1. Explain the difference between $display $monitor and $strobe with an example.

   $display
   - Prints the values immediately when the statement is executed.
   - Executes once each time it is encountered.
   - Example:
     ```
     initial begin
                         $display("Time: %0t, A = %0d", $time, A);
                          end
     ```

   $monitor
   - Continuously monitors and prints values whenever one of the variables in the statement changes.
   - Only needs to be declared once.
   - Example:
     ```
     initial begin
                         $monitor("Time: %0t, A = %0d", $time, A);
                          end
     ```

   $strobe
   - It is similar to `$display`, but it prints values at the end of the current simulation time step (after all events at that time are processed).
   - Ensures that all updates for the current time step are completed before printing.
   - Example:
     ```
     initial begin
                         $strobe("Time: %0t, A = %0d", $time, A);
                          end
     ```
2. Which system tasks are used to switch monitoring on and off? Explain with an example.
   - `$monitoron`: Resumes the monitoring that was previously stopped by `$monitoroff`.
   - `$monitoroff`: Suspends the monitoring that was started by `$monitor` without removing the monitoring statement.

   Example:
   ```
   module monitor_example;
             reg [3:0] A, B;
             initial begin
                  A = 4'b0000;
                  B = 4'b0001;

                  // Start monitoring
                  $monitor("Time: %0t, A = %0b, B = %0b", $time, A, B);

                  #5 A = 4'b0101;  // Update A at time 5
                  #5 B = 4'b1010;  // Update B at time 10

                  // Turn off monitoring
                  #10 $monitoroff;
                  A = 4'b1111;     // No output at time 20

                  // Turn on monitoring again
                  #5 $monitoron;
                  B = 4'b0000;     // Monitoring resumes at time 25
                end
              endmodule
   ```

3. Explain the following system tasks with an example:
    a. $stop
        - Suspends the simulation but does not end it.
        - Allows the user to inspect the current state of the simulation (usually in a simulator's interactive debugging mode).
        - The simulation can be resumed after stopping.
        - Commonly used during debugging to pause the simulation at specific points.
    b. $finish
        - Terminates the simulation entirely and exits the simulator.
        - Used when the simulation has completed its task or when you no longer need to simulate further.
        - No further simulation is possible after `$finish` is called.

    Example:
```
module stop_finish_example;
  reg [3:0] A;

  initial begin
    A = 4'b0000;

    // Some simulation steps
    #5 A = 4'b0101;

    // Pause the simulation here
    $display("Pausing simulation at time %0t, A = %0b", $time, A);
    $stop;  // Simulation pauses here; user can inspect the state

    // Resume simulation
    #5 A = 4'b1010;

    // End the simulation
    $display("Finishing simulation at time %0t, A = %0b", $time, A);
    $finish;  // Simulation ends here
  end
endmodule
```
4. Explain the following compiler directives with an example:
    a. ‘define
        - Used to create macros or symbolic constants that can be used throughout the code.
        - These macros are replaced by their value during compilation.
        - Useful for defining constants or simplifying code.

       Example:`define WIDTH 8
    b. ‘include
        - Used to include an external file into the current file.
        - This directive imports the contents of the specified file during compilation.
        - Commonly used to include reusable code, like header files, function definitions, or parameter files.
          Example: `include "params.v"

5. Explain the difference between == and === with an example.
   ==
   - Checks for logical equality.
   - It does not consider X or Z values in the comparison. If any operand has an X or Z, the result will be unknown (X).
   - They are used for general comparisons when X or Z values are unexpected.

   ===
   - Checks for exact bitwise equality, including X and Z values.
   - It considers X and Z as valid values and compares them directly. If both sides have X or Z in the same positions, the comparison can return true.
   - Useful when comparing for exact matches, including X and Z.

   Example:
```
module equality_example;
  reg [3:0] A, B;

  initial begin
    A = 4'b1010;  // A = 1010
    B = 4'b10X0;  // B = 10X0 (unknown value in the 3rd bit)

    // Using logical equality (==)
    if (A == B)
      $display("A == B is TRUE");
    else
      $display("A == B is FALSE");

    // Using case equality (===)
    if (A === B)
      $display("A === B is TRUE");
    else
      $display("A === B is FALSE");
  end
endmodule
```
6. What is a sensitivity list?
   A sensitivity list in Verilog is a list of signals that trigger the execution of a block of code, specifically within always blocks. When any signal in the sensitivity list changes value, the code inside the `always` block is executed. Sensitivity lists are essential for defining combinational or sequential logic.
7. Explain deposit and force commands.
   Force
   - The `force` command overrides the value of a signal or variable and forces it to hold a specific value until explicitly released using the `release` command.
   - It is commonly used for overriding signal values in testbenches, debugging, or during temporary control of signals.
   - Once forced, the signal retains the assigned value regardless of what the actual logic or other assignments would normally drive it to.

   Deposit

- The `deposit` command assigns a value to a signal immediately, but only once. Unlike `force`, it doesn't override the signal permanently; the signal can be updated again by other logic after the deposit.
- It is generally used for initializing or setting a value in a simulation without overriding normal behaviour permanently.
- Deposit sets the value of a signal once, but the signal can still be updated by other logic afterwards.

8. Explain freeze and drive with an example.
   Freeze
   - Freezing a signal means holding its value constant, regardless of any further changes in the design logic.
   - This can be achieved using the `force` command, which forces a signal to maintain a specific value.

   Drive
   - Driving a signal means assigning a value to it, typically as part of normal design logic.
   - Signals are driven by continuous assignments or procedural blocks in Verilog.
   - In contrast to freezing (forcing), driving signals can be done through standard assignments (= for procedural, or `assign` for continuous).
9. What does timescale 1ns/1ps mean?
   The `timescale` directive in Verilog defines the time unit and time precision for the simulation. The 1ns represents the time unit and 1ps represents the resolution or precision with which time is measured in the simulation.
10. Between variable and signal, which will update first and why?
    Signals (`wire`) typically update before variables (`reg`) in a simulation time step because continuous assignments (`assign`) are processed as part of the update event in the simulation cycle. Variables (`reg`), assigned in procedural blocks, update at the end of the current time step, after the procedural block has been evaluated.