INF5430

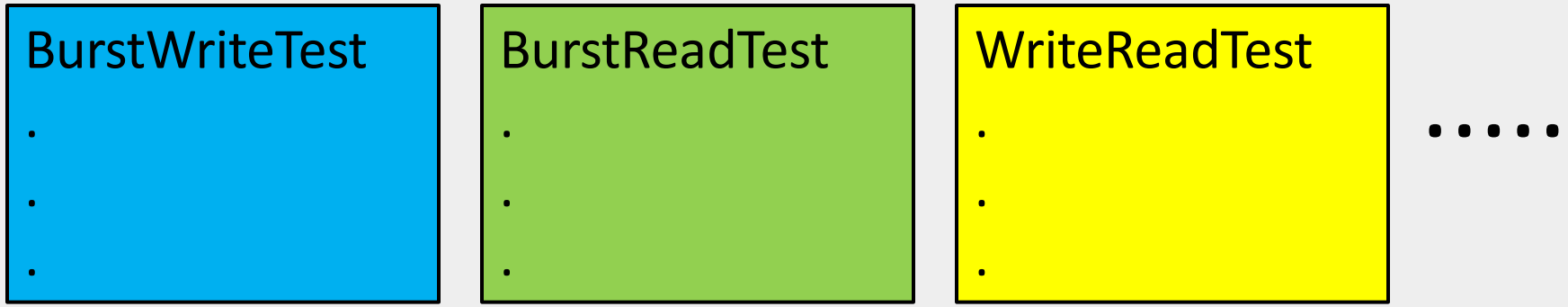
SystemVerilog for Verification

Chapter 9

Functional Coverage

Functional Coverage is:

