Verilog HDL

Assignment 4 Extended

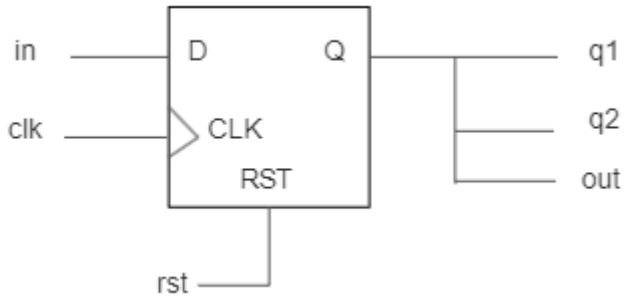
By: Magdy Ahmed Abbas Abdelhamid

Question 1

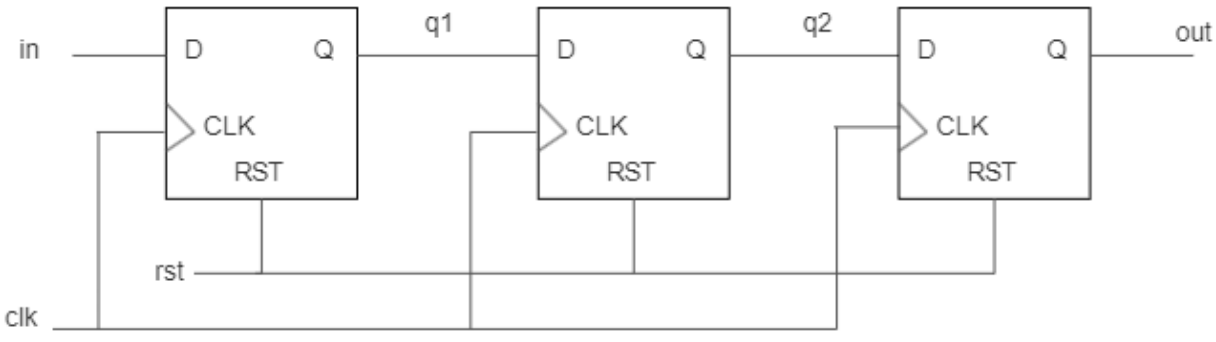
Hand - Drawn Schematic

1

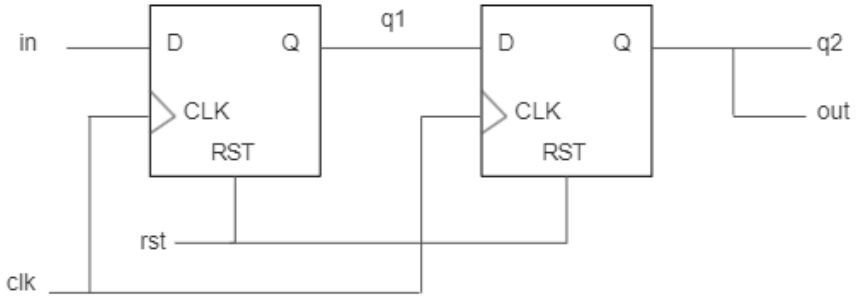
```
module reg_blocking1(clk, rst, in, out);
input in, rst, clk;
output reg out;

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
```



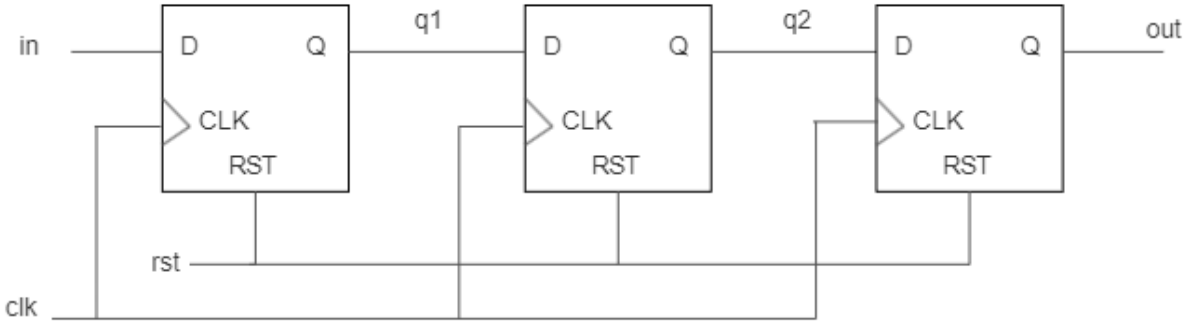

```
module reg_non_blocking1(clk, rst, in, out);
input in, rst, clk;
output reg out;

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
```



