
VLSI

SYSTEM VERILOG LOOPS CONCEPT

Presented By
Neeli Anu

CONTENTS

- what is loop
- Types of loops
- Loop syntaxes
- Codes on loops
- Comparsion table

WHAT IS LOOP:

- A loop is a programming concept that repeats a block of code multiple times based on a condition.

Loop can generate sequences of operations in both hardware design (RTL) and verification testbenches.

- **Repetition:**

The primary purpose of a loop is to execute a set of statements multiple times. Instead of writing the same code line by line for each iteration(the process of doing something again and again), a loop provides a concise way to achieve this.

- **Condition-based Execution:**

Loops in SV run while a given condition is true or for a set number of iterations. Once the condition fails or the count completes, control moves to the next statement after the loop.

- **Procedural Context:**

Loops in SystemVerilog can only be used within **procedural blocks**.

TYPES OF LOOP

I. For loop
II. While loop
III. Do while loop
IV. Foreach loop
V. Forever loop

FOR LOOP :

- A for **loop** is a programming structure used to **repeat a block of code a specific number of times** (e.g., "for each item in this list,)
- **Main use :** If the loop variable is declared inside the loop itself, the compiler understands that the variable belongs only to that specific loop. This keeps the code clean and reduces the chances of bugs.
- The for loop requires multiple statements within it to be enclosed by begin and end keywords.

This loop consists of three parts: **initialization**, **condition**, and **increment**

Syntax :

```
for ( [initialization]; <condition>; [modifier])
// Single statement
```

```
for (int i = 0; i < 5; i++) begin

$display("i = %0d", i);end
```

```
for (int i = 0; i < 10; i++)
 // For (int i=5; i<0 i --)
          begin
                // loop body
             end
```

I =Varaible like a local loop(integer

i++ = pre-increment based on codes numbers (1,2,3,4,5)

i -- =pre -Decrement number up to down(5,4,3,2,1)

Example:

```
module tb;
initial begin
 while(a<3)
 // while(a>3)
   begin
    $display("a=%0d",a);
    a++;
  end
end
```

```
module tb;
  initial begin
//for(int i=0;i<3;i++)
for (int i = 5; i >= 0; i--) begin
     $display("i = %0d", i);
   end

  end
endmodule
```

Output :

i = 5
i = 4
i = 3
i = 2
i = 1
i = 0

```
module tb;
        string array [5] = '{"apple", "orange", "pear",
"blueberry", "lemon"};

        initial
    begin
     for (int i = 0, j = 3; i < $size(array); i++)
     begin
      // array[i][j] = "JYD";
      array[i][j] = "J";
      $display ("array[%0d] = %s, %0dth index replaced by JYD",
i, array[i], j);
                    end
        end
endmodule
```

Output :

```
array[0] = appJe, 3th index replaced by JYD
array[1] = oraJge, 3th index replaced by JYD
array[2] = peaJ, 3th index replaced by JYD
array[3] = bluJberry, 3th index replaced by JYD
array[4] = lemJn, 3th index replaced by JYD
```

Array iteration:

The for loop initialization declares a local variable called i that represents index of any element in the array. The conditional expression checks that i is less than size of the array. The modifier increments the value of i so that every iteration of the for loop operates on a different index.

```
module tb;
        string array [5] = '{"apple", "orange", "pear", "blueberry", "lemon"};

        initial begin
                for (int i = 0; i < $size(array); i++)
                        $display ("array[%0d] = %s", i, array[i]);
        end
endmodule
```

Output:

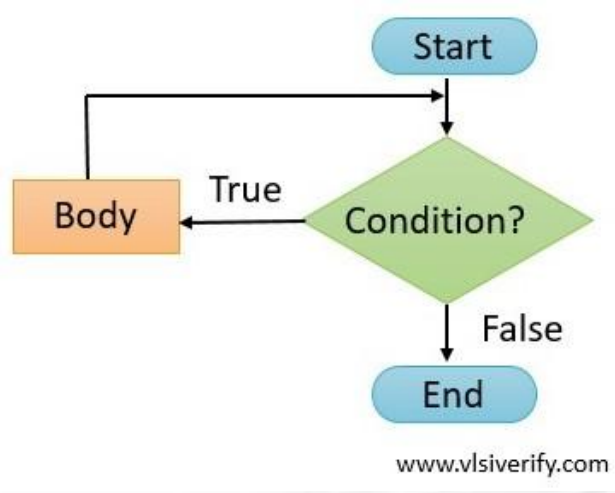
```
array[0] = apple
array[1] = orange
array[2] = pear
array[3] = blueberry
array[4] = lemon
```

WHILE LOOP:

A while loop is a control flow statement in SystemVerilog used to execute a block of code repeatedly as long as a specified condition is True.