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



- 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

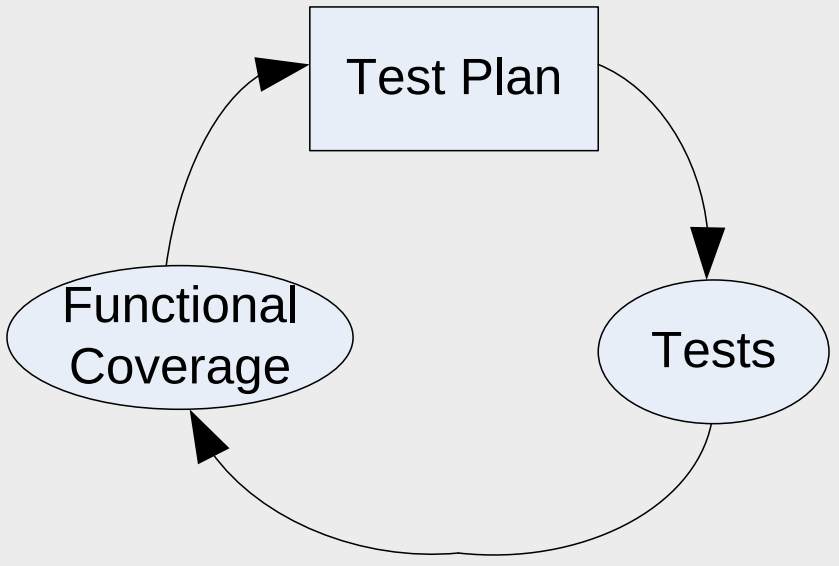
- Code coverage
- Cannot be automatically determined from the design

```
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

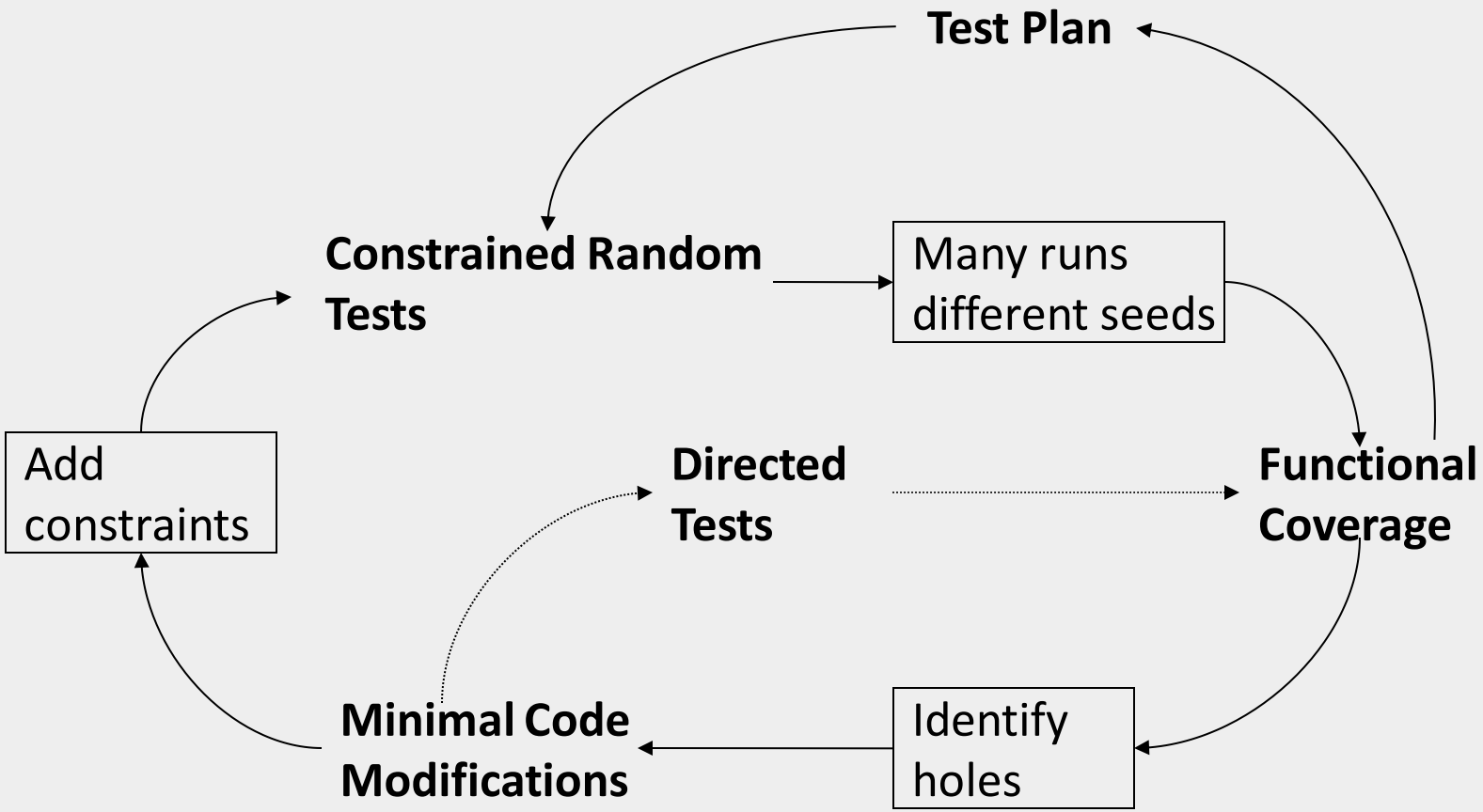
  .....
- Easy to obtain 100% code coverage on this model
- Impossible to obtain 100% functional coverage

Functional Coverage Verifies Tests Implement Test Plan



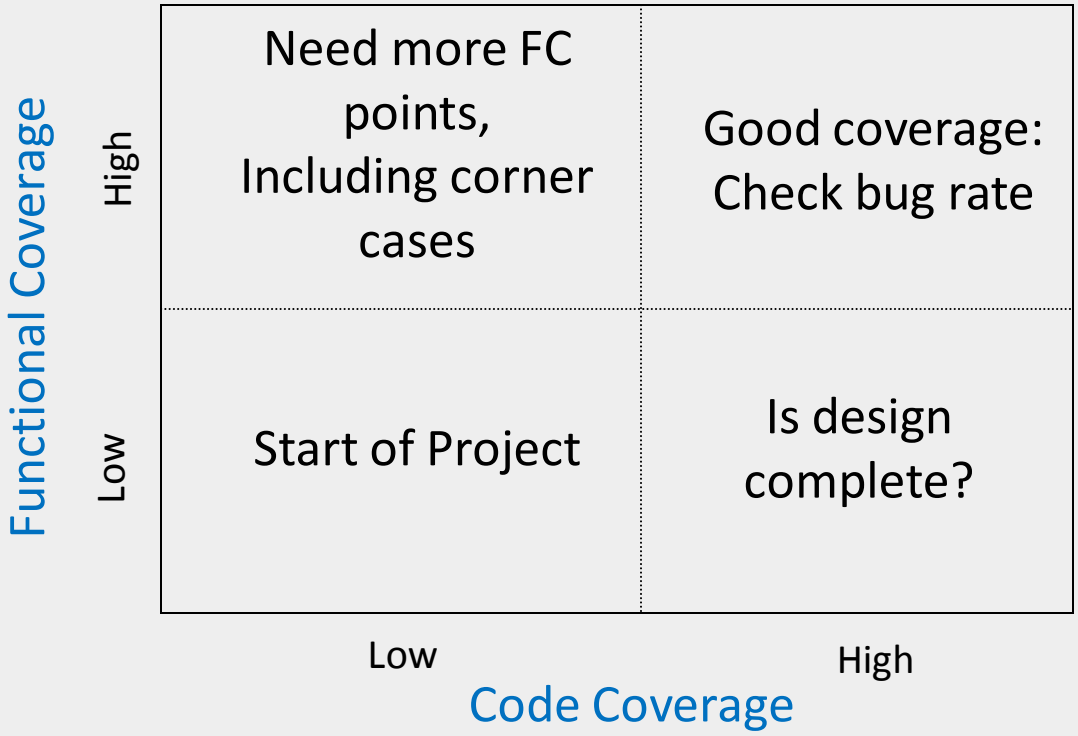
Functional Coverage is a metric for Verification Completeness

Indicates actions required to approach 100% functional coverage



9.3 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



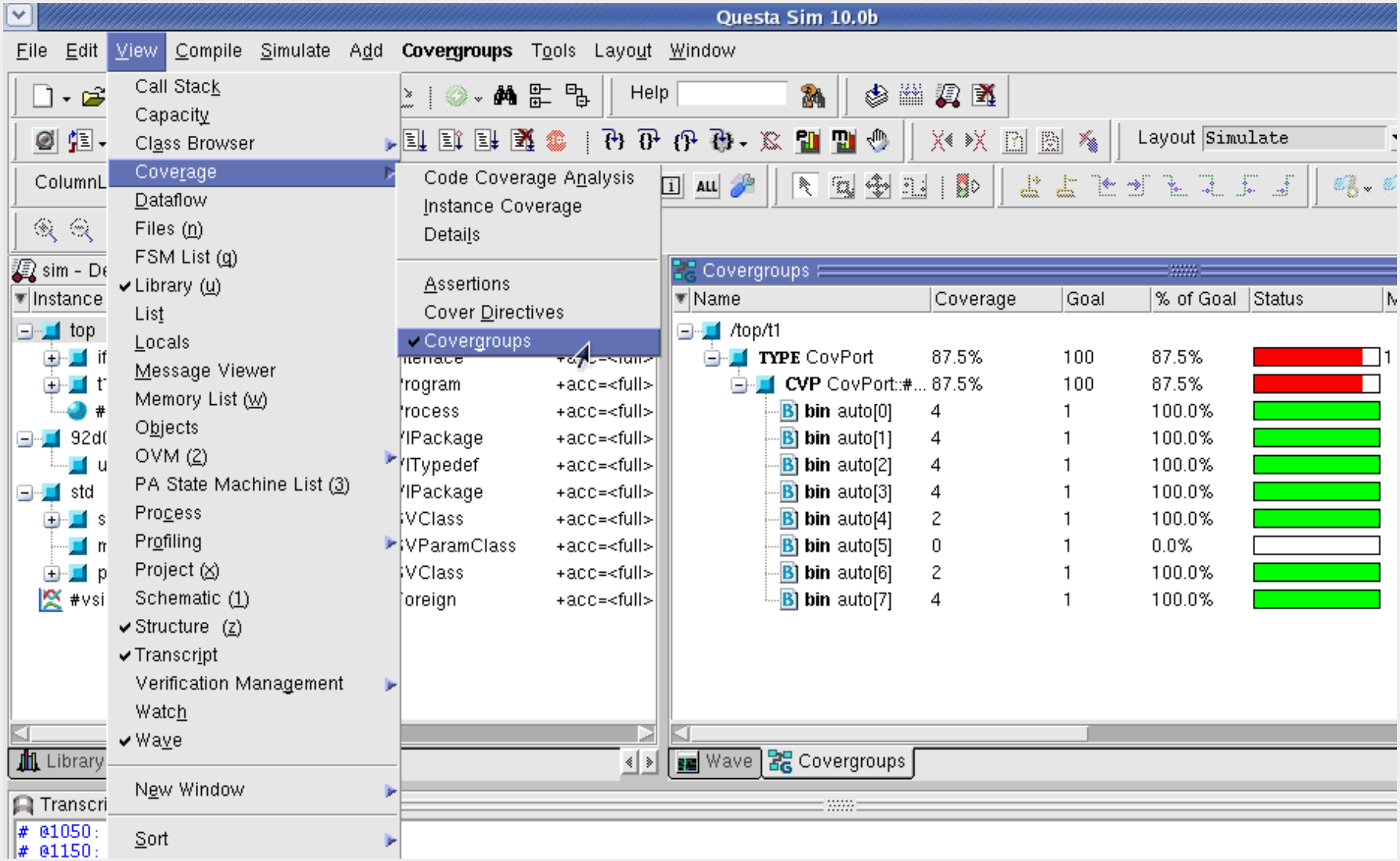
9.4 Simple Functional Coverage Example

```
program automatic test(busifc.TB ifc);
    class Transaction;
        rand bit [31:0] data;
        rand bit [ 2:0] port;
    endclass
    Transaction tr;
    covergroup CovPort;
        coverpoint tr.port;
    endgroup   // No semicolon!
    initial begin
        CovPort ck;
        ck = new();
        repeat (32) begin
            @ifc.cb;  // Wait a cycle
            tr = new();
            `SV_RAND_CHECK(tr.randomize);
            ifc.cb.port <= tr.port;
            ifc.cb.data <= tr.data;
            ck.sample();  // Gather coverage
        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
    2. 1 or more coverage points
    3. Cross coverage between 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

9.5.1 Defining a Covergroup in a Class

```
class Transactor;
    Transaction tr;
    mailbox #(Transaction) mbx;
    covergroup CovPort;
        coverpoint tr.port;
    endgroup
    function new(input mailbox #(Transaction) mbx);
        CovPort = new();
        this.mbx = mbx;
    endfunction
    task run();
        forever begin
            mbx.get(tr);
            @ifc.cb;
            ifc.cb.port <= tr.port;     // Send port into DUT via
            ifc.cb.data <= tr.data;     // <interface>.<clocking block>
            CovPort.sample();
        end
    endtask
endclass
```

9.6 Triggering in a Cover Group

Covergroup is triggered from:

1. A sample directive from procedural code

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

2. A blocking expression in the covergroup

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

A covergroup blocking expression can NOT be a wait. It can be:

- @(event)
- @(signal)
- @(posedge signal)
- @(negedge signal)

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

9.9 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.

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

default is 64

```
# 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:                    2          2
#         bin auto[0:3]                         19          1 Covered
#         bin auto[4:7]                         13          1 Covered
```

9.7.4 Sampling Expressions

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

```
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

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

```
 # Covergroup                          Metric  Goal/ Status
#                                              At Least
# ------------------------------------------------------------
#  TYPE /top/test/CovLen                100.0%  100 Covered
#     Coverpoint CovLen::len            100.0%  100 Covered
#         covered/total bins:             23     23
#         bin len[0]                       7      1 Covered
#         bin len[1]                      17      1 Covered
.......
#         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 // No semicolon
```

```
# TYPE /top/test/CovKind                    100.0%  100 Covered
# Coverpoint CovKind::#coverpoint__0#       100.0%  100 Covered
# covered/total bins:                        10      10
#    bin zero                                48       1 Covered
#    bin lo                                 252       1 Covered
#    bin hi[8]                               61       1 Covered
      ....
#    bin hi[15]                              70       1 Covered
#    default bin misc                       174         Occurred
```

9.7.7 Conditional Coverage

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

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

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

```
initial begin
    CovPort ck = new();
    #1ns ck.stop(); // Reset sequence stops collection of coverage data
    bus_if.reset <= 1;
    #100ns bus_if.reset = 0;
    ck.start(); // Start collecting coverage again.
    :
    :
end
```

9.7.8 Creating Bins for enumerated types

For enumerated types, a bin is created for each value

```
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.

```
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

```
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
```

Equivalent

- Can specify transitions of any length.

```
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);
```

Not Equivalent

Equivalent

9.7.10 Wildcard States and Transitions

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

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

Or use `wildcard` keyword

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

or

```
{3'bXX0};
{3'bXX1};
```

or

```
{3'bZZ0};
{3'bZZ1};
```

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

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

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

```
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.

```
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
```

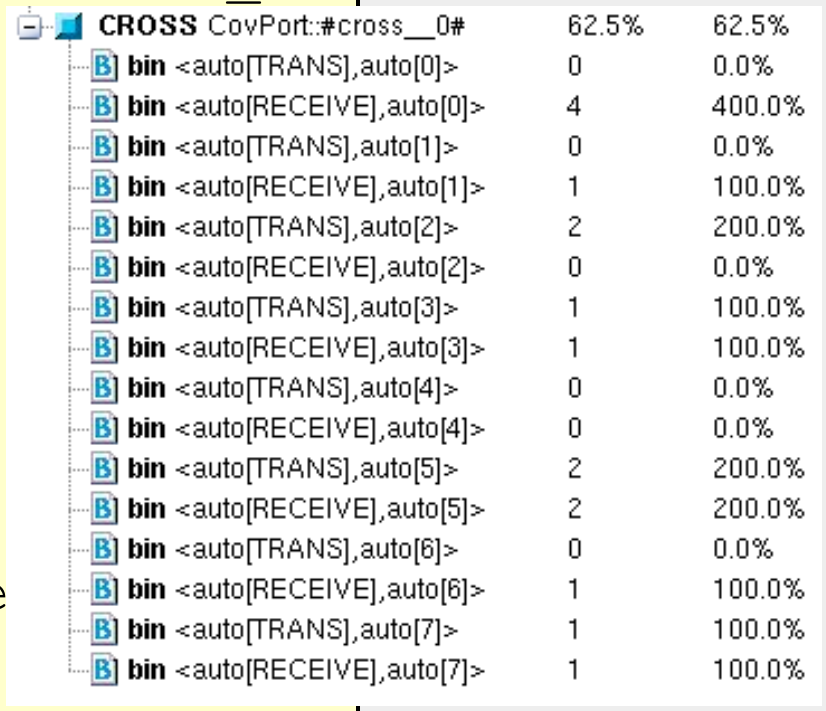
9.8 Cross Coverage

Cross coverage collects coverage on the intersection of 2 or more coverage points.

```
typedef enum {TRANS, RECEIVE} direction_e;
class Transaction;
   rand direction_e direction;
   rand bit [2:0] port;
endclass

Transaction tr;

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



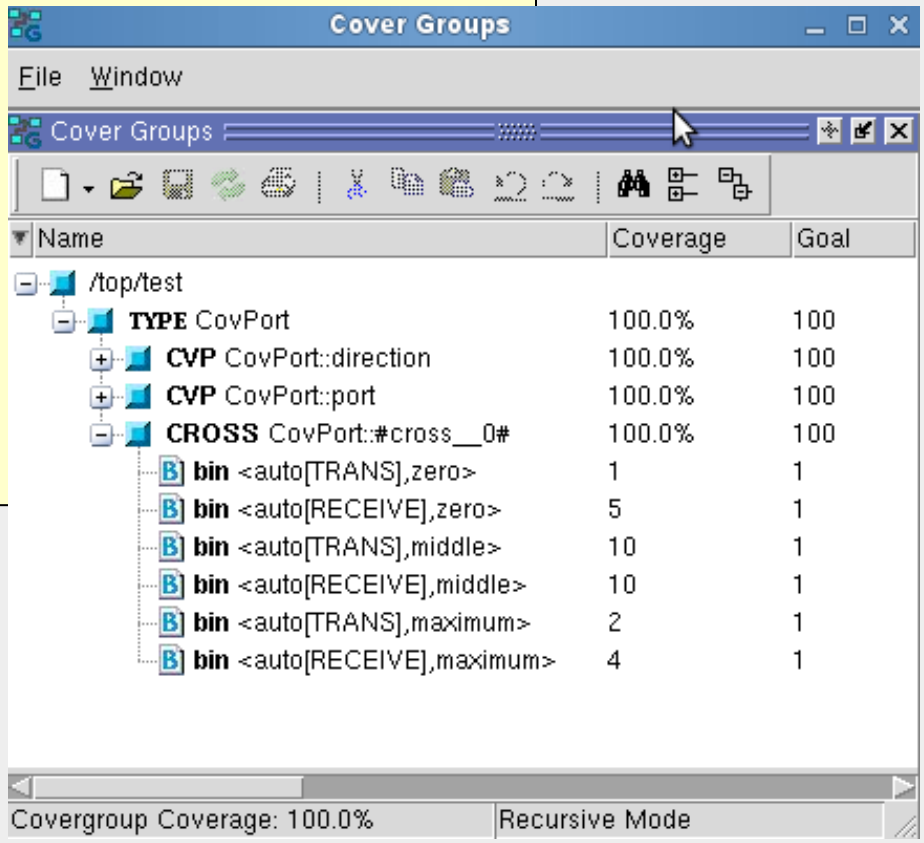
CROSS CovPort::#cross__0#	62.5%	62.5%
bin <auto[TRANS],auto[0]>	0	0.0%
bin <auto[RECEIVE],auto[0]>	4	400.0%
bin <auto[TRANS],auto[1]>	0	0.0%
bin <auto[RECEIVE],auto[1]>	1	100.0%
bin <auto[TRANS],auto[2]>	2	200.0%
bin <auto[RECEIVE],auto[2]>	0	0.0%
bin <auto[TRANS],auto[3]>	1	100.0%
bin <auto[RECEIVE],auto[3]>	1	100.0%
bin <auto[TRANS],auto[4]>	0	0.0%
bin <auto[RECEIVE],auto[4]>	0	0.0%
bin <auto[TRANS],auto[5]>	2	200.0%
bin <auto[RECEIVE],auto[5]>	2	200.0%
bin <auto[TRANS],auto[6]>	0	0.0%
bin <auto[RECEIVE],auto[6]>	1	100.0%
bin <auto[TRANS],auto[7]>	1	100.0%
bin <auto[RECEIVE],auto[7]>	1	100.0%

9.8.2 Labeling Cross Coverage Bins

To reduce the # of bins, create custom bins

