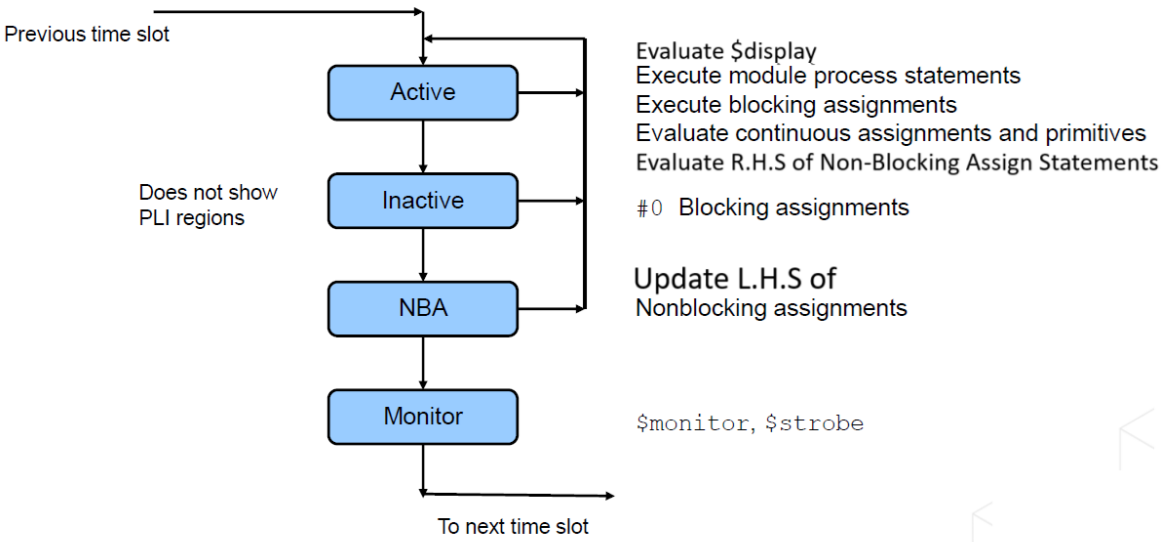
SystemVerilog Scheduler

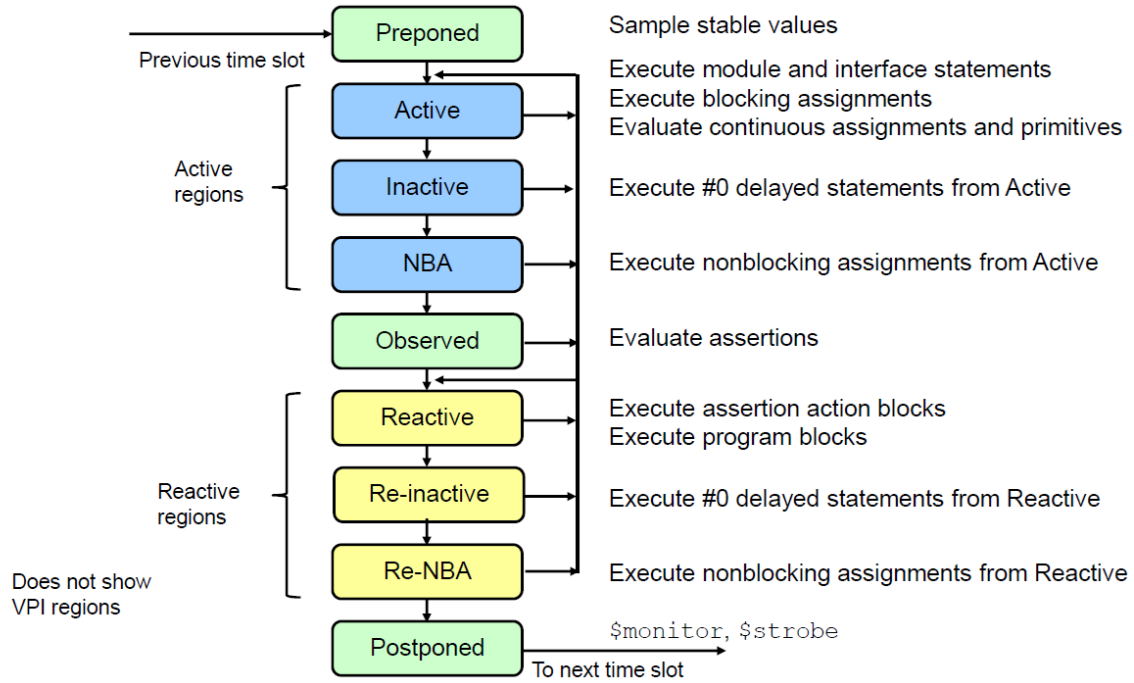
Youssef Nasser

بِسْمِ اللَّـهِ الرَّحْمَـٰنِ الرَّحِيمِ

Verilog Schedular



Remember Verilog is Part of SystemVerilog , so all regions will be copied their but adding new regions to the SV.



