ENCS5337: Chip Design Verification

Spring 2023/2024

Introduction to Coverage in SystemVerilog

Dr. Ayman Hroub

Outline

- Covergroups
- Coverpoints
- Bins
- Cross Coverage

Covergroup

- A Cover Group is a user-defined type specifying a coverage model
- The coverage model can be defined using the **`covergroup`** keyword
- It can be declared in a package/interface/module/program/class.
- A cover group contains:
  - A sampling event whose coverage can be manually sampled.
  - A **`coverpoint`** for each variable whose value must be sampled for coverage.

Covergroup Example

- You can define a variable of the **`covergroup`** name.
- You can instantiate the coverage model using **`new`**

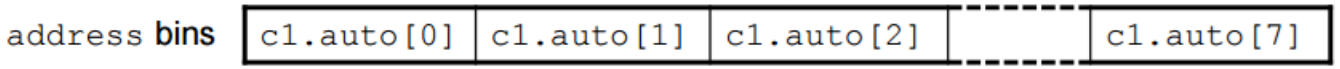
```
module example;
  logic clk;
  logic [2:0] address;
  logic [7:0] data;

  covergroup cg1 @(posedge clk);
    c1: coverpoint address;
    c2: coverpoint data;
  endgroup : cg1

  cg1 cover_inst = new();
  ...
endmodule
```

Bins

- Each **`coverpoint`** in a **`covergroup`** creates a series of counters, called bins, which are stored in a coverage database.
- When coverage is sampled, the bin corresponding to the variable value is incremented.
- By default, a **`coverpoint`** creates a single bin for every value in the variable range
- These are called automatic, or implicit, bins and are named using the identifier **`auto`** and the value for the bin.

Automatically Created Cover Point Bins Example

```
module example;
  logic clk;
  logic [2:0] address;
  logic [7:0] data;

  covergroup cg1 @(posedge clk);
    c1: coverpoint address;
    c2: coverpoint data;
  endgroup : cg1

  cg1 cover_inst = new();
  ...
endmodule
```



Explicit Bins

- You can define the bins explicitly:
  - To track only a subset of values.
  - To control what values increment which bin.

- Use the **`bins`** keyword, and provide a:
  - Bin name.
  - List of values or value ranges.
- With explicit bins, unlisted values are not tracked.
- Each **`coverpoint`** can have multiple bins clauses.

Explicit Bins Example

```
c1: coverpoint var1 {
  bins V = {1, 2, 5} ;
  }
```

No semicolon at end of coverpoint with explicit bins

Bin `V` increments for `var1` = 1, 2 or 5

Value list examples

```
{ [0:5], 10 }      - values 0-5 and 10
{ [0:5], [9:14] }  - values 0-5 and 9-14
{ 'h1, 'h2, 'hF }  - values 1, 2, 15
{ [1:9], [7:12] }  - range overlap allowed
{ [16:$] }         - range 16 to max value
```

Explicit Scalar and Vector Bins

- Scalar bin
  - A single bin that counts occurrences of any of the values in its list
  - Incremented for all values in value range list.
- Vector bin (array of bins)
  - A unique bin for each value in the range list.
  - Incremented when variable takes the corresponding value.
- You can mix scalar and vector bins for the same coverpoint.

Explicit Scalar and Vector Bins Example

Scalar example

```
cs: coverpoint var1 {
  // bin V increments for 1, 2 or 5
  bins V = {1, 2, 5} ;
}
```

Creates a single bin `cs.V`

Vector example

```
cv: coverpoint var1 {
  // bins V[1], V[2] and V[5]
  bins V[] = {1, 2, 5} ;
 }
```

`[]` for 3 values creates 3 bins

`cv.V[1], cv.V[2], cv.V[5]`

More on Explicit Scalar and Vector Bins

- **`illegal_bins`** specifies a bin for illegal values of the coverpoint variable. These values are automatically excluded from all other bins clauses in the same coverpoint, even if explicitly listed.
- The simulator issues an error when a sampled variable has an illegal value.

- **`ignore_bins`** specifies bins for ignored values. Once a value is defined in the list for ignored bins, it is automatically excluded from all other bins clauses in the same coverpoint,

- The **`default`** keyword specifies bins for values that do not appear in any other bins.

