EGC 455

SOC Design &Verification

UVM Tests and Components

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

UVM Test

- https://www.youtube.com/watch?v=2KyJHRqbnJU
- UVM is a library of classes and resources based on those classes
  - We use them to create testbenches.
- Creating tests with the Factory, so we can run multiple test in one compilation

SUNY – New Paltz
Division of Engineering Programs

Creating Tests with the Factory

```
if [file exists "work"] {vdel -all}
vlib work
vcom -f dut.f
vlog -f tb.f
vopt top -o top_optimized  +acc +cover=sbfec+tinyalu(rtl).

vsim top_optimized -coverage +UVM_TESTNAME=random_test

set NoQuitOnFinish 1
onbreak {resume}
log /* -r
run -all
coverage exclude -src ../../tinyalu_dut/single_cycle_add_and_xor.vhd -line 49 -code s
coverage exclude -src ../../tinyalu_dut/single_cycle_add_and_xor.vhd -scope /top/DUT/add_and_xor -line
49 -code b
coverage save random_test.ucdb


vsim top_optimized -coverage +UVM_TESTNAME=add_test

set NoQuitOnFinish 1
onbreak {resume}
log /* -r
run -all
coverage exclude -src ../../tinyalu_dut/single_cycle_add_and_xor.vhd -line 49 -code s
coverage exclude -src ../../tinyalu_dut/single_cycle_add_and_xor.vhd -scope /top/DUT/add_and_xor -line
```

SUNY – New Paltz
Elect. & Comp. Eng.

Setting up UVM top module

```
module top;
  import uvm_pkg::*;
`include "uvm_macros.svh"

  import   tinyalu_pkg::*;
`include "tinyalu_macros.svh"
 // instantiate bfm and DUT
  tinyalu_bfm      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));
// instead of passing the bfm to top level object as before, we use config database from UVM resource
 initial begin
  uvm_config_db #(virtual tinyalu_bfm)::set(null, "*", "bfm", bfm); // pass bfm using config_db
  run_test(); // Another feature of the UVM package; uses the command run input to run our test
end

endmodule : top
```

SUNY – New Paltz
Elect. & Comp. Eng.

```
class random_test extends uvm_test;
  `uvm_component_utils(random_test);  /* macro to register
random_test with the factory */
 virtual tinyalu_bfm bfm;

  function new (string name, uvm_component parent);
    super.new(name,parent);
    if(!uvm_config_db #(virtual tinyalu_bfm)::get(null,
"*","bfm", bfm)) // get bfm out of db using static function get
      $fatal("Failed to get BFM");
  endfunction : new
// run phase is like execute task previously
task run_phase(uvm_phase phase);
    random_tester    random_tester_h;
    coverage  coverage_h;
    scoreboard scoreboard_h;
```

SUNY – New Paltz
Elect. & Comp. Eng.

```
    phase.raise_objection(this);

    random_tester_h    =
new(bfm);
    coverage_h  = new(bfm);
    scoreboard_h = new(bfm);

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


random_tester_h.execute();
    phase.drop_objection(this);
  endtask : run_phase

endclass
```

```
class add_test extends uvm_test;
  `uvm_component_utils(add_test);

  virtual tinyalu_bfm bfm;

  function new (string name, uvm_component parent);
    super.new(name,parent);
    if(!uvm_config_db #(virtual
tinyalu_bfm)::get(null, "*","bfm", bfm))
      $fatal("Failed to get BFM");
  endfunction : new

 task run_phase(uvm_phase phase);
    add_tester   add_tester_h;
    coverage  coverage_h;
    scoreboard scoreboard_h;
```

SUNY – New Paltz
Elect. & Comp. Eng.

```
phase.raise_objection(this);

    add_tester_h = new(bfm);
    coverage_h   = new(bfm);
    scoreboard_h   = new(bfm);

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

    add_tester_h.execute();
    phase.drop_objection(this);
  endtask : run_phase

endclass
```

```
class random_tester;
  virtual tinyalu_bfm bfm;
  function new (virtual tinyalu_bfm b);
    bfm = b;
  endfunction : new
  protected virtual 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
```

```
protected virtual 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
```

```
  virtual task execute();
    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
    #500;
  endtask : execute

endclass : random_tester
```

```
class add_tester extends random_tester;

  function new (virtual tinyalu_bfm b);
    super.new(b);
  endfunction : new

  function operation_t get_op();
    bit [2:0] op_choice;
    return add_op;
  endfunction : get_op

endclass : add_tester
```

SUNY – New Paltz
Elect. & Comp. Eng.

```
class scoreboard;
   virtual tinyalu_bfm bfm;

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

 task execute();
   shortint predicted_result;
   forever begin : self_checker
     @(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
```



```
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};
    }
```

SUNY – New Paltz
Elect. & Comp. Eng.

```
    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

  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
```

Elect. & Comp. Eng.

Compiling and Referencing the UVM

- https://newpaltz.knowmia.com/K9sA
- UVM is a library of classes and resources based on those classes
  - We use them to create testbenches.
- Creating tests with the Factory, so we can run multiple test in one compilation

SUNY – New Paltz
Division of Engineering Programs

Using the UVM

- **Import uvm_pkg::* into all modules and packages**
- **Include uvm_macros.svh at the top of packages**

SUNY – New Paltz
Elect. & Comp. Eng.

Referencing the UVM

```
1  package memory_pkg;
2      import uvm_pkg::*;
3      `include "uvm_macros.svh"
4
5  `include "tester.svh"
6  `include "monitor.svh"
7  `include "scoreboard.svh"
8  endpackage // memory_pkg
9
```

Reference the UVM name space and include the macro definitions

SUNY – New Paltz
Elect. & Comp. Eng.

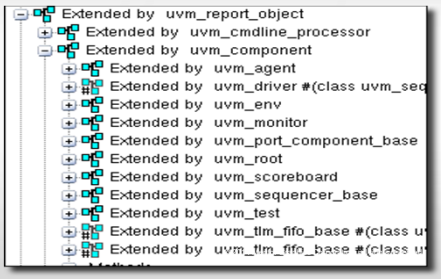
The root of all action

UVM Components

SUNY – New Paltz
Elect. & Comp. Eng.

uvm_component: Mother of all action

- Launched automatically by the UVM
- Contain methods that the UVM calls
- We fill in the methods



SUNY – New Paltz
Elect. & Comp. Eng.

uvm_components use Phases

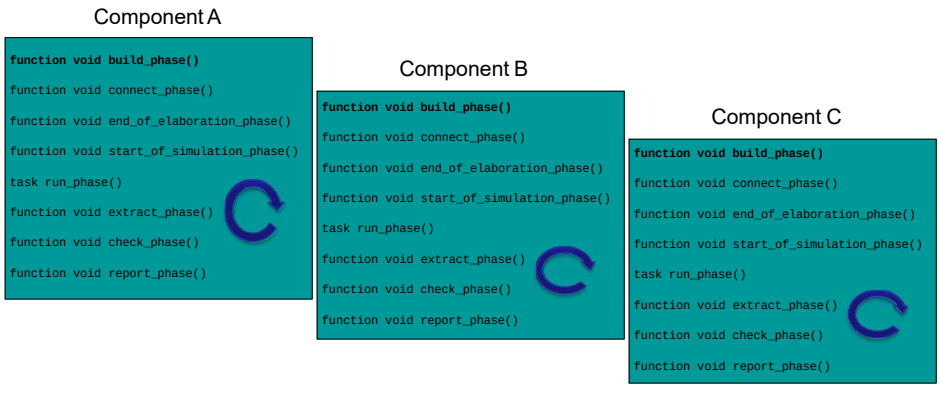
The UVM instantiates all components in their own threads and calls the phase methods in order.

function void build_phase()
function void connect_phase()
function void end_of_elaboration_phase() function void start_of_simulation_phase() task run_phase()
function void extract_phase()
function void check_phase() function void report_phase()

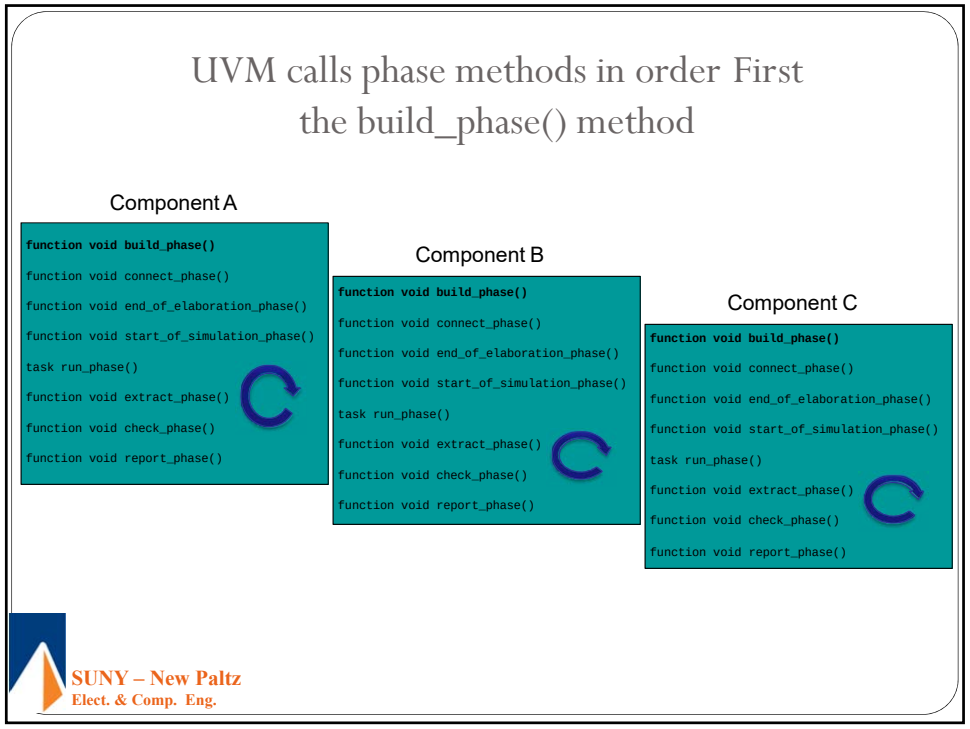
uvm_component

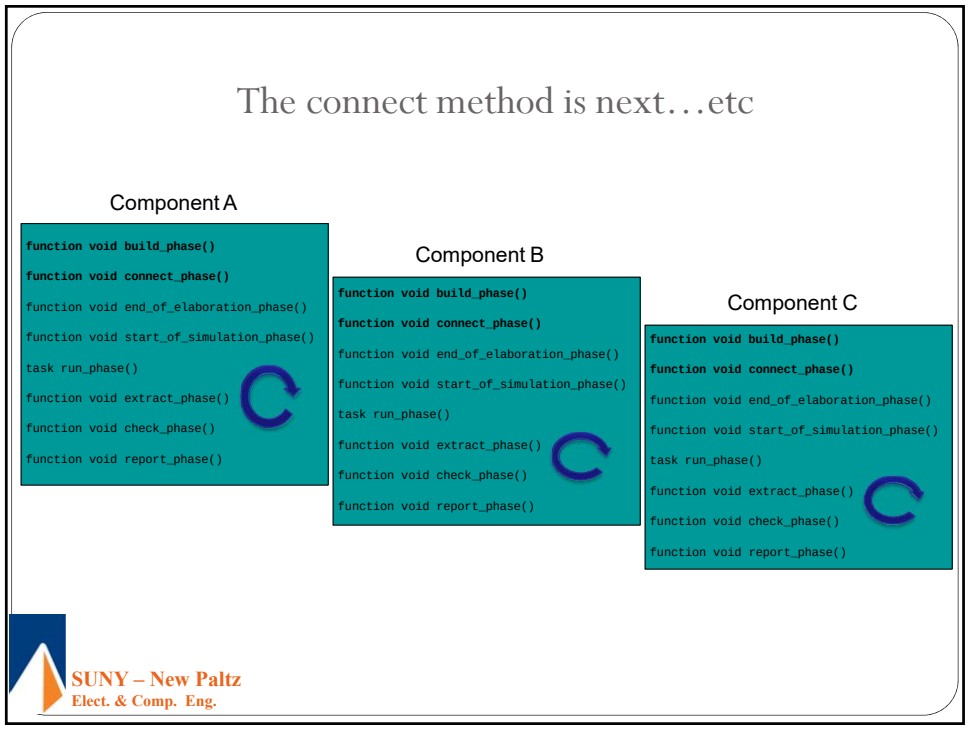
SUNY – New Paltz
Elect. & Comp. Eng.

UVM Instantiates Test
Test Instantiates other Components



SUNY – New Paltz
Elect. & Comp. Eng.


UVM calls phase methods in order First
the build_phase() method
Component A
function void build_phase()
function void connect_phase()
function void end_of_elaboration_phase()
function void start_of_simulation_phase()
task run_phase()
function void extract_phase()
function void check_phase()
function void report_phase()
Component B
function void build_phase()
function void connect_phase()
function void end_of_elaboration_phase()
function void start_of_simulation_phase()
task run_phase()
function void extract_phase()
function void check_phase()
function void report_phase()
Component C
function void build_phase()
function void connect_phase()
function void end_of_elaboration_phase()
function void start_of_simulation_phase()
task run_phase()
function void extract_phase()
function void check_phase()
function void report_phase()
SUNY – New Paltz
Elect. & Comp. Eng.


The connect method is next…etc
Component A
function void build_phase()
function void connect_phase()
function void end_of_elaboration_phase()
function void start_of_simulation_phase()
task run_phase()
function void extract_phase()
function void check_phase()
function void report_phase()
Component B
function void build_phase()
function void connect_phase()
function void end_of_elaboration_phase()
function void start_of_simulation_phase()
task run_phase()
function void extract_phase()
function void check_phase()
function void report_phase()
Component C
function void build_phase()
function void connect_phase()
function void end_of_elaboration_phase()
function void start_of_simulation_phase()
task run_phase()
function void extract_phase()
function void check_phase()
function void report_phase()
SUNY – New Paltz
Elect. & Comp. Eng.

The run_phase() method does the work

Component A

```
function void build_phase()
function void connect_phase()
function void end_of_elaboration_phase()
function void start_of_simulation_phase()
task run_phase()
function void extract_phase()
function void check_phase()
function void report_phase()
```

Component B

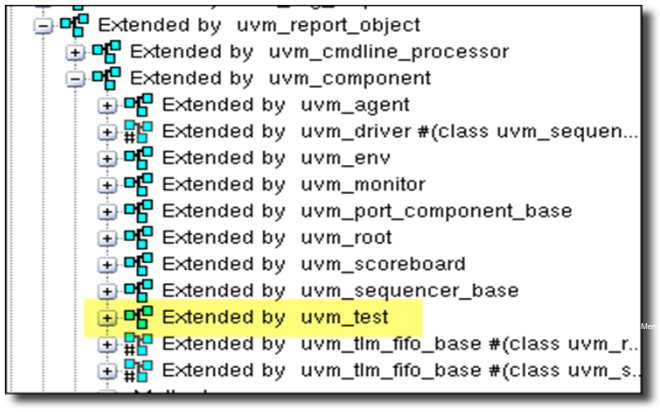
```
function void build_phase()
function void connect_phase()
function void end_of_elaboration_phase()
function void start_of_simulation_phase()
task run_phase()
function void extract_phase()
function void check_phase()
function void report_phase()
```

Component C

```
function void build_phase()
function void connect_phase()
function void end_of_elaboration_phase()
function void start_of_simulation_phase()
task run_phase()
function void extract_phase()
function void check_phase()
function void report_phase()
```

SUNY – New Paltz
Elect. & Comp. Eng.

The uvm_test is an uvm_component



SUNY – New Paltz
Elect. & Comp. Eng.

Running Multiple Tests

UVM Tests

SUNY – New Paltz
Elect. & Comp. Eng.

Running Tests with the UVM

```
if [file exists work] {vdel -all}
vlib work
vlog -f compile.f
vsim  top +UVM_TESTNAME=cat
set  NoQuitOnFinish 1
onbreak {resume}
run -all
vsim  top +UVM_TESTNAME=dog
run -all
```

```
# ----------------------------------------------------------------
# UVM-1.1b
# (C) 2007-2012 Mentor Graphics Corporation
# (C) 2007-2012 Cadence Design Systems, Inc.
# (C) 2006-2012 Synopsys, Inc.
# (C) 2011-2012 Cypress Semiconductor Corp.
# ----------------------------------------------------------------
#
# UVM_INFO @ 0: reporter [RNTST] Running test cat...
# UVM_INFO @ 0: uvm_test_top [cat] The cat says 'meow'
```

```
# ----------------------------------------------------------------
# UVM-1.1b
# (C) 2007-2012 Mentor Graphics Corporation
# (C) 2007-2012 Cadence Design Systems, Inc.
# (C) 2006-2012 Synopsys, Inc.
# (C) 2011-2012 Cypress Semiconductor Corp.
# ----------------------------------------------------------------
#
# UVM_INFO @ 0: reporter [RNTST] Running test dog...
# UVM_INFO @ 0: uvm_test_top [dog] En espanol, el perro dice 'guau, guau'
```

SUNY – New Paltz
Elect. & Comp. Eng.

Running Tests with the UVM

```
if [file exists work] {vdel -all}
vlib work
vlog -f compile.f
vsim  top +UVM_TESTNAME=cat
set  NoQuitOnFinish 1
onbreak {resume}
run -all
vsim  top +UVM_TESTNAME=dog
run -all
```

We tell the UVM what object to create using the **UVM_TESTNAME** plusarg.

How does this work?

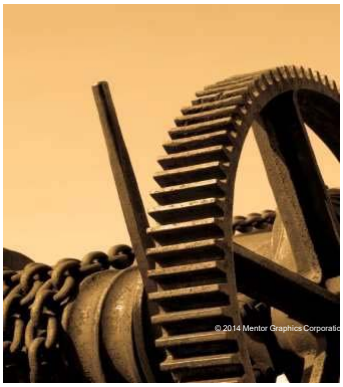
```
# ----------------------------------------------------------------
# UVM-1.1b
# (C) 2007-2012 Mentor Graphics Corporation
# (C) 2007-2012 Cadence Design Systems, Inc.
# (C) 2006-2012 Synopsys, Inc.
# (C) 2011-2012 Cypress Semiconductor Corp.
# ----------------------------------------------------------------
#
# UVM_INFO @ 0: reporter [RNTST] Running test cat...
# UVM_INFO @ 0: uvm_test_top [cat] The cat says 'meow'
```

```
# ----------------------------------------------------------------
# UVM-1.1b
# (C) 2007-2012 Mentor Graphics Corporation
# (C) 2007-2012 Cadence Design Systems, Inc.
# (C) 2006-2012 Synopsys, Inc.
# (C) 2011-2012 Cypress Semiconductor Corp.
# ----------------------------------------------------------------
#
# UVM_INFO @ 0: reporter [RNTST] Running test dog...
# UVM_INFO @ 0: uvm_test_top [dog] En espanol, el perro dice 'guau, guau'
```

SUNY – New Paltz
Elect. & Comp. Eng.

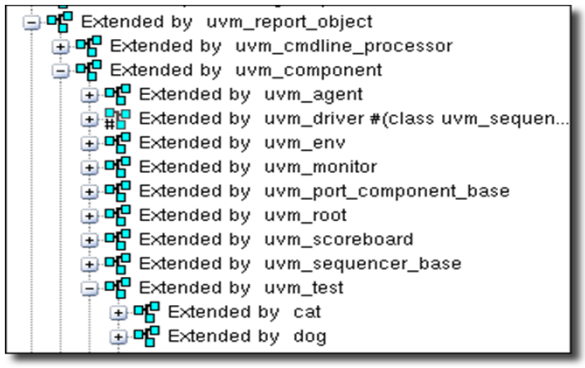
The UVM has a Built-in Factory

1. **We create classes that extend uvm_test**
1. **We call out the classes with the UVM_TESTNAMEplusarg**
2. **We start the test with the run_test()task (defined in uvm_pkg)**
3. **run_test()uses the UVM factory to create and launch a test object.**



SUNY – New Paltz
Elect. & Comp. Eng.

Creating a Test



Tests are classes that extend **uvm_test**

SUNY – New Paltz
Elect. & Comp. Eng.

Creating a Test

```
1  class cat extends uvm_test;
2    `uvm_component_utils(cat)
3
4    function new(string name, uvm_component parent);
5      super.new(name, parent);
6    endfunction : new
7
8    task run_phase(uvm_phase phase);
9      phase.raise_objection(this);
10     super.run();
11     `uvm_info("cat","The cat says 'meow'", UVM_INFO);
12     phase.drop_objection(this);
13   endtask
14 endclass : cat
```

Extend **uvm_test**

SUNY – New Paltz
Elect. & Comp. Eng.

Creating a Test

```
1  class cat extends uvm_test;
2    `uvm_component_utils(cat)
3
4    function new(string name, uvm_component parent);
5      super.new(name, parent);
6    endfunction : new
7
8    task run_phase(uvm_phase phase);
9      phase.raise_objection(this);
10     super.run();
11     `uvm_info("cat","The cat says 'meow'", UVM_INFO);
12     phase.drop_objection(this);
13   endtask
14 endclass : cat
```

Tell the UVM factory about this test

SUNY – New Paltz
Elect. & Comp. Eng.

Creating a Test

```
1  class cat extends uvm_test;
2    `uvm_component_utils(cat)
3
4    function new(string name, uvm_component parent);
5      super.new(name, parent);
6    endfunction : new
7
8    task run_phase(uvm_phase phase);
9      phase.raise_objection(this);
10     super.run();
11     `uvm_info("cat","The cat says 'meow'", UVM_INFO);
12     phase.drop_objection(this);
13   endtask
14 endclass : cat
```

Must have a constructor

SUNY – New Paltz
Elect. & Comp. Eng.

Creating a Test

```
1  class cat extends uvm_test;
2    `uvm_component_utils(cat)
3
4    function new(string name, uvm_component parent);
5      super.new(name, parent);
6    endfunction : new
7
8    task run_phase(uvm_phase phase);
9      phase.raise_objection(this);
10     super.run();
11     `uvm_info("cat","The cat says 'meow'", UVM_INFO);
12     phase.drop_objection(this);
13   endtask
14 endclass : cat
```

The UVM will call the **run_phase()** task

SUNY – New Paltz
Elect. & Comp. Eng.

Creating a Test

```
1  class cat extends uvm_test;
2    `uvm_component_utils(cat)
3
4    function new(string name, uvm_component parent);
5      super.new(name, parent);
6    endfunction : new
7
8    task run_phase(uvm_phase phase);
9      phase.raise_objection(this);
10     super.run();
11     `uvm_info("cat","The cat says 'meow'", UVM_INFO);
12     phase.drop_objection(this);
13   endtask
14 endclass : cat
```

The UVM passes us the phase object

SUNY – New Paltz
Elect. & Comp. Eng.

UVM Objections
"Hey, I'm working here!"

```
class cat extends uvm_test;
  `uvm_component_utils(cat)

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

  task run_phase(uvm_phase phase);
    phase.raise_objection(this);
    super.run();
    `uvm_info("cat","The cat says 'meow'", UVM_INFO);
    phase.drop_objection(this);
  endtask
endclass : cat
```

All threads have veto power over stopping the simulation
If any thread has a raised objection, the simulation continues

SUNY – New Paltz
Elect. & Comp. Eng.

Creating a Test

```
class cat extends uvm_test;
  `uvm_component_utils(cat)

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

  task run_phase(uvm_phase phase);
    phase.raise_objection(this);
    super.run();
    `uvm_info("cat","The cat says 'meow'", UVM_INFO);
    phase.drop_objection(this);
  endtask
endclass : cat
```

The raise_objection() method tells the UVM that we are not finished

SUNY – New Paltz
Elect. & Comp. Eng.

Creating a Test

```
1  class cat extends uvm_test;
2    `uvm_component_utils(cat)
3
4    function new(string name, uvm_component parent);
5      super.new(name, parent);
6    endfunction : new
7
8    task run_phase(uvm_phase phase
9      phase.raise_objection(this);
10     super.run();
11     `uvm_info("cat","The cat says 'meow'", UVM_INFO);
12     phase.drop_objection(this);
13   endtask
14 endclass : cat
```

The drop_objection() method tells the UVM that we have finished our test.

SUNY – New Paltz
Elect. & Comp. Eng.

Creating a Test

```
1  class cat extends uvm_test;
2    `uvm_component_utils(cat)
3
4    function new(string name,
5      super.new(name, parent);
6    endfunction : new
7
8    task run_phase(uvm_phase phase);
9      phase.raise_objection(this);
10     super.run();
11     `uvm_info("cat","The cat says 'meow'", UVM_INFO);
12     phase.drop_objection(this);
13   endtask
14 endclass : cat
```

The "this" keyword refers to the running instance of the cat object.

SUNY – New Paltz
Elect. & Comp. Eng.

Creating a Test

```
class dog extends uvm_test;

  `uvm_component_utils(dog)

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

  task run_phase(uvm_phase phase);
     phase.raise_objection(this);
     super.run();
     `uvm_info ("dog","En espanol, el perro dice 'guau, guau'", UVM_INFO);
     phase.drop_objection(this);
  endtask

endclass : dog
```

SUNY – New Paltz
Elect. & Comp. Eng.

Running the Test: Top Level

`run_test()` creates a test object and executes its **`run_phase()`** task

```
import uvm_pkg::*;
import cat_dog_pkg::*;

module top;

  initial run_test();

endmodule
```

```
if [file exists work] {vdel -all}
vlib work
vlog -f compile.f
vsim  top +UVM_TESTNAME=cat
set  NoQuitOnFinish 1
onbreak {resume}
run -all
vsim  top +UVM_TESTNAME=dog
run -all
```

SUNY – New Paltz
Elect. & Comp. Eng.

Driving Simulation with the UVM

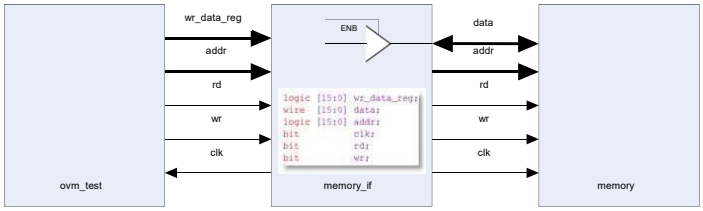
SUNY – New Paltz
Elect. & Comp. Eng.

Interfaces: Portal to the Class World

Classes don't have port lists.
How can we communicate with them?

SUNY – New Paltz
Elect. & Comp. Eng.

How do we connect the interface to the test?



SUNY – New Paltz
Elect. & Comp. Eng.

We use a package variable

```
1  package memory_pkg;
2     import uvm_pkg::*;
3
4     virtual interface memory_if global_mif;
5
6  `include "uvm_macros.svh"
7  `include "quiet_test.svh"
8  `include "verbose_test.svh"
9  endpackage: memory_pkg
10
```

SUNY – New Paltz
Elect. & Comp. Eng.

Pass the interface to package

```
import uvm_pkg::*;
import memory_pkg::*;

module top;

   memory_if mi();
   memory dut (mi.mem_mp);

   initial begin
      string test_name;

      memory_pkg::global_mif = mi;

      run_test();
   end

endmodule: top
```

SUNY – New Paltz
Elect. & Comp. Eng.

Creating a Test

```
class quiet_test extends uvm_test;

   `uvm_component_utils(quiet_test)

   virtual interface memory_if mif;

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

   endfunction : new

   virtual function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      mif = memory_pkg::global_mif;
   endfunction : build_phase

   virtual task run_phase(uvm_phase phase);
      int nloops=5;
      logic [3:0] tiny_addr;

      phase.raise_objection(this);
      endtask // run

endclass : quiet_test
```

Extend **`uvm_test`** and put this object in the factory.

SUNY – New Paltz
Elect. & Comp. Eng.

Creating a Test

```
1  class quiet_test extends uvm_test;
2
3     `uvm_component_utils(quiet_test)
4
5     virtual interface memory_if mif;
6
7     function new(string name, uvm_co
8        super.new(name,
9
10    endfunction : new
11
12    virtual function void build_phase(uvm_phase phase);
13       super.build_phase(phase);
14       mif = memory_pkg::global_mif;
15    endfunction : build_phase
16
17    virtual task run_phase(uvm_phase phase);
18       int nloops=5;
19       logic [3:0] tiny_addr;
20
21       phase.raise_objection(this);
40    endtask // run
41
42 endclass : quiet_test
43
```

Always create a constructor

SUNY – New Paltz
Elect. & Comp. Eng.

Creating a Test

```
1  class quiet_test extends uvm_test;
2
3     `uvm_component_utils(quiet_test)
4
5     virtual interface memory_if mif;
6
7     function new(string name, uvm_component parent);
8        super.new(name, parent);
9
10    endfunction : new
11
12    virtual function void build_phase(uvm_phase phase
13       super.build_phase(phase);
14       mif = memory_pkg::global_mif;
15    endfunction : build_phase
16
17    virtual task run_phase(uvm_phase phase);
18       int nloops=5;
19       logic [3:0] tiny_addr;
20
21       phase.raise_objection(this);
40    endtask // run
41
42 endclass : quiet_test
43
```

The UVM will call the **`build_phase()`** task before the **`run_phase()`** task

(more on `build_phase()` later)

SUNY – New Paltz
Elect. & Comp. Eng.

Creating a Test

```
1   class quiet_test extends uvm_test;
2
3      `uvm_component_utils(quiet_test)
4
5      virtual interface memory_if mif;
6
7      function new(string name, uvm_component parent);
8         super.new(name, parent);
9
10     endfunction : new
11
12     virtual function void build_phase(uvm_phase phase);
13        super.build_phase(phase);
14        mif = memory_pkg::global_mif;
15     endfunction : build_phase
16
17     virtual task run_phase(uvm_phase phase);
18        int nloops=5;
19        logic [3:0] tiny_addr;
20
21        phase.raise_objection(this);
40     endtask // run
41
42  endclass : quiet_test
43
```

The **run_phase()** task stimulates the DUT

SUNY – New Paltz
Elect. & Comp. Eng.

Getting the interface in build_phase()

memory_pkg.sv

```
1   package memory_pkg;
2      import uvm_pkg::*;
3
4      virtual interface memory_if global_mif;
5
6   `include "uvm_macros.svh"
```

top.sv

```
9      initial begin
10        memory_pkg::global_mif = mi;
11        run_test();
12     end
```

quiet_test.svh

```
12     virtual function void build_phase(uvm_phase phase);
13        super.build_phase(phase);
14        mif = memory_pkg::global_mif;
15     endfunction : build_phase
```

Take the interface out of the global variable in the package.

SUNY – New Paltz
Elect. & Comp. Eng.

Stimulating the DUT in the run_phase()task

```
17      virtual task run_phase(uvm_phase phase);
18         int nloops=5;
19         logic [3:0] tiny_addr;
20
21         phase.raise_objection(this);
22         mif.wr = 1'b1;
23         for (int i=0; i< 'h10; i++) begin
24            @(negedge mif.clk);
25            mif.wr_data_reg = i;
26            mif.addr = i;
27         end
28
29         uvm_report_info("MEMORY TEST", $psprintf("Running %0d loops",nloops));
30
31         repeat (nloops) begin
32            @(negedge mif.clk);
33            mif.wr = $random;
34            mif.rd = ~mif.wr;
35            tiny_addr = $random;
36            mif.addr = {12'd0,tiny_addr};
37            if (mif.wr)  mif.wr_data_reg = $random;
38         end
39         @(negedge mif.clk);
40         phase.drop_objection(this);
41      endtask // run
```


Stimulating the DUT in the run_phase()task

```
17      virtual task run_phase(uvm_phase phase);
18         int nloops=5;
19         logic [3:0] tiny_addr;
20
21         phase.raise_objection(this);
22         mif.wr = 1'b1;
23         for (int i=0; i< 'h10; i++) begin
24            @(negedge mif.clk);
25            mif.wr_data_reg = i;
26            mif.addr = i;
27         end
28
29         uvm_report_info("MEMORY TEST", $psprintf("Running %0d loops",nloops));
30
31         repeat (nloops) begin
32            @(negedge mif.clk);
33            mif.wr = $random;
34            mif.rd = ~mif.wr;
35            tiny_addr = $random;
36            mif.addr = {12'd0,tiny_addr};
37            if (mif.wr)  mif.wr_data_reg = $random;
38         end
39         @(negedge mif.clk);
40         phase.drop_objection(this);
41      endtask // run
```

Use the interface to access the signals

Creating a New Test

```
class verbose_test extends quiet_test;

   `uvm_component_utils(verbose_test)

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

   virtual task run_phase(uvm_phase phase);
      int nloops=5;
      logic [3:0] tiny_addr;

      phase.raise_objection(this);
      mif.wr = 1'b1;
      for (int i=0; i< 'h10; i++) begin
         @(negedge mif.clk);
         mif.wr_data_reg = i;
         mif.addr = i;
         `uvm_info("TESTER", $psprintf("addr: %2h  data: %2h  rd: %1b   wr:%1b",
                                       mif.addr, mif.data, mif.rd, mif.wr),UVM_INFO);
      end
```

Extend **quiet_test** class

Notify the UVM

Added `uvm_info to make it verbose

SUNY – New Paltz
Elect. & Comp. Eng.

Creating a New Test

```
class verbose_test extends quiet_test;

   `uvm_component_utils(verbose_test)

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

   virtual task run_phase(uvm_phase phase);
      int nloops=5;
      logic [3:0] tiny_addr;

      phase.raise_objection(this);
      mif.wr = 1'b1;
      for (int i=0; i< 'h10; i++) begin
         @(negedge mif.clk);
         mif.wr_data_reg = i;
         mif.addr = i;
         `uvm_info("TESTER", $psprintf("addr: %2h  data: %2h  rd: %1b   wr:%1b",
                                       mif.addr, mif.data, mif.rd, mif.wr),UVM_INFO);
      end
```

Must have own constructor

Calls **quiet_test** constructor

SUNY – New Paltz
Elect. & Comp. Eng.

Creating a New Test

```
class verbose_test extends quiet_test;

   `uvm_component_utils(verbose_test)

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

   virtual task run_phase(uvm_phase phase);
      int nloops=5;
      logic [3:0] tiny_addr;

      phase.raise_objection(this);
      mif.wr = 1'b1;
      for (int i=0; i< 'h10; i++) begin
         @(negedge mif.clk);
         mif.wr_data_reg = i;
         mif.addr = i;
         `uvm_info("TESTER", $psprintf("addr: %2h  data: %2h  rd: %1b  wr:%1b",
                                       mif.addr, mif.data, mif.rd, mif.wr),UVM_INFO);
      end
```

No **build_phase()** task!
Leverages **build_phase()** function from **quiet_test**

SUNY – New Paltz
Elect. & Comp. Eng.

Running the quiet_test

```
VSIM 2> vsim +UVM_TESTNAME=quiet_test top
# vsim +UVM_TESTNAME=quiet_test top
#
VSIM 3> run -all
# ----------------------------------------------------------------
# UVM-1.1b
# (C) 2007-2012 Mentor Graphics Corporation
# (C) 2007-2012 Cadence Design Systems, Inc.
#
# UVM_INFO @ 0: reporter [RNTST] Running test quiet_test...
# UVM_INFO @ 320: uvm_test_top [MEMORY TEST] Running 5 loops
# UVM_INFO verilog_src/uvm-1.1b/src/base/uvm_objection.svh(1120) @ 440: reporter [TEST_DONE] 'run' phase i
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    3
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [MEMORY TEST]     1
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# ** Note: $finish    : /tools/mentor/questa/10.1c_1/questasim/linux/../verilog_src/uvm-1.1b/src/base/uvm_
#    Time: 440 ns  Iteration: 55  Instance: /top
```

SUNY – New Paltz
Elect. & Comp. Eng.

Running the verbose_test

```
QuestaSim> do run.do
# QuestaSim vlog 10.1c_1 Compiler 2012.09 Sep  2 2012
#
# Top level modules:
#       top
# vsim +UVM_TESTNAME=verbose_test top
# ** Note: (vsim-3812) Design is being optimized...
# 1
# ----------------------------------------------------------------
# UVM-1.1b
# (C) 2007-2012 Mentor Graphics Corporation
# (C) 2007-2012 Cadence Design Systems, Inc.
#
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(215) @ 0: reporter [Questa UVM] QUESTA
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(217) @ 0: reporter [Questa UVM]  quest
# UVM_INFO @ 0: reporter [RNTST] Running test verbose_test...
# UVM_INFO ./bench/verbose_test.svh(20) @ 20: uvm_test_top [TESTER] addr: 00  data: xx  rd: 0   wr:1
# UVM_INFO ./bench/verbose_test.svh(20) @ 40: uvm_test_top [TESTER] addr: 01  data: 00  rd: 0   wr:1
# UVM_INFO ./bench/verbose_test.svh(20) @ 60: uvm_test_top [TESTER] addr: 02  data: 01  rd: 0   wr:1
# UVM_INFO ./bench/verbose_test.svh(20) @ 80: uvm_test_top [TESTER] addr: 03  data: 02  rd: 0   wr:1
# UVM_INFO ./bench/verbose_test.svh(20) @ 100: uvm_test_top [TESTER] addr: 04  data: 03  rd: 0   wr:1
# UVM_INFO ./bench/verbose_test.svh(20) @ 120: uvm_test_top [TESTER] addr: 05  data: 04  rd: 0   wr:1
```

SUNY – New Paltz
Elect. & Comp. Eng.

Steps to Create a UVM Test

- **Create a test object by extending uvm_test**
- **Create a package variable that will hold the interface**
- **In the top level, instantiate the interface and put a copy of the interface into the package variable**
- **In the test's build_phase()task, get the interface out of the package variable**
- **Stimulate the DUT in the run_phase()task**

SUNY – New Paltz
Elect. & Comp. Eng.

UVM Components

- https://www.youtube.com/watch?v=H7hIurC1BPg
- UVM Components are foundation of testbench structure
  - Step 1: Extend the *uvm_component* class or child class to define your component
  - Step 2: Use the *'uvm_component_utils()* macro to register this class with factory
  - Step 3: Provide at least the minimum *uvm_component* constructor
  - Step 4: Override the *UVM* phase methods as necessary

SUNY – New Paltz
Division of Engineering Programs

- Much simpler
- No bfm from top level
- No run task
- Key here is phases
  - Built phase: create three lower-level components
    - UVM calls their build phases
  - Connect phase
  - Run phase
- Has build phase: pulls down bfm

```
class random_test extends uvm_test;
  `uvm_component_utils(random_test); /* register  factory //

 random_tester tester_h;
  coverage      coverage_h;
  scoreboard    scoreboard_h;

  function void build_phase(uvm_phase phase);
    tester_h      = new("tester_h", this);
    coverage_h    = new("coverage_h",    this);
    scoreboard_h  = new("scoreboard_h",    this);
  endfunction : build_phase

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

endclass
```

SUNY – New Paltz
Elect. & Comp. Eng.

```
class random_tester;
`uvm_component_utils (random_tester)
  virtual tinyalu_bfm bfm;
  function new (string name, uvm_component parent);
    super.new(name, parent);
  endfunction : new

  virtual 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
```

- build phase: instead of constructor, to get bfm in config
- Run phase

```
  virtual      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

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

```
  task run_phase(uvm_phase phase);
    byte        unsigned       iA;
    byte        unsigned       iB;
    operation_t                op_set;
    shortint    result;

    phase.raise_objection(this);
    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
    #500;
    phase.drop_objection(this);
  endtask : run_phase
endclass : random_tester
```

```
class add_tester extends random_tester;

`uvm_component_utils(add_tester)

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

  function operation_t get_op();
    bit [2:0] op_choice;
    return add_op;
  endfunction : get_op

endclass : add_tester
```

SUNY – New Paltz
Elect. & Comp. Eng.

Coverage

- Same set up
  - basic constructor
  - Build phase
  - Run phase

```
class coverage extends uvm_component;
  `uvm_component_utils(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
```

SUNY – New Paltz
Elect. & Comp. Eng.

```
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};     }
   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
```

```
  function new (string name, uvm_component parent);
     super.new(name, parent);
     op_cov = new();
     zeros_or_ones_on_ops = new();
   endfunction : new
 function void build_phase(uvm_phase phase);
  if(!uvm_config_db #(virtual tinyalu_bfm)::get(null, "*","bfm", bfm))
       $fatal("Failed to get BFM");
endfunction : build_phase
    task run_phase(uvm_phase phase);
     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 : run_phase
endclass : coverage
```

UVM will get the build phase and run phase going on it own.
Phase objective not needed since stimulus are not being generated.

SUNY – New Paltz
Elect. & Comp. Eng.

```
class scoreboard extends
uvm_component;
  `uvm_component_utils(scoreboard);
  virtual tinyalu_bfm bfm;
  function new (string name, uvm_component
parent);
     super.new(name, parent);
  endfunction : new
function void build_phase(uvm_phase phase);
 if(!uvm_config_db #(virtual
tinyalu_bfm)::get(null, "*","bfm", bfm))
       $fatal("Failed to get BFM");
endfunction : build_phase
```

```
task run_phase(uvm_phase phase);
    shortint predicted_result;
    forever begin : self_checker
      @(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 : run_phase
endclass : scoreboard
```

SUNY – New Paltz
Elect. & Comp. Eng.

```
class add_test extends random_test;
  `uvm_component_utils(add_test);

  add_tester tester_h;

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

add_test leverages everything from random_test i.e. build phase, run phase. It does need its own constructor. All it needs tester_h variable to be set up.

SUNY – New Paltz
Elect. & Comp. Eng.

Other verification methodologies

SUNY – New Paltz
Division of Engineering Programs

UVM versus OVM

- The Universal Verification Methodology (UVM) is a standardized methodology for verifying integrated circuit designs.
- UVM is derived mainly from the **OVM** (Open Verification Methodology)
- **OVM** is based on the eRM (e Reuse Methodology) for the e Verification Language developed by Verisity Design in 2001.

SUNY – New Paltz
Elect. & Comp. Eng.

Configuring components in OVM and UVM

OVM

```
class my_env extends ovm_env;
...
  function void build();
    ahb_cfg = ahb_config::type_id::create("ahb_cfg");
    ahb_cfg.width = 16;
    // set additional fields
    set_config_object("*","ahb_cfg",ahb_cfg);
  endfunction
...
endclass

class my_ahb_agent extends ovm_component;
...
  function void build();
    ovm_object cfg;
    ahb_config my_cfg;
    assert(get_config_object("ahb_cfg",cfg,0);
    if (!$cast(my_cfg, cfg))
      ovm_report_error(...);
    ...
  endfunction
...
endclass
```

UVM

```
class my_env extends uvm_env;

...

 function void build();
   ahb_cfg = ahb_config::type_id::create("ahb_cfg");
   ahb_cfg.width = 16;
   // set additional fields
   uvm_config_db#(ahb_config)::set(
this,"ahb_agent","ahb_cfg",ahb_cfg);
 endfunction

...

endclass

 class my_ahb_agent extends uvm_component;

...

 function void build();
   ahb_config my_cfg;
   if (!uvm_config_db::ahb_config::get(
        this,"","ahb_cfg",my_cfg);
     `uvm_error(...)
   ...
 endfunction
...
endclass
```

SUNY – New Paltz
Elect. & Comp. Eng.

Comparing end of test in OVM and UVM

OVM	UVM

task run();
  seq.start(m_virtual_sequencer);
  global_stop_request();
endtask
	task run_phase(uvm_phase phase);
	 phase.raise_objection(this);
	 seq.start(m_virtual_sequencer);
	 phase.lower_objection(this);
	
	endtask

task run();
  ovm_test_done.raise_objection();
  seq.start(m_virtual_sequencer);
  ovm_test_done.drop_objection();
endtask
	task run_phase(uvm_phase phase);
	 phase.raise_objection(this , "started sequence");
	 seq.start(m_virtual_sequencer);
	 phase.drop_objection(this , "finished sequence");
	
	endtask

SUNY – New Paltz
Elect. & Comp. Eng.