1

Functional Coverage in SystemVerilog

Jason Sprott, Verilab

jason.sprott@verilab.com

Agenda

2

- Introduction
- Overview of cover property
- Overview of covergroup
- Compare cover property with covergroup
- Coding for coverage result analysis
- Coverage feedback

What Do We Cover?

3

- Specified functionality
  - Interface and internal protocols, expected usage, performance/QoS, configurations, etc.
- Design implementation features
  - Unspecified behaviour, FSMs, states or sequences not obvious from the outside
- Abstract data
  - Inter-related configurations (HW and SW), scenarios/sequences
- It's an important metric in determining completeness against a plan

SVA Cover Property

4

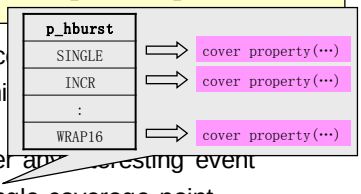
```
hburst_single_write : cover property (
   @(posedge hclk) (!hreset) throughout
   ((p_hburst == `SINGLE) && p_hwrite && ok_response)
);
```

- Similar to a concurrent assertion
- Used to monitor SVA sequences and properties
- Used to cover any interesting event
- Defines a single coverage point

SVA Cover Property

5

```
hburst_single_write : cover property (
   @(posedge hclk) (!hreset) throughout
   ((p_hburst == `SINGLE) && p_hwrite && ok_response)
);
```



- Similar to a c
- Used to moni properties
- Used to cover an… …esting event
- Defines a single coverage point

SVA Cover Property

6

```
hburst_single_write : cover property (
   @(posedge hclk) (!hreset) throughout
   ((p_hburst == `SINGLE) && p_hwrite && ok_response)
);
```

- Similar to a concurrent assertion
- Used to monitor SVA sequences and properties
- Used to cover any interesting event
- Defines a single coverage point
- Not related to the covergroup

What Coverage Results Are Collected?

7

```
sequence s_req2ack (
   req ##[1:10] ack ##1 (!ack)
);
c_req2ack : cover property (
   @(posedge clk) s_req2ack
);
```

```
property p_req2idle (
   @(posedge clk)
   s_req2ack |=> idle
);
c_req2idle : cover property (
   p_req2idle
);
```

- Results for Sequences
  - Number attempted
  - Number matched

  Recommendation:
  only cover sequences
- Results for Properties
  - Number attempted
  - Number passed
  - Number failed
  - Number vacuous passes

Turning A Property Into A Sequence

8

```
sequence s_req2ack (
   req ##[1:10] ack ##1 (!ack)
);
c_req2ack : cover property (
   @(posedge clk) s_req2ack
);
```

```
property p_req2idle (
   @(posedge clk)
   s_req2ack |=> idle
);
c_req2idle : cover property (
   p_req2idle
);
```

- We can turn a properties into sequence

```
req ##[1:10] ack ##1 (!ack) ##1 idle
```

- What's important is that we stick to a consistent methodology

Summary of cover property

9

	package	interface	module	program	class
cover property	✓	✓	✓	✓	✗

- Embedded in the design or separate module
- Good for design/RTL level coverage
- Handles complex temporal expressions
- Not good at mapping samples to multiple bins
- Can't be encapsulated in a class
  - Limits use in testbench

What is a covergroup? (1)

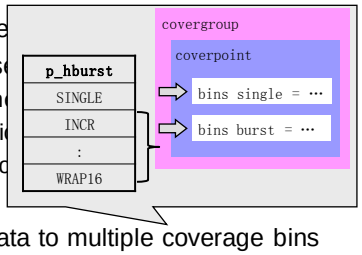
10

- User-defined type
- Encapsulates a set of coverage samples
  - Coverage with the same sample condition
  - State and transitions (sequences)
  - Cross coverage of samples can be defined
- Maps sampled data to multiple coverage bins

What is a covergroup? (1)

11

- User-defined type
- Encapsulates a s
  - Coverage with th
  - State and transiti
  - Cross coverage c