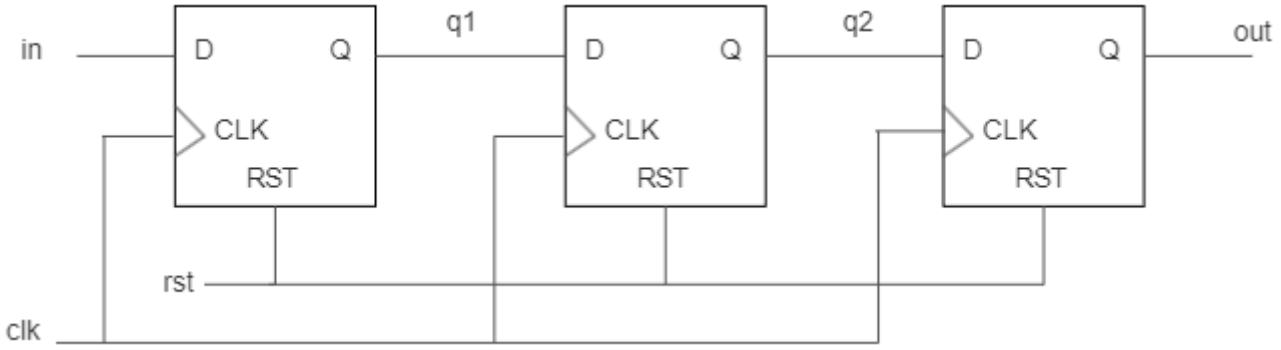
```
module reg_blocking2(clk, rst, in, out);
input in, rst, clk;
output reg out;

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
```



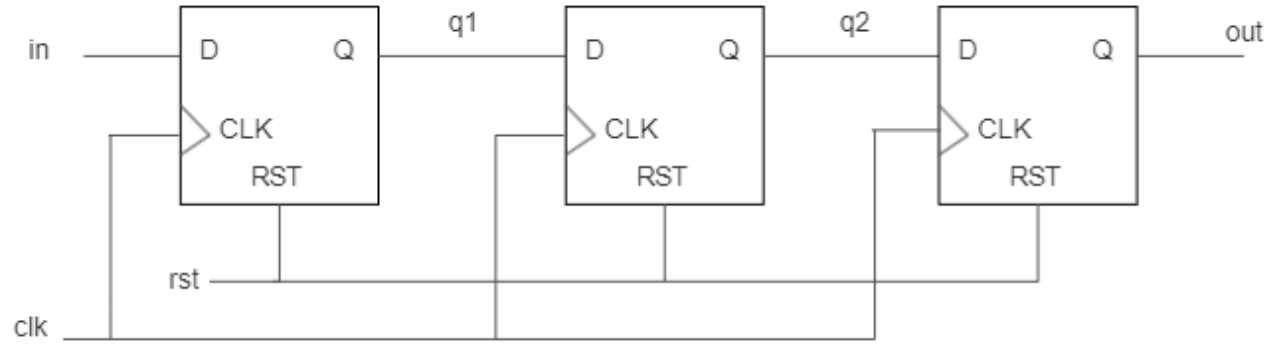

```
module reg_non_blocking2(clk, rst, in, out);
input in, rst, clk;
output reg out;

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
```

