Assignment_2

==========

Name: Ahmed Nour

Sheet ID: 16

Email: ahmedmohamednoour@gmail.com

==========

Q1)

Main Code for ALU:

```
module Nbit_ALU(out , in0 , in1);
parameter N_width = 4;
parameter OPCODE = 0;
input [N_width-1:0] in0 , in1;
output [N_width-1:0] out;

assign out = (OPCODE == 0)? in0+in1 : (OPCODE == 1)? in0|in1 : (OPCODE == 2)? in0-in1 : in0^in1;

endmodule
```

Testbench:

• Testbench 1: N=4 & OPCODE = 0

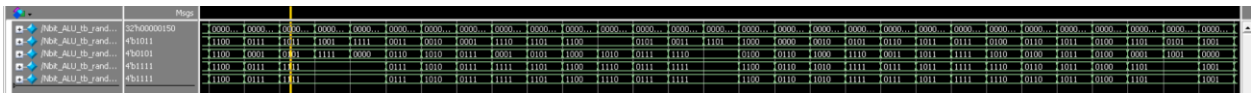
```
module Nbit_ALU_tb_randomized_add();
parameter N_width_tb = 4;
parameter OPCODE_tb = 0;

reg [N_width_tb-1:0] in0_tb , in1_tb , out_exp;
wire [N_width_tb-1:0] out_tb;

Nbit_ALU #(.N_width(N_width_tb), .OPCODE(OPCODE_tb)) DUT (out_tb , in0_tb , in1_tb);

//opcode = 0
integer i;
initial begin
  for(i=0; i<999; i=i+1)begin
    in0_tb= $random;
    in1_tb= $random;
    out_exp= in0_tb + in1_tb;
    #10;
    if(out_tb != out_exp)begin
        $display("error-ALU in addition");
    end
  end
  $display("No errors found in addition");
end

endmodule
```

```
view -new wave
# .main_pane.wave.interior.cs.body.pw.wf
add wave -position insertpoint  \
sim:/Nbit_ALU_tb_randomized_add/i \
sim:/Nbit_ALU_tb_randomized_add/in0_tb \
sim:/Nbit_ALU_tb_randomized_add/in1_tb \
sim:/Nbit_ALU_tb_randomized_add/out_exp \
sim:/Nbit_ALU_tb_randomized_add/out_tb
VSIM 28> run -all
# No errors found in addition
```

• **Testbench 1: N=4 & OPCODE = 1**

```
module Nbit_ALU_tb_randomized_or();
parameter N_width_tb = 4;
parameter OPCODE_tb = 1;

reg [N_width_tb-1:0] in0_tb , in1_tb , out_exp;
wire [N_width_tb-1:0] out_tb;

Nbit_ALU #(.N_width(N_width_tb), .OPCODE(OPCODE_tb)) DUT (out_tb , in0_tb , in1_tb);

//opcode = 1
integer i;
initial begin
  for(i=0; i<999; i=i+1)begin
    in0_tb= $random;
    in1_tb= $random;
    out_exp= in0_tb | in1_tb;
    #10;
    if(out_tb != out_exp)begin
        $display("error-ALU in OR");
    end
  end
  $display("No errors found in OR");
end

endmodule
```

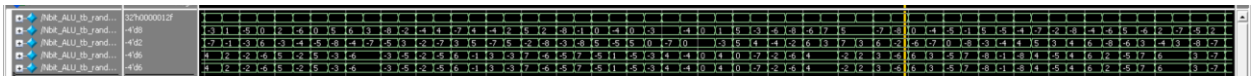
```
# Loading work.Nbit_ALU_tb_randomized_or(fast)
# Loading work.Nbit_ALU(fast)
VSIM 48> run
run
VSIM 49> run -all
# No errors found in OR
```



