Chapter 9 Functional Coverage Topics

• Functional coverage is
• Functional coverage is not
• Functional coverage strategies
  Strategies when code coverage or functional coverage is lacking
• Collecting coverage with covergroups
  Basics of collecting coverage with covergroups
• Coverage options
  Options that drive collection coverage or make the reports easier to read
• Bin manipulation
  For example: naming and excluding
• Transition coverage
  Track events over time
• Cross coverage
  Collects coverage on the intersection of coverage points
• Monitoring coverage during simulation
  Testbench can be steered into areas not covered, required for coverage driven verification
Functional Coverage is:

- A measure of which design features have been exercised.
- You’ve already performed functional coverage manually.

- BurstWriteTest
  - 
  - 
  -

- BurstReadTest
  - 
  - 
  -

- WriteReadTest
  - 
  - 
  -

- But how do you know if your new random testbench tests these design features?
- What if the designer disables a design feature?
Functional Coverage is not

- Code coverage
  - Line Coverage – how many lines of code executed
  - Path Coverage – which paths through the code and expressions have been executed
    - Case statement branches
    - If statement conditions
    - Expression coverage
      - \( y = (a | b) + (c & d); \)
  - Toggle Coverage – which single bit variables have had the values 0 or 1
  - FSM Coverage – Which state machine transitions and states have been visited
Functional Coverage is not

• Code coverage
• Cannot be automatically determined from the design

module dff(output logic q, input logic clk, d, reset);
    always @(posedge clk or negedge reset) begin
        q <= d;
    end
endmodule

• Functional coverage goals:
  1. Test loading of register with d = 0 and d=1
  2. Test resetting of register with q=0 and q=1
      ......
• Easy to obtain 100% code coverage on this model
• Impossible to obtain 100% functional coverage
Functional Coverage Verifies Tests
Implement Test Plan

Test Plan

Functional Coverage

Tests
Functional Coverage is a metric for Verification Completeness

Indicates actions required to approach 100% functional coverage

Test Plan

Constrained Random Tests → Many runs different seeds

Directed Tests → Functional Coverage

Add constraints → Minimal Code Modifications

Identify holes → Directed Tests

Directed Tests → Minimal Code Modifications

Minimal Code Modifications → Add constraints
9.2 Functional Coverage Strategies

• Gather information, not data
  • Consider a 1K fifo.
  • What design features do you want to collect coverage on?
• Only measure what you are going to use.
• Measuring completeness

<table>
<thead>
<tr>
<th>Functional Coverage</th>
<th>Code Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Need more FC points, Including corner cases</td>
<td>Is design complete?</td>
</tr>
<tr>
<td>Good coverage: Check bug rate</td>
<td></td>
</tr>
</tbody>
</table>

Start of Project | Low
program automatic test(busifc.TB ifc);

class Transaction;
    rand bit [31:0] data;
    rand bit [ 2:0] dst;
endclass
Transaction tr;
covergroup CovPort;
    coverpoint tr.dst;
endgroup
initial begin
    CovPort ck;
    ck = new();
    repeat (32) begin
        @ifc.cb;
        tr = new();
        `SV_RAND_CHECK(tr.randomize);
        ifc.cb.port <= tr.dst;
        ifc.cb.data <= tr.data;
        ck.sample();
    end
end end endprogram
### Questa Coverage Results

```
VSIM>coverage report -verbose
# COVERGROUP COVERAGE:
# ---------------------------------------------------------------
# Covergroup                  Metric      Goal/Status At Least
# ---------------------------------------------------------------
# TYPE /top/test/CovPort      100.0%      100 Covered
# Coverpoint CovPort::#coverpoint__0# 100.0%      100 Covered
# covered/total bins:         8           8
# bin auto[0]                 5           1 Covered
# bin auto[1]                 7           1 Covered
# bin auto[2]                 3           1 Covered
# bin auto[3]                 4           1 Covered
# bin auto[4]                 2           1 Covered
# bin auto[5]                 4           1 Covered
# bin auto[6]                 3           1 Covered
# bin auto[7]                 4           1 Covered
```
Coverage Results in the Questa GUI
9.5 Anatomy of a Cover Group

