Que 1:

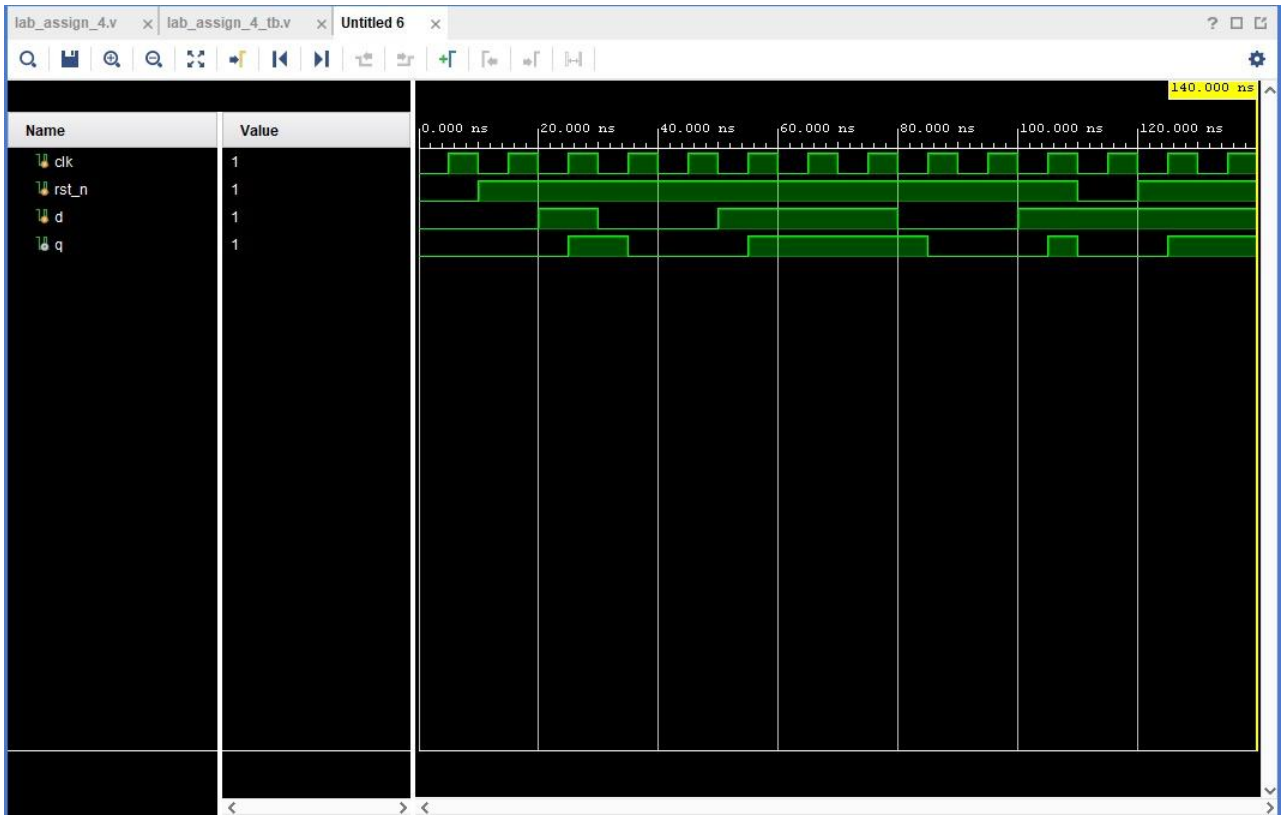
Non-blocking :

```
module d_flip_flop (
   input wire clk,
   input wire reset_n, // Active low reset
   input wire d,
   output reg q
);
   always @(posedge clk or negedge reset_n) begin
      if (!reset_n)
         q <= 0; // Reset the flip-flop
      else
         q <= d; // Capture the input
   end
endmodule
```

testbench:

```
module lab_assign_4_tb();
reg clk;
reg rst_n;
reg d;
wire q;
lab_assign_4 uut(
   .clk(clk),
   .rst_n(rst_n),
   .d(d),
   .q(q)
);
always #5 clk=~clk;
   initial begin
      clk=0;
      rst_n=0;
      d=0;
      #10 rst_n=1;
      #10 d=1;
      #10 d=0;
      #20 d=1;

      #10 rst_n=1;
      #10 d=1;
      #10 d=0;
      #20 d=1;

      #10 rst_n=0;
      #10 rst_n=1;

      #20 $finish;
```

```
    end

endmodule
```



