# Verilog HDL: Looping Statement

**Pravin Zode** 

#### Outline

- Loop Basics
- Forever loop
- Repeat loop
- While loop
- For loop
- Loop termination Disable Statement

#### Introduction

- Looping statements allow repetitive execution of code in behavioral blocks
- Primarily used in testbenches, simulation, and certain synthesis scenarios
- Four main types
  - Forever
  - Repeat
  - While
  - > for

#### **Forever Statement**

- Executes the enclosed statements indefinitely
- Typically used in initial blocks of testbenches for continuous processes.

```
initial begin
forever begin

clk = ~clk; // Invert clock signal continuously
#5; // 5 time unit delay
end
end
end
```

The forever loop cannot be synthesized into hardware, making it unsuitable for hardware design

# Example: Forever Statement (clock)

```
module clock gen(
       output reg clk // Clock output
3
4
5
       // Initialize clock and toggle indefinitely using a forever loop
6
       initial begin
         clk = 0; // Start clock at 0
8
        forever begin
         #5 clk = ~clk; // Toggle clock every 5 time units (10 time units period)
9
10
         end
11
       end
12
13
     endmodule
```

# **Example: Forever Statement**

```
module clock_gen;
 1
     //Example 1: Clock generation
     //Use forever loop instead of always block
 3
     reg clock;
 4
 5
     initial
 6
     begin
             clock = 1'b0;
 8
9
             forever #10 clock = ~clock; //Clock with period of 20 units
10
     end
11
12
     initial
              #100000 $finish;
13
14
15
     endmodule
```

## Repeat Statement

- Repeats a block of statements a fixed number of times
- Evaluates the count once at the start of the loop

Unlike the forever loop, the repeat loop is synthesizable into hardware.

# **Example: Repeat Statement**

```
1
     module repeat_loop_counter;
       reg [3:0] count; // 4-bit counter
 3
 4
       initial begin
 5
         count = 4'b0000; // Initialize count to 0
 6
 7
         repeat (10) begin // Repeat loop runs 10 times
 8
           #5 count = count + 1; // Increment count every 5 time units
           $display("Time = %0t | Count = %b", $time, count); // Print count
 9
10
         end
11
       end
     endmodule
12
```

# While Loop

- Executes a block of code repeatedly as long as a specified condition is true
- The condition is evaluated before each iteration.

```
1 initial begin
2     integer i = 0;
3     while (i < 10) begin
4     $\display(\text{"Value of i: \text{\chiod", i)};}
5     i = i + 1;
6     #10;
7     end
8     end</pre>
```

While loops are simulation-only and not synthesizable in hardware

## Example: While Loop

```
1
     module while loop counter;
 2
       reg [3:0] count; // 4-bit counter
 3
       integer i; // Loop variable
 4
 5
       initial begin
 6
         count = 4'b0000; // Initialize count to 0
 7
         i = 0;
 8
 9
         while (i < 10) begin // Loop until i reaches 10
10
           #5 count = count + 1; // Increment count every 5 ti
           $display("Time = %0t | Count = %b", $time, count);
11
12
           i = i + 1; // Increment loop variable
13
         end
14
       end
     endmodule
15
```

# For loop

- Similar to C-style for loops; combines initialization, condition, and increment in one statement
- Offers compact syntax for known iteration counts

```
1  initial begin
2  integer i;
3  for (i = 0; i < 10; i = i + 1) begin
4  $display("Loop iteration %0d", i);
5  #10;
6  end
7  end</pre>
```

For loops are synthesizable and suitable for hardware design

### Example: For loop

```
1
     module for loop counter;
       reg [3:0] count; // 4-bit counter
 2
 3
       integer i; // Loop variable
 4
 5
       initial begin
 6
         count = 4'b0000; // Initialize count to 0
 7
8
         for (i = 0; i < 10; i = i + 1) begin // Loop from 0 to 9
9
           #5 count = count + 1; // Increment count every 5 time units
           $display("Time = %0t | Count = %b", $time, count); // Print count
10
11
         end
12
       end
     endmodule
13
```

# Loop termination

- Verilog allows stopping a loop using the disable keyword.
- The disable function only works on named statement groups
- It is typically used after a fixed amount of time or within a conditional construct like if-else or case
- disable is useful for controlled termination of loops based on a condition.

# **Example: Loop termination**

```
1 ∨ module clock_generator(
         input EN, // Enable signal
 2
 3
         output reg CLK // Clock output
 4 \( \cdot \);
 5
 6
         initial begin
7
             CLK = 0; // Initialize clock to 0
8
             forever begin: loop ex
                 if (EN == 1)
10
                     #10 CLK = ~CLK; // Toggle clock every 10 time units
                 else
11 v
12
                     disable loop_ex; // Terminate loop if EN is 0
13
             end
14
         end
15
     endmodule
16
```

# **Comparison & Best Practices**

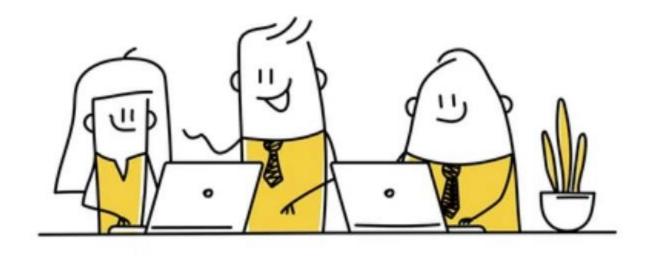
- forever: Use for continuous, never-ending processes (e.g., clock generation).
- repeat: Best for a predetermined, fixed number of iterations.
- while: Ideal when iterations depend on dynamic conditions.
- for: Preferred for concise loops with known iteration counts.
- General Tip: Always include exit conditions or simulation controls to avoid unintended infinite loops.

#### **Practical Use Cases**

- Testbench Clock Generation: Using forever to continuously toggle a clock signal.
- Simulation Data Generation: Using repeat to generate a fixed number of test vectors.
- Waiting for Conditions: Using while to poll a signal until a condition is met.
- Indexed Operations: Using for loops for initializing arrays or processing vector elements.

# Summary

- Looping statements in Verilog provide flexibility in behavioral modeling.
- Choose the loop type based on the desired iteration count and conditions.
- Always design with simulation control in mind to prevent unintentional infinite loops.



Thank you!

**Happy Learning**