DAY-83
#100DAYSRTL

“System Verilog:-Functional Coverage”

“Functional Coverage”:-

Functional coverage measures what functionalities/features of the design have been exercised by the tests. This can be useful in constrained random verification (CRV) to know what features have been covered by a set of tests in a regression. Functional coverage is a user-defined metric that measures how much of the design specification has been exercised in verification.

“Coverage Model”:-

- The coverage model is defined using the Covergroup construct.
- The covergroup construct is a user-defined type. The type definition is written once, and multiple instances of that type can be created in different contexts.
- Similar to a class, once defined, a cover group instance can be created via the new()operator. A covergroup can be defined as a module, program, interface, or class.
- Each covergroup specification can include,
- A clocking event that synchronizes the sampling of coverage points
  - ✓ A set of coverage points
  - ✓ Cross coverage between coverage points
  - ✓ Optional formal arguments
  - ✓ Coverage options

```
covergroup cov_grp;
  cov_p1: coverpoint a;
endgroup

cov_grp cov_inst = new();
@(abc) cov_inst.sample();
```

“Bins”:-

- A coverage point can be an integral variable or an integral expression. Each coverage point is associated with a “bin”. On each sample clock simulator will increment the associated bin value.
- The bins will automatically be created or can be explicitly defined.

“Automatic Bins(Implicit Bins)”:-

- An automatically single bin will be created for each value of the coverpoint variable range. These are called automatic, or implicit, bins.
- For an “n” bit integral coverpoint variable, a 2^n number of automatic bins will get created.

```
module cov;
  logic       clk;
  logic [7:0] addr;
  logic       wr_rd;

  covergroup cg @(posedge clk);
    c1: coverpoint addr;
    c2: coverpoint wr_rd;
  endgroup : cg
  cg cover_inst = new();
  ...
endmodule
```

- Below are the bins, will get created automatically,
  - for **addr:** c1.auto[0] c1.auto[1] c1.auto[2] … c1.auto[255]
  - for **wr_rd:** c2.auto[0]

“Explicit bins”:-

- “bins” keyword is used to declare the bins explicitly to a variable.
- A separate bin is created for each value in the given range of variable or a single/multiple bins for the rage of values.
- Bins are explicitly declared within curly braces { } along with the bins keyword followed by bin name and variable value/range, immediately after the coverpoint identifier.

```
module cov;
  logic       clk;
  logic [7:0] addr;
  logic       wr_rd;

  covergroup cg @(posedge clk);
    c1: coverpoint addr { bins b1 = {0,2,7};
                          bins b2[3] = {11:20};
                          bins b3   = {[30:40],[50:60],77};
                          bins b4[] = {[79:99],[110:130],140};
                          bins b5[] = {160,170,180};
                          bins b6    = {200:$};
                          bins b7 = default;}
    c2: coverpoint wr_rd {bins wrrd};
  endgroup : cg

  cg cover_inst = new();
  ...
endmodule
```

Result:-

bins b1 = {0,2,7 }; //bin "b1" increments for addr = 0,2 or 7
bins b2[3] = {11:20};//creates three bins b2[0],b2[1] and b2[3].and The 11 possible values are //distributed as follows: (11,12,13),(14,15,16) and (17,18,19,20) respectively.
bins b3 = {[30:40],[50:60],77}; //bin "b3" increments for addr = 30-40 or 50-60 or 77
bins b4[] = {[79:99],[110:130],140};//creates three bins b4[0],b4[1] and b4[3] with values 79-99,50-60 and 77 respectively
bins b5[] = {160,170,180};//creates three bins b5[0],b5[1] and b5[3] with values 160,170 and 180 respectively
bins b6 = {200:$}; //bin "b6" increments for addr = 200 to max value i.e, 255.
default bin;// catches the values of the coverage point that do not lie within any of the defined bins.

"Bins for Transition":-

- The transition of coverage point can be covered by specifying the sequence,

  value1 => value2

- It represents transition of coverage point value from value1 to value2.
- sequence can be single value or range,

  value1 => value2 => value3 ….
  range_list_1 => range_list_2

```
covergroup cg @(posedge clk);
  c1: coverpoint addr{ bins b1   = (10=>20=>30);
                       bins b2[] = (40=>50),(80=>90=>100=>120);
                       bins b3   = (1,5 => 6, 7);}
  c2: coverpoint wr_rd;
endgroup : cg

bins b1   = (10=>20=>30);                 // transition from 10->20->30
bins b2[] = (40=>50),(80=>90=>100=>120); // b2[0] = 40->50 and b2[1] = 80->90->100->120
bins b3 = (1,5 => 6, 7);}                 // b3 = 1=>6 or 1=>7 or 5=>6 or 5=>7
```

“Ignore Bins”:-

- A set of values or transitions associated with a coverage-point can be explicitly excluded from coverage by specifying them as ignore_bins.

```
covergroup cg @(posedge clk);
  c1: coverpoint addr{ ignore_bins b1 = {6,60,66};
                       ignore_bins b2 = (30=>20=>10); }
endgroup : cg
```

“Illegal Bins”:-

- A set of values or transitions associated with a coverage point can be marked as illegal by specifying them as illegal_bins.

```
covergroup cg @(posedge clk);
  c1: coverpoint addr{ illegal_bins b1 = {7,70,77};
                       ignore_bins b2 = (7=>70=>77);}
endgroup : cg
```