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
```

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
);

  // Initialize clock and toggle indefinitely using a forever loop
  initial begin
    clk = 0;         // Start clock at 0
    forever begin
      #5 clk = ~clk; // Toggle clock every 5 time units (10 time units period)
    end
  end

endmodule
```

Example : Forever Statement

```
module clock_gen;
//Example 1: Clock generation
//Use forever loop instead of always block
reg clock;

initial
begin
        clock = 1'b0;
        forever #10 clock = ~clock; //Clock with period of 20 units
end

initial
        #100000 $finish;

endmodule
```

Repeat Statement

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

```
repeat (<number_of_loops>)
  begin
    statement_1
    statement_2
        :
    statement_n
  end
```

```
initial
  begin
    CLK = 0;
    repeat (10)
      #10 CLK = ~CLK;
  end
```

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

Example : Repeat Statement

```
module repeat_loop_counter;
  reg [3:0] count;  // 4-bit counter

  initial begin
    count = 4'b0000; // Initialize count to 0

    repeat (10) begin  // Repeat loop runs 10 times
      #5 count = count + 1; // Increment count every 5 time units
      $display("Time = %0t | Count = %b", $time, count); // Print count
    end
  end
endmodule
```

While Loop

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

```
initial begin
  integer i = 0;
  while (i < 10) begin
    $display("Value of i: %0d", i);
    i = i + 1;
    #10;
  end
end
```

While loops are simulation-only and not synthesizable in hardware

Example : While Loop

```
module while_loop_counter;
  reg [3:0] count;  // 4-bit counter
  integer i;        // Loop variable

  initial begin
    count = 4'b0000; // Initialize count to 0
    i = 0;

    while (i < 10) begin  // Loop until i reaches 10
      #5 count = count + 1; // Increment count every 5 t
      $display("Time = %0t | Count = %b", $time, count);
      i = i + 1; // Increment loop variable
    end
  end
endmodule
```

For loop

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

```
initial begin
  integer i;
  for (i = 0; i < 10; i = i + 1) begin
    $display("Loop iteration %0d", i);
    #10;
  end
end
```

For loops are synthesizable and suitable for hardware design

Example : For loop

```
module for_loop_counter;
  reg [3:0] count;  // 4-bit counter
  integer i;        // Loop variable

  initial begin
    count = 4'b0000; // Initialize count to 0

    for (i = 0; i < 10; i = i + 1) begin  // Loop from 0 to 9
      #5 count = count + 1; // Increment count every 5 time units
      $display("Time = %0t | Count = %b", $time, count); // Print count
    end
  end
endmodule
```

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

```
module clock_generator(
    input EN,         // Enable signal
    output reg CLK    // Clock output
);

    initial begin
        CLK = 0;  // Initialize clock to 0
        forever begin: loop_ex
            if (EN == 1)
                #10 CLK = ~CLK; // Toggle clock every 10 time units
            else
                disable loop_ex; // Terminate loop if EN is 0
        end
    end

endmodule
```

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