• **Testbench 1: N=4 & OPCODE = 2**

```
module Nbit_ALU_tb_randomized_sub();
parameter N_width_tb = 4;
parameter OPCODE_tb = 2;

reg [N_width_tb-1:0] in0_tb , in1_tb , out_exp;
wire [N_width_tb-1:0] out_tb;

Nbit_ALU #(.N_width(N_width_tb), .OPCODE(OPCODE_tb)) DUT (out_tb , in0_tb , in1_tb);

//opcode = 2
integer i;
initial begin
  for(i=0; i<999; i=i+1)begin
    in0_tb= $random;
    in1_tb= $random;
    out_exp= in0_tb - in1_tb;
    #10;
    if(out_tb != out_exp)begin
        $display("error-ALU in subtraction");
    end
  end
  $display("No errors found in Subtraction");
end

endmodule
```

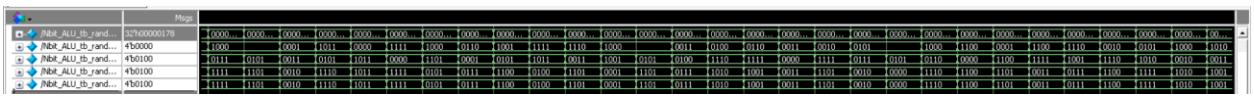
```
add wave -position insertpoint  \
sim:/Nbit_ALU_tb_randomized_sub/i \
sim:/Nbit_ALU_tb_randomized_sub/in0_tb \
sim:/Nbit_ALU_tb_randomized_sub/in1_tb \
sim:/Nbit_ALU_tb_randomized_sub/out_exp \
sim:/Nbit_ALU_tb_randomized_sub/out_tb
VSIM 54> run -all
# No errors found in Subtraction
```



• **Testbench 1: N=4 & OPCODE = 3**

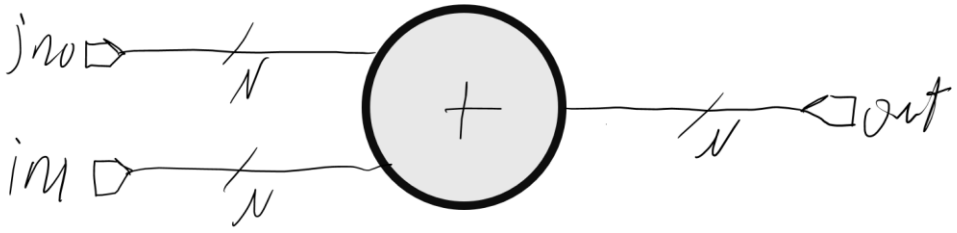
```
module Nbit_ALU_tb_randomized_xor();
parameter N_width_tb = 4;
parameter OPCODE_tb = 3;

reg [N_width_tb-1:0] in0_tb , in1_tb , out_exp;
wire [N_width_tb-1:0] out_tb;

Nbit_ALU #(.N_width(N_width_tb), .OPCODE(OPCODE_tb)) DUT (out_tb , in0_tb , in1_tb);

//opcode = 3
integer i;
initial begin
  for(i=0; i<999; i=i+1)begin
    in0_tb= $random;
    in1_tb= $random;
    out_exp= in0_tb ^ in1_tb;
    #10;
    if(out_tb != out_exp)begin
        $display("error-ALU in XOR");
    end
  end
  $display("No errors found in XOR");
end

endmodule
```

```
# Loading work.Nbit_ALU_tb_randomized_xor(fast)
# Loading work.Nbit_ALU(fast)
add wave -position insertpoint  \
sim:/Nbit_ALU_tb_randomized_xor/i \
sim:/Nbit_ALU_tb_randomized_xor/in0_tb \
sim:/Nbit_ALU_tb_randomized_xor/in1_tb \
sim:/Nbit_ALU_tb_randomized_xor/out_exp \
sim:/Nbit_ALU_tb_randomized_xor/out_tb
VSIM 58> run -all
# No errors found in XOR
```



