EGC 455

SOC Design &Verification

Parameterization, Factory & *Object Oriented Testbench*

Baback Izadi
Division of Engineering Programs
bai@engr.newpaltz.edu

1

Parameterization

- Parameterization is the cornerstone of UVM
- Two approaches
  - Static methods and static variables
  - Instantiation of classes and use of objects

SUNY – New Paltz
Division of Engineering Programs

2

Example of Parameters in a Module and its instantiation

```
module RAM #(awidth, dwidth) (
                        input wire [awidth-1:0] address,
                        inout wire [dwidth-1:0] data,
                        input we);

  initial $display("awidth: %0d  dwidth %0d",awidth, dwidth);
  // code to implement RAM
endmodule // RAM
```

```
module top;
        wire [ 7:0] address;
        wire [15:0] data;
        wire write_enable;

        RAM #(.awidth(8), .dwidth(16)) my_ram(address, data, write_enable);
endmodule // top
```

SUNY – New Paltz
Elect. & Comp. Eng.

3

Copy of our animal classes

```
virtual class animal;
  protected int age=-1;
  protected string name;
   function new(int a, string n);
     age = a;
     name = n;
  endfunction : new

  function int get_age();
      return age;
  endfunction : get_age

  function string get_name();
    return name;
  endfunction : get_name

  pure virtual function void
make_sound();
 endclass : animal
```

```
class lion extends animal;
   protected string       name;
   function new(int age, string n);
     super.new(age, n);
  endfunction : new

  function void make_sound();
     $display ("The lion, %s, says Roar", get_name());
  endfunction : make_sound

endclass : lion
```

```
class chicken extends animal;
    function new(int age, string n);
      super.new(age, n);
  endfunction : new

 function void make_sound();
     $display ("The Chicken, %s, says BECAWW",
get_name());
   endfunction : make_sound
 endclass : chicken
```

Elect. & Comp. Eng.

4

Instead of Parameter being a number, we make it a type, using static methods and variables

```
class animal_cage #(type T);

  protected static T cage[$];

  static function void cage_animal(T l);
    cage.push_back(l);
  endfunction : cage_animal

  static function void list_animals();
    $display("Animals in cage:");
    foreach (cage[i])
      $display(cage[i].get_name());
  endfunction : list_animals

endclass : animal_cage
```

```
module top;

  initial begin
    lion   lion_h;
    chicken  chicken_h;
    lion_h = new(15, "Mustafa");
    animal_cage #(lion)::cage_animal(lion_h);
    lion_h = new(15, "Simba");
    animal_cage #(lion)::cage_animal(lion_h);

    chicken_h = new(1, "Clucker");
    animal_cage #(chicken)::cage_animal(chicken_h);
    chicken_h = new(1, "Scratchy");
    animal_cage #(chicken)::cage_animal(chicken_h);

    $display("-- Lions --");
    animal_cage #(lion)::list_animals();
    $display("-- Chickens --");
    animal_cage #(chicken)::list_animals();
  end

endmodule : top
```

SUNY – New Paltz
Elect. & Comp. Eng.

5

Copy of our animal classes

```
virtual class animal;
  protected int age=-1;
  protected string name;

  function new(int a, string n);
    age = a;
    name = n;
  endfunction : new

  function int get_age();
      return age;
  endfunction : get_age

  function string get_name();
    return name;
  endfunction : get_name

  pure virtual function void
make_sound();
 endclass : animal
```

```
class lion extends animal;
  protected string       name;

  function new(int age, string n);
    super.new(age, n);
  endfunction : new

  function void make_sound();
    $display ("The lion, %s, says Roar", get_name());
  endfunction : make_sound

 endclass : lion
```

```
class chicken extends animal;
   function new(int age, string n);
     super.new(age, n);
  endfunction : new

  function void make_sound();
    $display ("The Chicken, %s, says BECAWW",
get_name());
  endfunction : make_sound
 endclass : chicken
```

Elect. & Comp. Eng.

6

Parameter type using non-static methods and variables – Instantiating animal cage

```
class animal_cage #(type T);

  protected T cage[$];

  function void cage_animal(T l);
    cage.push_back(l);
  endfunction : cage_animal

  function void list_animals();
    $display("Animals in cage:");
    foreach (cage[i])
      $display(cage[i].get_name());
  endfunction : list_animals

endclass : animal_cage
```

```
module top;
 lion   lion_h;
 chicken  chicken_h;
  animal_cage #(lion)  lion_cage;  // create a variable for lion
 animal_cage #(chicken) chicken_cage;
  initial begin
    lion_cage = new();
    lion_h = new(15, "Mustafa");
    lion_cage.cage_animal(lion_h);
    lion_h = new(15, "Simba");
    lion_cage.cage_animal(lion_h);
     chicken_cage = new();
    chicken_h = new(1, "Little Red Hen");
    chicken_cage.cage_animal(chicken_h);
     chicken_h = new(1, "Lady Clucksalot");
    chicken_cage.cage_animal(chicken_h);
     $display("-- Lions --");
    lion_cage.list_animals();
    $display("-- Chickens --");
    chicken_cage.list_animals();
  end
endmodule : top
```

SUNY – New Paltz
Elect. & Comp. Eng.

