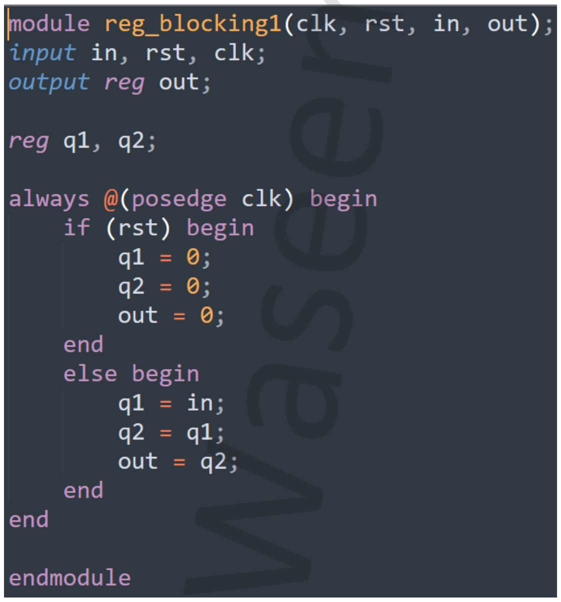
Counters & Shift Registers

1) Create a hand-drawn schematic for the following Verilog design

1-

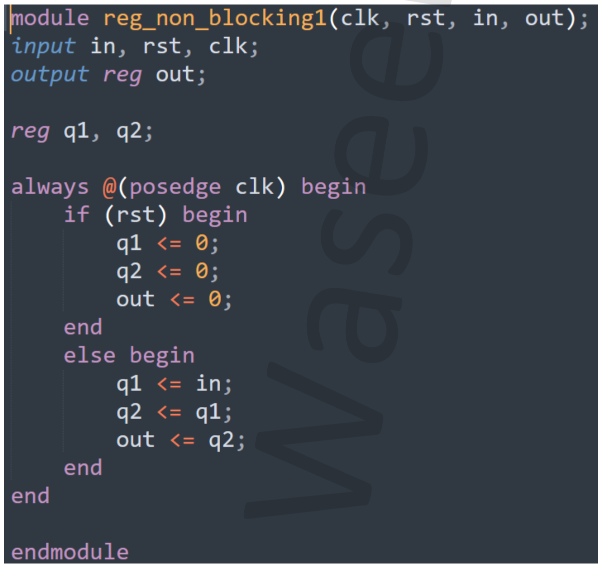


```verilog
module reg_blocking1(clk, rst, in, out);
input in, rst, clk;
output reg out;

reg q1, q2;

always @(posedge clk) begin
    if (rst) begin
        q1 = 0;
        q2 = 0;
        out = 0;
    end
    else begin
        q1 = in;
        q2 = q1;
        out = q2;
    end
end

endmodule
```

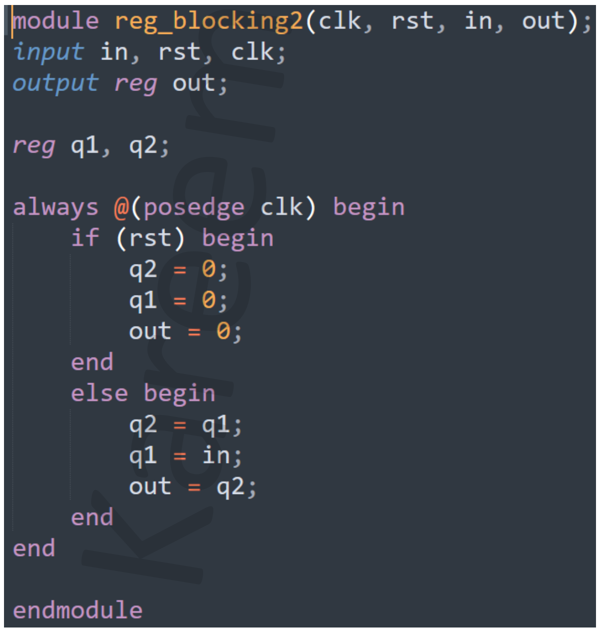


```verilog
module reg_non_blocking1(clk, rst, in, out);
input in, rst, clk;
output reg out;

reg q1, q2;

always @(posedge clk) begin
    if (rst) begin
        q1 <= 0;
        q2 <= 0;
        out <= 0;
    end
    else begin
        q1 <= in;
        q2 <= q1;
        out <= q2;
    end
end

endmodule
```

