CAL POLY
SAN LUIS OBISPO

EE 531: ADVANCED VLSI DESIGN

The SystemVerilog Program

Nishith N. Chakraborty

January, 2025

INTRODUCING THE PROGRAM CONSTRUCT

- Specifically for modeling the testbench environment
- Emphasis is not in modeling hardware details (wires, structure, and interconnections) but modeling environment in which hardware is verified
- Three basic purposes of `program`:
  - Provides entry point for execution of testbenches
  - Encapsulates program-wide data, tasks and functions
  - Provides scheduling for a reactive region
- Always enclosed as such:

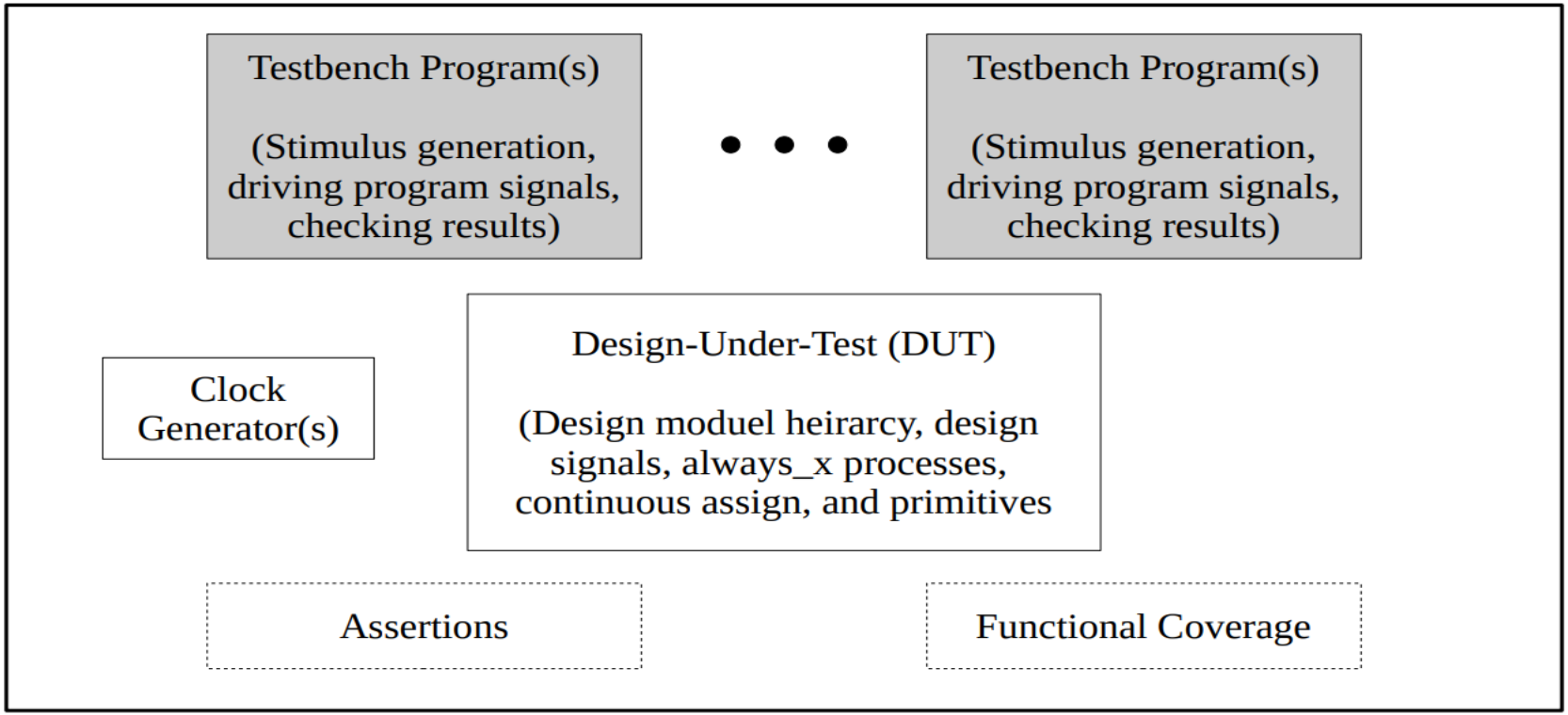
```
program
   ...
endprogram
```

MAKING OF A SYSTEMVERILOG PROGRAM

- Purpose: provide race-free interaction between design and testbench
- Can contain: data declarations, class definitions, subroutines, object instances, and one or more `initial` or `final` procedural blocks (no `always` blocks)
- Ability to initialize and individually instantiate `program` constructs enables use as *generalized* testbenches
- Example `program` declaration (structure similar to `module`):