```
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;`~~



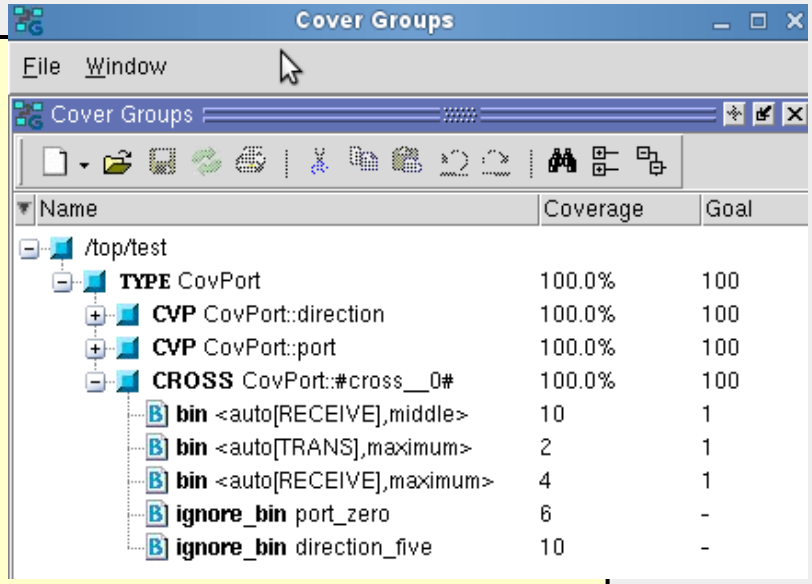
9.8.3 Excluding Cross Coverage Bins

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

```
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
```

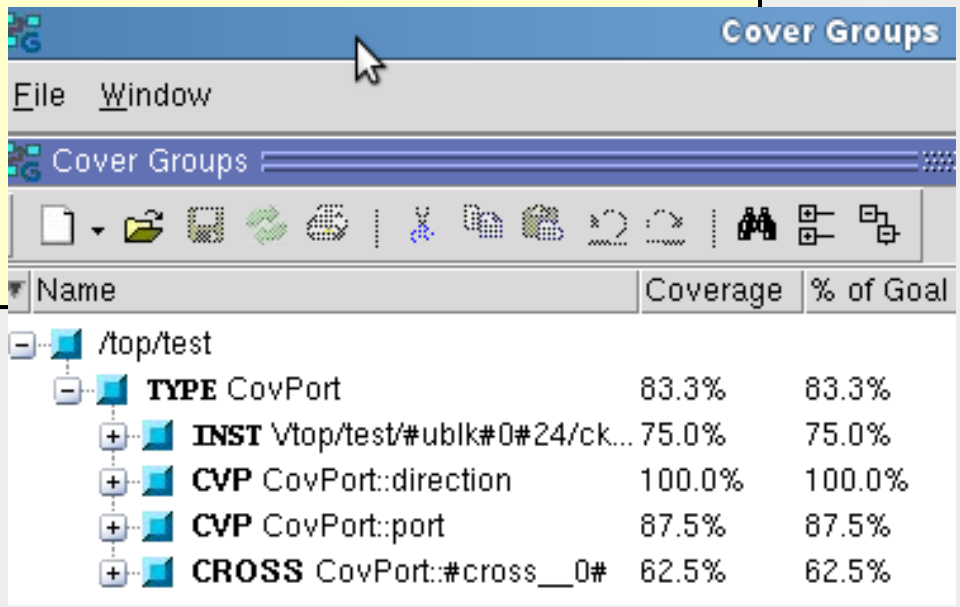
Equivalent



9.8.4 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.