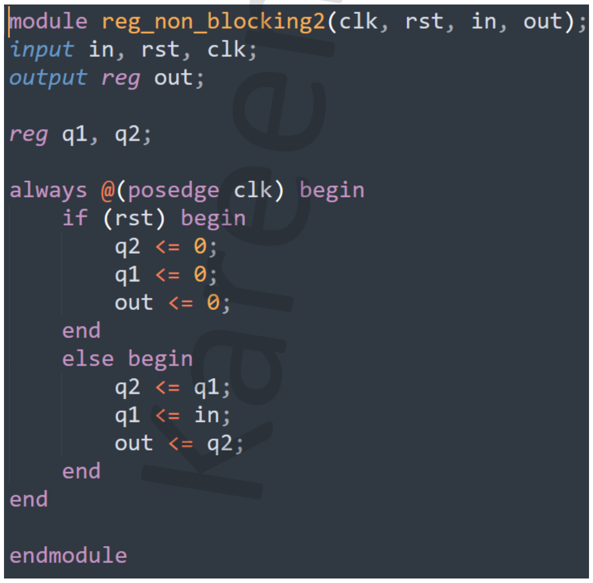


```verilog
module reg_blocking2(clk, rst, in, out);
input in, rst, clk;
output reg out;

reg q1, q2;

always @(posedge clk) begin
    if (rst) begin
        q2 = 0;
        q1 = 0;
        out = 0;
    end
    else begin
        q2 = q1;
        q1 = in;
        out = q2;
    end
end

endmodule
```

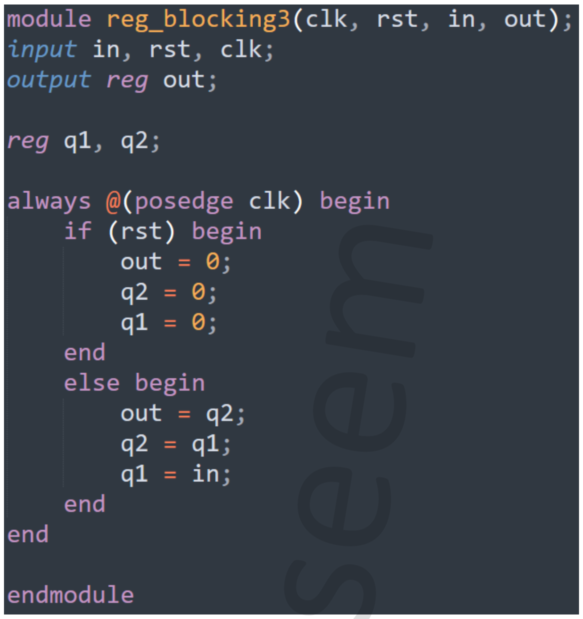


```verilog
module reg_non_blocking2(clk, rst, in, out);
input in, rst, clk;
output reg out;

reg q1, q2;

always @(posedge clk) begin
    if (rst) begin
        q2 <= 0;
        q1 <= 0;
        out <= 0;
    end
    else begin
        q2 <= q1;
        q1 <= in;
        out <= q2;
    end
end

endmodule
```

