nam služi da opredeli koji objekat nižeg hijerarhijskog nivoa će komunicirati preko kog ***bfm***-a.

```
env_config_h = new(.class_bfm(class_bfm), .module_bfm(module_bfm));
```

U predhodni slučajevima smo videli da klase nižeg nivoa izvlače ***bfm*** iz top nivoa, ali to može da izazove preplitanje viših i nižih nivoa hijerarije. U ovom slučaju imamo dva različita ***bfm***-a, moramo a kontrolišemo koji ćemo koristiti, moramo da odredimo ko će koristiti koji ***bfm***. To radimo kreiranjem ***config*** objekata na svakom nivou hijerarhije gde je to neophodno i prolazimo kroz kreirani objekat do sledećeg nivoa.
Pogledajm sada klasu ***env_config*** (iz fajla ***env_config.svh***)

```
class env_config;
 virtual tinyalu_bfm class_bfm;
 virtual tinyalu_bfm module_bfm;

 function new(virtual tinyalu_bfm class_bfm, virtual tinyalu_bfm module_bfm);
   this.class_bfm = class_bfm;
   this.module_bfm = module_bfm;
 endfunction : new
endclass : env_config
```

Primetićemo da u ovoj klasi imamo dve promenljive istog tipa ***virtual tinyalu_bfm,*** a to su ***class_bfm i module_bfm***. U našem konstruktoru očekujemo dve ovakve klase kao argumente i smeštemo ih u naše prethodno definisane hendlere.
Sada kada smo kreirali novi objekat ***env_config_h***, ubacujemo taj objekat u ***uvm_config_db***.

```
uvm_config_db #(env_config)::set(this, "env_h*", "config", env_config_h);
```

Hendleru ***uvm_config_db***-a predajemo tip ***env_config***, u ***set***–u kažemo da bilo šta što se nađe u ***testbench***-u, a što je iz ***env_h*** ili pripadajuće niže hijerarhije, kada se u tome naiđe na ***config***, to će biti zamenjeno sa ***env_config_h*** hendlerom.

Zatim kreiramo ***env*** korišćenjem ***factory***–ja. Pogledaćemo sada ***env*** (iz fajla ***env.svh***)

```
class env extends uvm_env;
  `uvm_component_utils(env);

  tinyalu_agent       class_tinyalu_agent_h,  module_tinyalu_agent_h;
  tinyalu_agent_config class_config_h, module_config_h;
```

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

  function void build_phase(uvm_phase phase);
    virtual tinyalu_bfm class_bfm;
    virtual tinyalu_bfm module_bfm;
    env_config env_config_h;

if(!uvm_config_db #(env_config)::get(this, "","config", env_config_h))
      `uvm_fatal("RANDOM TEST", "Failed to get CLASS BFM");

    class_config_h  = new(.bfm(env_config_h.class_bfm),  .is_active(UVM_ACTIVE));
    module_config_h = new(.bfm(env_config_h.module_bfm), .is_active(UVM_PASSIVE));

    uvm_config_db #(tinyalu_agent_config)::set(this, "class_tinyalu_agent_h*",
                                "config", class_config_h);

    uvm_config_db #(tinyalu_agent_config)::set(this, "module_tinyalu_agent_h*",
                                "config", module_config_h);



    class_tinyalu_agent_h  = new("class_tinyalu_agent_h",this);
    module_tinyalu_agent_h = new("module_tinyalu_agent_h",this);

  endfunction : build_phase

endclass
```

Ovde je bitno videti da se u ***build_phase*** upravlja ***config*** objektima. Prva stvar koja se preuzima iz ***uvm_config_db***–a je hendler za konfiguraciju okruženja ***env_config_h***. Zatim se kreiraju ***class_config_h*** i ***module_config_h***, a to su ***tinyalu_agent_config*** tipovi. Ako pogledamo u ***tinyalu_agent_config*** vidimo da u njemu imamo pokazivač na ***bfm***, a imamo i **promenljivu** koja kaže da li je nešto **aktivno** ili **pasivno**, to se vidi u kodu niže:

```
class tinyalu_agent_config;

  virtual tinyalu_bfm bfm;
  protected  uvm_active_passive_enum     is_active;

  function new (virtual tinyalu_bfm bfm, uvm_active_passive_enum
                    is_active);
    this.bfm = bfm;
    this.is_active = is_active;
  endfunction : new

  function uvm_active_passive_enum get_is_active();
    return is_active;
  endfunction : get_is_active
```

endclass : tinyalu_agent_config

Dakle vraćamo se ponovo na env.svh na linije 36 i 37 (videti deo koda niže) u kojima smo konstruisali ***tinyalu_agent_config*** objekte pričemu smo opredelili da agent ***class_bfm***-a bude aktivan, dok je agent ***module_bfm***-a pasivan, odnosno učestvovaće samo u monitoringu i generisanju rezultata.

```
class_config_h  = new(.bfm(env_config_h.class_bfm),  .is_active(UVM_ACTIVE));
module_config_h = new(.bfm(env_config_h.module_bfm), .is_active(UVM_PASSIVE));
```

Sada ovako formiranu konfiguraciju smeštamo u ***uvm_config_db***,

```
uvm_config_db #(tinyalu_agent_config)::set(this, "class_tinyalu_agent_h*",
                                "config", class_config_h);
uvm_config_db #(tinyalu_agent_config)::set(this, "module_tinyalu_agent_h*",
                                "config", module_config_h);
```

Kako protumačiti ove dve linije koda? Obe ove instance ***tinyalu_agent_config-a***, se referenciraju na ***config***, pri čemu ***class_tinyalu_agent_h*** se referencira na ***class_config_h***, dok se ***module_tinyalu_agent_h*** referencira na ***module_config_h***.

Sledi kreiranje agenata i njihovo povezivanje sa hendlerima:

```
class_tinyalu_agent_h  = new("class_tinyalu_agent_h",this);
module_tinyalu_agent_h = new("module_tinyalu_agent_h",this);
```

Pogledajmo sada ***build_phase*** iz ***tinyalu_agent*** (iz fajla ***tinyalu_agent.svh***):

```
function void build_phase(uvm_phase phase);

  if(!uvm_config_db #(tinyalu_agent_config)::get(this, "","config",
                                tinyalu_agent_config_h))
   `uvm_fatal("AGENT", "Failed to get config object");
  is_active = tinyalu_agent_config_h.get_is_active();

  if (get_is_active() == UVM_ACTIVE) begin : make_stimulus
    command_f = new("command_f", this);
    tester_h    = tester::type_id::create( "tester_h",this);
    driver_h    = driver::type_id::create("driver_h",this);
  end

  command_monitor_h = command_monitor::type_id::create("command_monitor_h",this);
  result_monitor_h  = result_monitor::type_id::create("result_monitor_h",this);

  coverage_h = coverage::type_id::create("coverage_h",this);
  scoreboard_h = scoreboard::type_id::create("scoreboard_h",this);

  cmd_mon_ap = new("cmd_mon_ap",this);
  result_ap  = new("result_ap", this);

endfunction : build_phase
```

Prva stvar koja se radi je preuzimanje ***tinyalu_agent_config_h*** hendlera iz ***uvm_config_db***-a. Zatim iz njega iščitavamo stanje varijable ***is_active***, odnosno proveravamo da li je agent aktivan. Ako agent jeste aktivan, tada

kreiramo komandni fifo ***command_f,*** zatim fabrički generišemo tester ***tester_h*** i drajver ***driver_h***, ukoliko agent nije aktivan preskačemo ovaj korak. U oba slučaja kreiramo ***command_monitor_h, result_monitor_h, covarage_h, scoreboard_h*** i kreiramo analysis portove ***cmd_mon_ap i result_ap***, time omogućavamo da agent stalno osluškuje magistralu i prati rezultate transakcija, bez obzira da li ih je on inicirao ili ne.