- Maps sampled data to multiple coverage bins

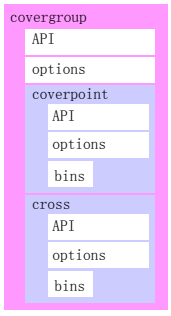
What is a covergroup? (1)

12

- User-defined type
- Encapsulates a set of coverage samples
  - Coverage with the same sample condition
  - State and transitions (sequences)
  - Cross coverage of samples can be defined
- Maps sampled data to multiple coverage bins
- Procedural interface for sampling and control
- Works with classes in an OO testbench

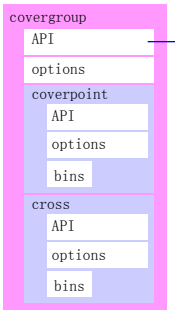
What is a covergroup? (2)

13



What is a covergroup? (3)

14

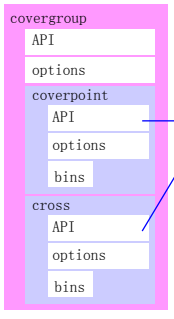


Procedural interface for group, e.g.

`sample()`	Triggers sampling of the group
`start()`	Start collecting coverage
`stop()`	Stop collecting coverage
`get_coverage()`	Get type coverage
`get_inst_coverage()`	Get instance coverage
`set_inst_name()`	Set instance name

What is a covergroup? (4)

15

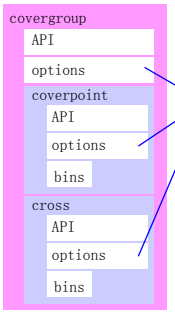


Procedural interface for coverpoint/cross, e.g.

`start()`	Start collecting coverage
`stop()`	Stop collecting coverage
`get_coverage()`	Get type coverage
`get_inst_coverage()`	Get instance coverage

What is a covergroup? (5)

16

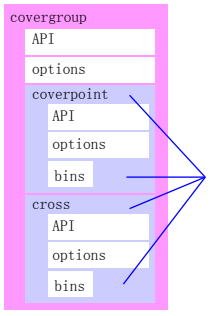


Coverage options for groups, coverpoint and cross, e.g.

`per_instance`	Only valid at group level, turns on tracking on a per instance basis
`at_least`	Minimum number of hits for each bin
`weight`	Weight used to compute the coverage score
`goal`	Target goal as a percentage
`name`	Name for covergroup instance
`comment`	Comment string for the associated element. It is saved in coverage database and appears in reports

What is a covergroup? (6)

17

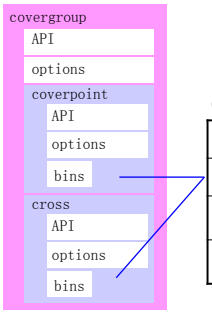


Filtering on individual bins, e.g.

`iff()`	Optional condition for disabling coverage on a coverpoint, cross, or bin
`ignore_bins`	Marks values or transitions to be ignored. These are excluded from coverage calculation
`illegal_bins`	Marks values or transitions as illegal in a coverpoint. If they occur a run-time error is issued
`binsof(x) intersect (y)`	Selects the bins of coverpoint x, whose values intersect with range y

What is a covergroup? (7)

18



Manually specified, or automatically generated bins, e.g.

`bins high = {[32'hFFFF_FF00:$]};`	1 bin covering a range
`bins other[8] = {[32'h0000_0021:32'hFFFF_FEFF]};`	8 bins across range
`bins my_states[] = { IDLE, REQ, ACK }`	3 bins. One for each state
`bins my_seq = (IDLE => REQ[*3] => ACK)`	1 bin for transition (with repetition)

Summary of covergroups

19

	package	interface	module	program	class
cover property	✓	✓	✓	✓	✗
covergroup	✓	✓	✓	✓	✓

- A must for coverage in an OO testbench
- Good for design/RTL level coverage
- Handles complex crosses and multiple bins
- Lots of filtering features
- Good control of coverage measurements

cover property vs covergroup

20

Now let's compare the two types of coverage

Sample Event (1)

21

```
c_single : cover property (
   @(posedge clk) iff (!reset)
   ((p_hburst == SINGLE) &&
ok_response)
);
```

```
covergroup cg_ahb @(posedge clk);
   cp_hbust : coverpoint p_hbust;
endgroup
```

