Sequential Logic Design

- Design the circuits in question 2, 3, and 4 using Verilog **and create a testbench** for each design to check its functionality. Run questalint making sure you 0 errors and generate the schematics

- Testbenches are advised to be a mix between randomization and directed testing taken into consideration realistic operation for the inputs. Please follow the testbench instructions for each question.

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

1-

```
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
```

```
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
```

```
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
```

```
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)

1. Implement 4-bit synchronous counter with asynchronous active low set using behavioral modelling that increment from 0 to 15
    - Input:
        - clk
        - set (sets all bits to 1)
        - out (4-bits)
2. Use the ripple counter done in Assignment 3 as a golden reference.
3. Test the above synchronous counter design using a self-checking testbench
    - Testbench should instantiate both counters.

3) Extend on the previous counter done in question 3 to have extra 2 single bit outputs (div_2 and div_4). Hint: Observe the output bits of the "out" bus to implement the following
    - div_2: output signal that acts as output clock with a frequency half the input clock signal.
    - div_4: output signal that acts as output clock with a frequency quarter the input clock signal.

Create a testbench that self-check the functionality of the div_2 and div_4.

4) 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

<u>Deliverables:</u>

1) The assignment should be submitted as a PDF file with this format <your_name>_Assignment3_Extra for example Kareem_Waseem_Assignment3_Extra
2) Snippets from the waveforms captured from QuestaSim for each design with inputs assigned values and output values visible.

Note that your document should be organized as 4 sections corresponding to each design above, and in each section, I am expecting the Verilog code, and the waveforms snippets with questalint snippet of 0 errors and schematic generated.