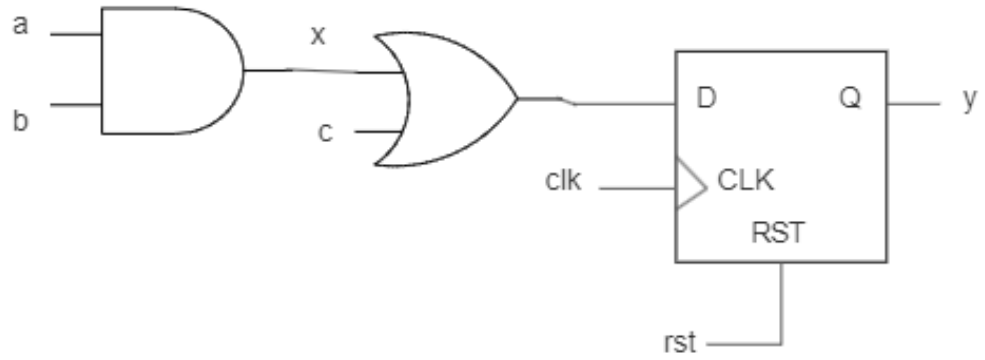



```
module reg_non_blocking3(clk, rst, in, out);
input in, rst, clk;
output reg out;

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
```

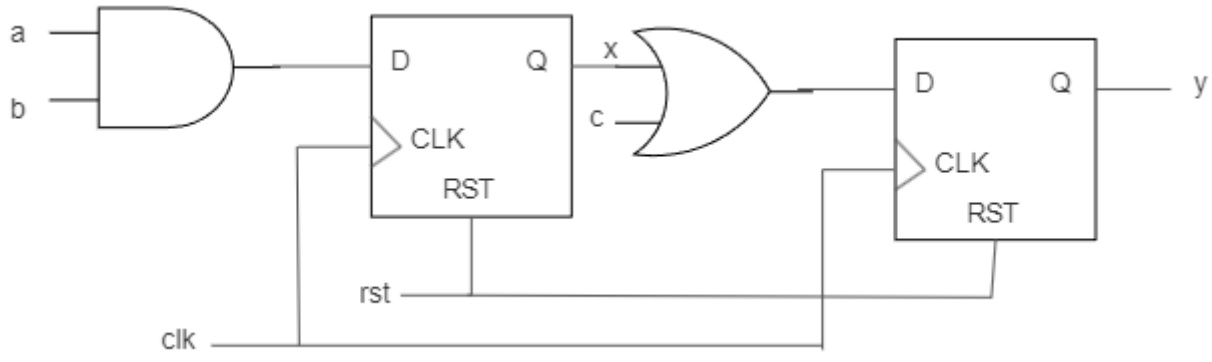


2

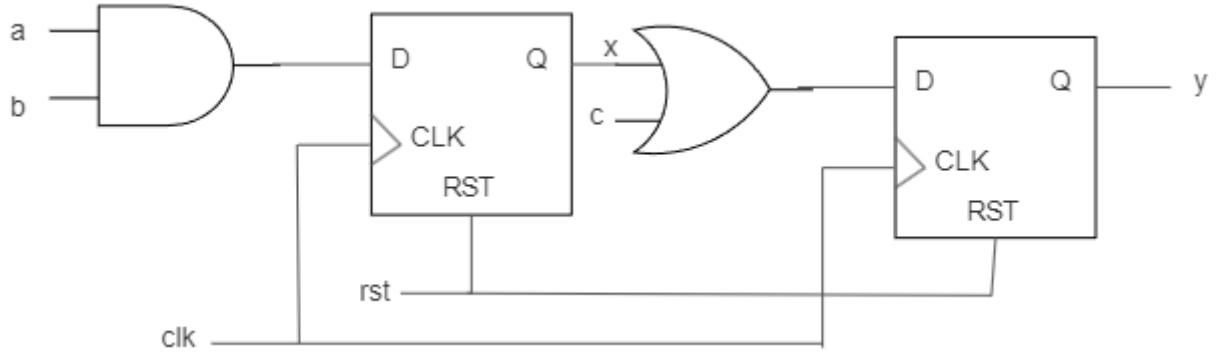
```
module comb_blocking1(clk, a, b, c, y);
input a, b, c, clk;
output reg y;

reg x;

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

endmodule
```



