VEŽBA 18

U okviru ove vežbe unapredićemo koncept ***tester*** klase, tako što ćemo u njoj ostaviti samo generisanje stimulus komandi i podataka, dok ćemo problem protokola predaje podataka ***bfm***-u izdvojiti u posebnu ***driver*** klasu.

U direktorijumu:

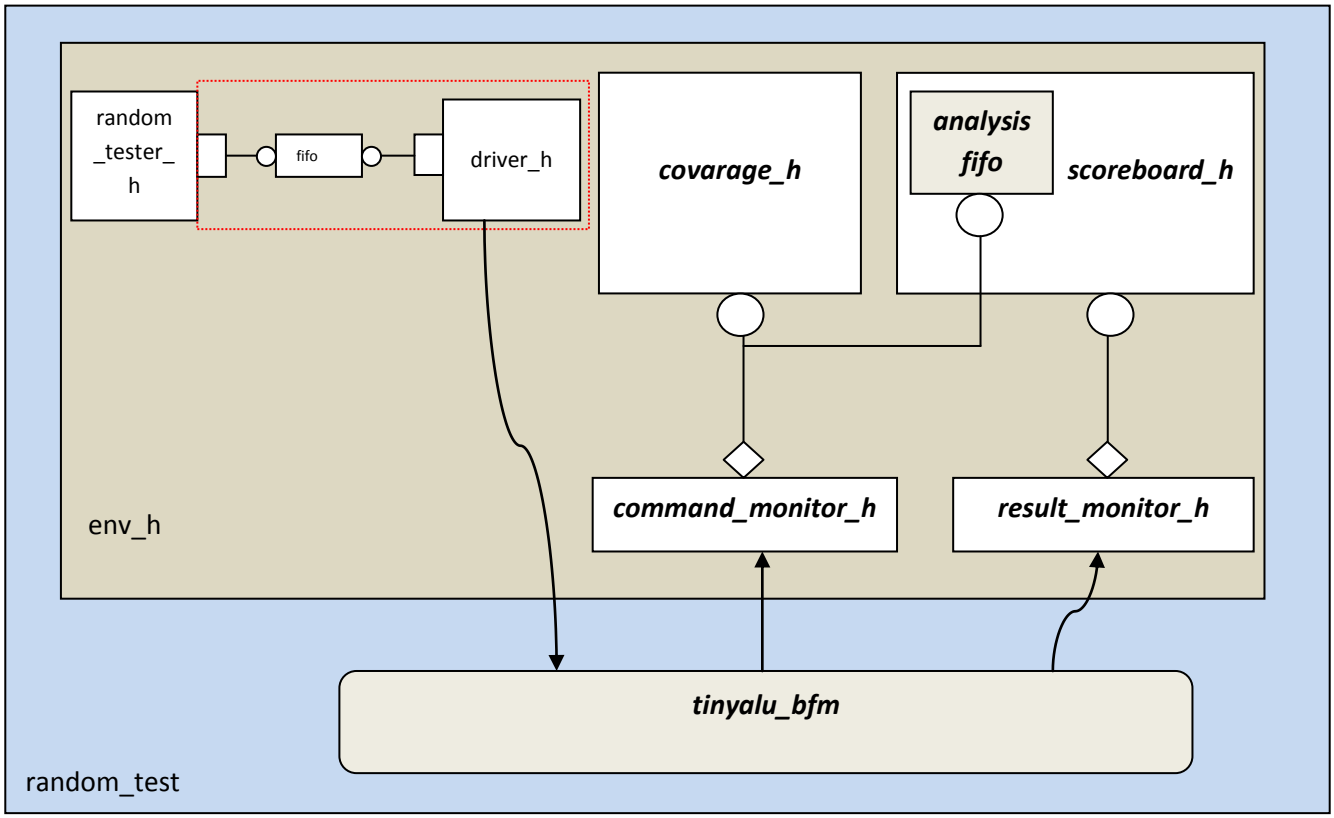
18_Put_and_Get_in_Action,

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

18_Put_and_Get_in_Action\tb_classes

nalaze se fajlovi: ***add_test.svh, add_tester.svh, base_tester.svh, command_monitor.svh, coverage.svh, driver.svh, env.svh, random_test.svh, random_tester.svh, result_monitor.svh, scoreboard.svh, vcs_base_tester.svh***

U prethodnoj vežbi (broj 17) imali smo izvor informacija kapsuliran u ***producer*** bloku, što je obuhvatalo generisanje podataka i postavljanje podataka na magistralu u skladu sa bfm protokolom. Ovakav pristup u kome jedan blok radi dvostruku funkciju nije dobra praksa. Dalako je bolje blok razdeliti na onaj koji generiše informacije i potom drugi blok koji te informacije predaje bfm-u u skladu sa potrebnim protokolom. Ovo razdvajanje je neophodno usvojiti kao praksu pre svega što u složenim sistemima oba ova problema predstavljaju vrlo zahtevne inženjerske zadatke i vrlo je praktično razdvojiti ih! Blok dijagram ovako unapređenog sistema dat je na slici 1.



Slika 1

Kao što je prikazano na *slici1* podelićemo operaciju na dve klase, jednu za odabir operacije, a drugu za interakciju sa ***BFM***-om. Videćemo kasnije da ova podela dozvoljava mnogo veću fleksibilnost u načinu na koji naš ***testbench*** radi. Označeni deo testbenča je nov. Termin ***driver_h*** se odnosi na objekat koji preuzima podatke iz ***testbench***-a i pretvara ih u signale na ***BFM***-u. Dodali smo ***driver_h*** našem ***testbench***-u i povezali ga sa objektom ***random_tester_h*** koristeći ***fifo***.

Pogledajmo sada ***base_tester***, fajl ***base_tester.svh***

```
`ifdef QUESTA
virtual class base_tester extends uvm_component;
`else
class base_tester extends uvm_component;
`endif

`uvm_component_utils(base_tester)
  virtual tinyalu_bfm bfm;

  uvm_put_port #(command_s) command_port;

  function void build_phase(uvm_phase phase);
    command_port = new("command_port", this);
  endfunction : build_phase

  pure virtual function operation_t get_op();

  pure virtual function byte get_data();

  task run_phase(uvm_phase phase);
    byte       unsigned       iA;
    byte       unsigned       iB;
    operation_t               op_set;
    command_s    command;

    phase.raise_objection(this);
    command.op = rst_op;
    command_port.put(command);
    repeat (1000) begin : random_loop
      command.op = get_op();
      command.A =  get_data();
      command.B =  get_data();
      command_port.put(command);
    end : random_loop
    #500;
    phase.drop_objection(this);
  endtask : run_phase

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

endclass : base_tester
```

Podsećamo se da ***base_tester*** klasu nasleđuje naš ***random_tester***, a njega nasleđuje ***add_tester.*** Za razliku od ***base_tester***-a iz vežbe broj 16, gde smo kroz njega direktno pristupali ***bfm***-u, (sećamo se preuzimali smo ga iz ***uvm_config_db*** –a), ovde ćemo podatke predavati ***put*** metodom našeg ***command_port***-a. U ovom primeru skriptom ***run.do*** pozivamo testove redom prvo ***random_tester***, zatim ***add_tester***. ***Base_tester*** generiše operande i komande iz metode ***run_phase*** i ovog puta ih prebacuje korišćenjem ***uvm_put_port*** klase, instancirani objekat po imenu ***command_port***, a tip podatka koji se predaje je struktura ***command_s***. (već viđeno u vežbi broj 17). Fragment koda oko ***run_phase base_tester***-a priložen je niže:

```
pure virtual function operation_t get_op();
pure virtual function byte get_data();

task run_phase(uvm_phase phase);
  byte       unsigned      iA;
  byte       unsigned      iB;
  operation_t              op_set;
  command_s    command;

  phase.raise_objection(this);
  command.op = rst_op;
  command_port.put(command);
  repeat (1000) begin : random_loop
    command.op = get_op();
    command.A =  get_data();
    command.B =  get_data();
    command_port.put(command);
  end : random_loop
  #500;
  phase.drop_objection(this);
endtask : run_phase
```

Vidimo da koristimo bloking metodu za predaju command varijable ***put***, a evo i motivacije, ovaj objekat ima isključivu namenu da formira komande i predaje ih ***driver***-u, on se mora prilagoditi ***driver***-ovom kapacitetu za prijem, odnosno apsorbciju ovih komandi i njihovu postavljanje na ***bfm***. i kako mu je to jedini zadatak, prirodno je da bude u stanju čekanja (blokade) sve dok ***driver*** ne preuzme podatke iz ***fifo***. U ***run_phase*** imamo ***raise_objection***, kreiramo ***rst_op*** komandu, postavljamo je u ***command_port,*** startujemo petlju i popunjavamo ***command***,i ponovo koristimo tu struct ***command***, postavljamo ***get_op*** i ***get_data***, i onda postavljamo u ***command_port***, ali pošto smo tu prethodno postavili reset operaciju, ovo će biti blokirano sve dok ***driver*** (kao korisnik) ne iščita aktuelni sadržaj ***fifo***. Uloga ***driver***-a je komunikacia sa DUT-om preko ***bfm***-a. Njegov kod dat je niže:

```
class driver extends uvm_component;
  `uvm_component_utils(driver)

  virtual tinyalu_bfm bfm;
```

```
  uvm_get_port #(command_s) command_port;

  function void build_phase(uvm_phase phase);
    if(!uvm_config_db #(virtual tinyalu_bfm)::get(null, "*","bfm", bfm))
      $fatal("Failed to get BFM");
    command_port = new("command_port",this);
  endfunction : build_phase

  task run_phase(uvm_phase phase);
    command_s   command;
    shortint    result;

    forever begin : command_loop
      command_port.get(command);
      bfm.send_op(command.A, command.B, command.op, result);
    end : command_loop
  endtask : run_phase

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

endclass : driver
```

U ovom slučaju driver ima ***uvm_get_port*** koji se kreira u ***build_phase***, a zatim u ***run_phase*** u beskonačnoj petlji preuzima ***command_s,*** pomoću ***get*** metode sa ***command_port***-a, interpretirane podatke i komandu predaje ***bfm***-u komandom ***send_op*** i automatski preuzima ***result***, koji uzgred u ovom bloku ne koristimo.

Povezivanje ***random_tester***-a sa ***driver***-om urađeno je u ***env*** objektu. ***Env*** objekat izlistan je u kodu niže. Pogledajmo njegov ***connect_phase***. Vidimo da je povezivanje urađeno korišćenjem ***uvm_tlm_fifo*** objekta, po imenu ***command_f*** koji prebacuje naše ***command_s*** podatke. Povezivanje ***scoreboard*** objekta, ***coverage*** objekta sa ***command_monitor***-om, odnosno ***result_monitor***-om obavljeno je na isti način kao u vežbi 16.

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

  random_tester    random_tester_h;
  driver    driver_h;
  uvm_tlm_fifo #(command_s) command_f;

  coverage   coverage_h;
  scoreboard scoreboard_h;
  command_monitor command_monitor_h;
  result_monitor  result_monitor_h;


  function void build_phase(uvm_phase phase);
```

```
    command_f = new("command_f", this);
    random_tester_h = random_tester::type_id::create("random_tester_h",this);
    driver_h        = driver::type_id::create("drive_h",this);

      coverage_h     =  coverage::type_id::create ("coverage_h",this);
      scoreboard_h    = scoreboard::type_id::create("scoreboard_h",this);
      command_monitor_h   = command_monitor::type_id::create("command_monitor_h",this);
      result_monitor_h= result_monitor::type_id::create("result_monitor_h",this);
    endfunction : build_phase

    function void connect_phase(uvm_phase phase);
      driver_h.command_port.connect(command_f.get_export);
      random_tester_h.command_port.connect(command_f.put_export);

      result_monitor_h.ap.connect(scoreboard_h.analysis_export);

      command_monitor_h.ap.connect(scoreboard_h.cmd_f.analysis_export);
      command_monitor_h.ap.connect(coverage_h.analysis_export);

    endfunction : connect_phase


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

endclass
```