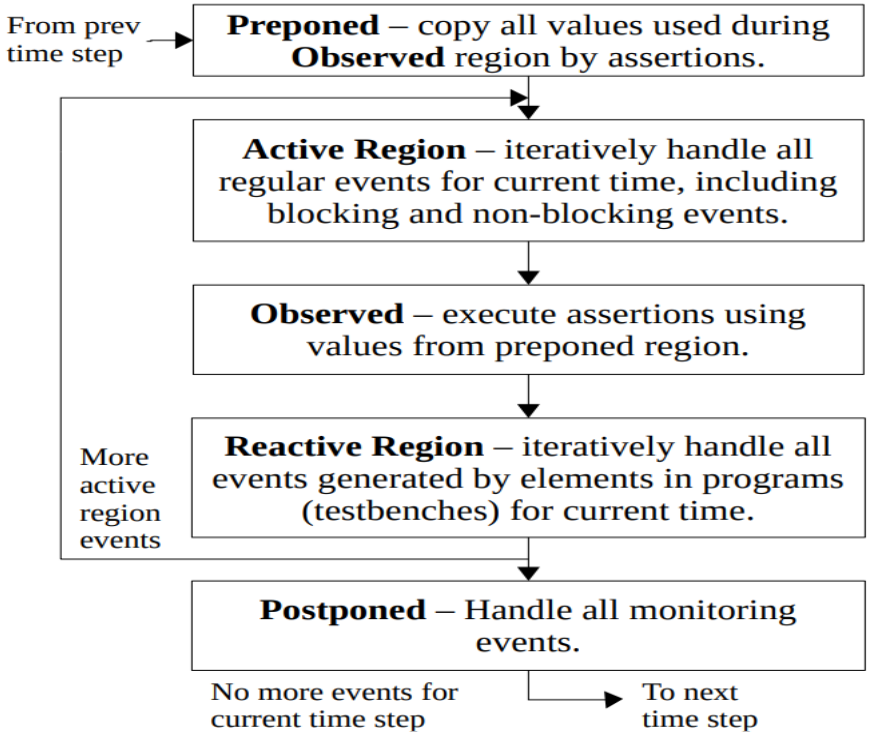
```
program testPrg
  (input logic clk,
   input logic [15:0] addr,
   inout logic [7:0] data);

  initial begin
    ...
  end
endprogram: testPrg
```

COMPLETE DESIGN/TESTBENCH HEIRARCHY



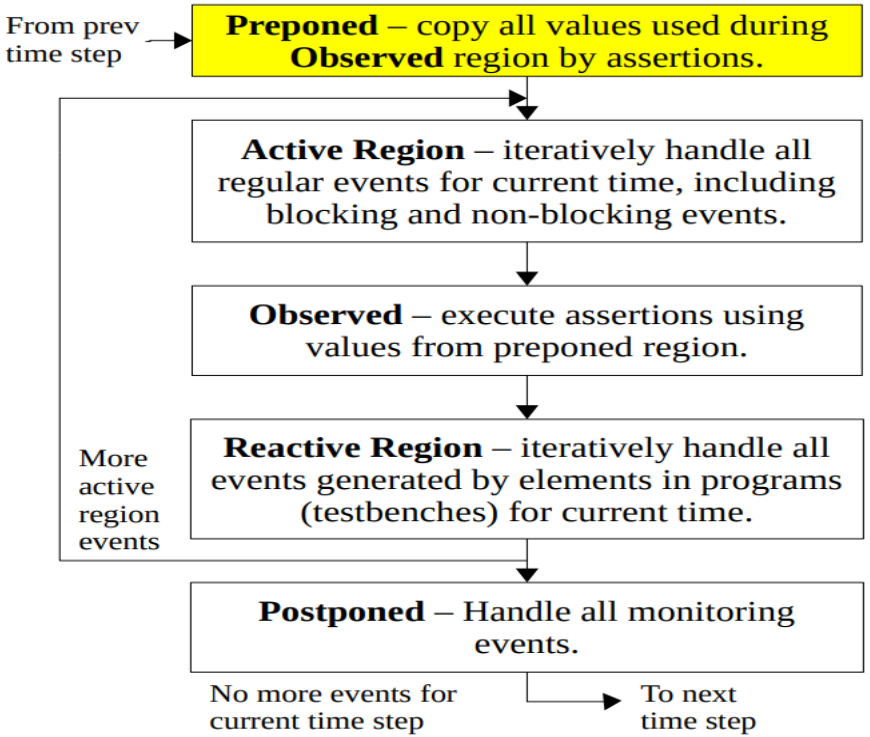
THE SIMULATION KERNEL

- Simulation handles all events generated from design based on new value changes
- Once all events handled, program executes for current time
- Program then generates new input values