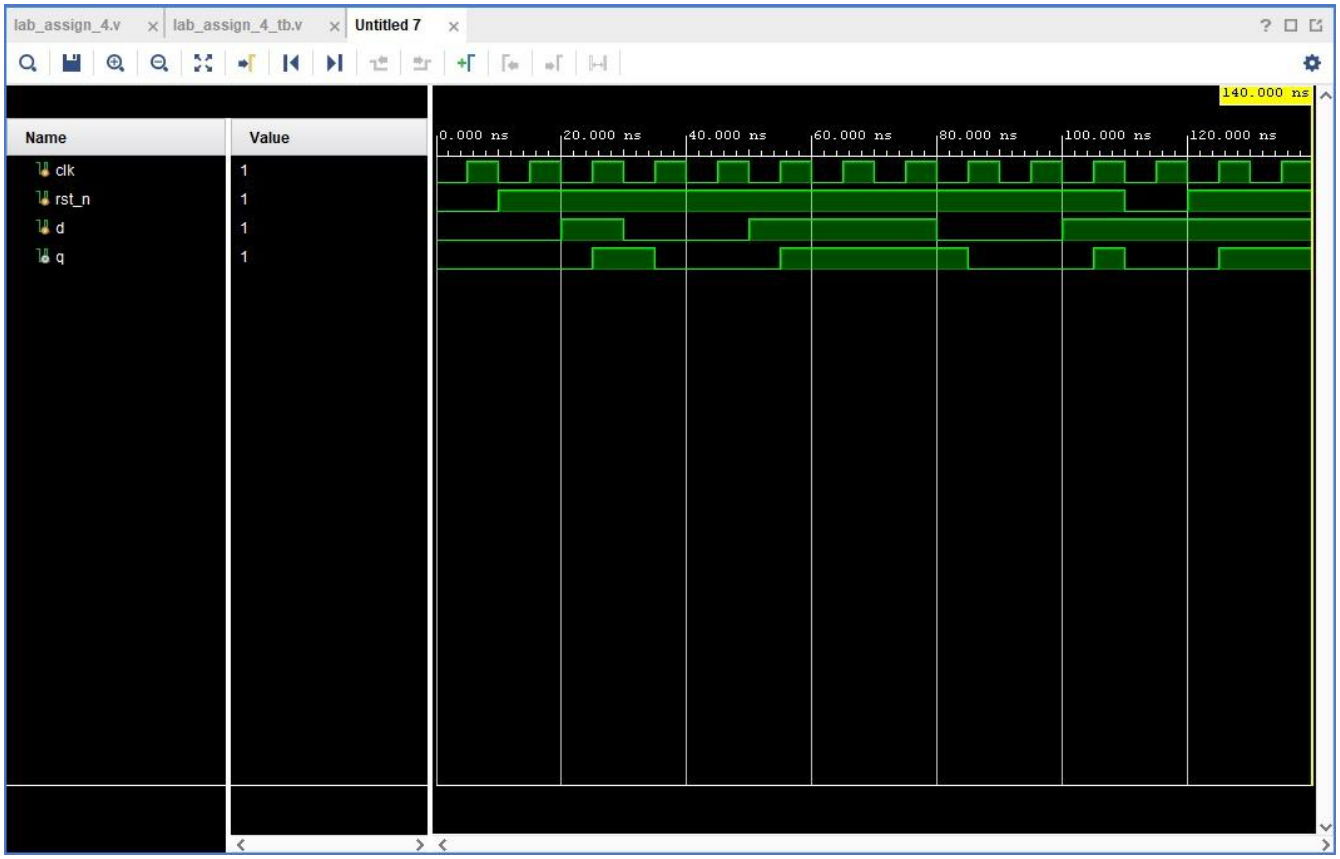
blocking:

```
module d_flip_flop (
    input wire clk,
    input wire reset_n, // Active low reset
    input wire d,
    output reg q
);
    always @(posedge clk or negedge reset_n) begin
        if (!reset_n)
            q = 0; // Reset the flip-flop
        else
            q = d; // Capture the input
    end
endmodule
```

