Module: SV for Verification
Section: Time Regions **Task:** Simulation Cycles

Task
Time Regions

---

1) **Explain the difference between "blocking assignment" and "non-blocking assignment" in SystemVerilog. With example code block which will evaluate the differences between both assignments:**

In SystemVerilog, blocking assignments and non-blocking assignments are used to assign values to variables, but they behave differently in terms of execution order and timing.

Name	**Blocking Assignments**	**Non-Blocking Assignments**
Execution	Executes sequentially within a procedural block	Executes concurrently, allowing multiple assignments to occur simultaneously.
Usage	Typically used for combinational logic.	Typically used for sequential logic.
Syntax	= operator	<= operator

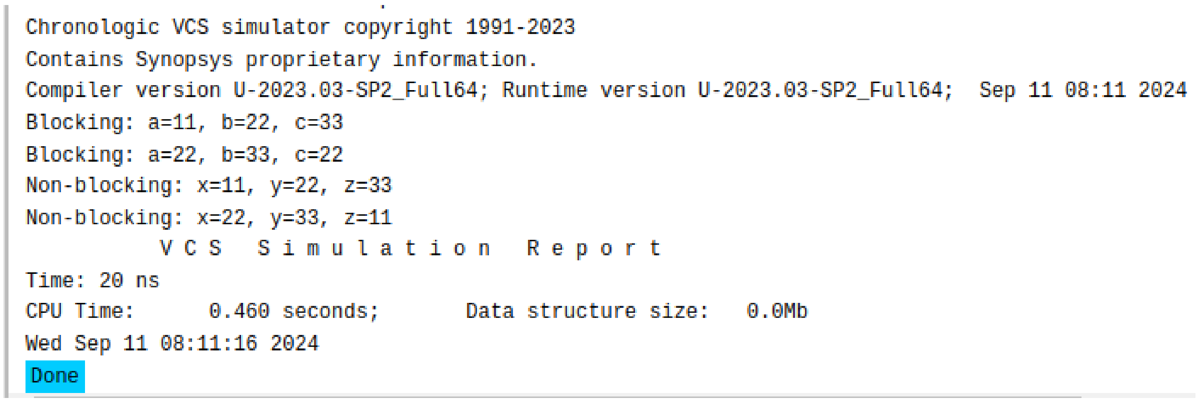
➢ **Code:**

```
// Author: Noman Rafiq
// Dated: Sep 11, 2024
module blocking_vs_nonblocking;
  reg [7:0] a, b, c;
  reg [7:0] x, y, z;

  // Blocking assignment
  initial begin
    a = 8'h11;
    b = 8'h22;
    c = 8'h33;
    $display("Before Blocking: a=%h, b=%h, c=%h", a, b, c);
    a = b;
    b = c;
    c = a;
    $display("After Blocking: a=%h, b=%h, c=%h", a, b, c);
  end

  // Non-blocking assignment
  initial begin
    x = 8'h11;
    y = 8'h22;
    z = 8'h33;
    $display("Before Non-blocking: x=%h, y=%h, z=%h", x, y, z);
    x <= y;
    y <= z;
    z <= x;
    #20;
    $display("After Non-blocking: x=%h, y=%h, z=%h", x, y, z);
  end
endmodule
```

➢ **Output:**



```
Chronologic VCS simulator copyright 1991-2023
Contains Synopsys proprietary information.
Compiler version U-2023.03-SP2_Full64; Runtime version U-2023.03-SP2_Full64;  Sep 11 08:11 2024
Blocking: a=11, b=22, c=33
Blocking: a=22, b=33, c=22
Non-blocking: x=11, y=22, z=33
Non-blocking: x=22, y=33, z=11
           V C S   S i m u l a t i o n   R e p o r t
Time: 20 ns
CPU Time:      0.460 seconds;       Data structure size:   0.0Mb
Wed Sep 11 08:11:16 2024
Done
```

➢ **Explanation:**

- **Blocking Assignment:**
  - The assignments **a = b**, **b = c**, and **c = a** are executed sequentially. The value of **a** is updated first, then **b**, and finally **c**.
- **Non-blocking Assignment:**
  - The assignments **x <= y**, **y <= z**, and **z <= x** are scheduled to occur simultaneously at the end of the time step.

2) How do program and scheduling semantics affect the execution of code?

In SystemVerilog, program and scheduling semantics impact how code is being executed, especially in simulations that involve both testbenches and design components. Here's how these concepts affect execution:

- **Program Blocks:**

    - **Testbench Code:** A program block is typically used for writing testbenches. It ensures that the testbench code runs in a controlled order relative to the DUT.
    - **Execution Order:** Program blocks execute after the RTL (register-transfer level) code, ensuring that the testbench cannot accidentally interfere with the RTL's internal processes.

- **Time Slot Scheduling:**

  Simulation Time is divided into time slots, and events within a time slot are scheduled into 9 major regions as given below:

        - **Preponed**
        - **Active Region**
        - **Inactive Region**
        - **NBA (Non-blocking Assignment) Region**
        - **Observe Region**
        - **Reactive Region**
        - **Re-Inactive Region**
        - **Re-NBA Region**
        - **Postponed Region**

- **Ordering of Events:**
    - Program blocks typically execute in the Reactive region, meaning they respond to the outputs of the DUT after the RTL execution has occurred.
    - The NBA region is often revisited if non-blocking assignments in the DUT trigger further events. This can lead to re-evaluation of logic before the simulation advances to the next time step.
    - Assertions are evaluated during the Observe region, after the values are updated but before time advances, ensuring that the design is checked based on the updated values.

