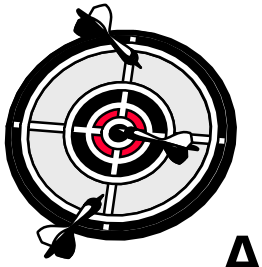
Agenda

DAY 3

8	Object Oriented Programming (OOP) – Inheritance
9	Inter-Thread Communications 
10	Functional Coverage 
11	SystemVerilog UVM Preview
CS	Customer Support

Unit Objectives

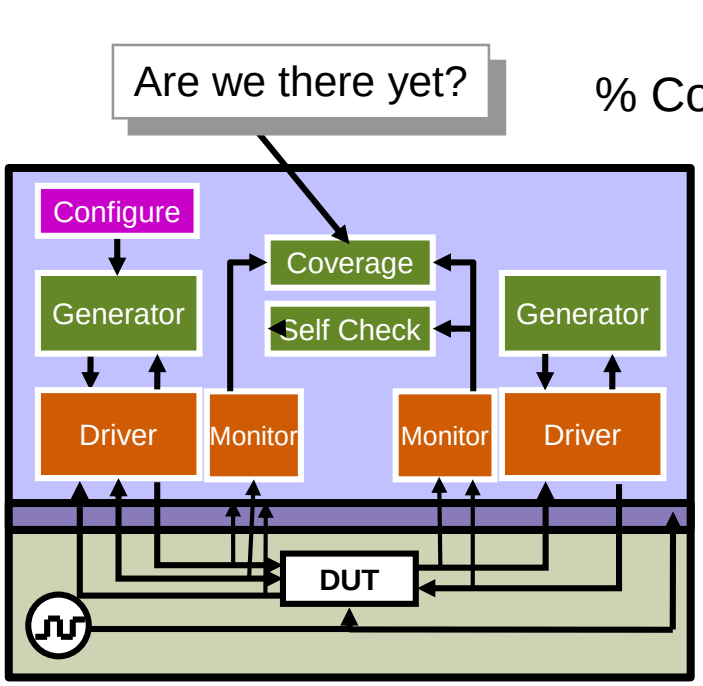


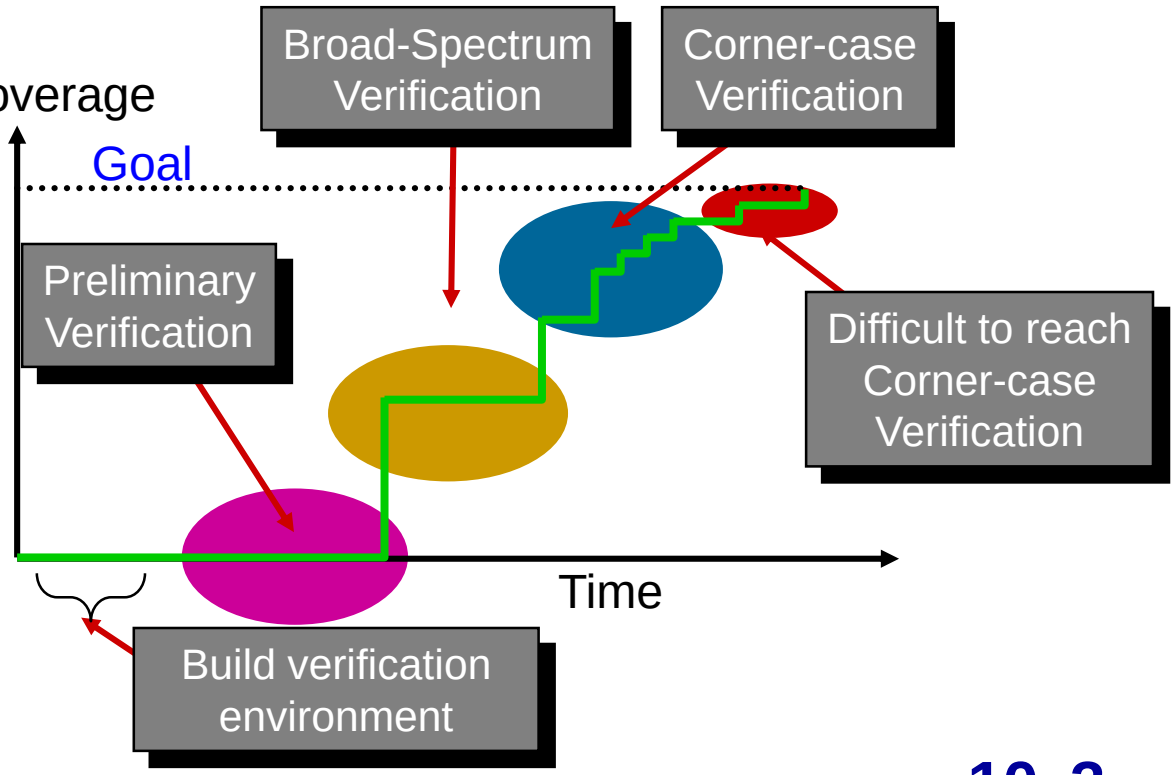
After completing this unit, you should be able to:

- **Define functional coverage structures**
  - Specify the coverage sample mechanisms
  - Define signals and variables to be sampled
  - Specify expected values that indicate functionality
  - Use parameters to make coverage instances unique
  - Use coverage attributes to customize individual coverage structures
- **Measure coverage dynamically**
- **Generate coverage reports after running VCS**

Phases of Functional Verification

- **Implement functional coverage to answer**
  - Has verification goal been reached?
  - When to switch to corner-case Verification?
  - When to write directed tests for difficult to reach corner-case Verification?