7

The Factory Pattern

Most visible design pattern in the UVM.

Flexible ways of creating objects, as opposed to hardcoding the objects

Used to create dynamically adaptable testbenches

SUNY – New Paltz
Division of Engineering Programs

8

Copy of our animal classes

```
virtual class animal;
  protected int age=-1;
  protected string name;

  function new(int a, string n);
    age = a;
    name = n;
  endfunction : new

  function int get_age();
      return age;
  endfunction : get_age

  function string get_name();
    return name;
  endfunction : get_name

  pure virtual function void
make_sound();
 endclass : animal
```

```
class lion extends animal;
  bit          thorn_in_paw = 0;
  function new(int age, string n);
    super.new(age, n);
  endfunction : new

  function void make_sound();
    $display ("The lion, %s, says Roar", get_name());
  endfunction : make_sound

endclass : lion
```

```
class chicken extends animal;
  function new(int age, string n);
    super.new(age, n);
  endfunction : new

  function void make_sound();
    $display ("The Chicken, %s, says BECAWW",
get_name());
  endfunction : make_sound
 endclass : chicken
```


9

Example: we want to read a list of animals and instantiate objects to represent them; Dynamic way of choosing our animals or randomly choose our animals. Impossible with hardcoding.

```
class animal_factory;
  static function animal make_animal(string species,
                                     int age, string name);
    chicken chicken;
    lion lion;
    case (species)
     "lion" : begin
       lion = new(age, name);
       return lion;
     end
      "chicken" : begin
       chicken = new(age, name);
       return chicken;
     end
      default :
       $fatal (1, {"No such animal: ", species});
       endcase // case (species)
  endfunction : make_animal
endclass : animal_factory
```

```
class animal_cage #(type T=animal);

  static T cage[$];

  static function void cage_animal(T l);
    cage.push_back(l);
  endfunction : cage_animal

  static function void list_animals();
    $display("Animals in cage:");
    foreach (cage[i])
      $display(cage[i].get_name());
  endfunction : list_animals

endclass : animal_cage
```


10

```
module top;
  initial begin  // used to test our factory
    animal animal_h;
    lion   lion_h;
    chicken  chicken_h;
    bit cast_ok;
    /* create a lion called Mustafa of age 15 and store it in animal_h   */
    animal_h = animal_factory::make_animal("lion", 15, "Mustafa");
    animal_h.make_sound(); // Use polymorphisim to invoke make_sound in lion
    /*  animal_h.thorn_in_paw generates error; only in lion class and not animal class.
    solution: convert animal object to lion object using cast */
    cast_ok = $cast(lion_h, animal_h); // $cast returns 1 of casting is successful
    if ( ! cast_ok)
      $fatal(1, "Failed to cast animal_h to lion_h");
     if (lion_h.thorn_in_paw) $display("He looks angry!"); // now thorn_in_paw works
    animal_cage#(lion)::cage_animal(lion_h);    // put Mustafa in lion cage
    if (!$cast(lion_h, animal_factory::make_animal("lion", 2, "Simba")))
      $fatal(1, "Failed to cast animal from factory to lion_h");
    animal_cage#(lion)::cage_animal(lion_h);
    if(!$cast(chicken_h ,animal_factory::make_animal("chicken", 1, "Clucker")))
      $fatal(1, "Failed to cast animal factory result to chicken_h");
```

Elect. & Comp. Eng.

11

top (Cont.)

```
    animal_cage #(chicken)::cage_animal(chicken_h);
    if(!$cast(chicken_h, animal_factory::make_animal("chicken", 1, "Boomer")))
      $fatal(1, "Failed to cast animal factory result to chicken_h");
    animal_cage #(chicken)::cage_animal(chicken_h);

    $display("-- Lions --");
    animal_cage #(lion)::list_animals();
    $display("-- Chickens --");
    animal_cage #(chicken)::list_animals();
  end
endmodule : top
```

Elect. & Comp. Eng.