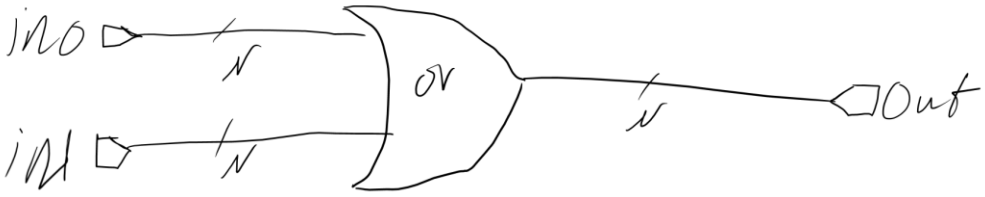
1) synthesized schematics of the design with OPCODE = 0

a) N=4, OPCODE=0



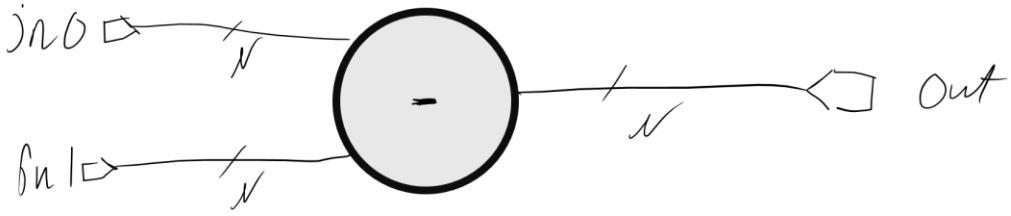
2) synthesized schematics of the design with OPCODE = 1

b) N=4, OPCODE=1



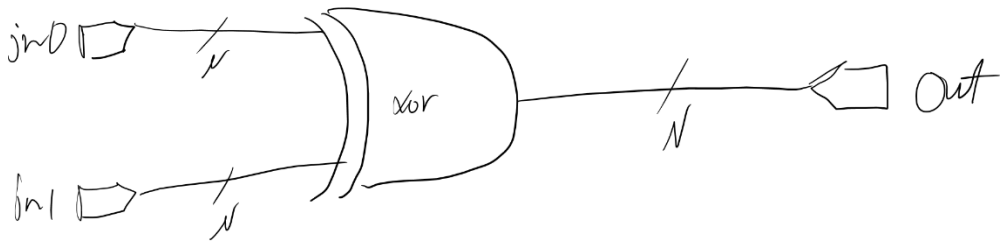
3) synthesized schematics of the design with OPCODE = 2

c) N=4, OPCODE=2



4) synthesized schematics of the design with OPCODE = 3

d) N=4, OPCODE=3



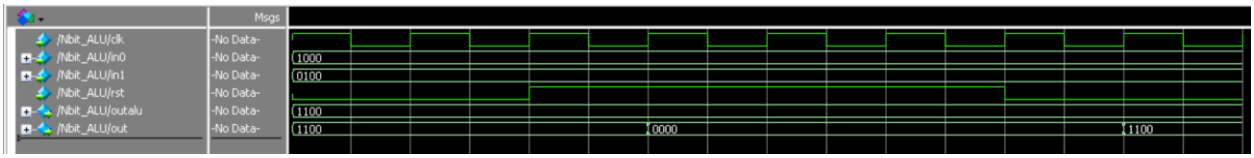
Q2)

Code:

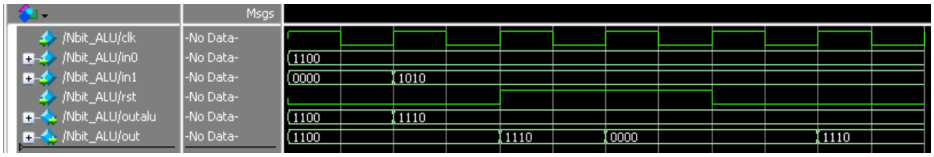
```
module Nbit_ALU(out, clk, rst, outalu , in0 , in1);
parameter N_width = 4;
parameter OPCODE = 0;

input [N_width-1:0] in0 , in1;
input clk , rst;
output [N_width-1:0] outalu;
output reg [N_width-1:0] out;

assign outalu = (OPCODE == 0)? in0+in1 : (OPCODE == 1)? in0|in1 : (OPCODE == 2)? in0-in1 : in0^in1;

always @(posedge clk)begin
    if (rst)
        out <= 0;
    else
        out <= outalu;
end
endmodule
```