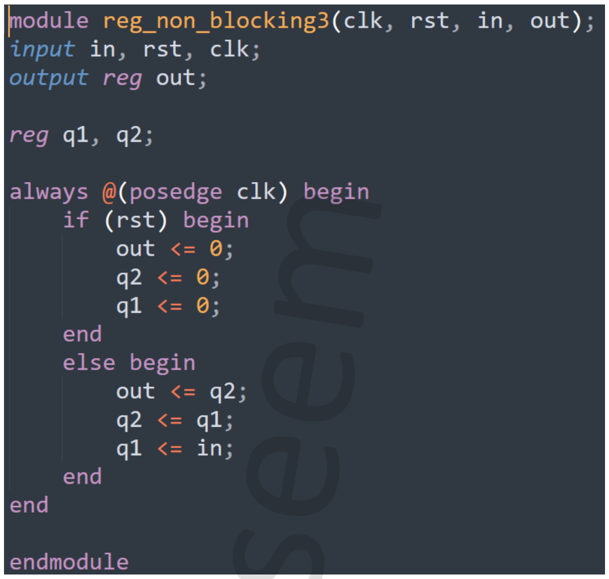


```
module reg_blocking3(clk, rst, in, out);
input in, rst, clk;
output reg out;

reg q1, q2;

always @(posedge clk) begin
    if (rst) begin
        out = 0;
        q2 = 0;
        q1 = 0;
    end
    else begin
        out = q2;
        q2 = q1;
        q1 = in;
    end
end

endmodule
```



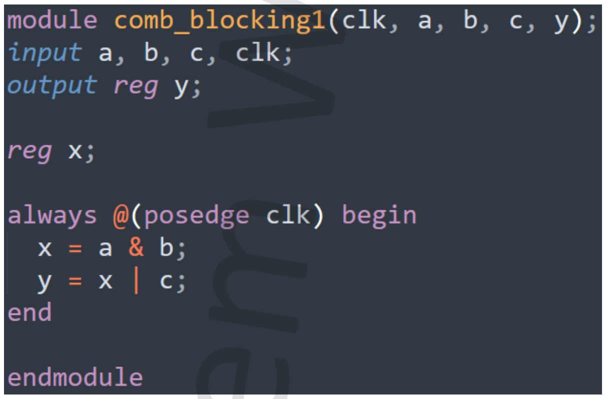
```
module reg_non_blocking3(clk, rst, in, out);
input in, rst, clk;
output reg out;

reg q1, q2;

always @(posedge clk) begin
    if (rst) begin
        out <= 0;
        q2 <= 0;
        q1 <= 0;
    end
    else begin
        out <= q2;
        q2 <= q1;
        q1 <= in;
    end
end

endmodule
```

2-

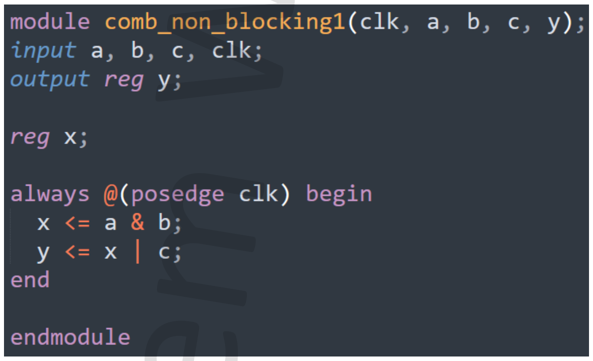


```
module comb_blocking1(clk, a, b, c, y);
input a, b, c, clk;
output reg y;

reg x;

always @(posedge clk) begin
  x = a & b;
  y = x | c;
end

endmodule
```

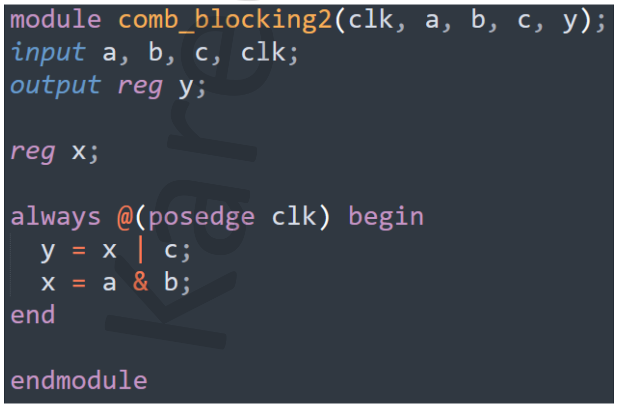


```
module comb_non_blocking1(clk, a, b, c, y);
input a, b, c, clk;
output reg y;

reg x;

always @(posedge clk) begin
  x <= a & b;
  y <= x | c;
end

endmodule
```