12

Object Oriented Testbench

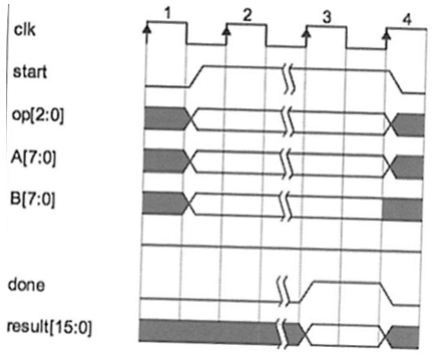
https://www.youtube.com/watch?v=JIzfTrVhDYU&t=211s

- Let's convert the module-based testbench into object –based testbench
- Top – The top-level module that instantiates the testbench class
- Testbench – The top-level class
- Tester – Drives stimulus
- Scoreboard – Checks that the TinyALU is working
- Coverage – Captures functional coverage information

SUNY – New Paltz
Division of Engineering Programs

13

Operation	Opcode
no_op	3'b000
add_op	3'b001
and_op	3'b010
xor_op	3'b011
mul_op	3'b100
unused	3'b101 - 3'b111



SUNY – New Paltz
Elect. & Comp. Eng.

14

Top-level Testbench

- All object-oriented testbench
  - Imports the class definitions
  - Instantiates the DUT and BFM and declares the testbench class variables
  - Instantiates and launches the testbench class

SUNY – New Paltz
Elect. & Comp. Eng.

15

- All class definitions and shared resources are stored in System Verilog packages

```
package tinyalu_pkg;
typedef enum bit[2:0]
        {no_op = 3'b000,
        add_op = 3'b001,
        and_op = 3'b010,
        xor_op = 3'b011,
        mul_op = 3'b100,
        rst_op = 3'b111} operation_t;
`include "coverage.svh"
`include "tester.svh"
`include "scoreboard.svh"
`include "testbench.svh"

endpackage : tinyalu_pkg
```

SUNY – New Paltz
Elect. & Comp. Eng.

16

BFM

```
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
```

SUNY – New Paltz
Elect. & Comp. Eng.

17

BFM (cont.)

```
task send_op(input byte iA, input byte iB, input
operation_t iop, output shortint alu_result);
       if (iop == rst_op)
begin
      @(negedge clk);
      op_set = iop;
      @(posedge clk);
      reset_n = 1'b0;
      start = 1'b0;
      @(posedge clk);
      #1;
      reset_n = 1'b1;
end
       else
              begin
              @(negedge clk);
              op_set = iop;
              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); //wait done
            start = 1'b0;
      end
     alu_result = result;
   end // else: !if(iop == rst_op)

 endtask : send_op

endinterface : tinyalu_bfm
```

Elect. & Comp. Eng.

18

```
module top;
 import  tinyalu_pkg::*;
`include "tinyalu_macros.svh"

  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)); //instantiate DUT

  tinyalu_bfm    bfm();  // instantiate bfm

  testbench   testbench_h;  // declare a variable to hold the testbench

  initial begin
    testbench_h = new(bfm); // instantiate new testbench and pass it a handle to the BFM
    testbench_h.execute(); // lunch execute method to verify the TinyALU
  end

endmodule : top
```

- Similar to module-based, except we replaced the stimulus, self-check, and coverage modules with testbench class.
- Testbench objects uses tasks in BFM to drive stimulus and watch the signals at output and coverage

Elect. & Comp. Eng.

19

```
class testbench;  // object oriented testbench uses objects rather than module to verify DUT
            virtual tinyalu_bfm bfm;  // virtual says this variable will be given a handle to the interface later
            tester   tester_h;
            coverage  coverage_h;
            scoreboard scoreboard_h;

            function new (virtual tinyalu_bfm b); // one way to the handle to the interface into the bfm var.
             bfm = b;
            endfunction : new

            task execute();
                        tester_h   = new(bfm);     // instantiate tester
                        coverage_h  = new(bfm);  // instantiate coverage
                        scoreboard_h = new(bfm);  // instantiate scoreboard

            fork
                        tester_h.execute();
                        coverage_h.execute();
                        scoreboard_h.execute();
            join_none

            endtask : execute
endclass : testbench
```

20

```
class tester;
  virtual tinyalu_bfm bfm; // define bfm variable
  function new (virtual tinyalu_bfm b);
    bfm = b;  // load bfm variable into constructor
  endfunction : new

  protected 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
```

- Difference of class and module-based tester
  - We use variable instead of portlist to access bfm.
  - We use execute () instead of an initial block

```
protected 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
```

SUNY – New Paltz
Elect. & Comp. Eng.

21

Tester (Cont.)

```
task execute(); // called by top level class
    byte        unsigned      iA;
    byte        unsigned      iB;
    shortint    unsigned      result;
    operation_t               op_set;
    bfm.reset_alu();
    op_set = rst_op;
    iA = get_data();
    iB = get_data();
    bfm.send_op(iA, iB, op_set, result);
    op_set = mul_op;
    bfm.send_op(iA, iB, op_set, result);
    bfm.send_op(iA, iB, op_set, result);
    op_set = rst_op;
    bfm.send_op(iA, iB, op_set, result);
```

```
repeat (10) begin : random_loop
      op_set = get_op();
      iA = get_data();
      iB = get_data();
      bfm.send_op(iA, iB, op_set, result );
      $display("%2h %6s %2h = %4h",iA,
op_set.name(), iB, result);
    end : random_loop
    $stop;
  endtask : execute
endclass : tester
```

SUNY – New Paltz
Elect. & Comp. Eng.