- cover property
  - Clocking event from RTL
- covergroup
  - Clocking event from RTL

Sample Event (2)

22

```
c_single : cover property (
   @(cover_me) iff (!reset)
   ((p_hburst == SINGLE) &&
ok_response)
);
...
-> cover_me; // in procedural code
```

```
covergroup cg_ahb @(posedge clk);
   cp_hbust : coverpoint p_hbust;
endgroup
```

- cover property
  - Clocking event from RTL
  - User defined event
- covergroup
  - Clocking event from RTL

Sample Event (2)

23

```
c_single : cover property (
   @(cover_me) iff (!reset)
   ((p_hburst == SINGLE) &&
ok_response)
);
...
-> cover_me; // in procedural code
```

```
event cover_me;

covergroup cg_ahb @(cover_me);
   cp_hbust : coverpoint p_hbust;
endgroup
...
-> cover_me; // in procedural code
```

- cover property
  - Clocking event from RTL
  - User defined event
- covergroup
  - Clocking event from RTL
  - User defined event

Sample Event (3)

24

```
c_single : cover property (
   @(cover_me) iff (!reset)
   ((p_hburst == SINGLE) &&
ok_response)
);
...
-> cover_me; // in procedural code
```

```
covergroup cg_ahb;
   cp_hbust : coverpoint p_hbust;
endgroup

cg_ahb.sample();
```

- cover property
  - Clocking event from RTL
- covergroup
  - Clocking event from RTL
  - User defined event
  - Can also use `sample()` method

State Coverage

25

```
c_single : cover property (
   @(posedge clk) iff (!reset)
   ((p_hburst == SINGLE) &&
    ok_response)
);
```

```
covergroup cg_ahb;
   cp_hburst : coverpoint p_hburst
               iff (!reset){
      bins single =
       {SINGLE} iff (ok_response);
      ...
   }
   ...
endgroup
```

- cover property
  - Condition on sample
  - Samples RTL state
- covergroup
  - Condition on sample
  - Samples RTL state
  - Has option for multi. bins covering other states

Sequence Coverage (Single Variable)

26

```
c_fsm_startup : cover property (
   @(posedge clk) iff (!reset)
   (state == IDLE) ##1
   (state == STARTUP) ##[1:5]
   (state == READY)
);
```

```
covergroup cg_foo;
 cp_fsm : coverpoint state
   iff (!reset){
     bins startup =
       (IDLE =>
        STARTUP[* 1:5] =>
        READY);
     ...
}
```

- cover property
  - Supports complex temporal relationships
- covergroup
  - Handles single variable sequences well
  - Handles repetition
  - Multiple bins possible

Sequence Coverage (Multi Variable)

27

```
c_req2ack : cover property (
   @(posedge clk) iff (!reset)
   (req ##[1:5] ack #1
    (!ack && !req))
);
```

- cover property
  - Supports complex temporal relationships
- covergroup
  - Doesn't handle this
  - However we can …

Hook-up A Covergroup To A Sequence

28

```
s_inst_commit : sequence(
   bit[7:0] inst_submitted;
   @(posedge clk)
   (req, inst_submitted = inst) ##[1:20]
   (commit, do_cover(inst_submitted));
);
```

```
c_inst_commit : cover property(s_inst_commit);
```

```
// triggers a covergroup sample
function void do_cover(bit[7:0] inst);
   // put data somewhere covergroup can see it
   ...
   // trigger sample
   cg_foo.sample();
endfunction
```

- We can attach actions to a sequence: e.g. store data in a local variable
- Or call function when it matches
- Function triggers covergroup sample
- The sequence must be used in a verification statement to be evaluated

