

DAY 59: 100 DAYS VERIFICATION CHALLENGE

Topic: Procedural Statements: initial, always, final, while, do-while, break-continue, fork-join, tasks, functions, forever loop

DAY 59 CHALLENGE:

1. Difference between final and initial blocks?
2. What is the use of always_ff?
3. Difference Between Always_comb and Always@(*)?
4. What is final block?
5. What is the use of break-continue statements? Explain with an example.
6. Difference between while and do while
7. What is fork-join and types of fork-join?
8. Difference between fork-join, fork-join_any, and fork-join_none
9. What is the main limitation of fork-join in Verilog, and how is this overcome in System Verilog?
10. How to kill process in fork/join ?
11. Illustrate how the errors of passing arguments to a function in incorrect order is eliminated in System Verilog.
12. What are the features added in System Verilog for function and task?
13. Why function has 0 simulation time?
14. What is callback?
15. What is a "void" function? Why do we use it?
16. What is "ref" & "const ref" function in System Verilog?
17. What is the difference between "forever" loop & "for" loop in System Verilog?
18. What is the use of "return" statement?

DAY: 54

Topic: Procedural statements: initial, always, final, while, do-while, break-continue, Fork-join, task, Function, Forever loop.

Solⁿ ① difference Between Final and Initial Blocks?

- The initial block occurs at the start of simulation whereas
- the final block occurs at the end of the simulation without having any delays. so its the same as a function call that can execute in zero simulation time.

Solⁿ ② what is the use of always_ff?

- always_ff is for representing flip-flop.
- always_latch is for representing a latch with enable.
- always_comb is for representing a combinational circuit.
- ∴ The SV always_ff procedure can be used to model synthesizable sequential logic behaviours.
- Or always_ff is specially designed for sequential CRT like ff it executes on either the posedge or negedge of the clk & variable used on the left side within this block are not used anywhere with the module.

Solⁿ ③ difference b/w Always_comb & always@(*)?

- | always_comb | always@(*) |
|--|--|
| ① always_comb automatically executes once at time zero. | ① always@(*) wait untill a change occurs on a signal in the inferred sensitivity list. |
| ② always_comb is sensitive to changes within the contents of a function. | ② always@(*) is only sensitive to change to the arguments of a function. |
| ③ always_comb does not permit multiple processes to write to the same variable | ③ always@(*) permit multiple processes to write to the same variable. |

Solⁿ ④ what is **Final Block**?

- Final Block is used at end of simulation time. Final Block is similar to initial Block. It will execute at zero simulation time. Final Block is used Final Keyword to start the block.

Solⁿ ⑤ what is the use of **break-Continue statements**? Explain with an example.

- Break and Continue keywords are used to control the loop flow.
- Both Break & Continue keywords can be used in all supported loops (while, do, while, forever, for, foreach, repeat).
- Break keyword is used to terminate the loop prematurely. Generally, Based on certain condition the loop is terminated.
- Continue keyword is used to jump the next iteration immediately without executing statements after the Continue keyword.

Solⁿ ⑥ Difference **Between while & do while**?

while

① while loop executes till the condition in loop meets fail.

- means while loop executes only when the condition is satisfied.

do while

① whereas do while is executed first in do block then it starts to execute the while loop.

- Similarly, do while, condition is specified after the loop ~~and~~ statements, even if the condition is not satisfied, the statement within the loop are executed at least once.

Solⁿ ⑦ what is **fork-join** & types of Fork-join?

- in Verilog, fork-join construct is used to create new thread which can run in parallel to code defined in another block.
- But in fork-join we must wait for all the process that was started inside the fork-join to end before moving next state ment.

Syntax :

fork
statement;
join

∴ as in System Verilog fork join helps to Synchronization Between the processes, inside the fork-join block the procedural blocks or statement execute parallelly."

① Types of fork-join: || There are three types:

① Fork-join ② fork-join any ③ fork-join none

Solⁿ ⑧ Difference Btw fork join, fork-join any & fork-join none

① fork join: When a begin end block is used within fork-block, the entire block to execute as a single process, with each statement executing sequentially

eg:

```
fork
Statement 1;
Statement 2;
join
```

```
fork
begin
Statement 1;
Statement 2;
end
begin
Statement 3;
Statement 4;
end
join
```

② fork-join any: The statements within fork join any executes such that if any one of the process within block completed, it comes out of the loop.

Syntax

fork
...
join any

③ fork-join none: it will come out from the block it does not wait for to complete any process in the block.

Date _____
Page _____

Solⁿ ⑨ what is the main limitation of fork join in Verilog, & how is this overcome in System Verilog?

- The main limitation of fork join construct in Verilog is that its static, that is, the execution of the code beyond the join is suspended until all the processes within the fork join are completed.

Solⁿ ⑩ How to kill process in fork join?