test bench:

```
module lab_assign_4_tb();
reg clk;
reg rst_n;
reg d;
wire q;
```

```
lab_assign_4 uut(
   .clk(clk),
   .rst_n(rst_n),
   .d(d),
   .q(q)
);
always #5 clk=~clk;
   initial begin
      clk=0;
      rst_n=0;
      d=0;
      #10 rst_n=1;
      #10 d=1;
      #10 d=0;
      #20 d=1;

      #10 rst_n=1;
      #10 d=1;
      #10 d=0;
      #20 d=1;

      #10 rst_n=0;
      #10 rst_n=1;

      #20 $finish;
   end
endmodule
```



explanation:
no, blocking and non-blocking assignments in a d flip flop outputs seem similar. Difference lies in a way they are executed during simulations.

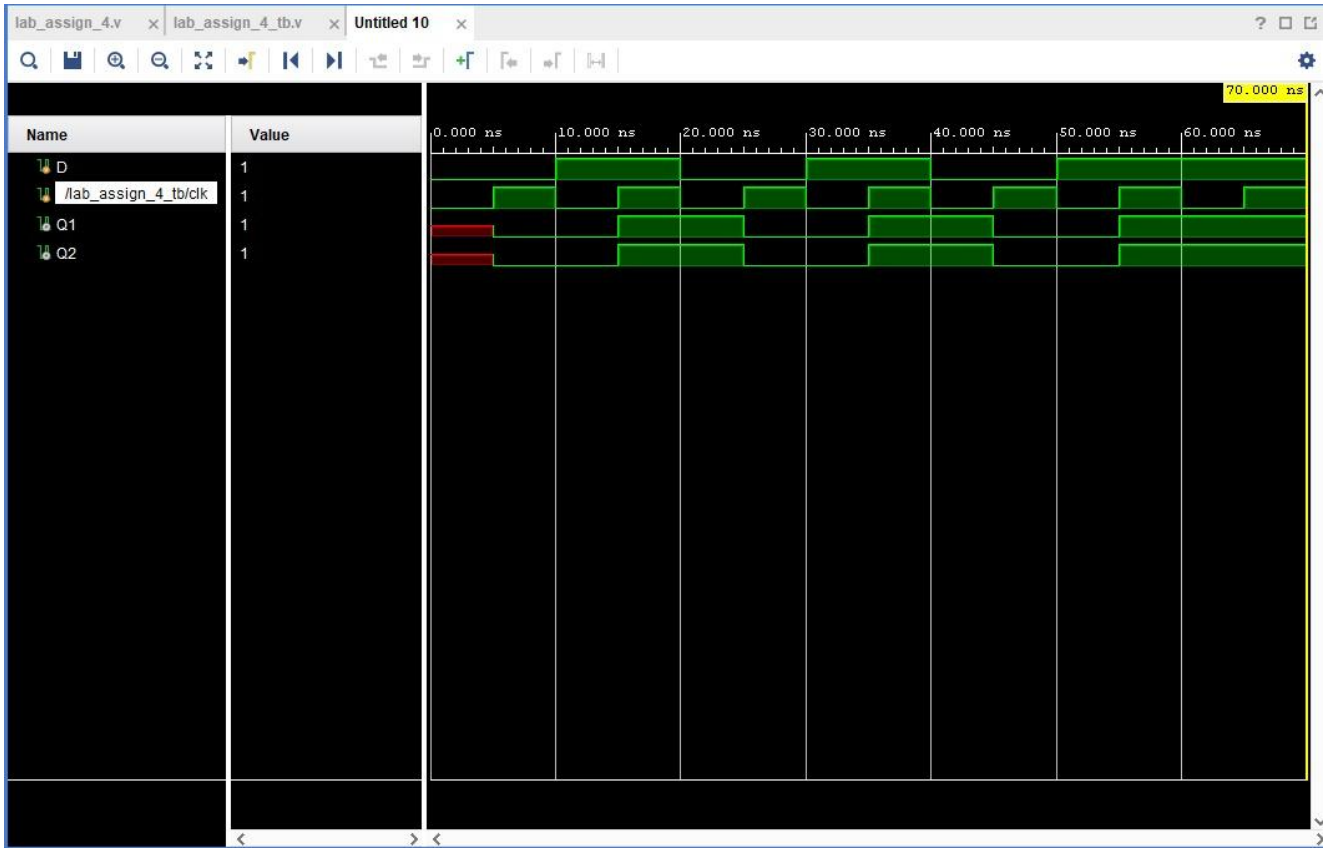
2-bitshifter(blocking)

```
module dff(D, clk, Q1, Q2); input D, clk;
output reg Q1, Q2;
always @(posedge clk) begin
Q1=D;
Q2=Q1;
end
endmodule
```

