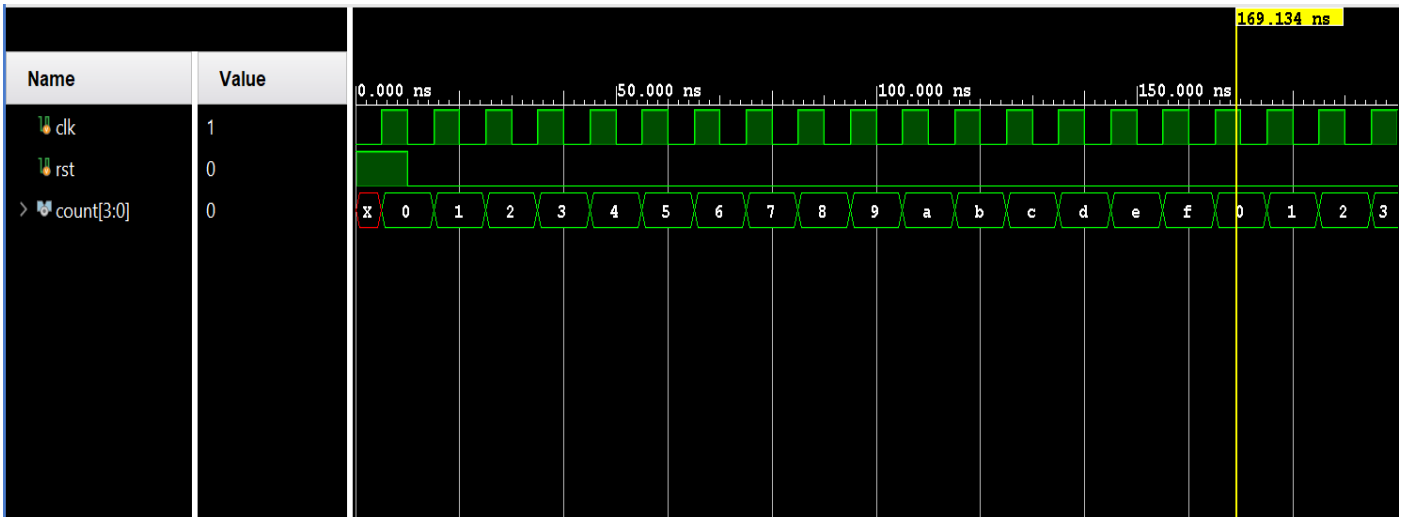
Create a 4-bit binary counter with synchronous reset.

CODE:

```
`timescale 1ns / 1ps

module binary_counter4bit(
input clk, rst,
output reg [3:0] count);
always@(posedge clk)begin
   if(rst)
      count<=4'b0;
   else begin
      if(count<=4'b1111)
         count<=count+1'b1;
      else
         count<=4'b0;
   end
end
endmodule
```

TESTBENCH:

```
module counter_tb();
   reg clk;
   reg rst;
   wire [3:0] count;
   // Clock generation using an always block
   binary_counter4bit dut(clk,rst,count);
   initial begin
      clk = 0; // Initialize clock
   end

   always #5 clk = ~clk; // Toggle clock every 5 time units
   // Reset logic
   initial begin
      rst = 1;   // Activate reset
      #10 rst = 0; // Deactivate reset after 10 time units
   end

   initial begin
      #200; // Run simulation for some time
      $stop; // Stop simulation
   end
endmodule
```



KEY NOTES:

- The initial block initializes clk to 0.
- The always block toggles clk every **5 time units**, meaning:
- The clock period is **10 time units** (5 up + 5 down).
- At **time = 0**, rst is set to 1, keeping the counter in reset mode.
- At **time = 10**, rst is set to 0, allowing the counter to start counting.
- The simulation **runs for 200 time units**, allowing the counter to increment several times.
- The $stop; command halts the simulation.

In Verilog, initial begin blocks are used for **one-time** execution in a simulation. These blocks execute sequentially **only once**, starting at time 0. You can use multiple initial blocks in a testbench. Both will start together at t=0 but execute independently.

- Each statement inside executes **sequentially** (line by line).
- Delays (#) can be used to introduce timing.

Declaration	Type	Example Usage
reg [3:0] a;	4-bit **single** register	Shift registers, counters, arithmetic operations.
reg a [3:0];	Array of **4 one-bit** registers	Individual bit storage, flags, separate bit control.