- in fork join, all active threads can be killed by calling `disable fork`.

Solⁿ ⑪ illustrate how the errors of passing arguments to a function in incorrect order is eliminated in System Verilog.

- in argument pass by name, arguments can be passed in any order by specifying the name of the subroutine argument.

Solⁿ ⑫ what are the features added in System Verilog for function and task?

- Basic difference related to delay btw Task & function remains the same here also.
- its optional to use `"begin-end"` block in task and functions in SV. `task - endtask`, `function ... endfunction` block are sufficient.
- To declare an automatic variable in a static task/function.
- System Verilog function can be static, automatic.
- void function can be called from task as well as function.

Solⁿ ⑬ why function has 0 simulation time?

- function has no delay or timing control statements are permitted.
- can have output arguments instead of the function return.
- in the invocation of a function there must be at least one argument to be passed.
- disable statements can't be used. & function cannot have non blocking assignments.

Solⁿ ⑭ what is call-back?

- Callbacks are the empty methods available by default. If its needed we can call them anything at all about the physical circuit, therefore the functions cannot contain any timing constructs.

Solⁿ ⑮ what is "void" function? why do we use it?

- Void function are simply functions that have no return value.
- They can be used wherever a statement is allowed.
- Void represents its function that does not return which was not possible in Verilog function.
- OR If you want to ignore the return value of function, simply cast it to "void" type.

Solⁿ ⑯ what is "ref" & Const "ref" function in System Verilog?

- When ref is used the variable points to the address of the variable, if any changes made to the ref variable within the method its updated globally & the changes are visible on all the scopes.
- But when the variable is declared as Const ref, the value of the variable cannot be changed, trying it throws an error.


Solⁿ ⑰ what is the use of "return" statement?

- Return is used to return a value from the function.
- It return the value in the type of return type specified in function definition.
- OR A return statement can return a value to the calling function.

Ques:6 Difference between while and do while loop?explain with example.

- A while loop is a control flow statement that executes statements repeatedly if the condition holds true else loop terminates.
- Do while loop is same as while loop, but it is guaranteed that the loop will run for at least 1 iteration before coming out of loop.

Brought to you by



▼ Languages & Libraries

Testbench + Design

SystemVerilog/Verilog ▼

UVM / OVM ⓘ

None ▼

Other Libraries ⓘ

None ▲

OVL 2.8.1

SVUnit 2.11 ▼

☐ Enable TL-Verilog ⓘ

☐ Enable Easier UVM ⓘ

☐ Enable VUnit ⓘ

▼ Tools & Simulators ⓘ

Mentor Questa 2021.3 ▼

testbench.sv

```
1 // while loop Example
2 module tb_while;
3     int num;
4
5     initial begin
6         while(num<5)begin //while loop example
7             $display("\tnum=%0d",num);
8             num++;
9         end
10    end
11 endmodule
12
13 //do while loop Example
14 module tb_do_while;
15     int var1;
16
17     initial begin
18         do begin
19             $display("\tvalue of variable=%0d",var1);
20             var1++;
21         end
22         while(var1<5);
23     end
24 endmodule
```

-timescale 1ns/1ns

Run Options ⓘ

-voptargs=+acc=npr

Run Time: 10 ms

☐ Use run.do Tcl file

☐ Use run.bash shell script

☐ Open EPWave after run

☐ Show output file after run

☐ Download files after run

► Examples

Log Share

```
#
# vsim -voptargs=+acc=npr
# run -all
#     num=0
#     num=1
#     num=2
#     num=3
#     num=4
#     value of variable=0
#     value of variable=1
#     value of variable=2
#     value of variable=3
#     value of variable=4
```

Ques:5 Example of break-continue.

DOULOS

▼ Languages & Libraries

Testbench + Design

SystemVerilog/Verilog ▼

UVM / OVM ?

None ▼

Other Libraries ?

None ▲
OVL 2.8.1
SVUnit 2.11 ▼

☐ Enable TL-Verilog ?
☐ Enable Easier UVM ?
☐ Enable VUnit ?

▼ Tools & Simulators ?

```
1 module tb;
2   int intA[10];
3
4   initial begin
5     for(int i=0; i<10; i++)begin
6       intA[i] = i;
7     end
8
9     for(int i=0; i<10; i++)begin
10      if(i==6)
11        break; //using break keyword
12      $display("\tArray[%0d]=%0d",i,intA[i]);
13    end
14    $display("\n-----\tAfter using Continue Keyword-----");
15
16    for(int i=0; i<10; i++)begin
17      if(i==6)
18        continue;
19    end
20    $display("\tArray=%p",intA);
21  end
22 endmodule
23
24
```

-timescale 1ns/1ns

Run Options ?