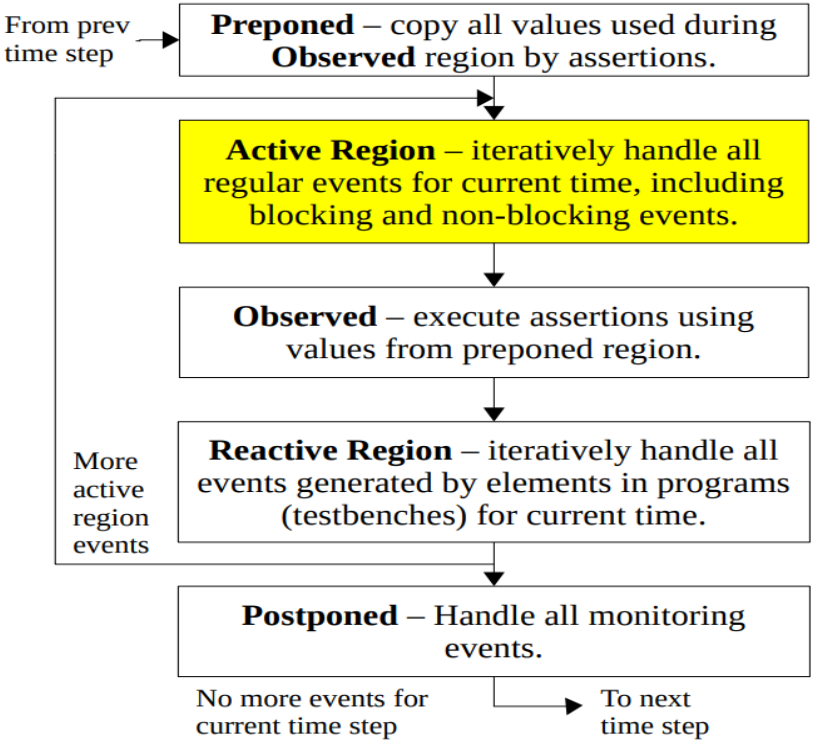
THE SIMULATION KERNEL

- **Preponed**
  - Time has just advanced from previous time step
  - No values to be updated in current step have appeared yet
  - All values from end of last step are still available
  - Prior step values are sampled/saved for later use