Simulation:

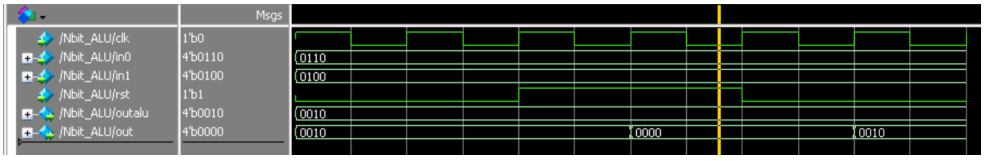
OPCODE = 0:



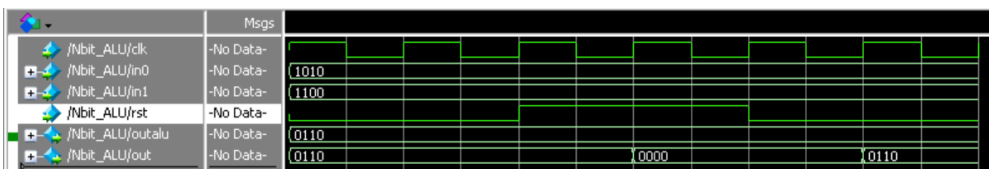
OPCODE = 1:



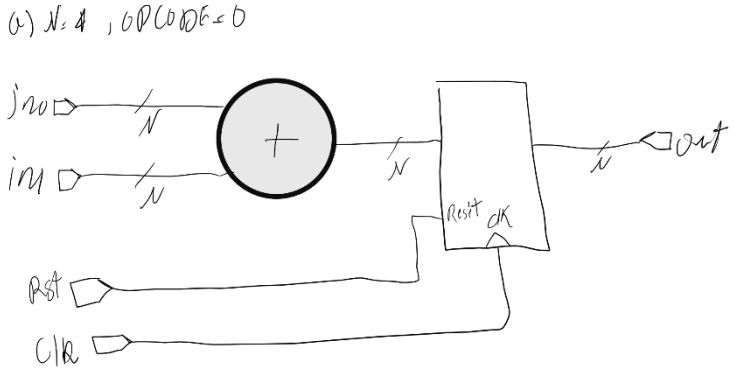
OPCODE = 2:



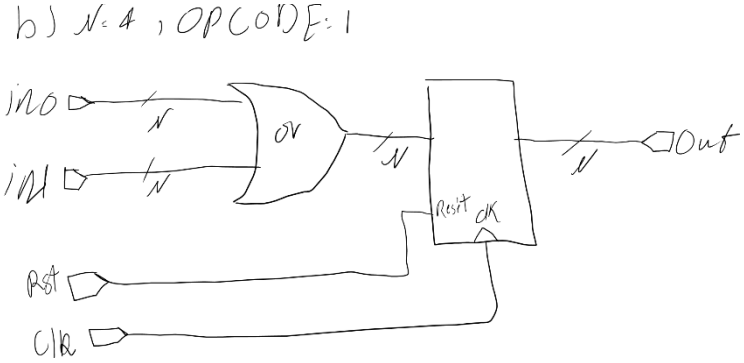
OPCODE = 3:



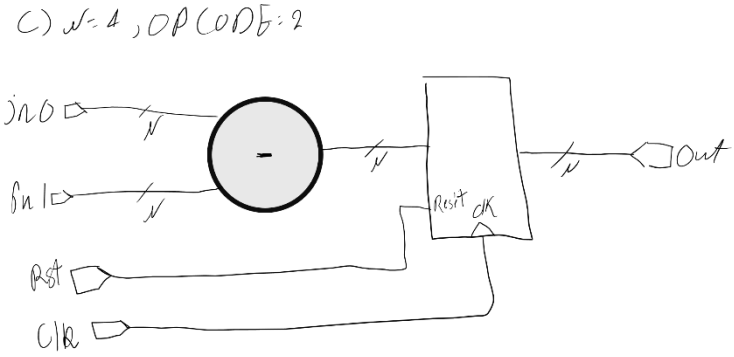
1) synthesized schematics of the design with OPCODE = 0



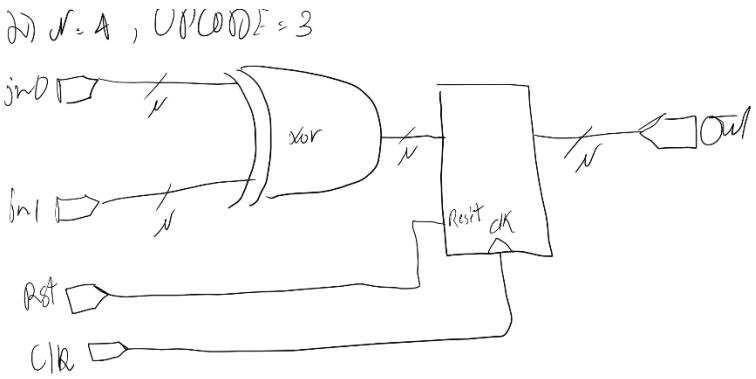
2) synthesized schematics of the design with OPCODE = 1



3) synthesized schematics of the design with OPCODE = 2



4) synthesized schematics of the design with OPCODE = 3

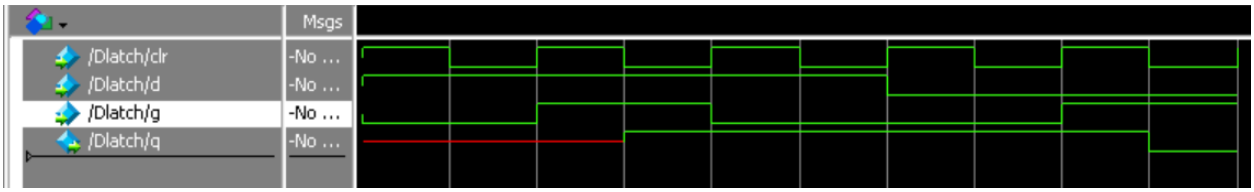


Q3)

Code:

```
1  module Dlatch(q, clr, d, g);
2      input clr, d, g;
3      output reg q;
4      always @(negedge clr) begin
5          if(g)
6              q <=d;
7      end
8
9
10 endmodule
```

Simulation:

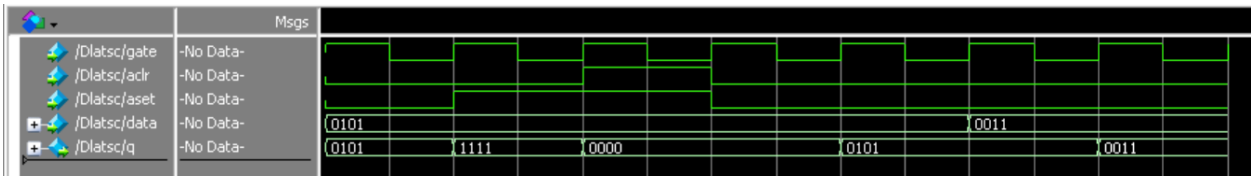


Q4)