SystemVerilog Scheduler

Youssef Nasser

How Are Assertions Evaluated in a Simulation Time Tick?

Active Region

In this region, the assertion "clock ticks" are detected, and assertions are scheduled.
(not executed) in the Observed region. Events originating in Observed region get
scheduled into the Active region (or Reactive Region) for execution.

Observed Region

The Observed region is meant for the evaluation of sequences, properties, and concurrent, not immediate.
assertions. Signal values remain constant during the Observed region.
Events originated in this region get scheduled into the Active and Reactive regions.
In general, here's what the Observed Region does.

- **Determine match of sequences.**
- **Start new attempts for sequences.**
- **Start new attempts for assertions.**
- **Resume evaluation of previous attempts.**
- **Schedule action blocks (*not* execute them – that's done in Reactive region).**

Reactive Region

The Reactive region executes statements from programs and checkers and action
blocks. Programs are intended for writing test-benches, as external environments
for designs, feeding stimuli, observing design evaluation results, and building tests
to exercise the design.

Preponed Region

This region is of importance in terms of understanding how the so-called sampling semantics of assertions work. How does the so-called *sampling edge* sample the variables in a property or a sequence is one of the most important concept you need to understand when designing assertions? As shown in Fig. 6.9 the important thing to note is that the variables used in assertions (property/sequence/expression) are sampled in the *preponed* region. What does that mean? It means (for example) if a sampled variable changes the same time as the sampling edge (e.g., clk) that the value of the variable will be the value it held—*before*—the clock edge.

```
@ (posedge clk) a |=> !b;
```

In the above sequence, let us say that variable "a" changes to "1" the same time that the sampling edge clock goes posedge clk (and assume "a" was "0" before it went to a "1"). Will there be a match of the antecedent "a"? No! Since "a" went from "0" to "1" the same time that clock went posedge clk, the sampled value of "a" at posedge clk will be "0" (from the preponed region) and not "1." This will not cause the property to trigger because the antecedent is not evaluated to be a "1." This will confuse you during debug. You would expect "1" to be sampled and the property triggered thereof. However, you will get just the opposite result.

