Exercises

1. For the class 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;
```

2. Expand the solution to Exercise 1 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`.
3. Expand the solution to Exercise 2 to cover the following test plan requirements:
   a. "The opcode shall take on the values ADD or SUB" (hint: this is 1 coverage bin).
   b. "The opcode shall take on the values ADD followed by SUB" (hint: this is a second coverage bin).
   Label the coverpoint `opcode_cp`.
4. Expand the solution to Exercise 3 to cover the test plan requirement, "Opcode must not equal DIV" (hint: report an error using `illegal_bins`).
5. Expand the solution to Exercise 4 to 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." Weight the cross coverage by 5.

Code

link in eda playground : https://edaplayground.com/x/bP6h

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

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

    // Covergroup defined to sample on posedge clk
    covergroup trans_cg @(posedge clk);
        option.per_instance = 1;
        opcode_cp   : coverpoint opcode
        {
            bins allOps[] = {ADD , SUB , MULT , DIV};
            bins add_or_sub = {ADD, SUB};
            bins add_then_sub = (ADD => SUB);
            illegal_bins div_is_illegal = {DIV}; // this will cancel bin[DIV]
in allOps[] bins
            /*
                i.e opcode_cp will be equivalent to this :
                opcode_cp   : coverpoint opcode
                {
                    bins allOps[] = {ADD , SUB , MULT};
                    bins add_or_sub = {ADD, SUB};
                    bins add_then_sub = (ADD => SUB);
                }
            */
        }
        operand1_cp : coverpoint operand1
        {
            bins min = {-128};
            bins zero = {0};
            bins max = {127};
            bins other = default;
        }
        opcode_x_operand1: cross opcode_cp , operand1_cp
        {
            bins add_or_sub_with_extremes1 =
            binsof(opcode_cp) intersect {ADD , SUB} &&
            binsof(operand1_cp) intersect {-128,127} ;

            bins add_or_sub_with_extremes2 =
            binsof(opcode_cp.add_or_sub) &&
            (binsof(operand1_cp.max) || binsof(operand1_cp.min));
        }

    endgroup
```

```
    function new();
        trans_cg = new();
    endfunction
endclass

program test;
    bit Running;
    Transaction t;

    // Clock process
    initial begin
        Running = 1;
        while (Running) begin
            #5 clk <= ~clk;
        end
        $display("Finished!!");
    end

    // Test and coverage logic
    initial begin
        t = new();
        while (t.trans_cg.get_coverage() < 100) begin
            @(posedge clk);
            void'(t.randomize());
        end
        Running = 0;
    end

    // Final coverage report
    final begin
        $display("Overall coverage      = %0.2f%%", $get_coverage());
        $display("Transaction coverage  = %0.2f%%",
t.trans_cg.get_coverage());
    end
endprogram
```