- **Impact on Execution:**
    - The program scheduling semantics ensure that the testbench can monitor and interact with the design at precise points, without disturbing its internal operation.
    - By dividing execution into different scheduling regions, SystemVerilog ensures predictable behavior between the testbench and DUT, critical for reliable simulation and verification.

3) **How does simulation relate to simulation time and time slots?**
In hardware simulation, simulation time models the passage of time in a digital circuit, progressing in discrete time steps, each representing a unit like a nanosecond. Events within each time step are processed in a series of time slots, ensuring correct event execution order and preventing race conditions.

   01. Time Slot Regions:

   i. **Preponed Region:** This is the first region in a time slot, used to sample signals before any other activity. Variables used in assertions are sampled here for evaluation later in the same time slot.
   ii. **Active Region:** Executes all active events like blocking assignments (=) and combinational logic updates. This is where most RTL logic gets executed and updated.
   iii. **Inactive Region:** Executes events that are delayed until after the active region in the same time slot, but still within the current simulation step. Non-critical updates or **#0** initialized variables often reside here.
   iv. **NBA (Non-blocking Assignment) Region:** Handles non-blocking assignments (<=), where the updates are postponed until the active region completes, allowing simultaneous evaluation of combinational logic.
   v. **Observe Region:** Evaluates assertions and monitors, which ensures they operate on the final values of the time slot. No further signal updates occur beyond this point.
   vi. **Reactive Region:** Processes events such as testbench stimulus and driver code, often including input and output activities that react to the updated signals from the NBA region.
   vii. **Re-Inactive Region:** Similar to the inactive region, but after the reactive region. It's used for additional events that need to be executed before moving to the next time slot.
   viii. **Re-NBA (Re-Non-blocking Assignment) Region:** Executes any leftover non-blocking assignments that need to be processed after the reactive region, before the simulation time advances.
   ix. **Postponed Region:** This is the final region, used mainly for event logging ($display, $monitor) and verification tasks that should reflect the final state of all signals in the current time slot.

Simulation advances to the next time slot only after all events in the current time slot are processed. This structure ensures correct sequencing of logic updates, testbench interactions, and assertions within a single time slot.

4) Explain how clocking block cycles work. What happens during each stage of a clocking block cycle?

A **clocking block cycle** in SystemVerilog manages the sampling and driving of signals in synchronization with a clock. Each cycle begins with a **clock event** (e.g., the positive edge of a clock), during which inputs are sampled and outputs are driven at specific stages of simulation.

1. **Before the clock edge**, signals listed as **inputs** in the clocking block are sampled in the **Preponed region** to ensure stable inputs.
2. The **Active and NBA (Non-blocking Assignment) regions** then handle the design's logic evaluation based on these sampled inputs.
3. **After the clock edge**, signals listed as **outputs** are driven to the design in the **Reactive region**, ensuring proper synchronization and avoiding race conditions.

When signals are referenced with the clocking block prefix, they adhere to the clocking block's timing. Without the prefix, the original signals are referenced, bypassing this timing control. This distinction ensures synchronization with the clock for inputs and outputs.

In summary, clocking blocks control signal sampling and driving with precision, allowing for clean interaction between the testbench and DUT.

5) What is a clocking block and what are the benefits of using clocking blocks inside an interface?

A **clocking block** in SystemVerilog is used to synchronize signal sampling and driving with a clock, ensuring consistent timing behavior between the testbench and the DUT. Clocking blocks define when signals are sampled before a clock edge (inputs) and driven after the edge (outputs), reducing race conditions.

When used inside **interfaces**, clocking blocks enhance modularity and clarity by centralizing signal timing management. Each **modport** in an interface can have its own clocking block, defining the direction and timing behavior of signals, which allows for both synchronous (clocked) and asynchronous signals to be handled cleanly. This structure abstracts timing concerns and reduces errors, ensuring proper synchronization of signals and improving overall verification quality.

By using clocking blocks with interfaces and modports, engineers ensure proper signal timing across different parts of a design, making testbenches easier to manage and less prone to timing issues.

6) **What is the difference between the following two ways of specifying skews in a clocking block?**
    a) input #1step req1;
    b) input #1ns req1;

The difference between **input #1step req1;** and **input #1ns req1;** relates to the delay before sampling the input signal in a clocking block. **#1step** introduces a delay equal to the smallest unit of time the simulator can manage, known as the global time precision, which is often extremely small and is used to ensure the signal is sampled as close to the clock event as possible.

This is particularly useful in verification, where precise timing is critical, such as in assertions. **#1ns**, on the other hand, introduces a fixed 1-nanosecond delay, which is much larger and may be used when a specific delay is required before sampling.

In summary, **#1step** offers ultra-fine, simulation-precision control over timing, while **#1ns** introduces a larger, fixed delay for scenarios requiring explicit timing offsets. Both serve different purposes depending on the timing requirements of the verification or design task.