• A covergroup can be defined in a package, module, program, interface, or class.
• Needs to be instantiated using `new()`.
• Contain:
  1. A clocking event
     • If not defined in the covergroup, use `sample()` construct.
  2. 1 or more coverage points
     • What you are going to collect coverage on.
  3. Cross coverage between coverage points
     • Intersection of 2 or more coverage points.
  4. Optional formal arguments
  5. Coverage options
• Recommendations
  1. Use clear names for covergroups
  2. Don’t define a covergroup in a data class.
  3. Label the coverpoints
class Transactor;
  Transaction tr;
mailbox #(Transaction) mbx;
covergroup CovPort;
  coverpoint tr.port;
endgroup
function new(input mailbox #(Transaction) mbx);
  CovPort = new(); // Instantiate covergroup
  this.mbx = mbx;
endfunction
endclass
9.6 Triggering in a Cover Group

Covergroup is triggered from:

1. A sample directive from procedural code

```vhdl
CovPort.sample();
```

2. A blocking expression in the covergroup

```vhdl
color_t color;
covergroup g1 @(posedge clk);
  coverpoint color;
endgroup

event trans_ready;
covergroup CovPort @(trans_ready);
  coverpoint ifc.cb.port;
endgroup
```
Exercise 1

For the code below, write a covergroup to collect coverage on the test plan requirement: “All ALU opcodes must be tested”.

Assume the opcodes are valid on the positive edge of signal $clk$.

typedef enum {ADD, SUB, MULT, DIV} opcode_e;

class Transaction;
    rand opcode_e opcode;
    rand byte operand1;
    rand byte operand2;
endclass

Transaction tr;
Exercise 1

For the code below, write a covergroup to collect coverage on the test plan requirement: “All ALU opcodes must be tested”.

Assume the opcodes are valid on the positive edge of signal clk.

typedef enum {ADD, SUB, MULT, DIV} opcode_e;

class Transaction;
   rand opcode_e opcode;
   rand byte operand1;
   rand byte operand2;
endclass

Transaction tr;

covergroup CovCode @ifc.cb;
   // If using an interface, @(posedge clk) if the clock is available.
   coverpoint tr.opcode;
endgroup
9.10 Coverage Options

- **Per-instance coverage**
  
  ```
  option.per_instance = 1;
  ```

- **Cover group comment**
  
  ```
  option.comment = "Setting bin middle from 1-6";
  ```

- **Name**
  
  ```
  option.name = "CovPort";
  ```

- **auto_bin_max**
- **weight**
9.7 Data Sampling

- Bins are automatically created for cover points
- For an n-bit expression, $2^n$ bins are created.
- The maximum number of bins can be reduced by setting the `auto_bin_max` option.

```vhdl
covergroup CovPort;
    option.auto_bin_max = 2;
    coverpoint tr.port;
endgroup
```

<table>
<thead>
<tr>
<th>Covergroup</th>
<th>Metric</th>
<th>Goal/ Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE /top/test/CovPort</td>
<td>100.0%</td>
<td>100 Covered</td>
</tr>
<tr>
<td>Coverpoint CovPort::#coverpoint_0#</td>
<td>100.0%</td>
<td>100 Covered</td>
</tr>
<tr>
<td>covered/total bins:</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>bin auto[0:3]</td>
<td>19</td>
<td>1 Covered</td>
</tr>
<tr>
<td>bin auto[4:7]</td>
<td>13</td>
<td>1 Covered</td>
</tr>
</tbody>
</table>
9.7.4 Sampling Expressions

- Expressions in coverpoints are allowed
- But check the report for the correct # of bins.

```plaintext
class Packet;
    rand bit [2:0] hdr_len;
    rand bit [3:0] payload_len;
    rand bit [3:0] kind;
endclass

Packet p;

covergroup CovLen;
    len16: coverpoint (p.hdr_len + p.payload_len);
    len32: coverpoint (p.hdr_len + p.payload_len + 5'b0);
endgroup
```

- `len16` coverpoint creates 16 bins
- `len32` coverpoint creates 32 bins
### 9.7.4 Sampling Expressions (cont.)

