1. Explain with an example, the different methods to code a clock in Verilog.
    - Using initial block: This method uses an initial block and an always block to generate a clock signal for simulation.

```
module tb_clock_generator;

  reg clk;  // Clock signal

  // Clock generation
  initial begin
    clk = 0;             // Initialize clock
    forever #5 clk = ~clk; // Toggle clock every 5 time units
  end

  initial begin
    // Testbench code
    #100;              // Run simulation for 100 time units
    $finish;           // End simulation
  end
endmodule
```

    - Using clock divider: A clock divider takes a high-frequency clock and generates a slower clock. This is useful for creating different clock frequencies.

```
module clock_divider (
  input wire clk_in,      // Input high-frequency clock
  input wire reset,       // Reset signal
  output reg clk_out      // Output slower clock
);

  reg [15:0] counter;     // Counter for dividing the clock

  always @(posedge clk_in or posedge reset) begin
    if (reset) begin
      counter <= 0;    // Reset counter
      clk_out <= 0;   // Reset output clock
    end else begin
      if (counter == 16'd49999) begin
        counter <= 0;    // Reset counter
        clk_out <= ~clk_out;  // Toggle the output clock
      end else begin
        counter <= counter + 1;  // Increment counter
      end
    end
  end

endmodule
```

    - **Using a Periodic Clock Generation Task:**For more complex scenarios, you can use a Verilog task to generate a clock.

```
module tb_clock_task;
```

```
    reg clk;  // Clock signal

    // Task to generate clock
    task generate_clock;
        input reg clk;
        begin
            clk = 0;
            forever #5 clk = ~clk; // Toggle clock every 5 time units
        end
    endtask

    // Call task in initial block
    initial begin
        generate_clock(clk);
        #100;  // Run simulation for 100 time units
        $finish; // End simulation
    end
endmodule
```

2. Write a Verilog code for:
    a. Synchronous reset.

```
module sync_reset (
    input wire clk,       // Clock signal
    input wire reset,     // Synchronous reset signal
    input wire [7:0] data_in, // Input data
    output reg [7:0] data_out // Output data
);

    always @(posedge clk) begin
        if (reset) begin
            data_out <= 8'b0;  // Reset the output data to zero
        end else begin
            data_out <= data_in; // Pass input data to output
        end
    end

endmodule
```

    b. Asynchronous reset.

```
module async_reset (
    input wire clk,       // Clock signal
    input wire reset,     // Asynchronous reset signal
    input wire [7:0] data_in, // Input data
    output reg [7:0] data_out // Output data
);

    always @(posedge clk or posedge reset) begin
```

```
        if (reset) begin
            data_out <= 8'b0;  // Reset the output data to zero immediately
        end else begin
            data_out <= data_in; // Pass input data to output
        end
    end
endmodule
```

3. Write a Verilog code to swap the contents of two registers:
    a. With a temporary register.

```
module swap_with_temp (
    input wire clk,      // Clock signal
    input wire reset,    // Reset signal
    input wire [7:0] A,  // Input register A
    input wire [7:0] B,  // Input register B
    output reg [7:0] outA, // Output register A after swap
    output reg [7:0] outB  // Output register B after swap
);

    reg [7:0] temp;  // Temporary register

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            outA = 8'b0;
            outB = 8'b0;
        end else begin
            temp = A;      // Store A in temp
            outA = B;      // Assign B to outA
            outB = temp;   // Assign temp (original A) to outB
        end
    end

endmodule
```

    b. Without a temporary register.

```
module swap_without_temp (
    input wire clk,      // Clock signal
    input wire reset,    // Reset signal
    input wire [7:0] A,  // Input register A
    input wire [7:0] B,  // Input register B
    output reg [7:0] outA, // Output register A after swap
    output reg [7:0] outB  // Output register B after swap
);

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            outA <= 8'b0;
            outB <= 8'b0;
        end else begin
```

```
            outA <= B;
            outB <= A;
          end
      end
    endmodule
```

4. Design the following using Verilog:
   a. XNOR gate.

```
module xnor_gate(input a, input b, output y);
        assign y = (a & b) | (~a & ~b)
endmodule
```

   b. D-FF.

```
module dff(input clk, input d, output q);
        reg d;
        initial d = 0;
        always @ (posedge clk) begin
                q <= d;
                        End
endmodule
```

   c. 2:1 MUX.

```
module mux2to1 (
   input wire a,       // Input 1
   input wire b,       // Input 2
   input wire sel,     // Select line
   output wire y       // Output
);

   assign y = (sel) ? b : a;  // If sel is 1, output b; otherwise, output a

endmodule
```

   d. 2-bit full adder.

```
module full_adder_2bit (
   input wire [1:0] A,  // 2-bit input A
   input wire [1:0] B,  // 2-bit input B
   input wire Cin,      // Carry-in
   output wire [1:0] Sum, // 2-bit Sum
   output wire Cout     // Carry-out
);

   // Internal wires for the carry bits
   wire c1, c2;

   // Full adder for the least significant bit (LSB)
   full_adder fa0 (.a(A[0]), .b(B[0]), .cin(Cin), .sum(Sum[0]), .cout(c1) );
   // Full adder for the most significant bit (MSB)
   full_adder fa1 (.a(A[1]), .b(B[1]), .cin(c1), .sum(Sum[1]), .cout(Cout));
endmodule

// Single-bit full adder module
```

```
module full_adder (
    input wire a,       // Input A
    input wire b,       // Input B
    input wire cin,     // Carry-in
    output wire sum,    // Sum output
    output wire cout    // Carry-out
);

    assign {cout, sum} = a + b + cin;

endmodule
```

5. Design a Verilog code to execute the following truth table:

Inputs		Outputs	
a	b	x	y
0	0	1	0
0	1	1	1
1	0	0	0
1	1	0	1

```
module truth_table(input a, input b, output x, output y);
        assign x = ~a;
        assign y =   b;
endmodule
```