DAY-84
#100DAYSRTL

"System Verilog:-Cross Coverage"

"Cross Coverage":-

- Cross Coverage is specified between the cover points or variables. Cross coverage is specified using the cross construct.
- Expressions cannot be used directly in a cross; a coverage point must be explicitly defined first.
- The cross-coverage allows having a cross-product (i.e. cartesian product) between two or more variables or coverage points within the same covergroup.
- In simple words, cross-coverage is nothing but a set of cross-products of variables or coverage points.

```
bit [3:0] a, b;
covergroup cg @(posedge clk);
  c1: coverpoint a;
  c2: coverpoint b;
  c1Xc2: cross c1,c2;
endgroup : cg
```

Cross coverage by cover_point name

```
bit [3:0] a, b;
covergroup cov @(posedge clk);
  aXb : cross a, b;
endgroup
```

Cross coverage by the variable name

- ✓ In the above example, each coverage point has 16 bins, namely auto[0]…auto[15].
- ✓ The cross of a and b (labeled aXb), therefore, has 256 cross products, and each cross product is a bin of aXb.

```
bit [3:0] a, b, c;
covergroup cov @(posedge clk);
  BC  : coverpoint b+c;
  aXb : cross a, BC;
endgroup
```

Cross coverage between variable and expression

- The coverage group cov has the same number of cross-products as the previous example, but in this case, one of the coverage points is the expression b+c, which is labeled BC.

"Code Practising":-

```
module Coverage;
  reg clk;
  bit [7:0] addr, data;
  bit [3:0] valid;
  bit en;
  covergroup c_group @(posedge clk);
   cp1: coverpoint addr && en; // labeled as cp1
   cp2: coverpoint data;       // labeled as cp2
   cp1_X_cp2: cross cp1, cp2; // cross coverage between two expressions
   valid_X_cp2: cross valid, cp2; // cross coverage between variable and expression
  endgroup : c_group
endmodule
```

"iff Constructs":-

- The expression within the iff construct provides feasibility to exclude the coverpoint in the coverage if an expression is evaluated as false.

"Code Practising":-

```
module func_coverage;
  logic [3:0] addr;
  covergroup c_group;
    cp1: coverpoint addr iff(!reset_n) // coverpoint addr is covered when reset_n is low.
  endgroup
  c_group cg = new();
endmodule
```

“bins of”:-

```
module tb;
bit [7:0] var1, var2;
covergroup c_group @(posedge clk);
  cp1: coverpoint var1 {
                         bins x1 = { [0:99] };
                         bins x2 = { [100:199] };
                         bins x3 = { [200:255] };
                       }
  cp2: coverpoint var2 {
                         bins y1 = { [0:74] };
                         bins y2 = { [75:149] };
                         bins y3 = { [150:255] };
                       }
  cp1_X_cp2: cross cp1, cp2 {
                         bins xy1 = binsof(cp1.x1);
                         bins xy2 = binsof(cp2.y2);
                         bins xy3 = binsof(cp1.x1) && binsof(cp2.y2);
                         bins xy4 = binsof(cp1.x1) || binsof(cp2.y2);
                       }
endgroup
endmodule
```

- ✓ In the above example, coverage points cp1 and cp2 include a set of cover bins for variables var1 and var2. The cp1_X_cp2 denotes the cross product of var1 and var2 with specified cross-bins.
- ✓ If no additional cross bins were specified (i.e. cp1_X_cp2: cross var1, var2), then it would have resulted in 9 cross products listed as:

```
<x1, y1>, <x1, y2>, <x1, y3>,
<x2, y1>, <x2, y2>, <x2, y3>,
<x3, y1>, <x3, y2>, <x3, y3>.
```

- ✓ The cross bin xy1: It results in 3 cross products listed as

```
<x1, y1>, <x1, y2>, <x1, y3>
```

- ✓ The cross bin xy2: It results in 3 cross products listed as

```
<x1, y2>, <x2, y2>, <x3, y2>
```

- ✓ The cross bin xy3: It results in 1 cross-product listed as

```
<x1, y2>
```

- ✓ The cross bin xy4: It results in 5 cross products listed as

```
<x1, y1>, <x1, y2>, <x1, y3>, <x2, y2>, <x3, y2>
```