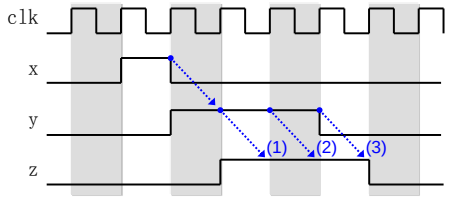
Sequences

29

A few things to remember with SVA …

SVA: Unwanted Matches Gotcha

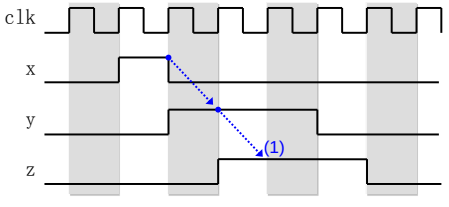
30




```
c_unwanted_matches : cover property(
  @(posedge clk)
  x ##[1:6] y ##1 z
);
```


SVA: Unwanted Matches Gotcha

31




```
c_first_match_only : cover property(
  @(posedge clk)
  first_match(x ##[1:6] y ##1 z)
);
```

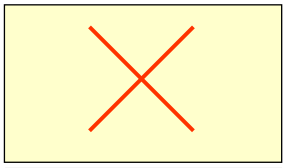

SVA: Some Things To Remember

32

- Sequences can have multiple matches, even if they have an endpoint, e.g. seq1.ended
- A property that is a sequence (no implication) behaves like first_match()
- The RHS (consequent) of a property behaves like first_match()
- The LHS (antecedent) cannot be a property

Mapping Samples onto Multiple Bins

33



```
enum {INC, DEC, NO_CHANGE} count_modes;
bit [31:0] address;

covergroup cg_ahb;
   cp_modes   : coverpoint count_modes;
   cp_address : coverpoint address {
      bins no_access = {[0:255]};
      bins other[8]  = {[256:$]};
   }
endgroup
```

- cover property
  - Doesn't handle this
- covergroup
  - Automatically creates bins for enums and variable ranges
  - Lots of useful features

cover property vs covergroup

Multiple Bins with SVA

34

It's clumsy but some automation is possible …

Using "generate" for Multiple Bins

35

Burst Type	
00	LINEAR
01	WRAP-4
10	WRAP-8
11	WRAP-16

X

Read/Write	
0	READ
1	WRITE

X

Response Type	
00	OK
01	RETRY
10	--
11	ERROR

We want to ignore this value

Total number of cover points to generate = 4 * 2 * 3 = 24

Using "generate" for Multiple Bins

36

```
sequence s_burst_xfer(burst_type, write_en, response_type);
   @(posedge clk)
      (xfer_started && (burst_o == burst_type) &&
      (wr_en == write_en)) ##[0:$]
      (resp_o == response_type);
endsequence : s_burst_xfer
```

Reuse the same sequence for each bin
The values for comparison are passed as arguments

Using "generate" for Multiple Bins

37

```
sequence s_burst_xfer(burst_type, write_en, response_type);
  @(posedge clk)
    (xfer_started && (burst_o == burst_type) &&
    (wr_en == write_en)) ##[0:$]
    (resp_o == response_type);
endsequence : s_burst_xfer
```

```
genvar i,j,k;
generate
  for(i=0;i<4;i++) begin : burst_type
    for(j=0;j<2;j++) begin : write_en
      for(k=0;k<4;k++) begin : response_type
        if(k != 2)
          c_burst_xfer_write : cover property( s_burst_xfer(i, j, k) );
      end
    end
  end
endgenerate
```

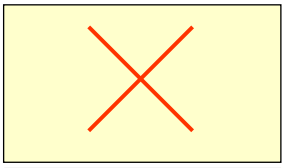
generate values for each cover property

Don't generate unwanted "bin"

Burst Type		Read/Write		Response Type	
00	LINEAR	0	READ	00	OK
01	WRAP-4	1	WRITE	01	RETRY
10	WRAP-8			10	–
11	WRAP-16			11	ERROR

Cross Coverage of Sampled Data

38



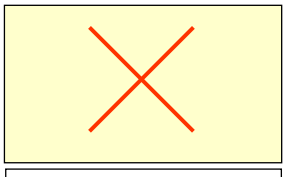
```
enum {M0,M1,M2,M3} src_master, dst_master;
covergroup cg_foo;
  ...
  cross_src_masterXdst_master : cross
     cp_src_master, cp_dst_master;

endgroup
```