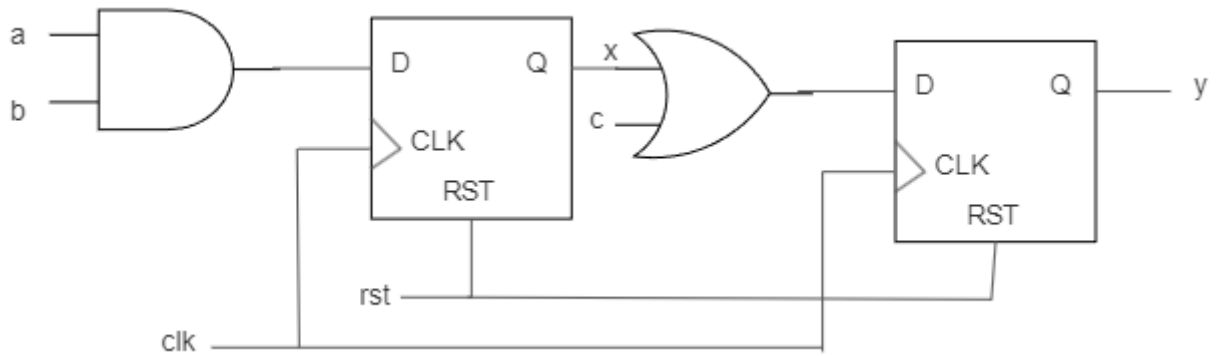

```
module comb_non_blocking1(clk, a, b, c, y);
input a, b, c, clk;
output reg y;

reg x;

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

endmodule
```



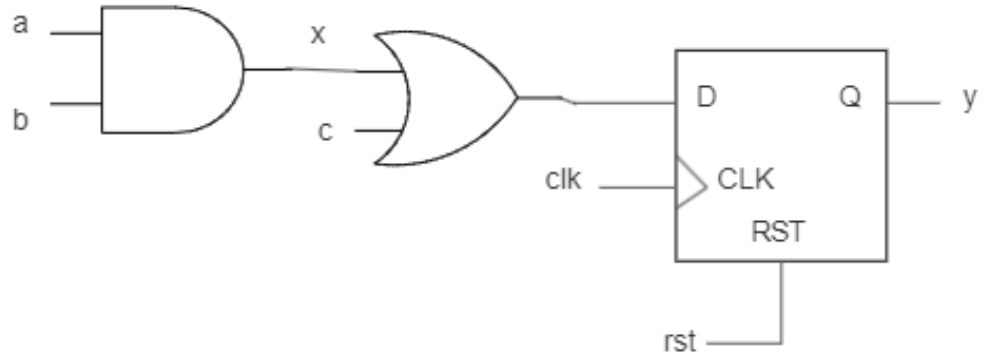
```
module comb_blocking2(clk, a, b, c, y);
input a, b, c, clk;
output reg y;

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
```