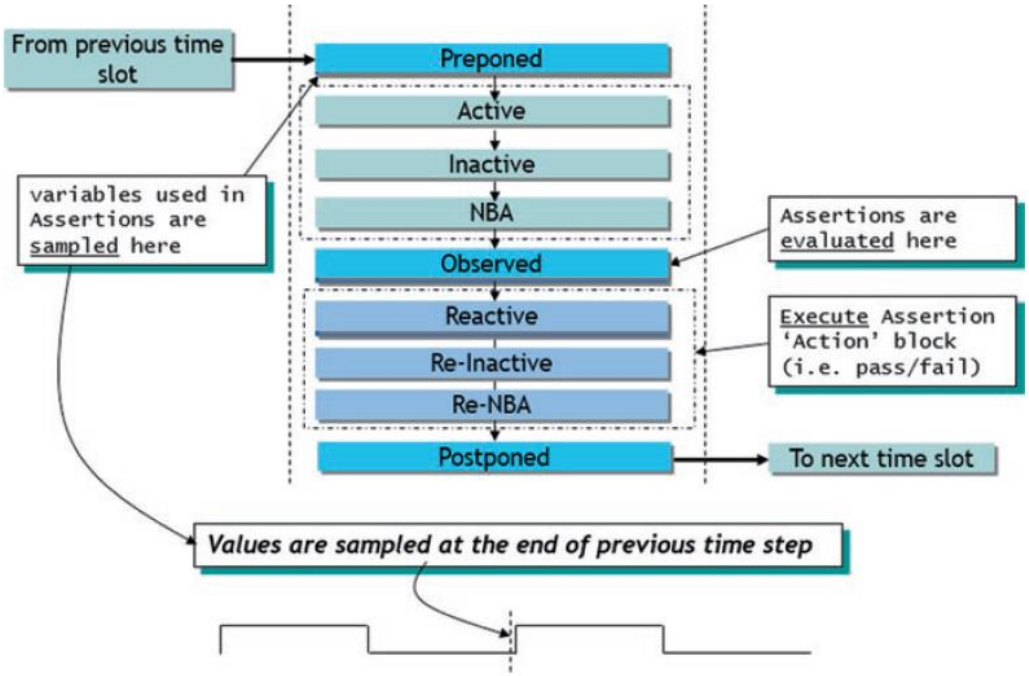


Fig. 6.9 Assertions variable sampling and evaluation/execution in a simulation time tick

SystemVerilog Scheduler

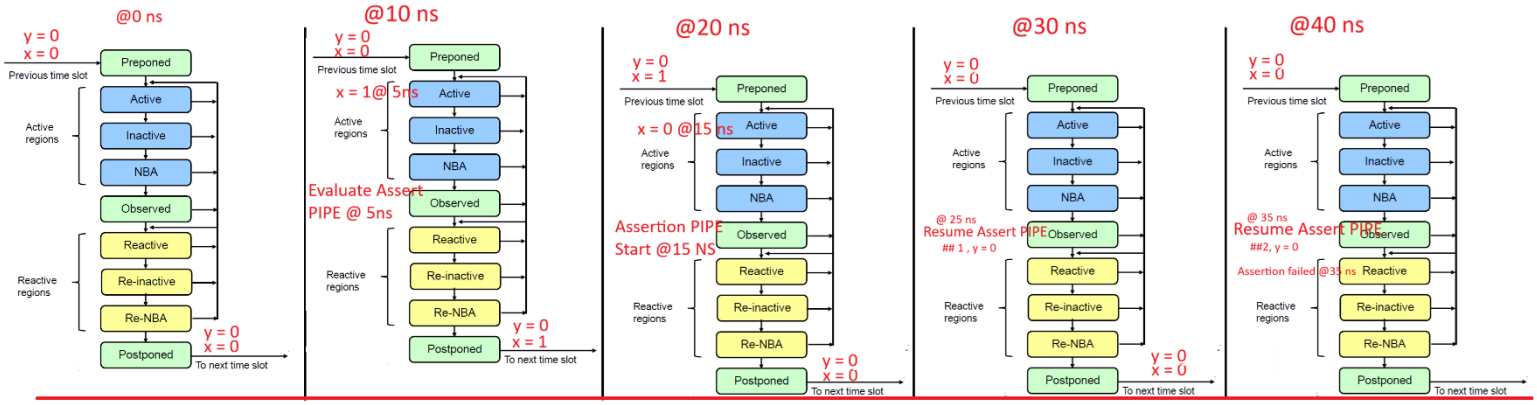
Youssef Nasser

Example

SV Code

```
module assertion_scheduling;

  bit clk;
  logic x,y;

  always #5 clk = ~ clk;

  default clocking my_clk
  @(posedge clk);
  endclocking

  initial begin
    $dumpfile("waveform.vcd");
    $dumpvars;
    clk          = 0;
    repeat (15) @(posedge clk);
    $finish;
  end

  initial begin
    x = 0;
    y = 0;

    @(posedge clk);
    x = 1;

    @(posedge clk); // 5ns
    x = 0;

  end

  property PIPE;
    x |-> ##2 y;
  endproperty

  A_PIPE : assert property (PIPE) ;


endmodule
```

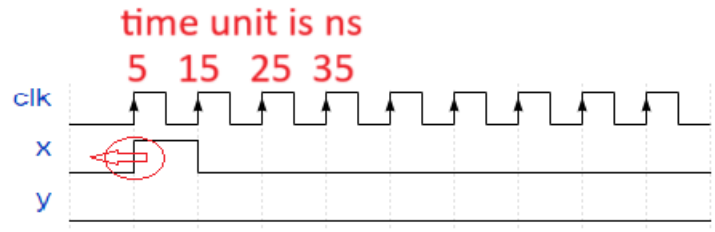
each block for 1 ns only but let it ticks each clock , omitted for easing.