- cover property
  - Doesn't handle this
- covergroup
  - Automatically handles crosses
  - Lots of filtering options

cover property vs covergroup

Using with Classes In Testbenches

39



```
class DmaTransactionCoverage;
 DmaTransaction t;

 covergroup cg_foo;
  cp_src_master : coverpoint t.src_master;
  cp_dst_master : coverpoint t.dst_master;
  ...
 endgroup

endclass
```

- cover property
  - Doesn't handle this
- covergroup
  - Can be instantiated in a class
  - Can reference members of a class

cover property vs covergroup

Handling Some Abstraction With SVA

40

We can't use classes, but we can use structs …

Using Structs With Cover Statements

41

```
typedef struct {
   bit [3:0] burst_type;
   bit       wr_en;
   bit [1:0] response;
} bus_transaction;
...
bus_transaction my_transaction;
```

```
sequence s_wrap8_write_retry(t);
   (t.burst_type == `WRAP8) && (t.wr_en == 1) &&
   (t.response == `RETRY);
endsequence : s_wrap8_write_retry

c_wrap8_write_retry : cover property (
   @cover_bus_transaction s_wrap8_write_retry(my_transaction)
);
```

Struct passsed as argument

Used in sequence

User-defined event

Cover specific transaction

Comparing Coverage Methods

42

	cover property	covergroup
In design	✓	✓
In a class (OO testbench)	×	✓
States	✓	✓
Single var. sequences	✓	✓
Complex TEs	✓	× *
Multiple bins	×	✓
Crosses	×	✓
Control goals & weights	×	✓

* We can get a sequence to trigger a covergroup sample

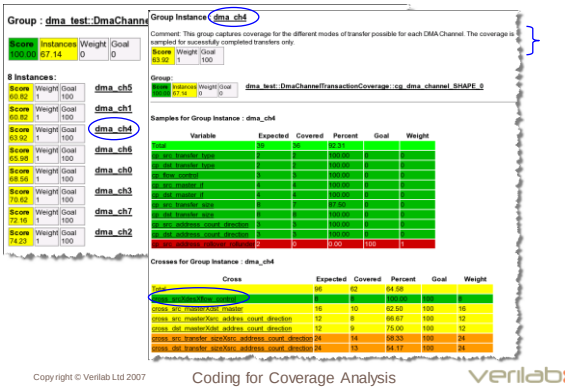
Coding for Coverage Analysis (1)

43

- Assume humans will be reading coverage reports
- Include comments for groups, coverpoints and crosses
- Rename covergroup instances to something meaningful

Sensible Names and Comments etc.

44



Coding for Coverage Analysis (2)

45

- Don't collect coverage you are not using
  - Be tough with this (it's easy to go overboard)
- Weed out unwanted auto-generated bins
- Use filtering to target only interesting samples

Filtering On Cross Bins

46

```
cross source, destination, flow_control {
  // Flow control on memory as source not supported
  ignore_bins mem_src_no_flowctrl  = binsof(source) intersect {MEMORY} &&
                                     binsof(flow_control) intersect {SRC};
  // Flow control on memory as destination not supported
  ignore_bins mem_dest_no_flowctrl = binsof(destination) intersect {MEMORY} &&
                                     binsof(flow_control) intersect {DEST};
}
```

source		destination		flow_control	
MEMORY	x	MEMORY	x	SRC	2 bins
PERIPHERAL		PERIPHERAL		DEST	2 bins
				DMA	

Coding for Coverage Analysis (3)

47

- Make sure goals and weights represent your intentions
- Watch out for sample points only used in crosses
- Only enable instance or type coverage when it's useful
  - Type coverage: cumulative for all instances of a covergroup
  - Instance coverage: is for a specific instance of a covergroup

Exclude Pointless Statistics

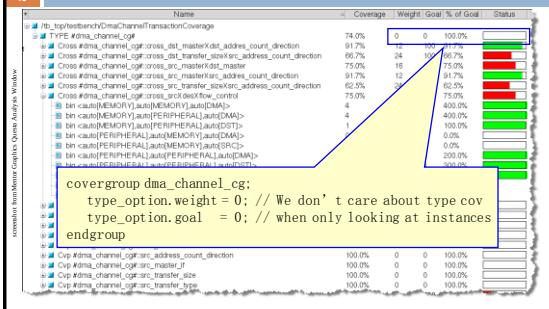
48

```
src_transfer_type : coverpoint t.src_transfer_type {
  option.goal      = 0; option.weight      = 0; // instance
  type_option.goal = 0; type_option.weight = 0; // type
}
```