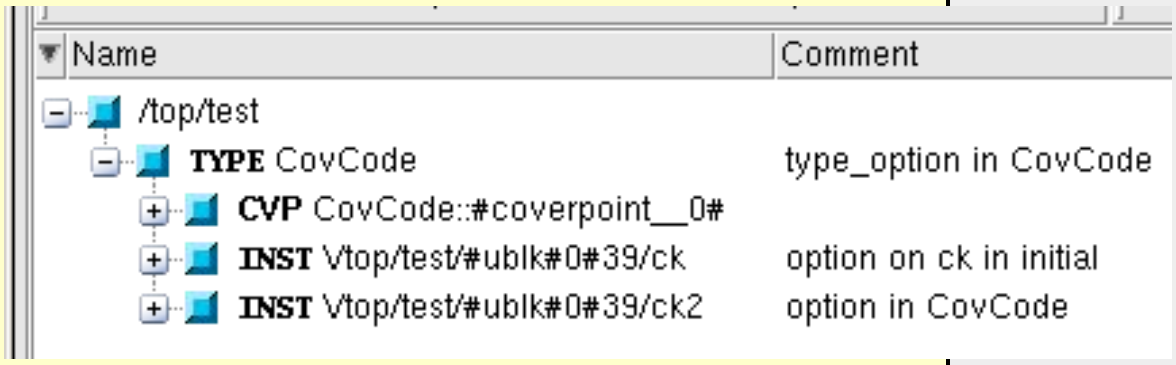
```
covergroup CovPort;
   option.per_instance = 1;
   direction: coverpoint tr.direction;
   port: coverpoint tr.port
   {option.weight = 0;}
   cross direction, port
   {option.weight = 2;}
endgroup
```



type_option vs option

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

```
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
```

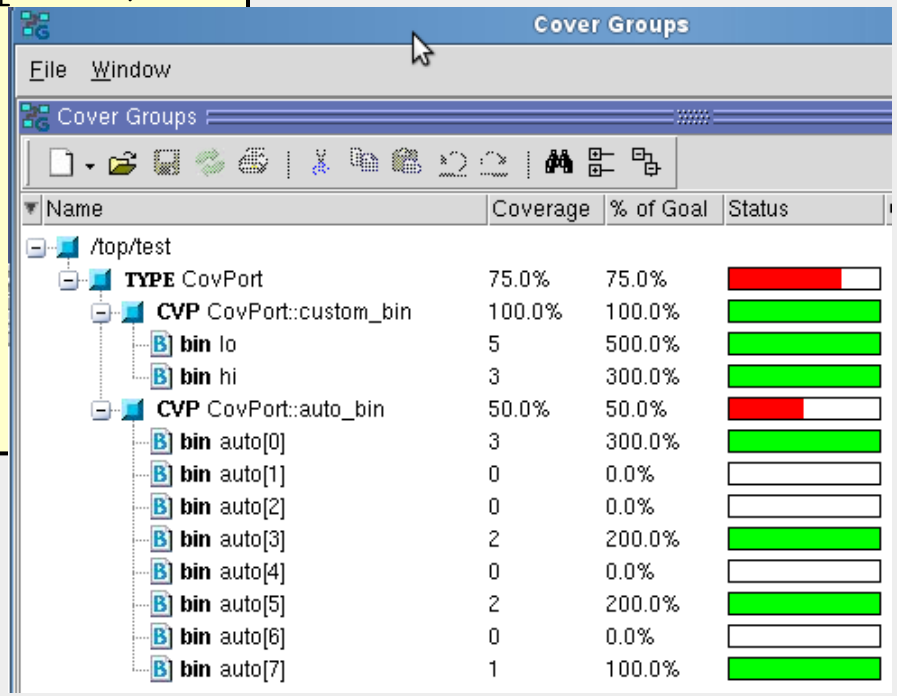


9.9.1 Pass Cover Group Args. by Value

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

```
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
```