Explicitly specify bins if expected # of bins is not a power of 2

```verbatim
covergroup CovLen;
  len: coverpoint (p.hdr_len + p.payload_len + 5'b0)
    
    {bins len[] = {[0:22]}; }
endgroup
```

<table>
<thead>
<tr>
<th># Covergroup</th>
<th>Metric</th>
<th>Goal/ Status</th>
</tr>
</thead>
<tbody>
<tr>
<td># TYPE /top/test/CovLen</td>
<td>100.0%</td>
<td>100 Covered</td>
</tr>
<tr>
<td># Coverpoint CovLen::len</td>
<td>100.0%</td>
<td>100 Covered</td>
</tr>
<tr>
<td># covered/total bins:</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td># bin len[0]</td>
<td>7</td>
<td>1 Covered</td>
</tr>
<tr>
<td># bin len[1]</td>
<td>17</td>
<td>1 Covered</td>
</tr>
</tbody>
</table>

......

| # | bin len[21] | 12 | 1 Covered |
| # | bin len[22] | 9 | 1 Covered |
9.7.6 Naming the cover bins

Define ranges for coverpoints and name the ranges

covergroup CovKind;
  coverpoint p.kind {
    bins zero = {0};
    bins lo = {[1:3], 5};
    bins hi[] = {[8:$]};
    bins misc = default;
  }
endgroup

$: shorthand for largest value for the sampled variable

default: misc holds all unchosen values
Exercise 2

Expand the last exercise to cover the test plan requirement, “Operand1 shall take on the values maximum negative (-128), zero, and maximum positive (127).” Define a coverage bin for each of these values as well as a default bin. Label the coverpoint `operand1_cp`.

typedef enum {ADD, SUB, MULT, DIV} opcode_e;

class Transaction;
    rand opcode_e opcode;
    rand byte operand1;
    rand byte operand2;
endclass

Transaction tr;
typedef enum {ADD, SUB, MULT, DIV} opcode_e;

class Transaction;
    rand opcode_e opcode;
    rand byte operand1;
    rand byte operand2;
endclass

Transaction tr;

covergroup CovCode @ifc.cb;
    operand1_cp: coverpoint tr.operand1{
        bins max_neg = {-128};
        bins zero = {0};
        bins max_pos = {127};
        bins misc = default;
    }
endgroup
9.7.7 Conditional Coverage

• Use `iff` to add a condition to a cover point

```verilog
covergroup CoverPort;
    coverpoint tr.port iff (!bus_if.reset);
endgroup
```

• Use `stop() / start()` to halt/resume collection of coverage

```verilog
initial begin
    CovPort ck = new();
    #1ns ck.stop();
    bus_if.reset <= 1;
    #100ns bus_if.reset = 0;
    ck.start();
    ...
end
```
9.7.8 Creating Bins for enumerated types

For enumerated types, a bin is created for each value

```haskell
typedef enum {INIT, DECODE, IDLE} fsmstate_e;
fsmstate_e pstate, nstate;
covergroup CovFSM;
    coverpoint pstate;
endgroup
```

To group multiple values in a single bin, define your own bins.

```haskell
covergroup new_bin;
    coverpoint pstate {
        bins non_idle = {INIT, DECODE};
        bins misc = default;
    }
endgroup
```
9.7.9 Transition Coverage

• Up to this point only considered static coverage.
• Can specify transition coverage

```plaintext
covergroup CovPort;
    coverpoint tr.port{
        bins zero_one = (0 => 1);
        bins zero_two = (0 => 2);
        bins zero_to_two = (0 => 1), (0 => 2);
        bins zero_to_two_alt = (0=>1,2);
    }
endgroup
```

• Can specify transitions of any length.

```plaintext
bins zero_one_two = (0=>1=>2);
bins zero_one_one_two = (0=>1=>1=>2);
bins zero_one_one_two_alt = (0=>1[*2]=2);
```