- Exclude coverpoints only used in cross by setting goals and weights to 0

Exclude Pointless Statistics

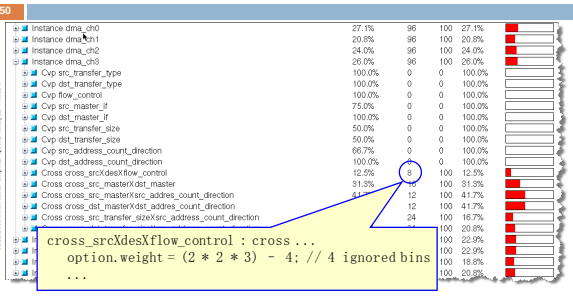
49



- Exclude group coverage by setting weights and goals

Accurate Weights For Cover/Cross

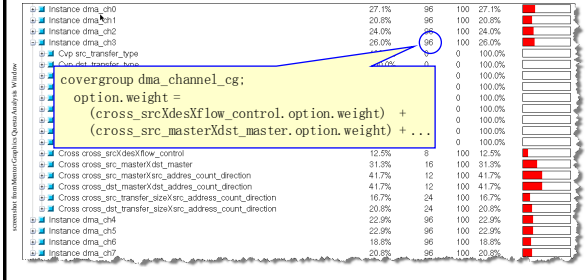
50



- cross goal = number of all non zero weight bins

Accurate Weights For Group

51



- group goal = number of cover and cross points with non-zero weight

Coding for Coverage Analysis (4)

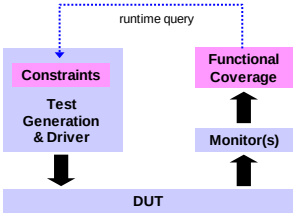
52

- Review coverage like any other code – there will be bugs

Coverage Feedback

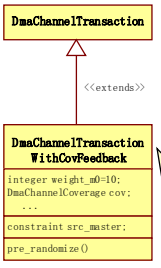
53

Useful Built-in Coverage Run-time Query Methods	
get_coverage()	Calculates type coverage (0-100)
get_inst_coverage()	Calculates instance coverage (0-100)
$get_coverage()	Overall coverage for all groups (0-100)



Extending Class For Coverage Feedback

54




```
class DmaChannelTransactionWithCovFeedback extends
      DmaChannelTransaction;

   // src_master weight distribution
   protected integer weight_src_m0=10;
   protected integer weight_src_m1=10;
   ...
   // allow import of coverage results
   protected DmaChannelTransactionCoverage cov;

   constraint src_master_dist_c {
      src_master dist {
         M0 := weight_src_m0,
         M1 := weight_src_m1,
         ...
   }

   function void pre_randomize();
      if (cov.dma_channel.src_master_m0.get_inst_coverage() == 100)
         weight_src_m0 = 1; // reduce probability of occurring
      ...
   endfunction : pre_randomize
endclass : DmaChannelTransactionWithCovFeedback
```


A Good Idea?

55

- Creating new tests (new constraints, or seed) and test grading is trivial by comparison
- Coverage results rarely map cleanly back to specific constraints
- You'll miss things
  - Coverage models evolve – they are not completed early
  - You will not know all the interesting states yet
- Running the same tests with new seeds can find bugs – even with perceived 100% coverage

Summary

56

- Coverage in design
  - Specified functionality and implementation
  - Scenarios with complex temporal relationships
  - Crosses between different samples
  - Handled using cover property and covergroups
- Coverage in testbench
  - Should fit into object-oriented testbench design
  - Complex stimulus scenarios
  - Data sampling decoupled from design
  - Best handled using covergroups
- Coverage feedback
  - Possible, but typically not the best solution

Functional Coverage in SystemVerilog

Jason Sprott, Verilab
jason.sprott@verilab.com