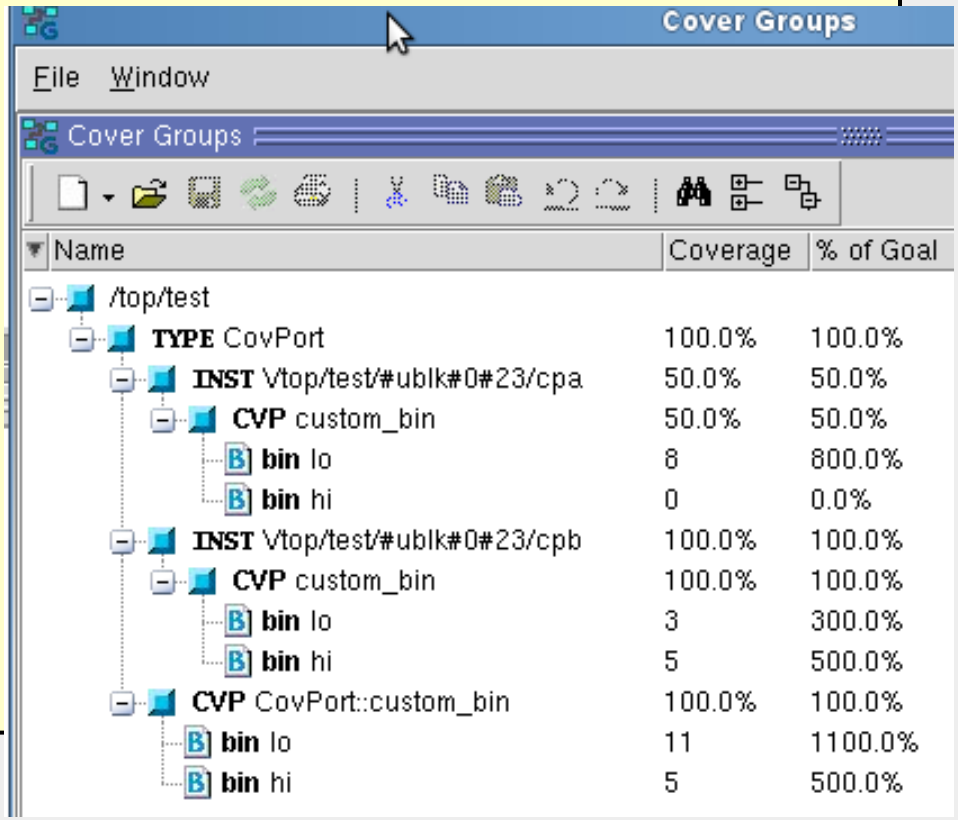


9.9.2 Pass Cover Group Args. by Ref.

To make covergroups even more generic pass args by reference

```
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
```



9.12 Measuring Coverage Statistics During Simulation

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

```
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
```

```
VSIM> vcover report –cvg –details coverage.ucdb –file coverage.rpt
```

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 solution

```
program automatic test(busifc.TB ifc);

  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, but @(posedge clk)
                               //    if the clock is available.
    coverpoint  tr.opcode;
  endgroup
  :
  :
  :
```

Coverage is collected just before events . So for posedge clock the opcode is sampled just before the posedge of clock.

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;
```

Exercise 2 solution

```
program automatic test(busifc.TB ifc);

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;

   coverpoint tr.opcode;

   operand1_cp: coverpoint tr.operand1{
      bins max_neg = {-128};
      bins zero = {0};
      bins max_pos = {127};
      bins misc = default;
   }
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).
3. "Opcode must not equal DIV" (hint: report an error using illegal_bins).

Label the coverpoint `opcode_cp`.

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

Exercise 3 solution

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

Exercise 4

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

Exercise 4 solution

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

```
$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() );
```

Exercise 5

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

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

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

Transaction tr;
```

Exercise 5 solution

```
……
covergroup CovCode @ifc.cb;

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

  operand1_cp: coverpoint tr.operand1{
    bins max_neg = {-128};
    bins zero = {0};
    bins max_pos = {127};
    bins misc = default;}

  opcode_operand1: cross opcode_cp, operand1_cp {
    ignore_bins operand1_zero = binsof(operand1_cp.zero);
    ignore_bins opcode_add_then_sub =
       binsof(opcode_cp.add_then_sub);
    option.weight = 5;}
endgroup
……
```