Code:

```
module Dlatsc(q, data, aset, aclr, gate);
    parameter LAT_WIDTH =4;

    input aset, aclr, gate;
    input [LAT_WIDTH-1:0] data;
    output reg [LAT_WIDTH-1:0]  q;

    always @(posedge gate or posedge aset or posedge aclr) begin
        if(aset)begin
            q <= (q | ~q);
        end
        if(aclr)begin
            q <= 0;
        end
        if(!aset && !aclr) begin
            q <= data;
        end

    end


endmodule
```

Simulation:



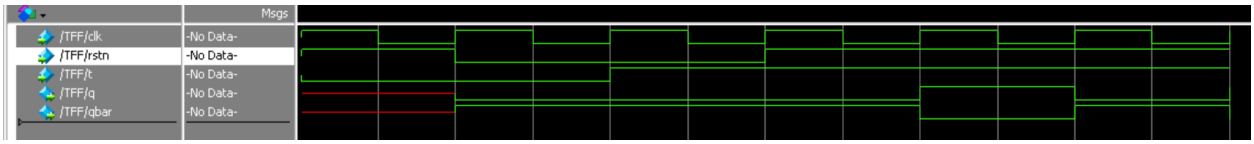
Q5)

a)

Code:

```
module TFF(q ,qbar, t, rstn, clk);
    input t, rstn, clk;
    output reg q;
    output qbar;

    assign qbar = ~q;

    always @(posedge clk or negedge rstn) begin
        if(!rstn)
            q <= 0;
        else begin
            if(t)
                q <= ~q;
        end
    end
endmodule
```

Simulation:

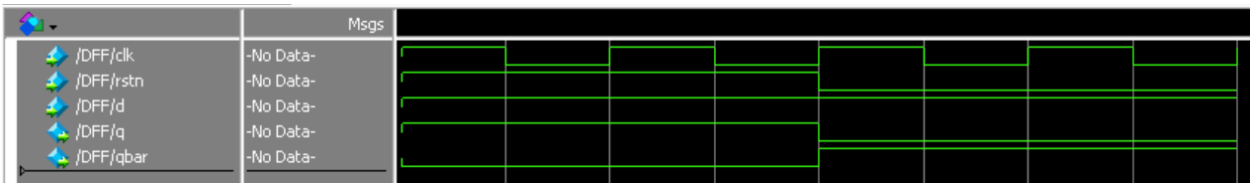


b)

Code:

```
module DFF(q ,qbar, d, rstn, clk);
    input d, rstn, clk;
    output reg q;
    output qbar;

    assign qbar = ~q;

    always @(posedge clk or negedge rstn) begin
        if(!rstn)
            q <= 0;
        else begin
            q <= d;
        end
    end

endmodule
```

Simulation:



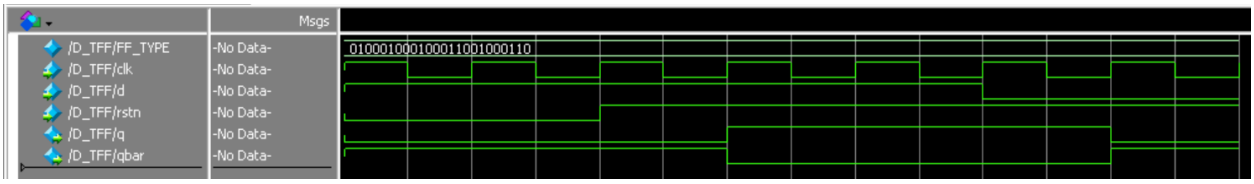
c)

Code:

```
module D_TFF(q ,qbar, d, rstn, clk);
    parameter FF_TYPE = "DFF";

    input d, rstn, clk;
    output reg q;
    output qbar;


    assign qbar = ~q;

    always @(posedge clk or negedge rstn) begin
        if(!rstn)
            q <= 0;
        else begin
            if(FF_TYPE == "DFF")begin
                q <= d;
            end
            else begin
                if(d)
                q <= ~q;
            end
        end
    end

endmodule
```

Simulation:

FF_TYPE = "DFF"



FF_TYPE = "TFF"