If the condition turns out to be **False**.The loop ends right there.

- Other wise it's shown Error or simulation after not shown empty.



Features	Description
✓ Condition check timing	○ Before entering the loop
✓ Runs minimum times	○ 0 times (if condition is false initially)
✓ Use case	○ When you **don't know** in advance how many times to repeat
✓ Can be infinite?	○ Yes, if condition is always true (e.g., while(1))
✓ Exit early?	○ Use break; inside loop to exit early
✓ Skip current iteration?	○ Use continue; to skip to next condition check

Syntax :

```
while (condition) begin
    // Statements to be executed
end
```

Example 1 :

```
module tb;
  int a;
 initial begin
 while(a<3)
// while(a>3)
 begin
$display("a=%0d",a);
 a++;
    end
  end
endmodule
```

Example -2 :

```
module tb;
  initial begin
 int cnt;
  while (cnt != 0)
begin
  $display ("cnt = %0d", cnt);
                    cnt++;
  end
   end
endmodule
```

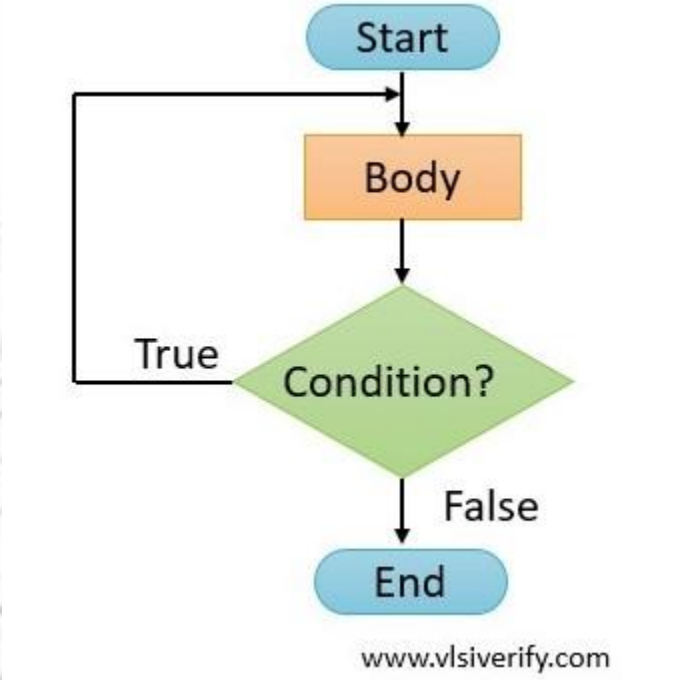
DO-WHILE :

- Executes the loop body *at least once*, and then checks the condition.
- If the condition is true after the first execution, the loop continues. If false, it terminates.
- This means the code inside the do-while loop is guaranteed to run at least one time, regardless of the initial condition

~~Similar to a while loop, but it guarantees that the code block will execute at least once before the condition is checked. The condition is evaluated after each iteration~~.

```
Initial begin
do {
    // code block
        } while (condition);


end
```



EXAMPLE 1:

```
module tb;
  int i = 0;

  initial begin
    do begin
      $display("Iteration %0d", i);
      i- -;
  //i++;
    end while (i < 5);
// Loop will run while this condition is true

end
endmodule
```

EXAMPLE -2 :

```
module tb;
  int a;

  initial begin
    do
      begin
        $display("a=%0d",a);
        a++;
      end
    // while(a<3);
     while(a>3);

  end
endmodule
```

Difference between while and do while loop

Feature / Aspect	while Loop	do...while Loop
Condition Check	Before the loop body is executed	After the loop body is executed
Minimum Executions	May not execute even once if condition is false	Executes **at least once**, even if condition is false
Syntax	while (condition) begin ... end	do begin ... end while (condition);
Use Case	When the code should **run only if condition is true**	When code **must run at least once**, regardless of condition
Common Error Risk	If condition is never true, code inside never runs	Can execute once even if logic doesn't require it
Condition First?	✅ Yes	❌ No
Guaranteed to Run Once?	❌ No	✅ Yes
Use When...	You want to check before run	You want to run at least once

Foreach :

- loop in SystemVerilog allows you to iterate over elements in arrays—both one-dimensional and multi-dimensional—without manually managing index variables.
- It makes your code cleaner, especially during simulations and when working with complex data structures.
- The foreach loop iterates through each index starting from 0. If there are multiple statements within the foreach loop, they have to be enclosed with begin and end keywords like all other procedural blocks.