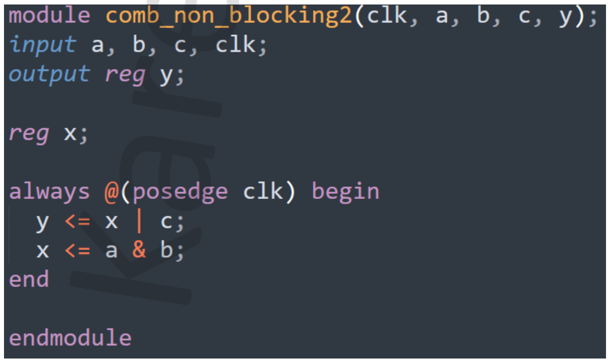


```
module comb_blocking2(clk, a, b, c, y);
input a, b, c, clk;
output reg y;

reg x;

always @(posedge clk) begin
  y = x | c;
  x = a & b;
end

endmodule
```

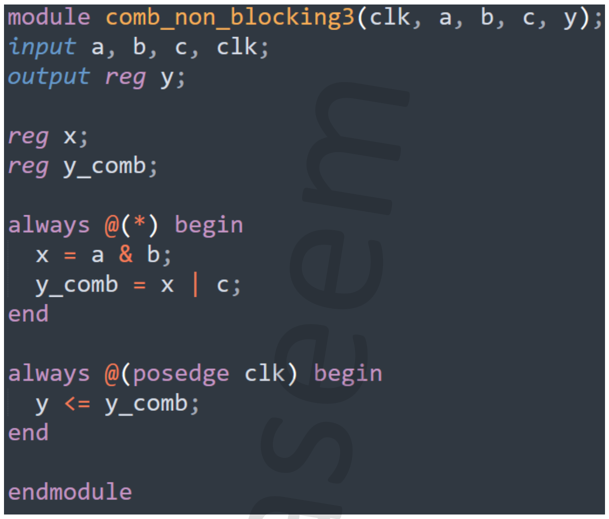


```
module comb_non_blocking2(clk, a, b, c, y);
input a, b, c, clk;
output reg y;

reg x;

always @(posedge clk) begin
  y <= x | c;
  x <= a & b;
end

endmodule
```


```
module comb_non_blocking3(clk, a, b, c, y);
input a, b, c, clk;
output reg y;

reg x;
reg y_comb;

always @(*) begin
  x = a & b;
  y_comb = x | c;
end

always @(posedge clk) begin
  y <= y_comb;
end

endmodule
```

2) Implement shift register with the following specs:

Parameter:

1. SHIFT_DIRECTION: specify shifting direction either LEFT or RIGHT, default = "LEFT"
2. SHIFT_AMOUNT: specify the number of bits to be shifted, possible values are 1, 2, 3, 4, 5, 6, 7. Default = 1

Ports:

1. Inputs:
    - clk
    - rst (async active high)
    - load: control signal if high, register should be loaded with the input "load_value"
    - load_value: value to be loaded to the register
2. Outputs:
    - PO (8 bits): parallel out which represent the register to be shifted.

Create 2 testbench to test the operation of the register when shifting right and shift amount is 2 and the other testbench to test the shifting left and shift amount is 1. The following specs should be tested:

1. Test reset that it forces the output to zero
2. Load signal to load a randomized value to the output
3. Test the shifting operation on the output
4. Load another randomized value to the output
5. Test the shifting again

Use Questasim waveform to check the above functionality

3) Implement a gray counter

Inputs:

- clk
- rst

Outputs:

- gray_out, 2-bit output

Hint: create a 2-bit binary counter counting 0, 1, 2, 3 and use the relation between the binary counter and the gray counter to assign the output bits. The most significant bit will be the same while the least significant bit of the gray out will be the reduction xor of the binary counter bits.

Create a testbench testing the following:

1. rst to force output to zero
2. remove reset and check the gray pattern from the waveform