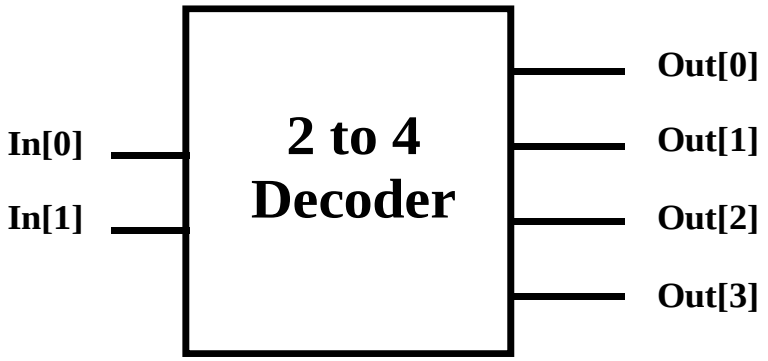




Combinational Logic Example

Goal: Check all combinations of input and output patterns

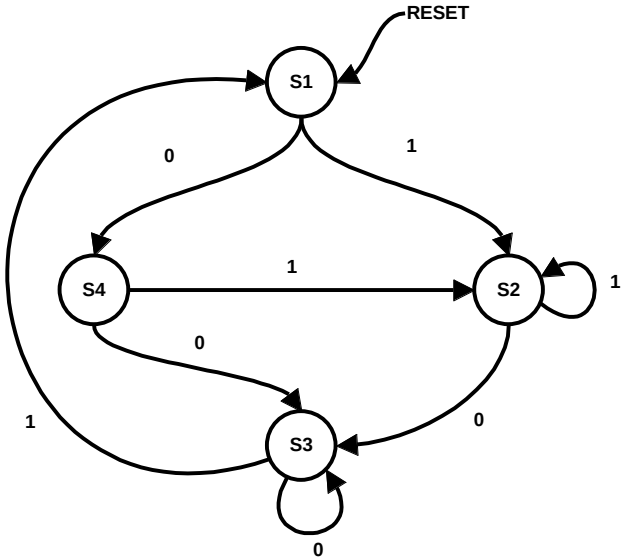
In[1]	In[0]	Out[3]	Out[2]	Out[1]	Out[0]
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



- Create state coverage bins to track input & output bit patterns
- Define sample timing for coverage bins
- Track state coverage bin results

State Transition Example

Goal: Check temporal state transitions



- Create state coverage bins to monitor states
- Create transition coverage bins to monitor state transitions

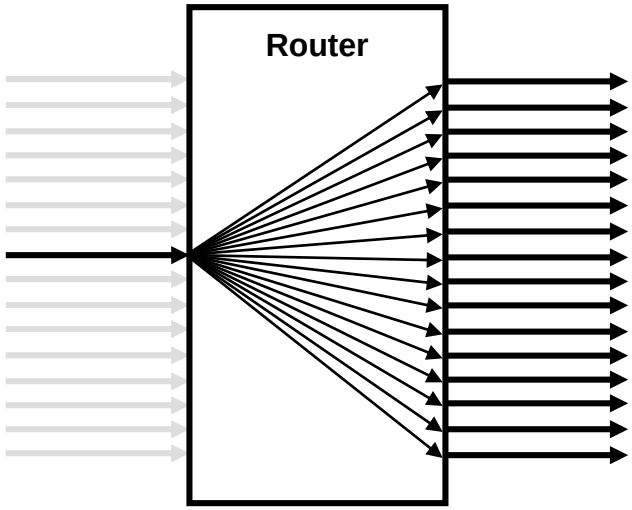
  S1 ⇒ S2 ⇒ S3 ⇒ S1 ⇒ S4 ⇒ S2 ⇒ S2 ⇒ S3 ⇒ S1

- Define sample timing for coverage bins
- Track transition coverage bin results

Cross Correlation Example

Goal: Check all input ports have driven all output ports

Input port	Output port
0	0
0	1
0	2
…	…
15	14
15	15



- Create cross coverage bins to correlate activities of input ports with respect to output ports
- Define sample timing for coverage bins
- Track cross coverage bin results

Functional Coverage in SystemVerilog

- **Create a covergroup**
  - Standalone - Recommended for reusability
  - Inside classes
- **Covergroup encapsulates:**
  - Coverage bins definitions
    - State
    - Transition
    - Cross correlation
  - Coverage bins sample timing definition
  - Coverage attributes e.g. coverage goal
- **Instantiate coverage object(s) from covergroup using new()**
  - Scope of variable visibility is same as subroutines
- **Query coverage object to determine progress of verification**
- **Only 2-state values covered**
  - Sampled values with X or Z excluded from coverage

Functional Coverage Example

Create coverage group

```
program automatic test;
 covergroup fcov() @(port_event);
  coverpoint sa;
  coverpoint da;
 endgroup: fcov
 bit[3:0]    sa, da;
 event       port_event;
 real        coverage = 0.0;
 fcov port_fc = new();

 initial while (coverage < 99.5) begin
    ...
    sa = pkt_ref.sa;
    da = pkt_ref.da;
    ->port_event;
//  port_fc.sample();          // alternative form of updating of bins
    coverage = $get_coverage();          // overall coverage
//  coverage = port_fc.get_inst_coverage(); // instance coverage
 end
endprogram: test
```

Declare & construct coverage object (required)

Query coverage result

State Bin Creation (Automatic)