22

```
class scoreboard;     // almost identical to the module version
        virtual tinyalu_bfm bfm;
        function new (virtual tinyalu_bfm b);
                bfm = b;
        endfunction : new
        task execute();
                shortint predicted_result;
                forever begin : self_checker        // forever replaces always block in module
                @(posedge bfm.done)
                #1;
                case (bfm.op_set)
                        add_op: predicted_result = bfm.A + bfm.B;
                        and_op: predicted_result = bfm.A & bfm.B;
                        xor_op: predicted_result = bfm.A ^ bfm.B;
                        mul_op: predicted_result = bfm.A * bfm.B;
                endcase // case (op_set)

                if ((bfm.op_set != no_op) && (bfm.op_set != rst_op))
                if (predicted_result != bfm.result)
                        $error ("FAILED: A: %0h  B: %0h  op: %s result: %0h",
                                bfm.A, bfm.B, bfm.op_set.name(), bfm.result);
                end : self_checker
        endtask : execute
endclass : scoreboard
```

23

Coverage

```
class coverage;
   virtual tinyalu_bfm bfm;
   byte       unsigned      A;
  byte       unsigned      B;
  operation_t  op_set;
   covergroup op_cov;

     coverpoint op_set {
       bins single_cycle[] = {[add_op : xor_op],
rst_op,no_op};
       bins multi_cycle = {mul_op};
        bins opn_rst[] = ([add_op:mul_op] => rst_op);
       bins rst_opn[] = (rst_op => [add_op:mul_op]);
        bins sngl_mul[] = ([add_op:xor_op],no_op =>
mul_op);
       bins mul_sngl[] = (mul_op => [add_op:xor_op],
no_op);
        bins twoops[] = ([add_op:mul_op] [* 2]);
       bins manymult = (mul_op [* 3:5]);
     }
   endgroup
```

```
covergroup zeros_or_ones_on_ops;

    all_ops : coverpoint op_set {
      ignore_bins null_ops = {rst_op,
no_op};}

    a_leg: coverpoint A {
      bins zeros = {'h00};
      bins others= {['h01:'hFE]};
      bins ones  = {'hFF};
    }

    b_leg: coverpoint B {
      bins zeros = {'h00};
      bins others= {['h01:'hFE]};
      bins ones  = {'hFF};
    }
```

24

```
op_00_FF:  cross a_leg, b_leg, all_ops {
      bins add_00 = binsof (all_ops) intersect {add_op} && (binsof (a_leg.zeros) || binsof (b_leg.zeros));
       bins add_FF = binsof (all_ops) intersect {add_op} &&(binsof (a_leg.ones) || binsof (b_leg.ones));

      bins and_00 = binsof (all_ops) intersect {and_op} &&(binsof (a_leg.zeros) || binsof (b_leg.zeros));

      bins and_FF = binsof (all_ops) intersect {and_op} &&(binsof (a_leg.ones) || binsof (b_leg.ones));

      bins xor_00 = binsof (all_ops) intersect {xor_op} &&(binsof (a_leg.zeros) || binsof (b_leg.zeros));

      bins xor_FF = binsof (all_ops) intersect {xor_op} &&(binsof (a_leg.ones) || binsof (b_leg.ones));

      bins mul_00 = binsof (all_ops) intersect {mul_op} &&(binsof (a_leg.zeros) || binsof (b_leg.zeros));

      bins mul_FF = binsof (all_ops) intersect {mul_op} &&(binsof (a_leg.ones) || binsof (b_leg.ones));

      bins mul_max = binsof (all_ops) intersect {mul_op} &&(binsof (a_leg.ones) && binsof (b_leg.ones));

      ignore_bins others_only =binsof(a_leg.others) && binsof(b_leg.others);
    }
endgroup
```

SUNY – New Paltz
Elect. & Comp. Eng.

25

```
function new (virtual tinyalu_bfm b);
  op_cov = new();
  zeros_or_ones_on_ops = new();
  bfm = b;
endfunction : new


task execute();
  forever begin  : sampling_block
    @(negedge bfm.clk);
    A = bfm.A;
    B = bfm.B;
    op_set = bfm.op_set;
    op_cov.sample();
    zeros_or_ones_on_ops.sample();
  end : sampling_block
endtask : execute

endclass : coverage
```

SUNY – New Paltz
Elect. & Comp. Eng.

26