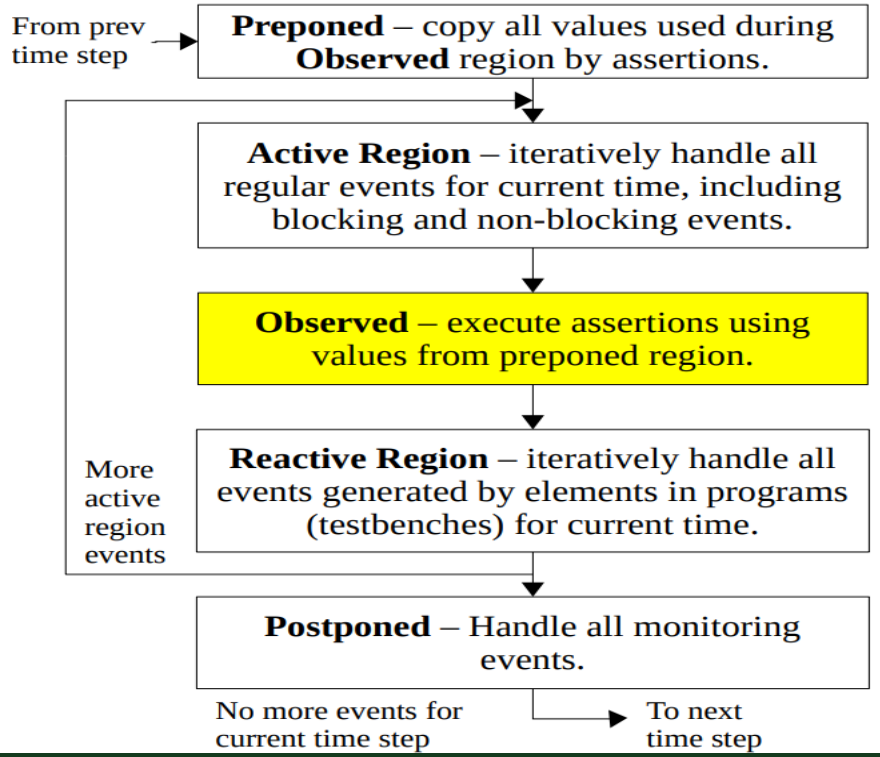
THE SIMULATION KERNEL

- **Active Region**
  - All regular events for current time retrieved from event list and handled
  - Several 0-delay events may be generated, several simulation cycles may be needed – loop
  - When no more regular events, then non-blocking updates are handled
  - All previous discussions have focused on this region



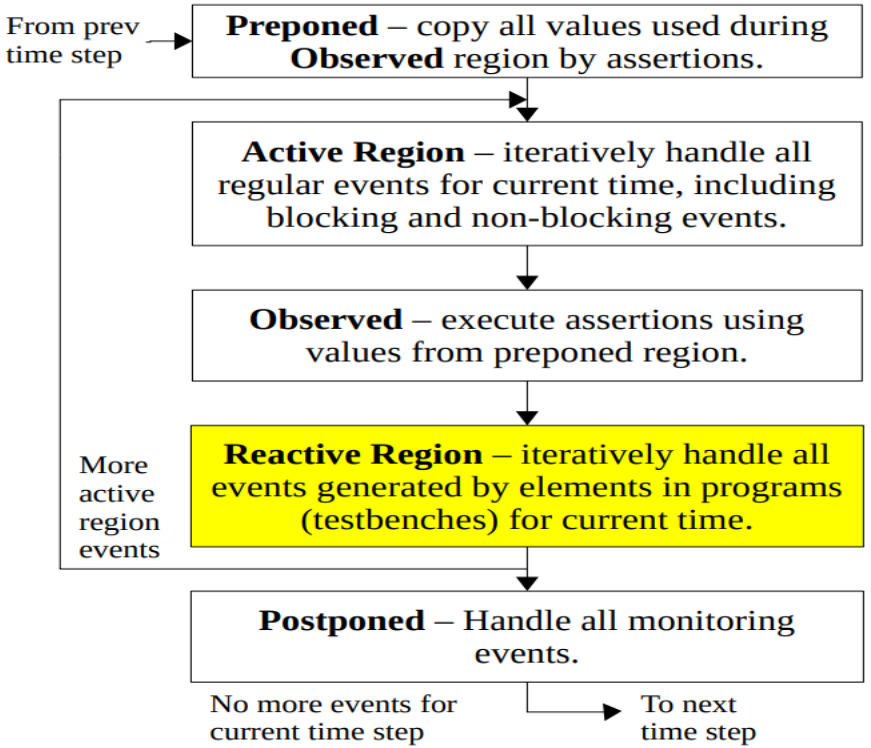
THE SIMULATION KERNEL

- **Observed**
  - Concurrent assertions execute based on values sampled in preponed region
  - Concurrent assertions are not based on value changes from events handled in active region



