# DAY 33 - 111 DAYS VERIFICATION CHALLENGE

Topic: System Tasks

Skill: Verilog

DAY 33 CHALLENGE:

## 1. Explain the difference between $display, $monitor & $strobe with an example?

**$display:**

- Displays information once when it is executed.
- The output is generated immediately when the statement is executed.

**$monitor:**

- Continuously monitors and displays information whenever any of the variables in its list change.
- The output is generated whenever there is a change in the monitored variables.

**$strobe:**

- Similar to $display, but it schedules the output to be displayed at the end of the current simulation time step.
- Ensures that the output reflects the final values of variables at the end of the time step.

```
module display_example;
  reg a, b, c;
  initial begin
    a = 0; b = 0; c = 0;
    // $display displays immediately when executed
    $display("Initial values - Time: %0t, a: %b, b: %b, c: %b", $time, a, b,
c);
    // $monitor monitors changes and displays when any variable changes
    $monitor("Monitor - Time: %0t, a: %b, b: %b, c: %b", $time, a, b, c);
    // Change the values with some delays
```

```
    #5 a = 1;
    #5 b = 1;
    #5 c = 1;
    // $strobe will display at the end of the current time step
    #10 $strobe("Strobe - Time: %0t, a: %b, b: %b, c: %b", $time, a, b, c);

    // Final values at the end
    #10 $display("Final values - Time: %0t, a: %b, b: %b, c: %b", $time, a,
b, c);
  end
endmodule
```

**output :**

```
Initial values - Time: 0, a: 0, b: 0, c: 0
Monitor - Time: 5, a: 1, b: 0, c: 0
Monitor - Time: 10, a: 1, b: 1, c: 0
Monitor - Time: 15, a: 1, b: 1, c: 1
Strobe - Time: 30, a: 1, b: 1, c: 1
Final values - Time: 40, a: 1, b: 1, c: 1
```

## 2. Which system tasks are used to switch monitoring on & off? Explain with example.

In Verilog, you can control the monitoring of signals using the system tasks $monitoron and $monitoroff. These tasks enable you to turn the monitoring feature on and off, respectively, allowing you to control when the $monitor task is actively displaying information.

**Example :**

```
module monitor_control_example;
  reg a, b, c;
  initial begin
    a = 0; b = 0; c = 0;
    $monitor("Monitor - Time: %0t, a: %b, b: %b, c: %b", $time, a, b, c);
    #5 a = 1; b = 0; c = 0;
    #5 b = 1;
    // Turn off monitoring
    #10 $monitoroff;
```

```
    // Change values while monitoring is off
    #5 a = 0; c = 1;
    // Turn on monitoring
    #10 $monitoron;
    // Change values while monitoring is on
    #5 b = 0;
    #10 $finish;
  end
endmodule
```

**Output :**

```
Monitor - Time: 5, a: 1, b: 0, c: 0
Monitor - Time: 10, a: 1, b: 1, c: 0
Monitor - Time: 25, a: 0, b: 1, c: 1
Monitor - Time: 30, a: 0, b: 0, c: 1
```

## 3. Explain the following System tasks with an example:

### I. $stop

The $stop system task is used to pause the simulation and return control to the simulator. It is typically used for debugging purposes, allowing you to examine the state of the simulation at a specific point. After a $stop is encountered, you can choose to continue the simulation, restart it, or stop it completely.

### II. $finish

The $finish system task is used to terminate the simulation completely. When $finish is encountered, the simulation stops and control is returned to the operating system. This task is typically used at the end of the testbench to indicate that the simulation is complete.

**Example :**

```
module stop_finish_example;
  reg a, b, c;
  initial begin
    a = 0; b = 0; c = 0;
    $display("Initial values - Time: %0t, a: %b, b: %b, c: %b", $time, a, b,
c);
    #5 a = 1; b = 1;
```

```
    // Use $stop to pause the simulation
    #5 $stop;
    // Change the values after $stop
    #5 c = 1;
    // Use $finish to terminate the simulation
    #10 $finish;
  end
endmodule
```