testbench:

```
module tb_dff;
   // Inputs
   reg D;
   reg clk;
   // Outputs
   wire Q1, Q2;

   // Instantiate the D flip-flop module
   dff uut (
```

```
    .D(D),
    .clk(clk),
    .Q1(Q1),
    .Q2(Q2)
);

// Clock generation
initial begin
    clk = 0;
    forever #5 clk = ~clk; // Generate a clock with a period of 10 units
end

// Stimulus
initial begin
    // Initialize input
    D = 0;

    // Apply test cases
    #10 D = 1;  // Change D to 1
    #10 D = 0;  // Change D to 0
    #10 D = 1;  // Change D to 1
    #10 D = 0;  // Change D to 0
    #10 D = 1;  // Change D to 1

    #20 $finish; // End simulation
end

// Monitor outputs
initial begin
    $monitor("Time: %0t | D: %b | Q1: %b | Q2: %b", $time, D, Q1, Q2);
end
```



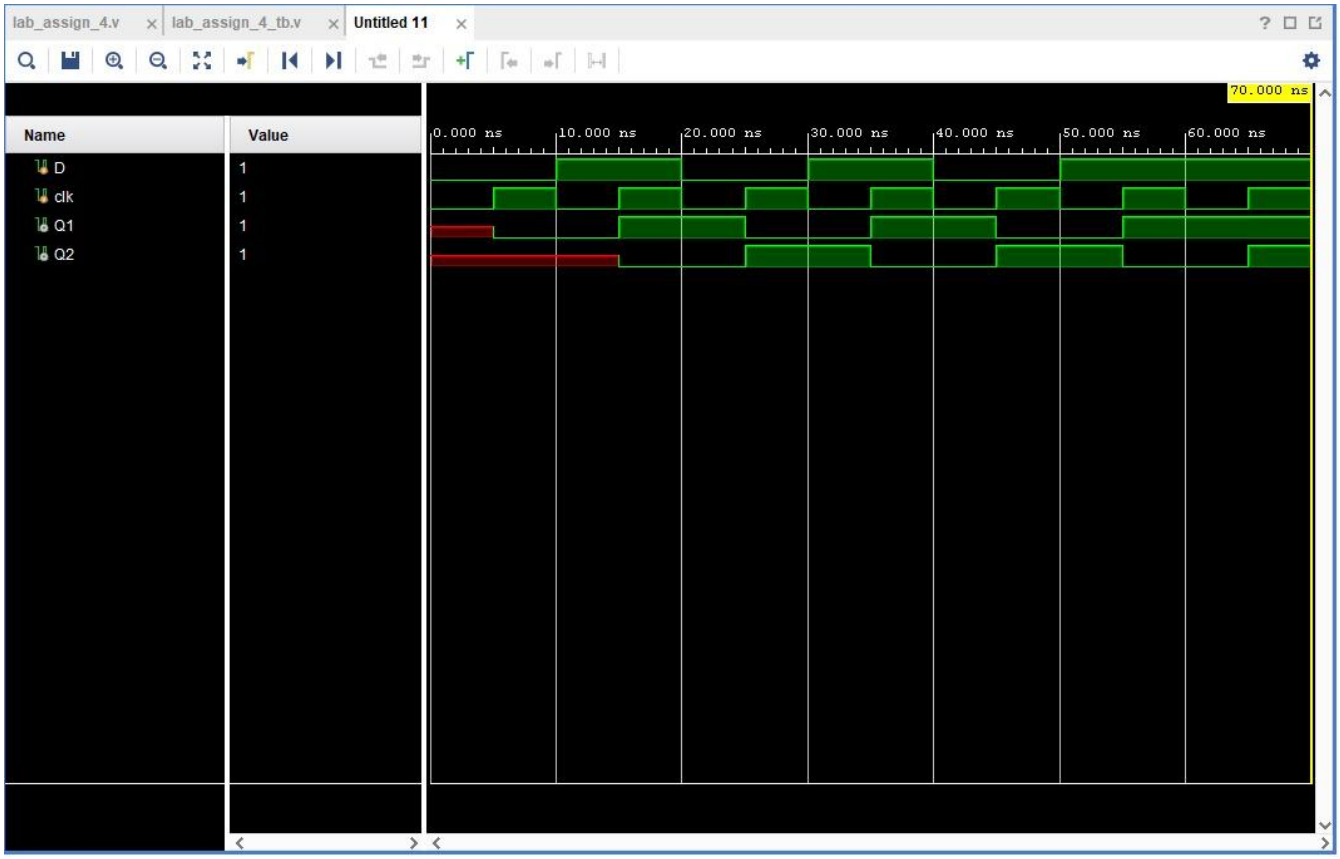
2-bitshifter(non-blocking)

```
module dff(D, clk, Q1, Q2); input D, clk;
output reg Q1, Q2;
always @(posedge clk) begin
Q1=D;
Q2=Q1;
end

endmodule
```

testbench:

```
module tb_dff;
   // Inputs
   reg D;
   reg clk;
   // Outputs
   wire Q1, Q2;

   // Instantiate the D flip-flop module
   dff uut (
```

```
    .D(D),
    .clk(clk),
    .Q1(Q1),
    .Q2(Q2)
);