- **SystemVerilog can automatically create state bins**

```
bit [3:0] sa,da;
covergroup cov1 @(posedge rtr_io.clock);
  coverpoint sa;  //16 bins
  coverpoint da;  //16 bins
  compare: coverpoint (sa > da); // 2 bins
  concat: coverpoint {sa, da} {  // 256 bins
    option.auto_bin_max = 256;   // else 64 bins default
  }
endgroup: cov1
```

- **Bin name is "auto[value_range]"**
  - The value_range are the value range which triggered that bin
- **Maximum number of bins is set by auto_bin_max**
  - Controlled using the auto_bin_max attribute
  - Bins are allocated with equal number of states

Measuring Coverage

Without auto binning:

- **Coverage is:**

$$\frac{\text{\# of bins covered (have at\_least hits)}}{\text{\# of total bins}}$$

With auto binning:

auto_bin_max **limit the number of bins used in the coverage calculation**

- **Coverage is:**

$$\frac{\text{\# of bins covered (have at\_least hits)}}{\text{min(possible values for data type | auto\_bin\_max)}}$$

Automatic State Bin Creation Example

```
bit[3:0] sa, da;
covergroup cov_addr @(rtr_io.cb);
  coverpoint sa;
  coverpoint da;
  option.auto_bin_max = 2; // maximum of 2 auto-created bins
endgroup: cov_addr
…
cov_addr cov1 = new();

sa = 1; da = 8;
@(rtr_io.cb);
$display("%0d covered", $get_coverage());

sa = 9; da = 9;
@(rtr_io.cb);
$display("%0d covered", $get_coverage());

sa = 3; da = 5;
@(rtr_io.cb);
$display("%0d covered", $get_coverage());
```

(50% covered)

Var	Bin	#Hit
sa	auto[0:7]	1
da	auto[8:15]	1

(75% covered)

Var	Bin	#Hit
sa	auto[0:7]	1
	auto[8:15]	1
da	auto[8:15]	2

(100% covered)

Var	Bin	#Hit
sa	auto[0:7]	2
	auto[8:15]	1
da	auto[0:7]	1
	auto[8:15]	2

State and Transition Bin Creation (User)

- Define state bins using ranges of values
- Define transition bins using state transitions

```
covergroup MyCov() @(cov_event);
  coverpoint port_number {
    bins s0 = { 0 };                        // one bin for port_number == 0
    bins lo = { [0:7] };                    // creates one state bin
    bins hi[] = { [8:15] };                 // creates 8 state bins
                                            // hi_8 through hi_f
    ignore_bins ignore = { 16, 20 };        // ignore if hit
    illegal_bins bad = default;             // terminates simulation if hit
                                            // default refers to undefined values
    bins t0 = (0 => 8, 9 => 0);             // creates one transition bin
    bins t1[] = (8, [0:7] => [8:15]);       // creates 72 transition bins
    bins other_trans = default sequence;    // all other single state transitions
//  illegal_bins bad_trans = default sequence; // terminates simulation
  }
endgroup: MyCov
```

Cross Coverage Bin Creation

- VCS can automatically create cross coverage bins

```
covergroup cov1() @(rtr_io.cb);
  coverpoint sa;
  coverpoint da;
  cross sa, da;
  cross p_cnt, mde; //implicit bins for p_cnt,mde
endgroup: cov1
```

- Users can also define cross bins to cover*

```
src: coverpoint sa; //can label coverpoints
dest: coverpoint da { bins d1 = { 0 }; bins d2 = default; }
cx: cross src, dest {
  bins lo = binsof(src) intersect {[0:4]};
  bins mid = binsof(src) && binsof(dest.d1); //expression
  bins hi = ! binsof(src) intersect {[0:7]};
}
```

Wildcard Bins

- **Can use wildcards to express bin ranges**
  - Use wildcard keyword in bin definition
  - Similar to ==? operator
  - ?, x or z may be used as wildcard
    - ? preferred to avoid confusion when covering 4-state variables

```
covergroup myCov() @(cov_event);
  coverpoint port_number {
    wildcard bins b1 = {4'b01??};     // {[4'b0100:4'b0111]}
    wildcard bins b2[] = {4'b011? => 4'b?000}; // 4 transitions
    wildcard bins w1[2] = {4'b11??}; // w1[4'b1100,4'b1101] and
                                     // w1[4'b1110,4'b1111]
  }
endgroup: myCov
```

Bin Coverage - `with`

- **Use `with` to limit values that are included in a bin**
  - The `with` clause specifies that only those values that satisfy the given expression (for which the expression evaluates to true) are included in the bin
  - Expression has access to sampled value via `item`

```
covergroup cg0;
  gfc_cp: coverpoint gfc {
    bins lo[] =  {[0:8'h0f]}      with (item % 2  == 0);
    bins med[] = {[8'h10:8'hc0]} with (item % 3  == 0);
  }
endgroup
```

Expression(s) must be true to update bin(s)

Creates 8 bins for lo
Creates 60 bins for med

Specifying Sample Event Timing

```
covergroup definition_name [(argument_list)] [sample_event];
  [label:] coverpoint coverage_point  {  ...  }
}
```

- ***sample_event*** **defaults to `with function sample()`**
  - Can be overridden with one of:
    - user defined arguments to sample() method
    - @([specified_edge] signals | variables)
- **Bins are updated asynchronously as the sample_event occurs**
  - To update bins at end of simulation time slot
    - set **`type_option.strobe* = 1`** in covergroup

Parameterized Sample Method

- **Can override `sample()` with a triggering function `sample()` that accepts arguments**

```
covergroup pkt_cg with function sample(Packet pkt);
```

  - Allows sampling of coverage data from contexts other than the scope enclosing the covergroup declaration
  - Can be called from
    - automatic task or function.
    - sequence or property of a concurrent assertion
    - procedural block
  - Formal arguments of the sample method should
    - only designate a coverpoint or conditional guard expression
    - be different from list of covergroup arguments since they belong to the same lexical scope as the formal arguments to the covergroup
    - not designate an output direction

Parameterized Sample Method: Example

```
covergroup pkt_cg with function sample(Packet pkt);
  src: coverpoint pkt.sa;
  dst: coverpoint pkt.da;
  cross src, dst;
endgroup: pkt_cg
pkt_cg p_cg = new();

function bit Packet::check();
  if (actual_pkt.compare(ref_pkt, message)) begin
    p_cg.sample(actual_pkt);
  ...
endfunction: check
```

Determining Coverage Progress

- `$get_coverage()` returns testbench coverage percentage as a `real` value

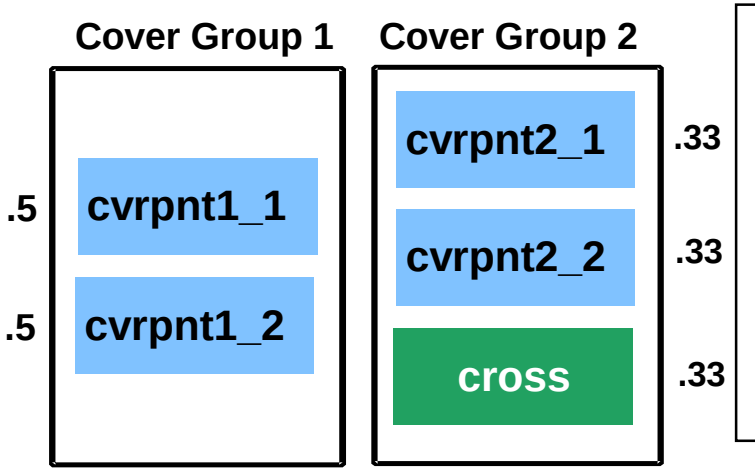
```
covergroup cov1(ref int x, int y) @(rtr_io.cb);
    mode1: coverpoint x {
    bins S_1[] = { [34:78], [1:27] }; }
endgroup: cov1

program automatic Main;
  int a;
  real cov_percent;
  cov1 c_obj1 = new(a, b);
  initial forever begin
    cov_percent = $get_coverage();
    $display("%0d covered\n", cov_percent);
    ...
  end
endprogram: Main
```

Coverage Measurement Example

- Each `covergroup` contributes equally to total coverage
- Each `coverpoint`/`cross` block contributes equally to the covergroup's coverage
- Attributes may affect contributions

Total Coverage:
(Group 1 % x 0.5) + (Group 2 % x 0.5) = Coverage %

Cover Group 2 Coverage:
(cvrpnt2_1 % x .33) + (cvrpnt2_2 % x .33) + (cross % x .33) = *group coverage %*



For cvrpnt2_1 coverpoint in Group 2 assume:

option.`at_least` = 1
total bins = 10
bins having at least 1 hit = 6

the coverpoint's contribution is:
.33 x .6 x 1 = .2 (20%) to its covergroup
.2 x .5 = .1 (10%) to the total coverage

Coverage Attributes (1/2)

- **Coverage attributes are defined for entire coverage groups or for individual coverpoints**
  - Defined using **`option`** or **`type_option`** keyword
  - Attributes at the coverage group level may be overridden at the sample and cross level
  - Attributes may be different for each instance of a coverage object by passing arguments

```
covergroup cov1(int var1) @(cov_event);
    option.auto_bin_max = var1; // entire group

    coverpoint x {
        option.auto_bin_max = 4; // just for x
        bins lower = { [1:8] };
        ...
     }
endgroup: cov1
```

Coverage Attributes (2/2)

- **Coverage attributes are of two kinds**
  - Instance-specific coverage attributes
    - `option.<attribute> = <value>;`
  - Type-specific (static) coverage attributes
    - `type_option.<attribute> = <value>;`
- **Attributes available for both kinds - *weight, comment***
  - `option.<common_attr>`
    - Will work only if attribute `option.per_instance = 1`
    - Will save separate instance based coverage
  - Use `type_option.<attr>` for cumulative coverage attributes

```
covergroup cov1 @(cov_event);
  coverpoint sa { type_option.weight = 0; }
  coverpoint da { type_option.weight = 0; }
  cross sa, da;  //only cross is covered
endgroup: cov1
```

Major Coverage Attributes

- **`at_least` (1):**
  - Minimum hits for a bin to be considered covered
- **`auto_bin_max` (64):**
  - Maximum number automatically created bins
    - Each bin contains equal number of values
- **`comment`*:**
  - A comment that is saved in the database and appears in the report
- **`weight` (1)*:**
  - Multiplier for coverage bins
- **`per_instance` (0):**
  - Specifies whether to also collect coverage statistics per instance for a coverage group
  - Cumulative statistics always collected

* also available as a `type_option`

Control and Query of Covergroups

- **Methods for control and query of covergroups**
  - `sample()` only works on a covergroup instance
  - All others work on covergroup, coverpoint or cross

Method	Function
`void sample()`	Triggers sampling of the covergroup – only works on covergroup
`real get_coverage()`	Calculates type coverage number (0...100)
`real get_inst_coverage()` `(Not supported yet by VCS)`	Calculates the coverage number (0...100) – Must set `type_option.merge_instances = 1`
`void start()`	Starts collecting coverage information
`void stop()`	Stops collecting coverage information

Parameterized Coverage Group

- Variables passed by reference or value
- Variables used in guard expressions

```
covergroup MyCov(ref reg[3:0] data,
                 reg reset_l,
                 input reg[3:0] middle)
   coverpoint data { // sampled parameter
      bins s_lo (0:middle) ;
      bins s_hi[] (middle+1:15) iff (reset_l);
   }
endgroup: MyCov
program automatic Example(router_io.TB rtr_io);
   reg [3:0] mydata; reg reset_l;
   MyCov cov1;
   initial begin
   cov1 = new(mydata, reset_l, 7);
   reset_l = 0; mydata=3;
   cov1.sample();
   #1 reset_l = 1; cov1.sample(); end
endprogram: Example
```

must be ref – covered variable

pass by value

reset_l – guard variable
can be ref or pass by value

guard expression

Bin s_lo (0:7),
8 bins s_hi[8]…s_hi[15]

data not sampled when
reset_l=0

data sampled this time

Coverage Result Reporting Utilities

- **VCS writes coverage data to a binary database file**
  - The database directory is named simv.vdb
- **Generate HTML report:**

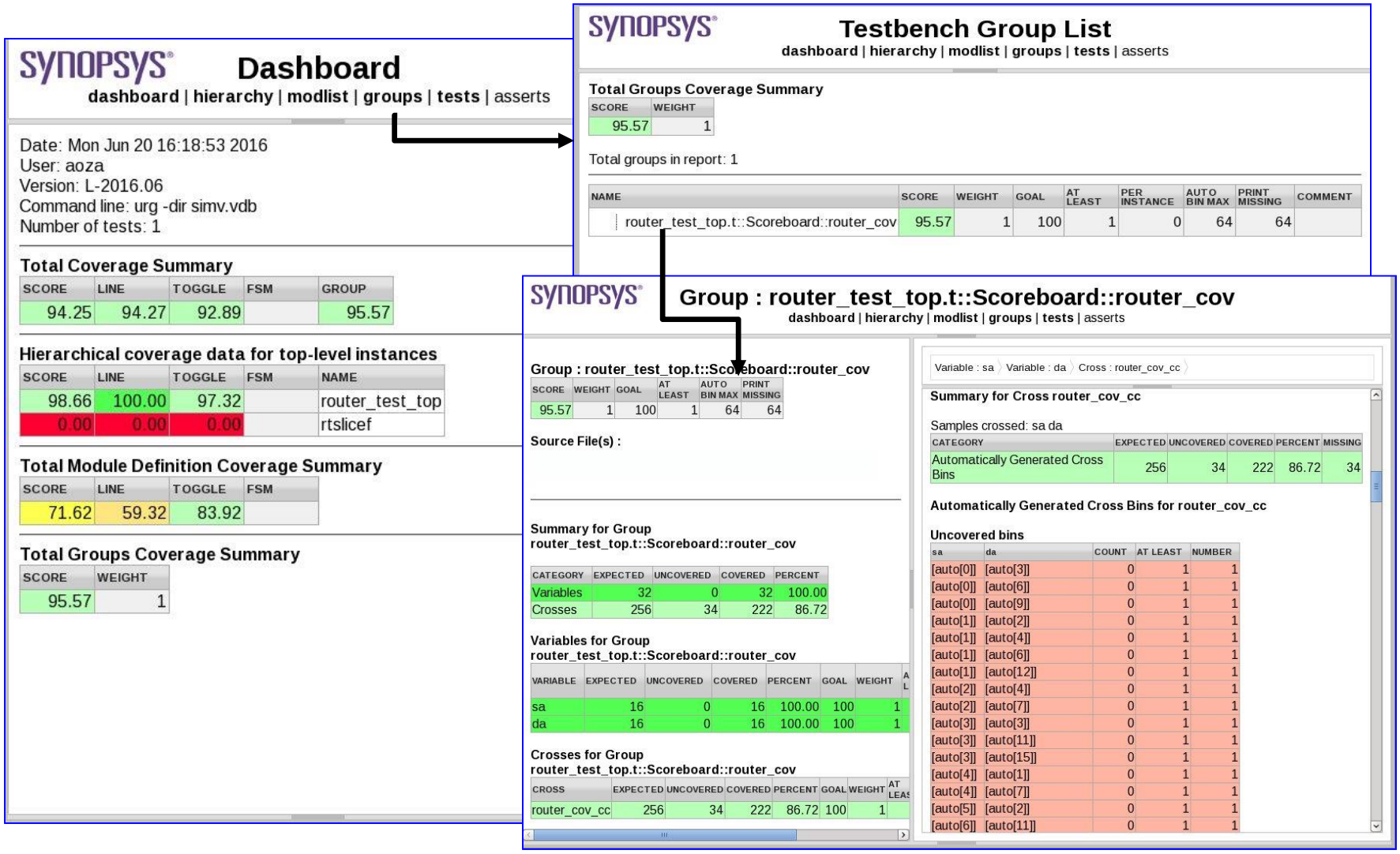
```
urg -dir <directory>
example: urg -dir simv.vdb
```

- **Generate Text Report:**

```
urg -dir <directory> -format text
example: urg -dir simv.vdb -format text
```

- **The data in all coverage database files in that directory are merged and reported**

Sample URG HTML Report



SYNOPSYS®

Dashboard

dashboard | hierarchy | modlist | groups | tests | asserts

Date: Mon Jun 20 16:18:53 2016
User: aoza
Version: L-2016.06
Command line: urg -dir simv.vdb
Number of tests: 1

Total Coverage Summary

SCORE	LINE	TOGGLE	FSM	GROUP
94.25	94.27	92.89		95.57

Hierarchical coverage data for top-level instances

SCORE	LINE	TOGGLE	FSM	NAME
98.66	100.00	97.32		router_test_top
0.00	0.00	0.00		rtslicef

Total Module Definition Coverage Summary

SCORE	LINE	TOGGLE	FSM
71.62	59.32	83.92	

Total Groups Coverage Summary

SCORE	WEIGHT
95.57	1

SYNOPSYS®

Testbench Group List

dashboard | hierarchy | modlist | groups | tests | asserts

Total Groups Coverage Summary

SCORE	WEIGHT
95.57	1

Total groups in report: 1

NAME	SCORE	WEIGHT	GOAL	AT LEAST	PER INSTANCE	AUTO BIN MAX	PRINT MISSING	COMMENT
router_test_top.t::Scoreboard::router_cov	95.57	1	100	1	0	64	64	

SYNOPSYS®

Group : router_test_top.t::Scoreboard::router_cov

dashboard | hierarchy | modlist | groups | tests | asserts

Group : router_test_top.t::Scoreboard::router_cov

SCORE	WEIGHT	GOAL	AT LEAST	AUTO BIN MAX	PRINT MISSING
95.57	1	100	1	64	64

Source File(s) :

Summary for Group router_test_top.t::Scoreboard::router_cov

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
Variables	32	0	32	100.00
Crosses	256	34	222	86.72

Variables for Group router_test_top.t::Scoreboard::router_cov

VARIABLE	EXPECTED	UNCOVERED	COVERED	PERCENT	GOAL	WEIGHT	A L
sa	16	0	16	100.00	100	1	
da	16	0	16	100.00	100	1	

Crosses for Group router_test_top.t::Scoreboard::router_cov

CROSS	EXPECTED	UNCOVERED	COVERED	PERCENT	GOAL	WEIGHT	AT LEAS
router_cov_cc	256	34	222	86.72	100	1	

Variable : sa 〉 Variable : da 〉 Cross : router_cov_cc 〉

Summary for Cross router_cov_cc

Samples crossed: sa da

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT	MISSING
Automatically Generated Cross Bins	256	34	222	86.72	34

Automatically Generated Cross Bins for router_cov_cc

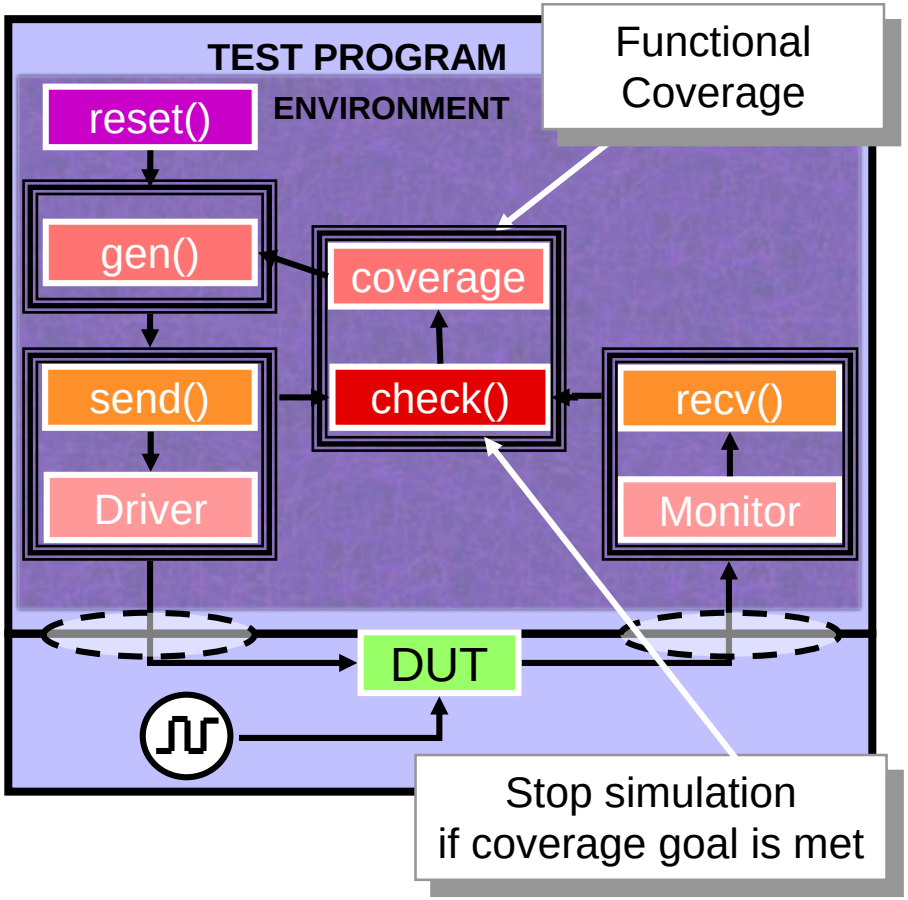
Uncovered bins

sa	da	COUNT	AT LEAST	NUMBER
[auto[0]]	[auto[3]]	0	1	1
[auto[0]]	[auto[6]]	0	1	1
[auto[0]]	[auto[9]]	0	1	1
[auto[1]]	[auto[2]]	0	1	1
[auto[1]]	[auto[4]]	0	1	1
[auto[1]]	[auto[6]]	0	1	1
[auto[1]]	[auto[12]]	0	1	1
[auto[2]]	[auto[4]]	0	1	1
[auto[2]]	[auto[7]]	0	1	1
[auto[3]]	[auto[3]]	0	1	1
[auto[3]]	[auto[11]]	0	1	1
[auto[3]]	[auto[15]]	0	1	1
[auto[4]]	[auto[1]]	0	1	1
[auto[4]]	[auto[7]]	0	1	1
[auto[5]]	[auto[2]]	0	1	1
[auto[6]]	[auto[11]]	0	1	1

Lab 6 Introduction

60 min

Implement Functional Coverage, Packages using an OOP Methodology



Part 1: Implement functional coverage

Compile & Simulate

Part2: Packages and Methodology

Unit Objectives Review

Having completed this unit, you should be able to:

- **Define functional coverage structures**
  - Specify the coverage sample mechanisms
  - Define signals and variables to be sampled
  - Specify expected values that indicate functionality
  - Use parameters to make coverage instances unique
  - Use coverage attributes to customize individual coverage structures
- **Measure coverage dynamically**
- **Generate coverage reports after running VCS**

Appendix

Advanced Coverage Topics

- Merging Coverage Results
- Covergroup Exclusion
- Test Grading

Merging Coverage Results

Merging Coverage Results

- **Providing multiple coverage directories to URG creates merged result**

```
% urg -dir test1/simv.vdb -dir test2/simv.vdb -dir ...
```

  - Does not merge the database

- **Merge Database using**

```
% urg -dbname mrgdb -dir t1.vdb -dir t2.vdb
```

  - merged database looks as if single test produced the results

- **Recommendation**
  - Create separate coverage database for each test
  - Only merge coverage databases of tests that passed
  - Re-run all tests to collect coverage if design/testbench has changed a lot

Merging Coverage Results - Parallel Merging

- **Parallel Merging**
  - Distributes the merging operations on several machines
    - Can specify a set of machines as a list
    - Can use the LSF or GRID options
  - Can provide improvement in performance for merge
    - e.g. improved the time from 3 hrs to 27 min (6.7x) using 8 CPUs

```
% urg -parallel \
      -noreport \
      -dir test1/simv.vdb \
      -dir test2/simv.vdb \
      ...
```

Use **-noreport** to merge databases, but avoid generating URG report.

- **For help**

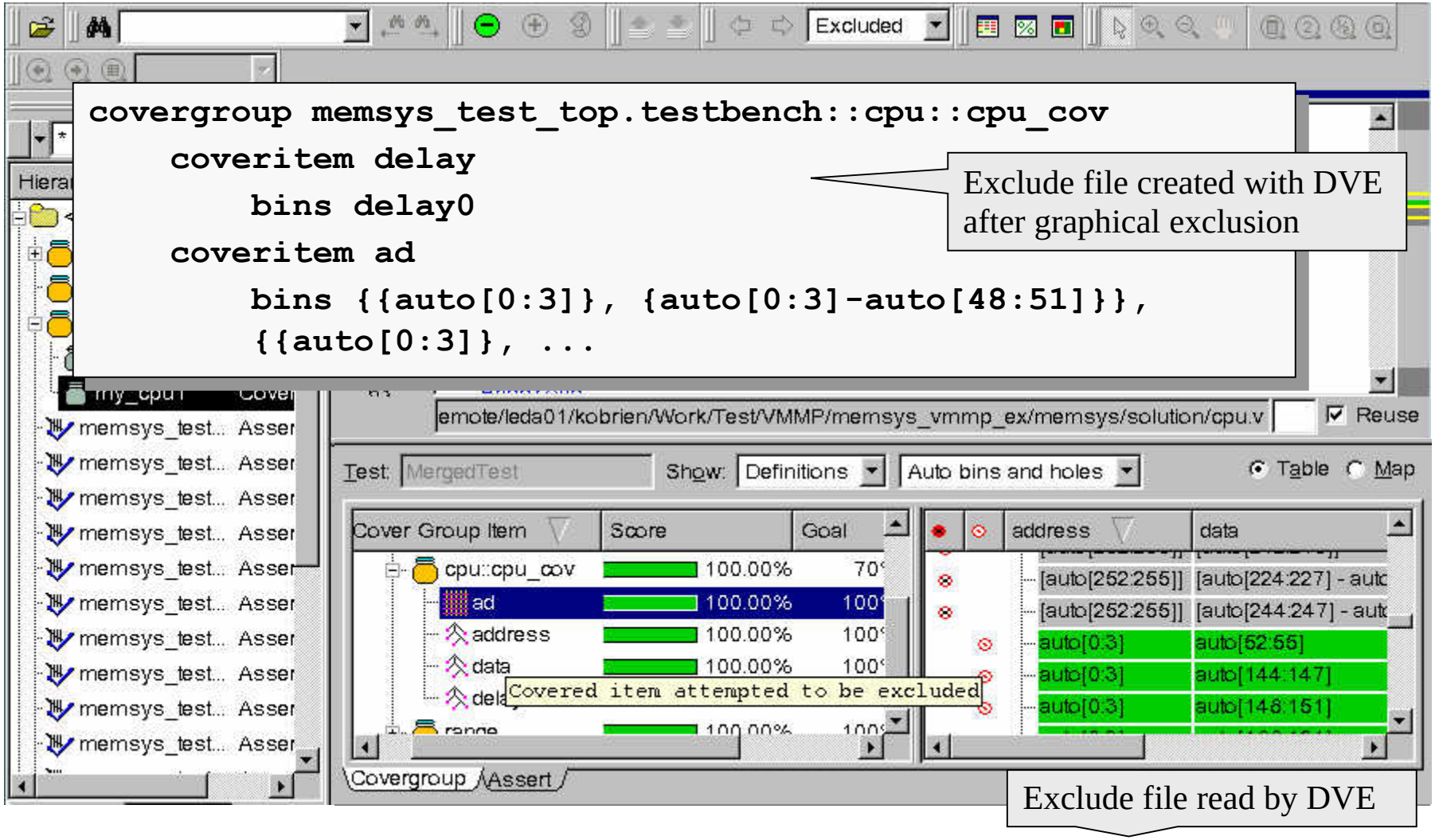
```
%> urg -parallel -help
```

Covergroup Exclusion

Covergroup Exclusion

- **Part of common methodology excluding various types of coverage (code, assert, covergroup,…) as needed**
  - Exclude file passed to URG post-simulation
    - Graphically create exclude file using DVE
    - User-created exclude file
  - Exclude don’t-care objects
  - Exclude uncoverable objects
- **Allow exclusion of covergroups, coverpoints, crosses, and bins**
  - Instance and module level groups
  - User-defined and automatic bins and bin ranges
- **URG prints summary of excluded data**

Exclusion through DVE




```
% dve -cov -covdir simv.vdb -elfile test1.el
```

Exclusion through URG

```
% urg -dir simv.vdb -elfile test.el
```

- reports excluded bins

```
//sample test.el file
covergroup test.delay_cov
   coveritem delay
    bins delay0
    bins delay_2
    bins delay_7
    bins delay_8
    bins ignore_vals
    ...
```

Summary for Variable delay

CATEGORY	EXPECTED	UNCOVERED	COVERED	PERCENT
User Defined Bins	6	0	6	100.00

User Defined Bins for delay

Uncovered bins

NAME	COUNT	AT LEAST	NUMBER	EXCLUSION
delay0	44	50	1	Excluded

Excluded/Illegal bins

NAME	COUNT	
delay_2	0	excluded
delay_7	0	excluded
delay_8	0	excluded
ignore_vals	0	excluded

Covered bins

NAME	COUNT	AT LEAST	EXCLUSION
delay_1	53	50	

Test Grading

Test Grading

- **For given set of tests with coverage results, provide a minimal set of tests that give the highest coverage score**
  - Useful for building “dead-or-alive”(/ “sanity” / “approval” /) test set
    - Run before checking-in design/TB changes, to perform basic verification
  - First set of tests to run (highest priority) as a basic regression set
- **Run URG with `-grade` option**

```
% urg -grade [ quick | greedy | score ]
```

  - “Tests” page show tests in graded order
    - Scores show incremental contribution for each test
  - Remember: test score is combination of test & seed
    - High scoring test with seed X might get low score with seed Y
  - URG doesn’t consider simulation time
    - Test with best score might be the longest
      - Refer to ‘Simulation Time’ column in URG report

Test Grading: `-grade` options

- **`greedy` (default)**
  - Tests put in a best-first order based on test usefulness
  - Shows cumulative, standalone, and incremental scores for each test
  - Grading algorithm is quadratic in the number of tests: $O(N^2)$
  - Takes long time to complete
- **`quick`**
  - Tests put in a best-first order based on test usefulness
  - Shows cumulative and incremental scores for each test
  - Grading algorithm is linear in the number of tests: O(N)
- **`score`**
  - Tests put in default order
  - Shows standalone scores only
  - Grading algorithm is linear in the number of tests: O(N)

Test Grading: Example

- Tests report example with per-test grading (score)

```
% urg -dir ./simv.vdb -grade score
```

Tests

dashboard | hierarchy | modlist | groups | tests | asserts

Total Coverage Summary

SCORE	GROUP
91.93	91.93

Total tests in report: 6

Tests are in the order found in the database (the same as the order shown by urg -show availabletests).

Scores are the standalone scores for each test.

SCORE	GROUP	Sim Time	Seed	48Seed	NAME
73.44	73.44	--	1419613496	1419613496	**./simv.vdb/test_gen_4**
74.74	74.74	--	1141850969	1141850969	**./simv.vdb/test_gen_1**
74.22	74.22	--	--	--	**./simv.vdb/test**
70.57	70.57	--	43287995	43287995	**./simv.vdb/test_gen_5**
70.05	70.05	--	943521536	943521536	**./simv.vdb/test_gen_2**
69.53	69.53	--	1715377402	1715377402	**./simv.vdb/test_gen_3**

dashboard | hierarchy | modlist | groups | tests | asserts

Test Grading: Additional Options

- **`goal [<R>]`**
  - Stops grading when the cumulative score reaches R
  - If not specified, the program will process all specified tests
- **`timelimit <N>`**
  - Stops grading after N minutes have passed
  - Only those tests that are graded before the time limit is hit are included in the graded list
- **`maxtests <N>`**
  - Stops after N tests have been graded
- **`minincr <R>`**
  - Stops grading when no remaining tests would add at least R percentage points to the total cumulative score. Only used for the greedy algorithm
- **`reqtests <file>`**
  - When used with **greedy**, specifies reading a list of test names from file for inclusion in the grading report. These tests are included at the top of the graded list, regardless of their scores or effectiveness for coverage