Question 2

SHIFT REGISTER

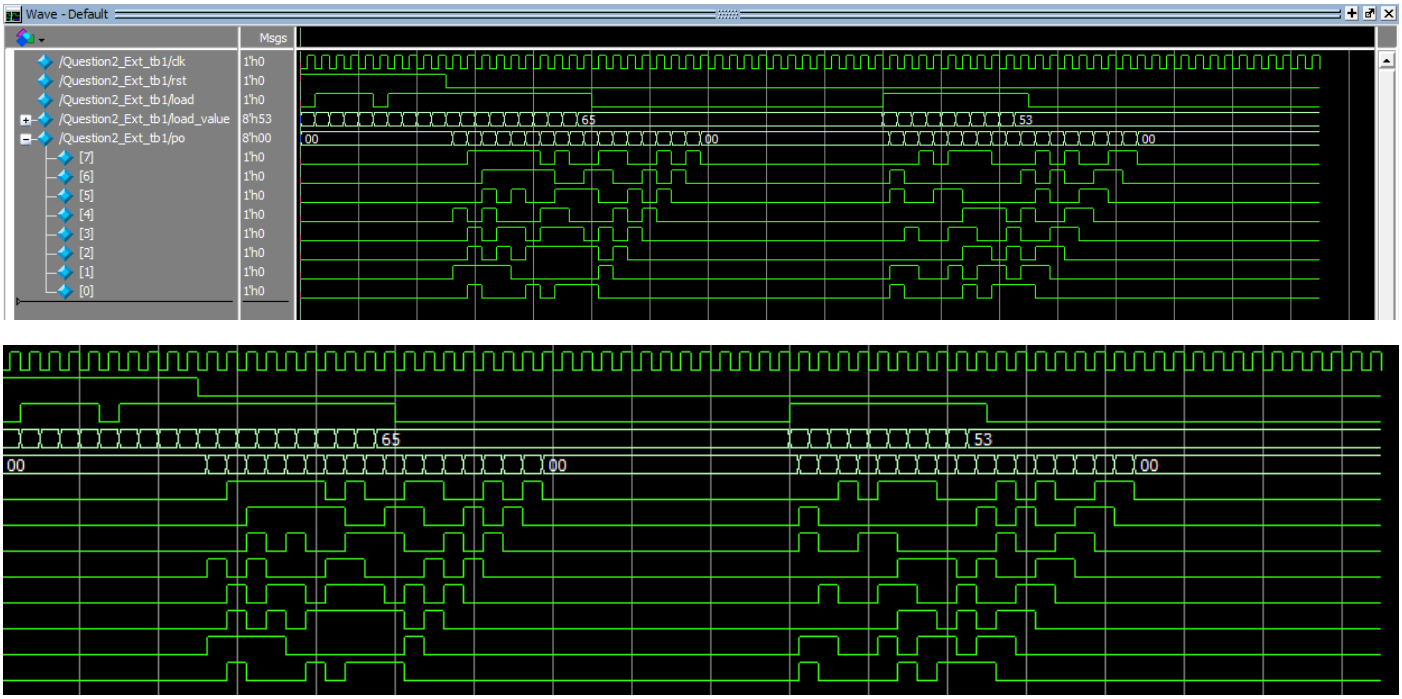
Verilog Code:

```
module Question2_Ext (clk, rst, load, load_value, PO);

parameter SHIFT_DIRECTION = "LEFT";
parameter SHIFT_AMOUNT = 1;

input clk, rst, load;
input [7: 0] load_value;
output reg [7: 0] PO;

always @(posedge clk or posedge rst) begin
   if (rst)
      PO <= 0;
   else if (load)
         PO <= load_value;
   else
      case (SHIFT_DIRECTION)
        "LEFT" : PO ≤ PO << SHIFT_AMOUNT;
              // PO <= {PO[7 – SHIFT_AMOUNT: 0], SHIFT_AMOUNT[1'b0]};
        "RIGHT" : PO ≤ PO >> SHIFT_AMOUNT;
               // PO <= {SHIFT_AMOUNT[1'b0], PO[7: SHIFT_AMOUNT]};
      endcase
end

endmodule
```

Testbench Code 1:

```
module Question2_Ext_tb1 ();
parameter N1 = "LEFT";
parameter N2 = 1;
reg clk, rst, load;
reg [7: 0] load_value;
wire [7: 0] po;
Question2_Ext #(. SHIFT_DIRECTION(N1), . SHIFT_AMOUNT(N2))
DUT(. clk(clk), . rst(rst), . load(load), . load_value(load_value), . PO(po));
initial begin
  clk = 0;
  forever
    #25 clk = ~clk;
end
initial begin
  rst = 1;
  repeat (10) begin
    load = $random;
    load_value = $random;
    @(negedge clk);
    if (po ! = 0) begin
      $display("The Shift Register Design is Wrong! ");
      $stop;
    end
  end
  rst = 0; load = 1;
  repeat (10) begin
    load_value = $random;
    @(negedge clk);
  end
  load = 0;
  repeat (20) @(negedge clk);
  load = 1;
  repeat (10) begin
    load_value = $random;
    @(negedge clk);
  End
  load = 0;
  repeat (20) @(negedge clk);
  $stop;
end
endmodule
```

Waveform 1:

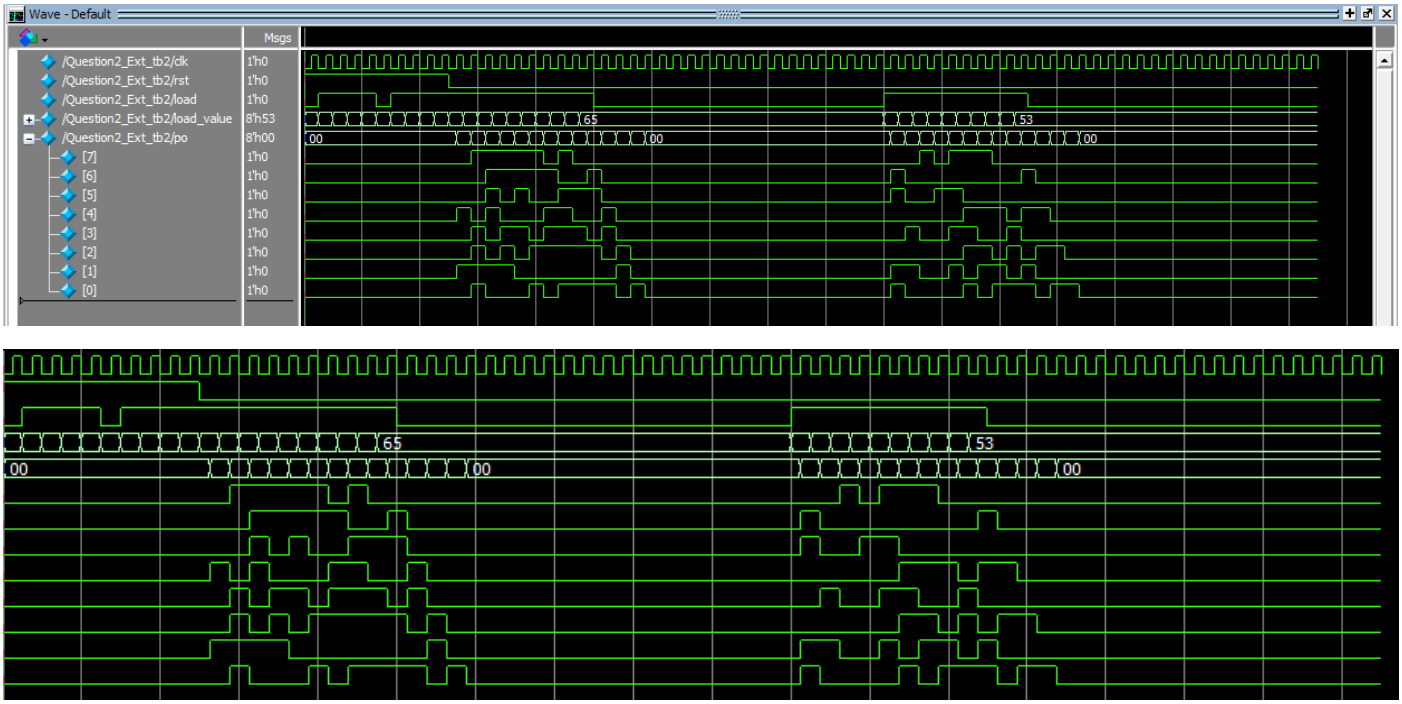


1. Test reset that it forces the output to zero
2. Load signal to load a randomized value to the output
3. Test the shifting operation on the output
4. Load another randomized value to the output
5. Test the shifting again

Testbench Code 2:

```
module Question2_Ext_tb2();

parameter N3 = "RIGHT";
parameter N4 = 2;
reg clk, rst, load;
reg [7: 0] load_value;
wire [7: 0] po;
Question2_Ext #(.SHIFT_DIRECTION(N3), .SHIFT_AMOUNT(N4))
DUT(.clk(clk), .rst(rst), .load(load), .load_value(load_value), .PO(po));
initial begin
  clk = 0;
  forever
    #25 clk = ~clk;
End
initial begin
  rst = 1;
  repeat (10) begin
    load = $random;
    load_value = $random;
    @(negedge clk);
    if (po != 0) begin
      $display("The Shift Register Design is Wrong! ");
      $stop;
    end
  end
  rst = 0; load = 1;
  repeat (10) begin
    load_value = $random;
    @(negedge clk);
  end
  load = 0;
  repeat (20) @(negedge clk);
  load = 1;
  repeat (10) begin
    load_value = $random;
    @(negedge clk);
  end
  load = 0;
  repeat (20) @(negedge clk);
  $stop;
end
endmodule
```

Waveform 2:



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

Question 3

Gray Counter

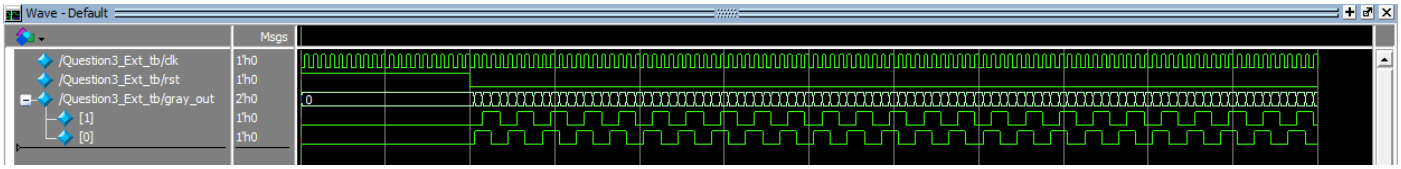
Verilog Code:

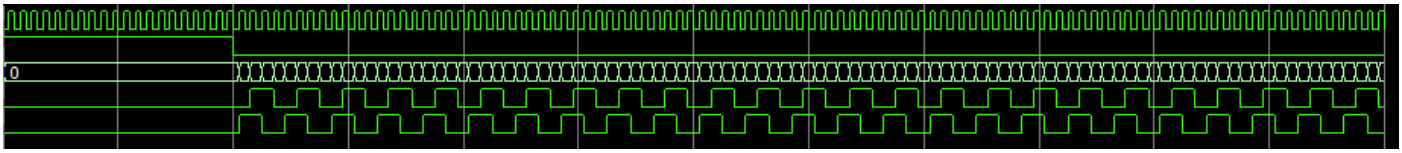
```
module Question3_Ext (clk, rst, gray_out);

input clk, rst;
output [1: 0] gray_out;

reg [1: 0] binary_cnt;

always @(posedge clk or posedge rst) begin
   if (rst)
      binary_cnt  <=  0;
   else begin
      binary_cnt  <=  binary_cnt  +  1;
   end
end

assign gray_out[1]  =  binary_cnt[1];
assign gray_out[0]  =  ^binary_cnt;

endmodule
```

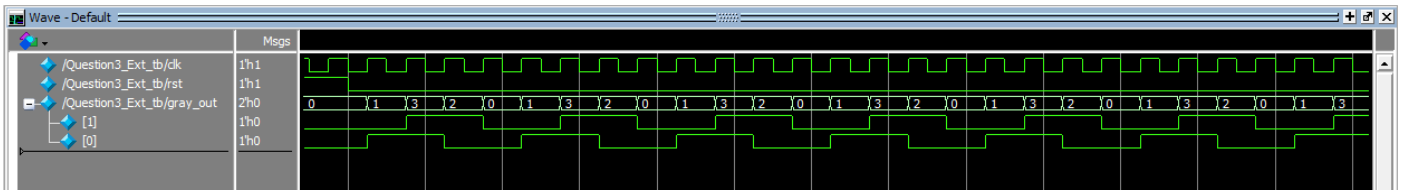
Testbench Code:

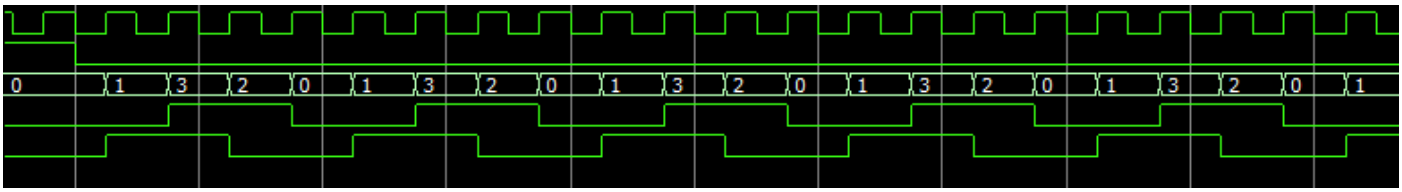
```
module Question3_Ext_tb();

reg clk, rst;
wire [1: 0] gray_out;

Question3_Ext DUT(. clk(clk), . rst(rst), . gray_out(gray_out));

initial begin
clk = 0;
forever
#25 clk = ~clk;
end

initial begin
// initialize all Stimulus Signals
rst = 1;
////////////////////////
// Test Async Signals //
////////////////////////
repeat (20) begin
@(negedge clk);
if (gray_out ! = 0) begin
$display("Async Control Signals – Q output is not Correct");
$stop;
end
end
//////////////////////////////////////////////
// Test Gray Counter Output – Check Waveform //
//////////////////////////////////////////////
rst = 0;
repeat (100) @(negedge clk);
$stop;
end

endmodule
```

Waveform:









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