• To repeat value 1 for 3, 4, or 5 times use 1[*3:5]
9.7.10 Wildcard States and Transitions

Suppose we want to collect coverage on `port` being even or odd

```verilog
covergroup CovPort;
    coverpoint tr.port[0] {
        bins even = {1'b0};
        bins odd = {1'b1};
    }
endgroup

Or use `wildcard` keyword

```verilog
covergroup CovPort;
    coverpoint tr.port {
        wildcard bins even = {3'b??0};
        wildcard bins odd = {3'b??1};
    }
endgroup
```
9.7.11 Ignoring Values

• Suppose, due to the design, port will never exceed the value 5
• One solution is to create a custom bin

```vhdl
covergroup CovPort;
    coverpoint tr.port{
        bins zero_to_five[] = {[0:5]};
    }
endgroup
```

• Another solution is to use the `ignore_bins` construct.

```vhdl
covergroup CovPort;
    coverpoint tr.port{
        ignore_bins hi = {6,7};
    }
endgroup
```
9.7.12 Illegal Bins

If certain ranges of a variable are illegal (you think!) define them as illegal.

```verilog
covergroup CovPort;
  coverpoint tr.port{
    bins zero_to_five[] = {[0:5]};
    illegal_bins no_hi = {6,7};
  }
endgroup

or

covergroup CovPort;
  coverpoint tr.port{
    ignore_bins hi = {6,7};
    illegal_bins no_hi = {6,7};
  }
endgroup
```
Exercise 3

Expand the last exercise to cover the following test plan requirements:

1. “The opcode shall take on the values ADD or SUB” (hint: this is 1 coverage bin).
2. “The opcode shall take on the values ADD followed by SUB” (hint: this is a second coverage bin).

Label the coverpoint `opcode_cp`.

```cpp
typedef enum {ADD, SUB, MULT, DIV} opcode_e;
class Transaction;
    rand opcode_e opcode;
    rand byte operand1;
    rand byte operand2;
endclass
Transaction tr;
```
typedef enum {ADD, SUB, MULT, DIV} opcode_e;

class Transaction;
    rand opcode_e opcode;
    rand byte operand1;
    rand byte operand2;
endclass

Transaction tr;

opcode_cp: coverpoint tr.opcode{
    bins add_sub = {ADD, SUB};
    bins add_then_sub = (ADD=>SUB);
    illegal_bins no_div = {DIV};
}
typedef enum {TRANS, RECEIVE} direction_e;
class Transaction;
    rand direction_e direction;
    rand bit [2:0] port;
endclass

covergroup CovPort;
    cross tr.direction, tr.port;
endgroup

covergroup CovPort;
    direction: coverpoint tr.direction;
    port: coverpoint tr.port;
    cross direction, port;
endgroup

9.8 Cross Coverage

Cross coverage collects coverage on the intersection of 2 or more coverage points.
9.8.2 Labeling Cross Coverage Bins

To reduce the # of bins, create custom bins

```vhdl
covergroup CovPort;
  direction: coverpoint tr.direction;
  port: coverpoint tr.port{
    bins zero = {0};
    bins middle = {[1:6]};
    bins maximum  = {7};
  }
  cross direction, port;
endgroup

