Module: SV for Verification
Section: Randomization **Task:** Randomization of Variable

Task
Randomization of Variable

➢ **Why do we use randomized stimulus to verify the design?**
As designs become more complex, it becomes increasingly challenging to create a complete set of test stimuli to verify their functionality. While you can write directed test cases to check specific features, it's nearly impossible to cover every scenario as the number of features grows. Moreover, the most challenging to find bugs often arise from interactions between features, which are difficult to catch with a checklist approach.

To address this, constrained-random testing (CRT) is used. Unlike directed tests that target known issues, CRT generates random stimuli to uncover unexpected bugs. By applying constraints, you ensure that the random tests are both valid and relevant.

Although setting up a CRT environment requires more effort than creating directed tests, once established, it allows you to run numerous tests automatically and efficiently. This approach significantly boosts productivity by shifting the workload from manual test creation and checking to automated.

➢ **Why you should not randomize an object in the class constructor.**
The constructor in SystemVerilog is primarily used to initialize an object's variables to a known state, ensuring predictable and consistent behavior. Randomizing an object in the constructor is not recommended because it limits flexibility and control in your test environment.

Tests often need to adjust constraints, enable or disable specific conditions, or modify randomization weights before the object is randomized. If randomization occurs too early, you might have to discard and redo the process, leading to inefficiencies and potential issues. By keeping randomization separate from the constructor, you can better manage when and how an object is randomized, allowing for more targeted and effective testing.

➢ **How can you disable randomization on a variable?**
The **rand_mode()** method in SystemVerilog is used to control whether a random variable is enabled or disabled during randomization. When a random variable is set to inactive, it behaves as if it was never declared as **rand** or **randc**—meaning it won't be affected by the **randomize()** method. Instead, its value is treated as a fixed state by the solver during randomization.

By default, all random variables are active when they are first declared.

➢ **Can randomization fail in your code? Give at least 2 scenarios with code and logical explanation.**

Yes, randomization can fail if your code has conflicting or out-of-range constraints, so we should always check the status. If not checked, the variables may get unexpected values, causing the simulation to fail.

1. **Conflicting Constraints:**
   If the constraints applied to random variables are conflicting or impossible to satisfy, the randomization will fail.
   - **Code Snippet:**

```
class Packet;
  rand bit [7:0] data;
  rand bit [3:0] addr;

  constraint valid_data {
    data > 100;
    data < 50; // Conflicting constraint
  }
endclass

module test;
  initial begin
    Packet pkt = new();
    if (!pkt.randomize()) begin
      $display("Randomization failed due to conflicting constraints!");
    end else begin
      $display("Data: %0h, Addr: %0h", pkt.data, pkt.addr);
    end
  end
endmodule
```

2. **Out-of-Range Constraints:**
   If the constraints applied to random variables are conflicting or impossible to satisfy, the randomization will fail.
   - **Code Snippet:**

```
class Packet;
  rand bit [3:0] addr;

  constraint valid_addr {
    addr inside {8'd16, 8'd32}; // Values outside the 4-bit range
  }
endclass

module test;
  initial begin
    Packet pkt = new();
    if (!pkt.randomize()) begin
      $display("Randomization failed due to out-of-range constraints!");
    end else begin
      $display("Addr: %0h", pkt.addr);
    end
```

```
  end
endmodule
```

➢ **What is a “randomize ()” method and what does it do? Is it used as a function or as a task?**
randomize( ) is a method which is used as a function that assigns random values to any variable in the class that has been labeled **rand** or **randc,** and also makes sure that all active constraints are obeyed.

The **randomize( )** method is a function, not a task. This means it executes and returns a result in the same simulation time unit. Specifically, it returns a bit value: 1 if the randomization is successful and 0 if it fails.