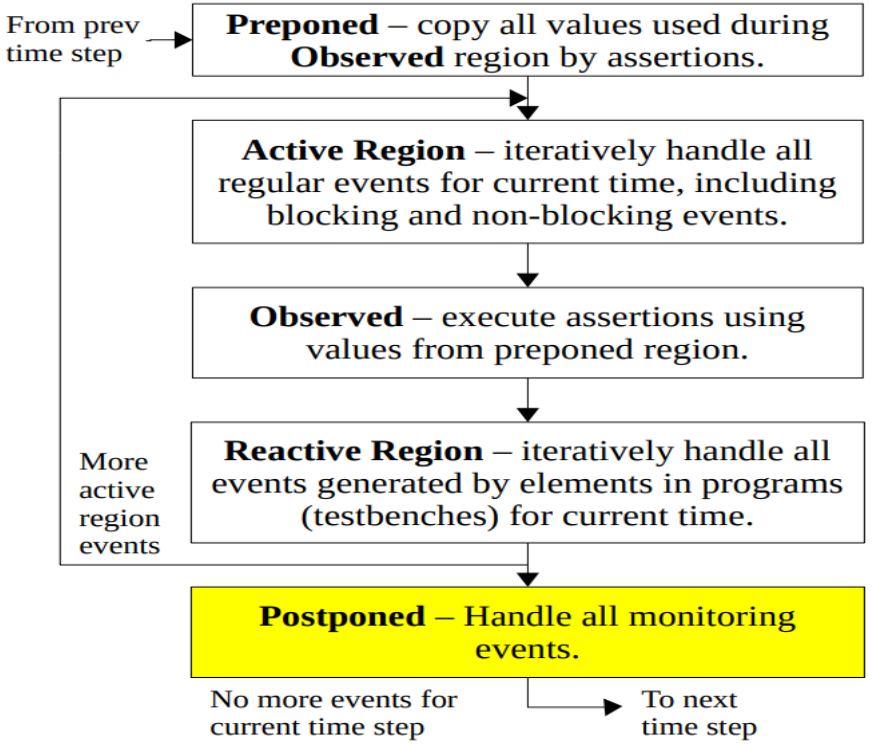
THE SIMULATION KERNEL

- **Reactive Region**
  - Region for regular and non- blocking events generated by program constructs
  - Functionality of testbench
  - Executed just like active region: regular then non-blocking
  - May cause more active region events (testbench change, design executes) – loop
  - Testbench reacts to current time results

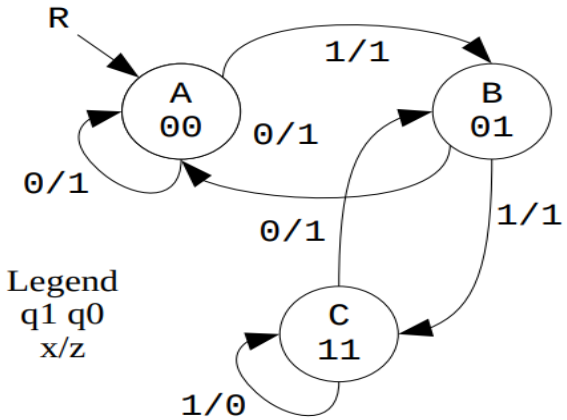


THE SIMULATION KERNEL

- **Postponed**
  - Region where monitoring events are executed
  - From this point, go to next time step



OLD EXAMPLE: FSMBEHAVIOR




```
module FSMbehavior
  (input  logic x, ck, r_l,
   output logic z);
  enum {A, B, C} state;       // state variable

  // state memory AND next state logic in one always block
  always_ff @(posedge ck, negedge r_l) begin
    if(~r_l)                  // active low reset
      state <= A;
    else
      case (state)
        A: state <= (x) ? B : A;
        B: state <= (x) ? C : A;
        C: state <= (x) ? C : B;
        default: state <= A; // always include default!
      endcase
  end

  // output logic with always_comb
  always_comb begin
    z = 1'b1;                 // z starts with 1 - may override
    if (state == C) z = ~x;   // OK - z is updated no matter
what!
  end
endmodule: FSMbehavior
```

EXAMPLE: FSM TESTBENCH WITH PROGRAM TOP MODULE

- Top testbench module instantiates DUT and at least one program

```
module topTest;
  logic ck, x, z, r_l; // internal to top testbench module
                       // provides connections between
                       // DUT and the FSM testbench program

  FSMbehavior dut(.*);
  fsmProg     tb(.*);

  // global signals ck and r_l excited at the top
  //   these happen regardless of DUT or program
  //   independent of specific test
  initial begin: I
    $monitor($time, " Current State = %s, Output = %b",
             dut.state.name, z);
    ck = 0; r_l = 0;
    r_l <= #1 1;
    forever #5 ck = ~ck;
  end
endmodule: topTest
```

EXAMPLE: FSM TESTBENCH WITH PROGRAM TOP MODULE

- Top testbench module instantiates DUT and at least one program

```
program fsmProg
  (input  logic ck,  // only input is clock – implicit FSM
   output logic x);  // output is x – will excite DUT FSM

  initial begin: J   // the implicit FSM
    x <= 1;
    @(posedge ck);   // test A -> B
    @(posedge ck);   // test B -> C
    @(posedge ck);   // test C -> C
      x <= 0;
    @(posedge ck);   // test C -> B
    @(posedge ck);   // test B -> A
    @(posedge ck);   // test A -> A
    #1 $finish;
  end
endprogram: fsmProg
```

Thank you!