Vector Bins Size

- For a vector bin of unconstrained size
  - Each unique value in the list has its own bin.
  - Duplicate values are not retained.
  - Illegal and ignored values are removed after the values are distributed.

- For a vector bin of a specified size
  - The values are distributed as evenly as possible by order of appearance in the list, with the later bins getting any extra values.
  - Duplicate values are retained, so can show up in multiple bins.
  - Illegal and ignored values are removed after the values are distributed.

`illegal_bins` and `ignore_bins` Example

```
logic [3:0] var1;

ce: coverpoint var1 {
  // 1 bin for illegal values {0, 15}
  illegal_bins a = { 0, 15 };
  // 1 bin for ignored values {13, 14}
  // (value 15 is illegal)
  ignore_bins  b = { [13:15] };
  // 1 bin for {2, 3}
  bins c     = { 2, 3 };
  // 3 bins e[1], e[2], e[6]
  bins e[]   = { [0:2], 2, 6 };
  // 2 bins - d[0] = {9,10,11,9}
  //        - d[1] = {12}
  bins d[2] = { [9:11], 9, [12:15] };
  // 1 bin for {4,5,7,8},
  bins f     = default;
}
```

Covergroup Coverage – How Many Bins?

```
typedef enum bit[2:0] {ADDI, SUBI, ANDI, XORI, JMP, JMPC, CALL} op_t;
typedef enum bit[1:0] {REG0, REG1, REG2, REG3} regs_t;

op_t       opc;
regs_t     regs;
logic[7:0] data;

covergroup cg @(posedge clk);
  c1: coverpoint opc { bins op[] = { [ADDI:$] }; }
  c2: coverpoint regs;
  c3: coverpoint data { bins low[] = { [0:'h0F] } ;
                        bins high  = { ['h10:'hFF] } ;
                      }
endgroup

cg cg1 = new();
```

c1: 7 explicit vectored bins named c1.op[ADDI] to c1.op[CALL]

c2: 4 automatic vectored bins named c2.auto[REG0] to c2.auto[REG3]

c3: 16 explicit vectored bins named c3.low[0] to c3.low[15] and 1 explicit scalar bin named c3.high

What Is a Cover Cross?

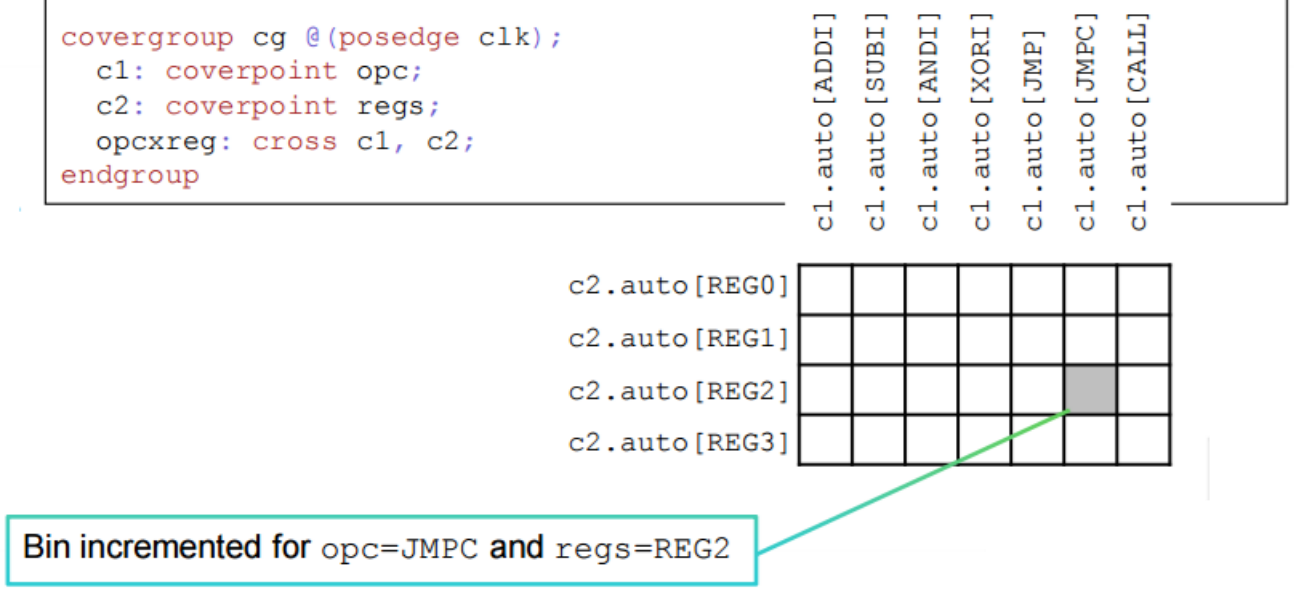
- A Cover Cross is a user-defined item in a coverage model specifying a cross-product of cover points to cover and optionally a condition guarding its sampling.