bins misc = default;
```
9.8.3 Excluding Cross Coverage Bins

• Sample 9.34:

covergroup CovDst34;
    dst: coverpoint tr.dst;
    {bins dst[] = {0:$};
     kind: coverpoint tr.kind {
       bins zero   = {0};
       bins lo     = {[1:3]};
       bins hi[]   = {[8:$]};
       bins misc   = default;
     }
    cross kind, dst{
        ignore_bins hi = binsof(dst) intersect {7};
        ignore_bins md = binsof(dst) intersect{0} &&
            binsof(kind) intersect {[9:11]};
        ignore_bins lo = ninsof(kind.lo)};
endgroup

Note the [] as with dynamic arrays
## 9.8.3 Excluding Cross Coverage Bins

### Sample 9.34:

<table>
<thead>
<tr>
<th>kind</th>
<th>dst</th>
<th>dst_0</th>
<th>dst_1</th>
<th>dst_2</th>
<th>dst_3</th>
<th>dst_4</th>
<th>dst_5</th>
<th>dst_6</th>
<th>dst_7</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>lo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>misc</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hi_8</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hi_9</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hi_a</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hi_b</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hi_c</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hi_d</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hi_e</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hi_f</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **dst**
- **dst_0**
- **dst_1**
- **dst_2**
- **dst_3**
- **dst_4**
- **dst_5**
- **dst_6**
- **dst_7**

- **kind**
- **zero**
- **lo**
- **misc**
- **hi_8**
- **hi_9**
- **hi_a**
- **hi_b**
- **hi_c**
- **hi_d**
- **hi_e**
- **hi_f**

### Code Examples:

- `ignore_bins lo = binsof(kind.lo)`
- `ignore_bins hi = binsof(dst) intersect{7}`
- `ignore_bins md = binsof(dst) intersect{0} && binsof(kind) intersect{[9:11]}`
9.8.3 Excluding Cross Coverage Bins

- As before use `ignore_bins` to reduce the # of cross coverage bins
- Use `binsof & intersect` to specify cross coverage bins to ignore

```vhdl
covergroup CovPort;
  direction: coverpoint tr.direction;
  port: coverpoint tr.port {
    bins zero = {0};
    bins middle = {[1:6]};
    bins maximum = {7};
  }

  cross direction, port {
    ignore_bins port_zero = binsof(port) intersect {0};
    ignore_bins port_0 = binsof(port.zero);
    ignore_bins trans_five = binsof(port) intersect {5} &&
      binsof(direction) intersect {TRANS};
  }
endgroup
```

Equivalent
### 9.8.3 Excluding Cross Coverage Bins (cont.)

<table>
<thead>
<tr>
<th>direction</th>
<th>zero</th>
<th>middle</th>
<th>maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANS</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>RECEIVE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```c
ignore_bins port_zero = binof(port) intersect {0};

ignore_bins port_zero = binof(trans_five = binof(port) intersect {5} && binof(direction) intersect {TRANS});
```
9.8.4 Excluding Cover Points from the Total Coverage Metric

• Suppose you define a cover point just to be used for cross coverage
• Use `weight` to ignore the coverage contribution of this cover point.

```plaintext
covergroup CovPort;
    option.per_instance = 1;
    direction: coverpoint tr.direction;
    port: coverpoint tr.port
    {option.weight = 0;}
    cross direction, port
    {option.weight = 2;}
endgroup
```
type_option vs option

- `type_option.<option>` specifies an option for the covergroup
- `option.<option>` specifies an option for an instance of the covergroup

```verilog
covergroup CovCode;
  coverpoint tr.opcode;
  option.per_instance = 1;
  type_option.comment = "type_option in CovCode";
  option.comment = "option in CovCode";
endgroup

initial begin
  CovCode ck, ck2;
  ck = new();
  ck2 = new();
  tr = new();
  ck.option.comment = "option on ck in initial";
  ....
end
```
Exercise 4

Expand the last exercise to:
1) Collect coverage on the test plan requirement, “The opcode shall take on the values ADD or SUB when operand1 is maximum negative or maximum positive value.”
2) Weight the cross coverage by 5

```cpp
typedef enum {ADD, SUB, MULT, DIV} opcode_e;

class Transaction;
    rand opcode_e opcode;
    rand byte operand1;
    rand byte operand2;
endclass

Transaction tr;
```
typedef enum {ADD, SUB, MULT, DIV} opcode_e;

class Transaction;
    rand opcode_e opcode;
    rand byte operand1;
    rand byte operand2;
endclass