~~✓ foreach (array_name[index]) begin~~
~~✓ // statements using array_name[index]~~
~~✓ End~~

~~Multi dimensional~~
~~✓ foreach (array_name[i, j]) begin~~
~~✓ // access array_name[i][j]~~
~~✓ end~~

```
foreach(<variable>[<iterator>])
        // Single statement

foreach(<variable>[<iterator>]) begin
        // Multiple statements
end
```

Works with Various Array Types:

It provides a unified way to iterate over:

- **Fixed-size (Static) Arrays:** int my_array[10];
- **Dynamic Arrays:** int dynamic_array[];
- **Queues:** int my_queue[$];
- **Associative Arrays:** string my_map[string];

Static array [fixed size]

EX1:

```
module foreach_example;
 int a[5] = '{10, 20, 30, 40, 50};

 initial begin
  foreach (a[i]) begin
   $display("a[%0d] = %0d", i,
a[i]);
  end
 end
endmodule  A
```

Output :

```
a[0] = 10
a[1] = 20
a[2] = 30
a[3] = 40
a[4] = 50
```

Ex:2

```
module tb;
 int a[4];
 initial
  begin

  foreach(a[i])
   a[i] = i;
  foreach(a[i])
   $display("a[%0d]=%0d",i,a[i]);

 end
endmodule
```

Output :

```
a[0]=0
a[1]=1
a[2]=2
a[3]=3
```

Dynamic Array

```
module foreach_dynamic;
 int da[];

 initial begin
  da = new[4];
  da = '{1, 3, 5, 7};

  foreach (da[i]) begin
   $display("da[%0d] = %0d", i,
da[i]);
  end
 end
endmodule
```

Output :

```
da[0] = 1
da[1] = 3
da[2] = 5
da[3] = 7
```

Associative Array

```
module foreach_assoc;
 int aa[string];

 initial begin
  yogi["a"] = 100;
  yogi["b"] = 200;

  foreach (aa[i]) begin
  $display("yogi[%s] = %0d", i,aa[i]);
        end
           end
            endmodule
```

Output :

yogi[a] = 100
yogi [b] = 200

Dimensional Array 1D:

```
module tb;
 int array[5] = '{1, 2, 3, 4, 5};
 initial
  begin
  foreach (array[i])
   begin
   $display("array[%0d] = %0d", i, array[i]);
  end
 end
Endmodule
```

Output:

array[0] = 1
array[1] = 2
array[2] = 3
array[3] = 4
array[4] = 5

Multi Dimensional Array

```
module tb;
 int array[3][3][3] = '{'{'{1, 10, 100}, '{2, 20, 200}, '{3, 30, 300}},
                      '{'{4, 40, 400}, '{5, 50, 500}, '{6, 60, 600}},
                      '{'{7, 70, 700}, '{8, 80, 800}, '{9, 90, 900}}
                     };
 initial
   begin
   foreach (array[i,j, k])
     begin
     $display("array[%0d][%0d][%0d] = %0d", i,j, k, array[i][j][k]);
   end
 end
endmodule
```

URL : https://www.edaplayground.com/x/YzJ5

Output :

```
array[0][0][0] = 1
array[0][0][1] = 10
array[0][0][2] = 100
array[0][1][0] = 2
array[0][1][1] = 20
array[0][1][2] = 200
array[0][2][0] = 3
array[0][2][1] = 30
array[0][2][2] = 300
array[1][0][0] = 4
array[1][0][1] = 40
array[1][0][2] = 400
array[1][1][0] = 5
array[1][1][1] = 50
array[1][1][2] = 500
array[1][2][0] = 6
array[1][2][1] = 60
array[1][2][2] = 600
array[2][0][0] = 7
array[2][0][1] = 70
array[2][0][2] = 700
array[2][1][0] = 8
array[2][1][1] = 80
array[2][1][2] = 800
array[2][2][0] = 9
array[2][2][1] = 90
array[2][2][2] = 900
```

Forever :

The forever loop creates an infinite loop that continuously executes a statement or a block of statements. This loop continues indefinitely until it is explicitly terminated by a $finish system call or a break.

If the loop contains only one statement, the begin and end keywords can be omitted.

Syntax on Forever:

The basic syntax for a forever loop is straightforward

```
systemverilog
forever begin
    // Statements to be executed repeatedly
end
```

Functionality

Infinite Execution:

Unlike other loops (like for, while, or repeat) that iterate a fixed number of times or based on a condition, the forever loop runs endlessly.