- You can track cross-products of:
  - Coverpoints within the covergroup.
  - Other scope variables

- Use the **`cross`** keyword and provide a:
  - Cross name (optional).
  - List of coverpoints and/or variables.
- Bins are created for every cross combination.

Cover Cross Example

```
typedef enum bit[2:0] {ADDI, SUBI, ANDI, XORI, JMP, JMPC, CALL} op_t;
typedef enum bit[1:0] {REG0, REG1, REG2, REG3} regs_t;
op_t       opc;
regs_t     regs;

covergroup cg @(posedge clk);
  c1: coverpoint opc;
  c2: coverpoint regs;
  opcxreg: cross c1, c2;
endgroup
```



Cover Cross Bins

- A Coverage Bin is a tool-defined or user-defined counter associated with a cover point value set, a cover point value transition set, or a cover cross tuple set.

- For cross-products the tool automatically creates a unique bin for each tuple.

Cover Cross Bins Example

```
logic [1:0] a;
logic [3:0] b;
logic c;
covergroup cg @(posedge clk);
  bcp: coverpoint b {
     bins b1    = { [9:12] };     //one bin b1
     bins b2[]  = { [13:15] };    //3 bins: b2[13], b2[14], b2[15]
     bins restofb[] = default;    //9 bins: [0] ... [8] not in cross
    }
  ccp: coverpoint c;              // two automatic bins
  axbxc: cross a, bcp, ccp;       // 32 bins = a(4) x bcp(4) x ccp(2)
endgroup : cg
```

Implicit coverpoint for `a`

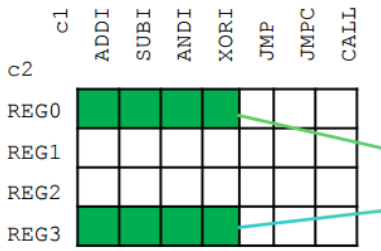
```
axbxc.<a.auto[0], b.b1, c.auto[0]>
axbxc.<a.auto[0], b.b1, c.auto[1]>
axbxc.<a.auto[0], b.b2[13], c.auto[0]>
axbxc.<a.auto[0], b.b2[13], c.auto[1]>
axbxc.<a.auto[0], b.b2[14], c.auto[0]>
....
```

Crosses created

Explicit Cross Bin and Select Expressions

- You can create explicit cross coverage bins:
  - **`binsof`** selects specific bins from a coverpoint.
  - **`intersect`** filters bin selection to specified value ranges.
- You can use **`!, &&, ||`** on resulting selections.
- SystemVerilog by default automatically creates a single bin for every product of the cover cross.
- It does not include any coverpoint bins declared with the **`default`** range or any declared with **`illegal_bins`** or **`ignore_bins`**.

Explicit Cross Bin and Select Expressions Example




```
typedef enum bit[2:0] {ADDI, SUBI, ANDI,
                       XORI, JMP, JMPC,
                       CALL} op_t;
typedef enum bit[1:0] {REG0, REG1,
                       REG2, REG3} regs_t;
op_t       opc;
regs_t     regs;

covergroup cg @(posedge clk);
  c1: coverpoint opc ;
  c2: coverpoint regs ;
  opxr : cross c1, c2 {
    bins x1 =
      binsof(c1) intersect {[ADDI:XORI]} &&
      binsof(c2) intersect {REG0, REG3};
                        }
endgroup
```

x1 – track logical opcodes (ADDI–XORI) on REG0 and REG3

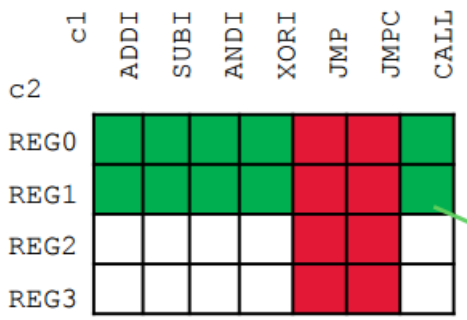
Defining Cover Cross with Illegal and Ignored Cross Bins

- Use **`ignore_bins`** to exclude bins from a cross:
  - Even if selected elsewhere in the same cross.