-voptargs=+acc=npr

Run Time: 10 ms

☐ Use run.do Tcl file
☐ Use run.bash shell script
☐ Open EPWave after run
☐ Show output file after run
☐ Download files after run

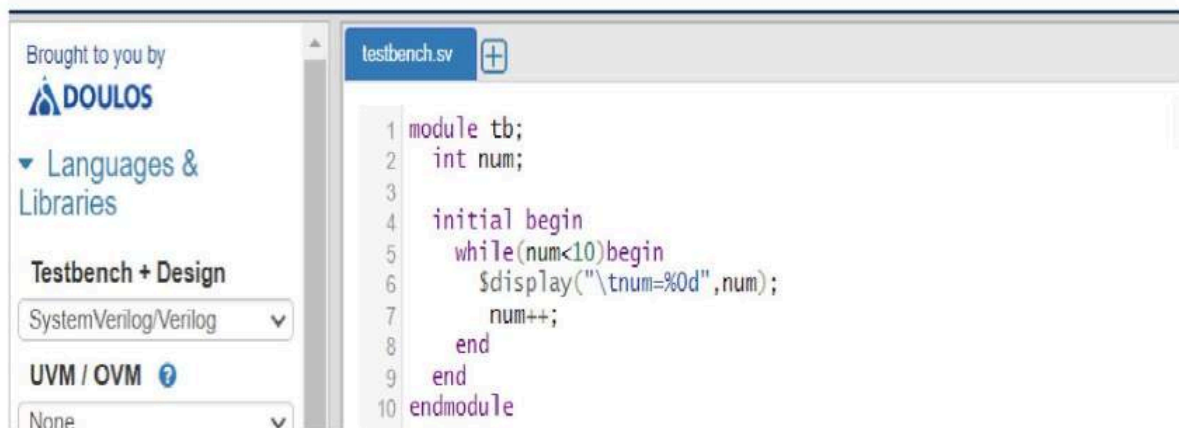
► Examples

Log Share

```
# vsim -voptargs=-tcl -npi
# run -all
#   Array[0]=0
#   Array[1]=1
#   Array[2]=2
#   Array[3]=3
#   Array[4]=4
#   Array[5]=5
#
# ----- After using Continue Keyword-----
#   Array='{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}'
# exit
```

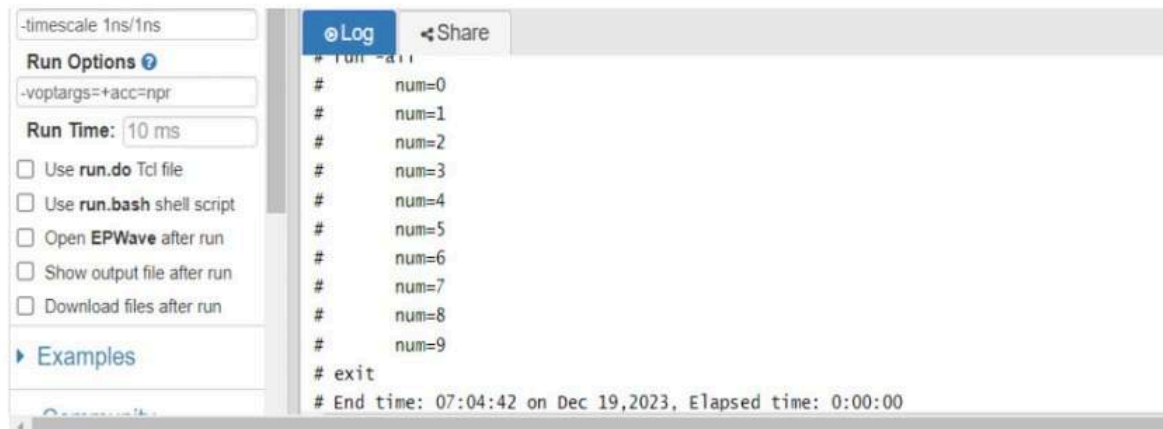

Difference between while and do while loop:

- In the while loop, a condition is checked first, and if it holds true statements will be executed else the loop terminates.
- In do while loop, even if a condition is not true, a loop can execute at once.



The screenshot shows the DOULOS IDE interface. On the left, there is a sidebar with the text "Brought to you by DOULOS" and a section titled "Languages & Libraries" containing "Testbench + Design" with dropdowns for "SystemVerilog/Verilog" and "UVM / OVM". The main editor window displays a file named "testbench.sv" with the following code:

```
1 module tb;
2   int num;
3
4   initial begin
5     while(num<10)begin
6       $display("\tnum=%0d",num);
7       num++;
8     end
9   end
10 endmodule
```




The screenshot shows the execution log of the testbench. On the left, there are "Run Options" including a timescale of "1ns/1ns", a target of "+acc=npr", and a run time of "10 ms". There are several checkboxes for additional actions like "Use run.do Tcl file", "Use run.bash shell script", "Open EPWave after run", "Show output file after run", and "Download files after run". The main log window shows the output of the simulation:

```
# Run -all
# num=0
# num=1
# num=2
# num=3
# num=4
# num=5
# num=6
# num=7
# num=8
# num=9
# exit
# End time: 07:04:42 on Dec 19,2023, Elapsed time: 0:00:00
```


Ques:2 Example for always_comb and always@(*).