SystemVerilog Scheduler

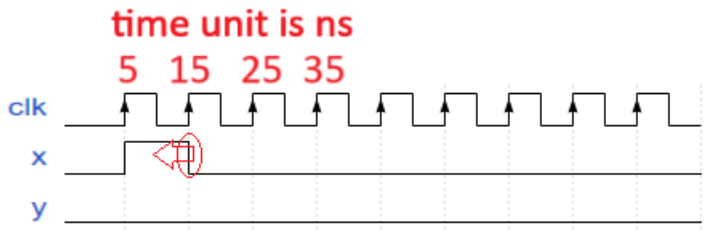
Youssef Nasser

Debugging assertion in waveform , clock period is 10 ns , starting from 0 so posedge clk timing are: @5ns , @15ns , @25ns and 35@ns

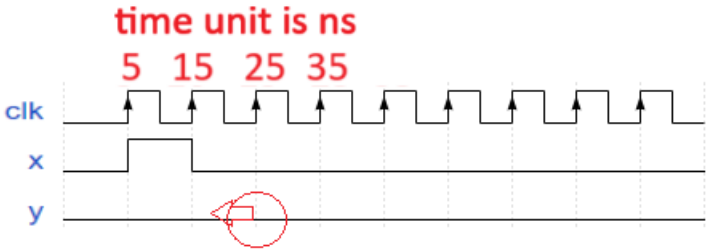


Assertion is evaluated each posedge clock

@5ns It will check the preponed region which is left from 5ns , it will check x at 4ns , x is 0 so assertion will not start.



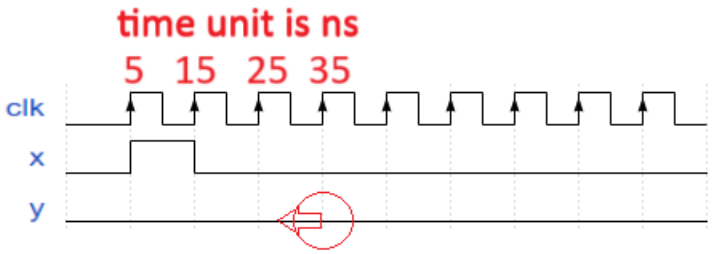
@15 ns It will check the preponed region which is left from 15ns , it will check x at 14ns , x is 1 so assertion will start.



@25 ns , it will check preponed region for x exactly @24 ns , it's 0 so no more assertion attempt for that property.

Since assertion is overlapping implication so it will count from that cycle "15ns to 25ns" which is first cycle.

@25 ns will check the preponed region which is left from 25ns , it will check y at 24ns , y is 0 so it saves value of y at ##2 is 0.



"25ns to 35ns" which is second cycle.

@35 ns will check the preponed region which is left from 35ns , it will check y at 34ns , y is 0 so it saves value of y at ##2 is 0.

In the reactive region assertion will take an action weather it pass or fail , @35 assertion will fail since after 2 clock cycles from antecedent x is 1 , the value of y is 0.

VCS log

```
** Error: Assertion error.
#    Time: 35 ns Started: 15 ns  Scope: preponed.A_PIPE File: testbench.sv
Line: 36
```

EDAPLAYGROUND link

https://edaplayground.com/x/rez6

immediate assertions – scheduler

https://verificationacademy.com/forums/t/event-region-for-immediate-and-deferred-immediate-assertions/40336/2

assert final?

.triggered and wait

Static

Dynamic

Active region work with initial and always? What is the references

Youssef Nasser

Extra Notes for finish and stop msgs

Diagnostic Messages for $stop and $finish Tasks

When using the `$stop` and `$finish` task, you can provide an optional argument to print diagnostic messages related to the simulation termination. There are three possible argument values, each resulting in a different diagnostic message:

Argument Value	Diagnostic Message
0	Prints nothing
1	Prints simulation time and location (default)
2	Prints simulation time, location, and statistics about memory and CPU time used in simulation

Use VCS to check the log.

Reference link
https://circuitcove.com/system-tasks-simulation-control/

- **$finish(0);**
  VCS LOG
  “ “ empty

- **$finish(1); or $finish; → DEFAULT IS $finish(1) or finish 😊**
  VCS LOG
  **** Note: $finish : testbench.sv(16)**

- **$finish(2); or $finish**
  **** Note: Data structure takes 5067368 bytes of memory**
  Process time 0.01 seconds
  $finish : testbench.sv(16)

  Note if you writing anything instead of $finish(0); $finish;
  $finish(1),$finish(2);
  it will be default.
  for example $finish(3); it’s equivalent to $finish;