Procedural Context:

forever is a procedural statement, meaning it must be enclosed within procedural blocks like initial, always, or task blocks. It cannot be used directly inside a module.

Simulation Use:

Forever loops are primarily used in testbenches for simulation purposes and are not synthesizable, meaning they cannot be used to describe hardware for synthesis.

Time Delays: It is crucial to include a time delay (#) within a forever loop to prevent the simulation from hanging in a zero-delay loop.

Time Delays:

It is crucial to include a time delay (#) within a forever loop to prevent the simulation from hanging in a zero-delay loop.

Common uses

Clock Generation:

A frequent use of the forever loop is generating clock signals in a testbench. The loop can be used to toggle a clock signal at regular intervals.

Stimulus Generation:

Testbenches often utilize forever loops within tasks to continuously generate and apply stimuli to the design under test.

Monitoring and Sampling:

Drivers and monitors in verification components within methodologies like UVM often use forever loops inside tasks to continuously observe bus activity or sample signals

Terminating the loop

Since a forever loop runs indefinitely, mechanisms are necessary to terminate it when required:

$Finish:

The $finish system task can be used to stop the simulation entirely, thus ending the forever loop.

break statement:

The break statement allows prematurely exiting the loop based on a specified condition.

Example :

```
module forever_example;
 int count;
 initial begin
  forever begin
   $display("Value of count = %0d", count);
   count++;
   #5;
  end
 end

 initial begin
  #30;
  $finish;
 end
endmodule
```

Output:

```
Value of count = 0
Value of count = 1
Value of count = 2
Value of count = 3
Value of count = 4
Value of count = 5
```

Example with a break statement

```
module forever_example;
 int count;
 initial begin
  forever begin
   $display("Value of count = %0d", count);
   count++;
   if(count == 10) break;
  end
 end
endmodule
```

Output:

```
Value of count = 0
Value of count = 1
Value of count = 2
Value of count = 3
Value of count = 4
Value of count = 5
Value of count = 6
Value of count = 7
Value of count = 8
Value of count = 9
```

A always block inside another procedural block

A compilation error is expected when always block is used inside another procedural block. In such a case, a forever block can be used.

```
module tb;
  int count;
  initial begin
    always begin // can not use inside other procedural block
      $display("Value of count = %0d", count);
      count++;
      #5;
    end
  end

  initial begin
    #30;
    $finish;
  end
endmodule
```

Output :

```
Following verilog source has syntax error :
  "testbench.sv", 7: token is 'always'
      always begin // can not use inside other
procedural block
            ^
1 error
```

Clock Generation

```
    module tb;
  reg clk = 0;

  initial begin
    forever begin
      #5 clk = ~clk;
    end
  end
endmodule
```

This code Verilog or Sv

```
          module tb;
  reg clk = 0;

  always begin
    #5 clk = ~clk;
  end
endmodule
```

This code does not contain a $display or $finish, so it won't print anything by itself — and it will run forever (until you manually stop it).

Repeat Loop :

The repeat loop in SystemVerilog is used to execute a block of code a fixed number of times.

Syntax on repeat loop :

```
repeat(<number>)
    begin   // <number> can be variable or fixed value

            end
```

expression:

A positive integer value that decides how many times the loop executes.
The loop ends automatically after repeating the block the given number of times

Example:

Based on the array size, array elements are printed.

The string “Repeat it” is printed three times.

```
module tb;
 int array[5] = '{100, 200, 300, 400, 500};
 int i;
 initial begin
  repeat ($size(array)) begin
   $display("array[%0d] = %0d", i, array[i]);
   i++;
  end

  repeat(3)
   $display(“Yogendra ");
 end
```

Output :

```
 array[0] = 100
array[1] = 200
array[2] = 300
array[3] = 400
array[4] = 500
Yogendra
Yogendra
Yogendra
```

Testbench: Using repeat with break and continue

```
module tb;
 int i = 0;

 initial begin
  $display("Start of repeat loop with break and continue\n");

  repeat (10) begin
   i++;

   // Skip the 3rd yogendra
   if (i == 3) begin
    $display("Skip yogendra %0d using continue", i);
    continue;
   end

   // Stop the loop at the 7th yogendra
   if (i == 7) begin
    $display("Break the loop at yogendra %0d", i);
    break;
   end

   $display("Yogendra %0d executed", i);
  end

  $display("\nEnd of repeat loop");
 end
endmodule
```

Output :

Yogendra 1 executed
Yogendra 2 executed
Skip yogendra 3 using continue
Yogendra 4 executed
Yogendra 5 executed
Yogendra 6 executed
Break the loop at yogendra 7

End of repeat loop


THANK
YOU