VEŽBA 3

Na ovoj vežbi ćemo se upoznati sa ***BFM***-om što znači **Bus Functional Model.**

U direktorijumu: ***03_Interfaces_and_BFMs***

nalaze se verilog fajlovi: **top.sv, tinyalu_bfm.sv, tinyalu_pkg.sv , coverage.sv, scoreboard.sv, tester.sv**

Verilog fajl ***top.sv*** predstavlja vrh hijerarije i sastoji se iz sledećih podblokova:

```
module top;
tinyalu_bfm    bfm();
tester    tester_i    (bfm);
coverage   coverage_i  (bfm);
scoreboard scoreboard_i(bfm);
tinyalu DUT  (.A(bfm.A), .B(bfm.B), .op(bfm.op),
              .clk(bfm.clk), .reset_n(bfm.reset_n),
              .start(bfm.start), .done(bfm.done), .result(bfm.result));
endmodule : top
```

Ovde treba zapaziti da je povezivanje podblokova sistema realizovano korišćenjem BFM-a. Kao što smo videli na početku ove vežbe, sam naziv BFM dolazi kao skraćenica koja označava funkcionalni model magistrale. Osnovna prednost ovako deklarisane magistrale je u tome što možemo složenu strukturu signala koji povezuju različite blokove grupisati unutar jedinstvenog BFM-a i zatim umesto povezivanja signal po signal, kao što se radi klasično u VHDL-u ili Verilogu dovoljno je povezivanje uraditi postavljanjem formalne instance ovako formirane magistrale. System Verilog jezik nam omogućava kreiranje BFM-ova upotrebom njegove ***interface*** konstrukcije.

Možemo uočiti različito povezivanje prva tri bloka, kod kojih samo prosledimo u listu portova ***bfm***, dok naš blok koji se testira, njegovoj instanci DUT prosleđujemo signale iz ***bfm***-a jedan po jedan, ovo je posledica činjenice da je sam tinyalu dizajn napisan klasično u VHDL-u i ne podržava BFM tehniku povezivanja.

Fokusirajmo se sada na ***tinyalu_bfm.sv*** . On je deklarisan kao interface, u njemu su deklarisani signali koji su potrebni za povezivanje našeg kompletnog DUT-a. Zanimljivo je uočiti da pored ovih signala System Verilog dozvoljava da se unutar interface-a postavi generator takta, zatim taskovi, u našem slučaju imamo task za resetovanje i task za slanje operacije. Praktično ovakvo proširenje koncepta magistrale, omogućava da se formalno u magistralu spakuje kompletna infrastruktura koja je potrebna za pokretanje i povezivanje celog testbench-a.

```
interface tinyalu_bfm;
  import tinyalu_pkg::*;

  byte      unsigned     A;
  byte      unsigned     B;
  bit       clk;
  bit       reset_n;
  wire [2:0]  op;
  bit       start;
  wire      done;
```

```
wire [15:0]  result;
operation_t  op_set;

assign op = op_set;

initial begin
  clk = 0;
  forever begin
    #10;
    clk = ~clk;
  end
end


task reset_alu();
  reset_n = 1'b0;
  @(negedge clk);
  @(negedge clk);
  reset_n = 1'b1;
  start = 1'b0;
endtask : reset_alu

task send_op(input byte iA, input byte iB, input operation_t iop, output shortint alu_result);

  op_set = iop;

  if (iop == rst_op) begin
    @(posedge clk);
    reset_n = 1'b0;
    start = 1'b0;
    @(posedge clk);
    #1;
    reset_n = 1'b1;
  end else begin
    @(negedge clk);
    A = iA;
    B = iB;
    start = 1'b1;
    if (iop == no_op) begin
      @(posedge clk);
      #1;
      start = 1'b0;
    end else begin
      do
        @(negedge clk);
```

```
        while (done == 0);
        start = 1'b0;
      end
    end // else: !if(iop == rst_op)

  endtask : send_op

endinterface : tinyalu_bfm
```

Prednosti ovakvog kapsuliranja magistrale sa pratećim taskovima i izvorom takta možemo videti unutar modula ***tester.*** Ovaj blok korišćenjem funkcija generiše slučajnim izborom operacije i operande. Pored toga u njegovom "***initial begin***" bloku postavljena je kompletna sekvenca pobude sistema, što podrazumeva pozivanje reseta, zatim u petlji 1000 pozivanja slučajnih operacija sa slučajnim operandima.

```
module tester(tinyalu_bfm bfm);
  import tinyalu_pkg::*;

  function operation_t get_op();
    bit [2:0] op_choice;
    op_choice = $random;
    case (op_choice)
      3'b000 : return no_op;
      3'b001 : return add_op;
      3'b010 : return and_op;
      3'b011 : return xor_op;
      3'b100 : return mul_op;
      3'b101 : return no_op;
      3'b110 : return rst_op;
      3'b111 : return rst_op;
    endcase // case (op_choice)
  endfunction : get_op

  function byte get_data();
    bit [1:0] zero_ones;
    zero_ones = $random;
    if (zero_ones == 2'b00)
      return 8'h00;
    else if (zero_ones == 2'b11)
      return 8'hFF;
    else
      return $random;
  endfunction : get_data

  initial begin
    byte      unsigned      iA;
```

```
    byte      unsigned     iB;
    operation_t            op_set;
    shortint   result;

    bfm.reset_alu();
    repeat (1000) begin : random_loop
      op_set = get_op();
      iA = get_data();
      iB = get_data();
      bfm.send_op(iA, iB, op_set, result);
    end : random_loop
    $stop;
  end // initial begin
endmodule : tester
```

Blok ***coverage*** je prilagođen upoterbi bfm-a, pri čemu je suština postavljanja ***covergoup***-a i korpi ostala ista, jedina razlika ogleda se u tome što se pristup signalima sada radi preko bfm-a, dok je u prethodnoj vežbi kada je coverage rađen direktno u testbech-u, pristup signalima bio direktan.

Scoreboard blok je takođe prilagođen pristupu signalima preko bfm-a, dok je sistemska funkcionalnost nepromenjena.

U konceptu rada sa System Verilog interface-ima, odnosno BFM-ovima, pogotovo kada se dublje upustimo u objektno orijentisanu verifikaciju, pristup BFM-u sa više različitih strana uz rešavanje problema rukovanja istim postaće ključni zadaci koje UVM mora da rešava.