- Use **`illegal_bins`** to specify illegal bins:
  - Even if selected or excluded elsewhere in same cross.

Cover Cross with Illegal and Ignored Cross Bins Example

```
typedef enum bit[2:0] {ADDI, SUBI, ANDI,
                       XORI, JMP, JMPC,
                       CALL} op_t;
typedef enum bit[1:0] {REG0, REG1,
                       REG2, REG3} regs_t;
op_t       opc;
regs_t     regs;

covergroup cg @(posedge clk);
  c1: coverpoint opc ;
  c2: coverpoint regs ;
  opxr : cross c1, c2 {
    bins x3 = ! binsof(c2) intersect {[REG2:REG3]};
    ignore_bins x4 = binsof(c1) intersect {[JMP:JMPC]};
                    }
endgroup
```



x3 – select all bins not of REG2 or REG3, but ignore all JMP or JMPC bins (x4)

Defining Easier Cover Cross Bins

- Complex cross expressions can be avoided using dedicated coverpoints:
  - Define multiple coverpoints for required values or ranges.
  - Cross coverpoints directly

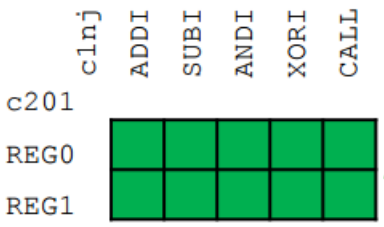
- You can have as many coverpoints as you wish on the same variable, as long as each is uniquely named.

Defining Easier Cover Cross Bins Example

```
// declarations as before
op_t        opc;
regs_t      regs;

covergroup cg @(posedge clk);
  c1   : coverpoint opc;
  c1nj: coverpoint opc {
    bins opnj={[ADDI:XORI],CALL};}

  c2   : coverpoint regs;
  c201: coverpoint regs {
    bins rg01={REG0,REG1};}

  opxr : cross c1nj, c201;

endgroup
```

Select all opcodes except JMP & JMPC

Select REG0 & REG1



Select all bins not of REG2 or REG3, but ignore all JMP or JMPC bins

One bin – opxr.<opnj,rg01>

How to Use Cover Groups in Classes (1)

- To define the cover group in the class definition is an intuitive way to define the coverage model for the class.
- Each instance of the class will track coverage separately.
- Define the class member cover group instance.
- The declaration creates an instance variable of an anonymous cover group type.
  - You cannot create another instance of that type

How to Use Cover Groups in Classes (2)

- Construct the cover group instance
- Define cover group sampling
  - For a design or test component class object, define a sampling event.
  - For a data class object, call the cover group method **`sample()`**

How to Use Cover Groups in Classes: Example

```
class covclass;

  logic [2:0] address;
  logic [7:0] data;

  covergroup cg1;
    c1: coverpoint address;
    c2: coverpoint data;
  endgroup : cg1

  function new();
    cg1 = new();
  endfunction
endclass

covclass one = new();
initial begin
  ...
  one.cg1.sample();
```

A class covergroup instance is not separately named

Defining a Cover Point Bin for Value Transitions

- Coverpoints can also track transitions:
  - Single value change
  - Sequence of values
  - Multiple changes between arrays of values

```
typedef enum bit[2:0] {ADDI, SUBI, ANDI, XORI, JMP, JMPC, CALL} op_t;
op_t       opc;

covergroup cg;
  c1: coverpoint opc {bins adsu=(ADDI => SUBI);
                      bins suan=(ADDI => SUBI => ANDI);
                      bins su3= (ADDI,SUBI => ANDI); }
endgroup
```

1 transition

1 sequence transition

(ADDI => ANDI), (SUBI => ANDI)

```
(ADDI => SUBI)        : 1 transition
(JMP => JMPC => CALL): sequence
(SUBI => ANDI,XORI)   : 2 transitions
                      : (SUBI => ANDI)
                      : (SUBI => XORI)
(ADDI[*3]) : consecutive repetition
             (ADDI => ADDI => ADDI)
```