## Explain following compiler directives with an example:

### I. `define

The `define directive is used to define macros in Verilog. It allows you to create constants or parameterized text that can be used throughout your code. This can make your code more readable and maintainable.

```
`define WIDTH 8
`define MESSAGE "Hello, Verilog!"

module define_example;
  reg [`WIDTH-1:0] data;

  initial begin
    data = 8'b10101010;
    $display(`MESSAGE);
    $display("Data: %b", data);
  end
endmodule
```

### II. `include

The `include directive is used to include the contents of another file into the current file. This can be useful for including common definitions, modules, or testbenches, thereby promoting code reuse.

```
`include "defines.v"

module include_example;
  reg [`WIDTH-1:0] data;

  initial begin
    data = 8'b10101010;
    $display(`MESSAGE);
    $display("Data: %b", data);
  end
endmodule
```

## 5. Explain the difference between == and === with an example.

**==**

- Purpose: Checks for logical equality.

 Behavior: Compares two values, treating x and z as unknowns.

 Result: If any bit of the operands is x or z, the result is x (unknown).

===

 Purpose: Checks for case equality.

 Behavior: Compares two values, considering x and z as valid bits.

 Result: Returns 1 if the operands are exactly the same, including x and z values, and 0 otherwise.

**Example :**

```
module equality_example;
  reg [3:0] a, b;
  initial begin
    a = 4'b1010;
    b = 4'b1010;
    $display("a == b : %b", a == b);  // Logical equality: 1 (true)
    $display("a === b: %b", a === b); // Case equality:  1 (true)
    b = 4'b10x0;
    $display("a == b : %b", a == b);  // Logical equality: x (unknown)
    $display("a === b: %b", a === b); // Case equality:  0 (false)
    a = 4'b10x0;
    b = 4'b10x0;
    $display("a == b : %b", a == b);  // Logical equality: x (unknown)
    $display("a === b: %b", a === b); // Case equality:  1 (true)
  end
endmodule
```

## 6. What is a sensitivity list?

A sensitivity list in Verilog specifies the set of signals that trigger the execution of a procedural block, such as an always block. When any signal in the sensitivity list changes value, the procedural block is executed.

## 7. Explain deposit and force commands.

In Verilog simulations, deposit and force are commands used to set the value of variables or signals during simulation.

- Deposit (deposit):

- Used to set the value of a signal or variable immediately without holding it.
- The signal can still change due to other processes in the simulation.

- Force (force):
  - Used to set and hold a signal or variable to a specific value, overriding other assignments.
  - The signal remains at this value until it is released using the release command.

```
initial begin
  // Deposit example
  a = 0;
  #5 a = 1; // After 5 time units, 'a' is set to 1 (deposit)

  // Force example
  force b = 1; // 'b' is forced to 1
  #5 release b; // After 5 time units, 'b' is released and can change freely
end
```

## 8. Explain freeze & drive with an example.

 Freeze:

- Used to hold the current value of a signal or variable, preventing it from changing.
- Typically used in simulations to stabilize a signal temporarily.

 Drive:

- Used to drive a signal to a specific value, overriding other assignments.
- Similar to force, but it typically implies continuous driving.

```
reg a, b, c;
initial begin
  a = 1;
  b = 0;
  // Freeze example
  freeze a; // Hold the current value of 'a'
  #10 unfreeze a; // Allow 'a' to change after 10 time units
```

```
  // Drive example
  drive c = 1; // Continuously drive 'c' to 1
  #10 stop_drive c; // Stop driving 'c' after 10 time units
end
```

## 9. What does timescale 1 Ns/1 Ps mean?

In Verilog, the `timescale directive specifies the time unit and the time precision of the simulation.

- 1 ns: Time unit, the base time measurement unit in the simulation.
- 1 ps: Time precision, the smallest time step that can be resolved in the simulation.

## 10. Between variable and signal, which will be updated first & why?

In Verilog, variables (reg, integer, etc.) and signals (wire, tri, etc.) are updated differently within procedural blocks and concurrent blocks.

- Variables (reg):
    - Updated immediately within the procedural blocks (always, initial, etc.) when the block executes.
    - Changes take effect in the current simulation time step.
- Signals (wire):
    - Updated through continuous assignments or concurrent blocks.
    - Changes take effect at the end of the current simulation time step after all procedural blocks have executed.