# CSCE 616 – Hardware Design Verification

# Lab – 9 Coverage

--SAUMIL GOGRI

## Objective – Write and measure Coverage

- Coverage
- Code Coverage
- Functional Coverage
- Covergroup/Coverpoint/Bins
- SV-Syntax

## Coverage

Coverage measures the quality of set of tests.

Coverage gives us the ability to see what has not been covered.

Coverage metrics helps to gauge progress, assess effectiveness, and help determine when the design is robust enough for tapeout

Types of Coverage:

- 1. Code Coverage
- 2. Functional Coverage

## Code Coverage(implicit)

- Code Coverage is the implementation coverage.
- Code Coverage answers the question:

  Is there a piece of code that has not been covered?
- Very useful in initial verification phase, when we run directed/constrained-random testcases.
- It gives insight on what portion of design is not exercised by your current test suite.
- Increases when we run regressions.

Many types of code coverage are there, but the ones required for our lab:

1. Block Coverage
2. Expression Coverage
3. Toggle Coverage

## Functional Coverage(explicit)

- Functional Coverage is the specification coverage.
- Functional Coverage answers the question:
  - Is this feature or functionality covered?

- Verification Engineer defines the goals for functional Coverage.
- It is only as good as your coverage metrics.

  If number of cover groups written are less, and we are getting 100%coverage, it doesn’t mean we have covered all the functions, it simply means those 5 functions represented by 5 cover groups are covered

## Terms Related to Coverage:

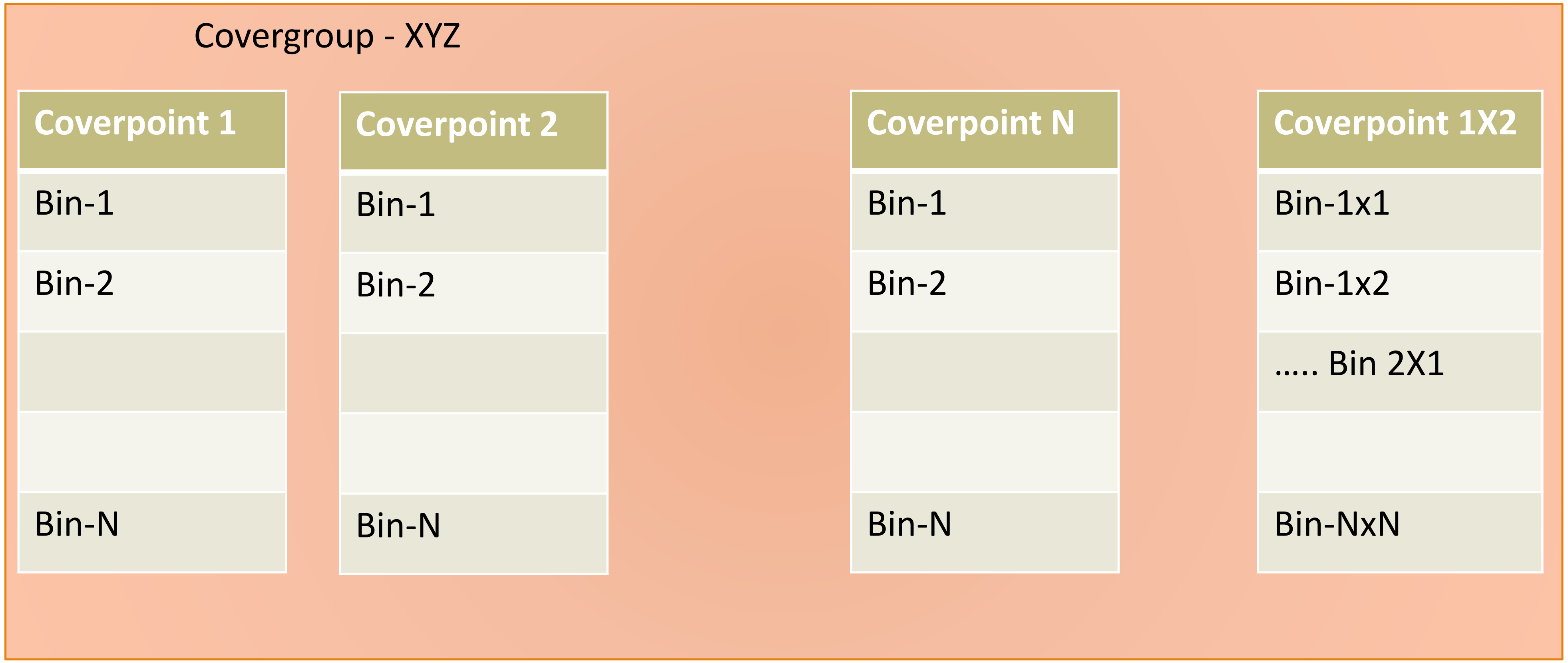
COVER GROUP: Coverage of related attributes is grouped in Cover-Groups.

ITEM: Each attribute is an item.

BUCKET or BIN: Each possible value of attribute.

EVENT: Recording of value of an item occurs when a particular event happens.

# Terms Related to Coverage:



# Types of Coverpoints, Bins

- Basic: Scalar Expression, example: int, byte, enum
- Transition: Transition of one already defined group item.
- Cross: Cross product of 2 or more already defined group items

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

## SV-Syntax

```
logic [7:0] addr;
logic      wr_rd;
covergroup cg;
   c1: coverpoint addr;
   c2: coverpoint wr_rd;

   cross_c1__c2 : cross c1,c2;
  endgroup : cg
  cg cover_inst = new();

  cg.sample();
```

Below bins will be created:

for addr: c1.auto[0] c1.auto[1] c1.auto[2] … c1.auto[255]

for wr_rd:   c2.auto[0], c2.auto[1]

for cross_c1__c2  : we have 256*2 = 512 bins

These are automatically created so it is call automatic/implicit bins

## SV-Syntax

Explicit Bins:

```
c1: coverpoint addr { 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[2].and The 10 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[2] with values 79-99,110-130
                                                                   and 140 respectively

                      bins b5    = {200:$}; //bin “b6” increments for addr = 200 to max value i.e, 255

                      bins b6 = default;} // catches the values of the coverage point that do not lie within any of the defined bins
```

## SV-Syntax

**Ignore Bins:**

```
covergroup cg;
c1: coverpoint addr          { ignore_bins b1 = {6,60,66};
                               ignore_bins b2 = {[0:2]}; }
```

**Illegal Bins:**

```
covergroup cg;
c1: coverpoint addr          { illegal_bins b1 = {6,60,66};
                               illegal_bins b2 = {[0:2]}; }
```

## Thank you