Brought to you by



▼ Languages & Libraries

Testbench + Design

SystemVerilog/Verilog ▼

UVM / OVM ?

None ▼

Other Libraries ?

None
OVL 2.8.1
SVUnit 2.11

☐ Enable TL-Verilog ?

☐ Enable Enable UVM ?

testbench.sv +

SV/Verilog Testbench

```
1 module tb;
2   logic a,b,c;
3   logic y,z;
4
5   always@(a)
6     y=a^b^c;
7
8   always_comb
9     z=a^b^c;
10  initial begin
11    a=0; b=0; c=0;
12    #5 a=0; b=1; c=0;
13    #5 a=0; b=0; c=1;
14    #5 a=1; b=1; c=1;
15  end
16  initial
17    $monitor("| t=%0t  a=%0d,b=%0d,c=%0d | y=%0d | z=%0d |", $time,a,b,c,y,z);
18 endmodule
```

-timescale 1ns/1ns

Run Options ?

-voptargs=+acc=npr

Run Time: 10 ms

☐ Use run.do Tcl file

☐ Use run.bash shell script

☐ Open EPWave after run

☐ Show output file after run

☐ Download files after run

Log Share

```
# Loading work.tb(fast)
#
# vsim -voptargs=+acc=npr
# run -all
# | t=0  a=0,b=0,c=0 | y=0 | z=0 |
# | t=5  a=0,b=1,c=0 | y=0 | z=1 |
# | t=10 a=0,b=0,c=1 | y=0 | z=1 |
# | t=15 a=1,b=1,c=1 | y=1 | z=1 |
# exit
```


Ques:16 Example of "ref" and "Const ref" function in System Verilog.

The screenshot shows the DOULOS IDE interface. On the left, the 'Languages & Libraries' panel is set to 'SystemVerilog/Verilog'. The 'Testbench + Design' section shows 'SystemVerilog/Verilog' selected. The 'UVM / OVM' section is set to 'None'. The 'Other Libraries' section lists 'None', 'OVL 2.8.1', and 'SVUnit 2.11'. The 'Run Options' section is expanded, showing options like 'Use run.do Tcl file', 'Use run.bash shell script', 'Open EPWave after run', 'Show output file after run', and 'Download files after run'. The main editor displays a System Verilog code snippet for a module 'tb2' that uses a function 'sum' with 'ref' parameters. The code is as follows:

```
1 //Pass by reference
2 module tb2;
3   int num1,num2,result;
4
5   function int sum( ref int num1,num2);
6     num1 = num1+num2;
7     num2 = num1+num2;
8     return num1+num2;
9   endfunction
10
11   initial begin
12     num1 = 10;
13     num2 = 20;
14     $display("\n\tnum1=%0d,num2=%0d",num1,num2);
15     result = sum(num1,num2);
16     $display("\tnum1=%0d,num2=%0d,result=%0d",num1,num2,result);
17   end
18 endmodule
```

The output window shows the simulation results:

```
CPU time: .219 seconds to compile + .233 seconds to elab + .255 seconds to link
Chronologic VCS simulator copyright 1991-2021
Contains Synopsys proprietary information.
Compiler version S-2021.09; Runtime version S-2021.09; Dec 19 12:36 2023

num1=10,num2=20
num1=30,num2=50,result=80
VCS Simulation Report
Time: 0 ns
```

The screenshot shows the DOULOS IDE interface with the same settings as the first image. The main editor displays a similar System Verilog code snippet, but the function 'sum' is declared with 'const ref' parameters. The code is as follows:

```
1 //Pass by reference
2 module th2;
3   int num1,num2,result;
4
5   function int sum( const ref int num1,num2);
6     num1 = num1+num2;
7     num2 = num1+num2;
8     return num1+num2;
9   endfunction
10
11   initial begin
12     num1 = 10;
13     num2 = 20;
14     $display("\tnum1=%0d,num2=%0d",num1,num2);
15     result = sum(num1,num2);
16     $display("\tnum1=%0d,num2=%0d,result=%0d",num1,num2,result);
17   end
18 endmodule
```

The output window shows a compilation error:

```
Error-[IUCV] Invalid use of 'const'
testbench.sv, 7
'const' variable is either driven or connected to a non-const variable.
Variable 'num2' declared as 'const' cannot be used in this context
Source info: num2 = (num1 + num2);

2 errors
CPU time: .150 seconds to compile
```