Module: SV for Verification
Section: Writing a Complete Testbench **Task:** Clocking Block

Task 1

Clocking and Program Blocks

➢ **What is the issue with the below code and how can that issue be tackled?**

```
bit shot_put;
bit javelin;
initial begin
  @(posedge clk);
  fork
    shot_put = $random();
    javelin = $random();
  join_none
  @(shot_put or javelin)
    throw = (shot_put || javelin);
end
```

The issue with the initial code lies in the unsynchronized execution of the **fork/join_none** block and the event control **@(shot_put or javelin)**. The **fork/join_none** allows the random assignments to happen in parallel, but the event control could trigger prematurely, evaluating **throw** before the random values are assigned, resulting in an incorrect outcome based on the initial **0** values.

A better approach is to use **fork/join** or an explicit **wait** to ensure the random assignments complete before evaluating **throw**. This guarantees synchronization, as the **join** ensures both random assignments finish before moving on, and **wait** ensures the variables get non-zero values before assigning to **throw**.

By addressing this, we eliminate ambiguity, ensuring correct execution.

- **Fixed Code:**

```
bit shot_put;

bit javelin;

bit throw;

initial begin

@ (posedge clk);
```

```
fork

shot_put = $urandom_range(1);

javelin = $urandom_range(1);

join // Ensures that both threads have finished before continuing further

@ (shot_put or javelin);      // Wait for change

throw = (shot_put || javelin);

end
```

➢ **What is the issue with the below code and how can that issue be tackled?**

```
event wicket, batsman;
initial begin
  @(posedge clk);
  fork
    forever begin
      @batsman;
      repeat (cricket+1) @(posedge clk);
      ->wicket;
    end
    forever begin
      ->batsman;
      @wicket;
      cricket += 1;
    end
  join_none
end
```

In this code, there is a risk of deadlock due to the use of blocking event triggers (**->**). This happens when one **forever** block triggers an event before the other block has a chance to process it, causing a situation where the first block waits indefinitely for an event that has already occurred.

To address this, we use non-blocking event triggers (**->>**). Non-blocking triggers ensure that events are handled correctly, preventing one block from missing events triggered by the other. This adjustment avoids the deadlock issue and ensures proper synchronization between the two **forever** blocks.

- **Fixed Code:**

```
event wicket, batsman;

initial begin

    @ (posedge clk);

    fork

        forever begin

            @ (batsman);  // Wait for the batsman event

            repeat (cricket + 1) @ (posedge clk);

            ->>wicket;    // Non-blocking trigger for the wicket event

        end


        forever begin

            ->>batsman;   // Non-blocking trigger for the batsman event

            @ (wicket);   // Wait for the wicket event

            cricket += 1;

        end

    join_none

end
```