1. Differentiate between task and function.
   Task:

   - Can have time delays (#, @, or wait).
   - Can call other tasks and functions.
   - Does not return a value.
   - Used for more complex operations.

   Function:

   - Cannot have time delays.
   - It can only call other functions.
   - Must return a value.
   - Used for combinational logic.
2. What is PLI? What are its applications?
   PLI is an interface in Verilog that allows integration between Verilog code and external programming languages like C. It enables calling functions and accessing simulation data outside of Verilog.
   Applications
   - Extending simulator capabilities.
   - Performing operations not supported in Verilog directly.
   - Debugging and data analysis.
3. What are virtual and pure virtual functions in Verilog?
   Virtual Functions: Not directly applicable in Verilog as it doesn't support object-oriented concepts like in C++.
   Pure Virtual Functions: Refer to abstract methods in object-oriented languages; Verilog doesn't have this concept.
4. What is DPI? What are its applications?
   DPI is a standardised interface in SystemVerilog (an extension of Verilog) to call functions written in C/C++ directly from SystemVerilog and vice versa.
   Applications
   - Allows to combine high-level software and hardware design.
   - Used for integrating custom algorithms and models written in C/C++ into Verilog simulations.
5. Explain the following system tasks in Verilog:
   a. $random
      Generates a random number.
   b. $readmemb
      Reads memory contents from a binary file into a memory array.
   c. $readmemh
      Reads memory contents from a hex file into a memory array.
6. Explain the following file support operations in Verilog:
   a. $fopen
      Opens a file for reading or writing.
   b. $fscan
      Reads formatted data from a file.
   c. $fdisplay
      Writes formatted data to a file.
   d. $fwrite
      Writes unformatted data to a file.

   e. $fclose

      Closes an open file.

7. Design and execute the following:
   a. T flip-flop

```
module T_flipflop(
   input wire T,
   input wire clk,
   input wire reset,
   output reg Q
);
   always @(posedge clk or posedge reset) begin
      if (reset)
         Q <= 0;
      else if (T)
         Q <= ~Q;
   end
endmodule
```

   b. 4-bit shift register

```
module shift_register(
   input wire clk,
   input wire reset,
   input wire serial_in,
   output reg [3:0] Q
);
   always @(posedge clk or posedge reset) begin
      if (reset)
         Q <= 4'b0000;
      else
         Q <= {Q[2:0], serial_in};
   end
endmodule
```

   c. FSM to detect the overlapping sequence 01010

```
module fsm_01010 (
   input wire clk,
   input wire reset,
   input wire din,
   output reg dout
);
   reg [2:0] state;
   parameter S0 = 3'b000, S1 = 3'b001, S2 = 3'b010, S3 = 3'b011, S4 = 3'b100;

   always @(posedge clk or posedge reset) begin
      if (reset)
         state <= S0;
      else begin
         case (state)
            S0: state <= (din == 0) ? S1 : S0;
            S1: state <= (din == 1) ? S2 : S1;
            S2: state <= (din == 0) ? S3 : S0;
            S3: state <= (din == 1) ? S4 : S1;
            S4: state <= (din == 0) ? S3 : S0;
            default: state <= S0;
        endcase
    end
end

assign dout = (state == S4);
endmodule
```