Transaction tr;
covergroup CovCode @ifc.cb;
    operand1_cp: coverpoint tr.operand1{
        bins max_neg = {-128};
        bins zero = {0};
        bins max_pos = {127};
        bins misc = default;
    }
endgroup

opcode_cp: coverpoint tr.opcode{
    bins add_sub = {ADD, SUB};
    bins add_then_sub = (ADD=>SUB);
    illegal_bins no_div = {DIV};
}

code_operand1: cross opcode_cp, operand1_cp {
    // Ignore bins operand1_zero = binsof(operand1_cp.zero); // Ignore operand1 = 0
    // or
    ignore_bins operand1_zero = binsof(operand1_cp) intersect{0}; // Ignore operand1 = 0
    // Removes bins {* , add_then_sub}
    ignore_bins opcode_add_then_sub = binsof(opcode_cp.add_then_sub); // Ignore opcode = add_then_sub
    // But not this since it does not compile with a not very descriptive syntax error.
    ignore_bins opcode_add_then_sub = binsof(opcode_cp) intersect{ADD=>SUB};

    option.weight = 5;
}
9.8.1 Pass Cover Group Args. by Value

Reuse covergroups by passing in arguments to `new()`:

```vhdl
covergroup CovPort (int mid);
    custom_bin: coverpoint tr.port
        {bins lo = {[0:mid-1]};
        bins hi = {[mid:$]};
    }
    auto_bin: coverpoint tr.port;
endgroup

initial begin
    CovPort cp;
    cp = new(5);
end
```
9.8.2 Pass Cover Group Args. by Ref.

To make covergroups even more generic pass args by reference

```verilog
covergroup CovPort(ref bit [2:0] port, input int mid);
    option.per_instance = 1;
    custom_bin: coverpoint port
        {bins lo = {[0:mid-1]};
        bins hi = {[mid:$]};
    }
endgroup

initial begin
    CovPort cpa, cpb;
    tr = new();
    cpa = new(tr.port_a, 6);
    cpb = new(tr.port_b, 2);
end
```
9.11 Measuring Coverage Statistics During Simulation

- Evaluate coverage statistics during simulation to:
  - Quit the simulation
  - Change constraints
  - Evaluate how simulation is progressing

```plaintext
initial begin
  forever begin
    repeat (4) @ifc.cb;
    $display("%t: Instantiation total coverage is %f", $time,
             ck.get_coverage());
    $display("%t: Covergroup total coverage is %f", $time,
             CovPort::get_coverage());
    $display("%t: Instantiation Dir x port coverage is %f", $time,
             ck.dir_port.get_coverage());
    $display("%t: Covergroup Dir x port coverage is %f", $time,
             CovPort::dir_port.get_coverage());
  end
end
```
Changes to Questa Covergroups between 6.5b and later versions

In Questa 6.5b and older the bins for a covergroup were displayed by opening up a covergroup in the covergroups window. Now, in later versions they are not. How do I go back to the old functionality?

merge_instances=1: Coverage is computed by merging instances together as the union of coverage of all instances.

merge_instances=0: Coverage is computed as the weighted average of instances.

Bins required

Bins not required

type_option.merge_instances = 1;

vsim -cvg63
Exercise 5

Assuming that your covergroup is called CovCode and the instantiation name of the covergroup is ck expand the last exercise to:

1) Display the coverage of coverpoint operand1_cp referenced by the instantiation name
2) Display the coverage of coverpoint opcode_cp referenced by the covergroup name

typedef enum {ADD, SUB, MULT, DIV} opcode_t;

class Transaction;
    rand opcode_t opcode;
    rand byte operand1;
    rand byte operand2;
endclass

Transaction tr;
Exercise 5

Assuming that your covergroup is called `CovCode` and the instantiation name of the covergroup is `ck` expand the last exercise to:

1) Display the coverage of coverpoint `operand1_cp` referenced by the instantiation name

2) Display the coverage of coverpoint `opcode_cp` referenced by the covergroup name

```plaintext
typedef enum {ADD, SUB, MULT, DIV} opcode_t;

class Transaction;
    rand opcode_t opcode;
    rand byte operand1;
    rand byte operand2;
endclass

Transaction tr;

$display("%t: Coverpoint ck.operand1_cp coverage is %f", $time, ck.operand1_cp.get_coverage());
$display("%t: Covergroup CovCode::opcode_cp is %f", $time, CovCode::opcode_cp.get_coverage());
```