// Clock generation
initial begin
    clk = 0;
    forever #5 clk = ~clk; // Generate a clock with a period of 10 units
end

// Stimulus
initial begin
    // Initialize input
    D = 0;

    // Apply test cases
    #10 D = 1;  // Change D to 1
    #10 D = 0;  // Change D to 0
    #10 D = 1;  // Change D to 1
    #10 D = 0;  // Change D to 0
    #10 D = 1;  // Change D to 1

    #20 $finish; // End simulation
end

// Monitor outputs
initial begin
    $monitor("Time: %0t | D: %b | Q1: %b | Q2: %b", $time, D, Q1, Q2);
end
```



explanation:
here there is a shift in output in non-blocking. When you compare both the blocking and non-blocking.

Que 2:

Verilog code:

```
module assignment_4(
input wire [3:0] num,
output reg is_prime
);
   always @(*) begin
      case (num)
         4'd2, 4'd3, 4'd5, 4'd7, 4'd11, 4'd13: is_prime=1;
         default: is_prime=0;
      endcase
   end
endmodule
```

testbench code:

```
module assignment_4_tb();
reg [3:0] num;
wire is_prime;

assignment_4 uut(
   .num(num),
   .is_prime(is_prime)
);
```

```
    initial begin
      for(num=0;num<16;num=num+1)begin
      #10;
      end
    end
endmodule
```

simulation results: