

Assignment_2

Name: Ahmed Nour

Sheet ID: 16

Email: ahmedmohamednoour@gmail.com

Q1)

Main Code for ALU:

```
1 module Nbit_ALU(out , in0 , in1);
2   parameter N_width = 4;
3   parameter OPCODE = 0;
4   input [N_width-1:0] in0 , in1;
5   output [N_width-1:0] out;
6
7   assign out = (OPCODE == 0)? in0+in1 : (OPCODE == 1)? in0|in1 : (OPCODE == 2)? in0-in1 : in0^in1;
8
9   endmodule
```

Testbench:

- **Testbench 1: N=4 & OPCODE = 0**

```

1 module Nbit_ALU_tb_randomized_add();
2     parameter N_width_tb = 4;
3     parameter OPCODE_tb = 0;
4
5     reg [N_width_tb-1:0] in0_tb , in1_tb , out_exp;
6     wire [N_width_tb-1:0] out_tb;
7
8     Nbit_ALU #(.N_width(N_width_tb), .OPCODE(OPCODE_tb)) DUT (out_tb , in0_tb , in1_tb);
9
10    //opcode = 0
11    integer i;
12    initial begin
13        for(i=0; i<999; i=i+1)begin
14            in0_tb= $random;
15            in1_tb= $random;
16            out_exp= in0_tb + in1_tb;
17            #10;
18            if(out_tb != out_exp)begin
19                $display("error-ALU in addition");
20            end
21        end
22        $display("No errors found in addition");
23    end
24
25 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
VSIM28> run -all
# No errors found in addition
```

| | Hops | |
|------------------|------------|--|
| • Nbr_AU_b_rnd_0 | 3670000000 | |
| • Nbr_AU_b_rnd_1 | | |
| • Nbr_AU_b_rnd_2 | | |
| • Nbr_AU_b_rnd_3 | | |
| • Nbr_AU_b_rnd_4 | | |
| • Nbr_AU_b_rnd_5 | | |

- **Testbench 1: N=4 & OPCODE = 1**

```

1 module Nbit_ALU_tb_randomized_or();
2 parameter N_width_tb = 4;
3 parameter OPCODE_tb = 1;
4
5 reg [N_width_tb-1:0] in0_tb , in1_tb , out_exp;
6 wire [N_width_tb-1:0] out_tb;
7
8 Nbit_ALU #(.N_width(N_width_tb), .OPCODE(OPCODE_tb)) DUT (out_tb , in0_tb , in1_tb);
9
10 //opcode = 1
11 integer i;
12 initial begin
13     for(i=0; i<999; i=i+1)begin
14         in0_tb= $random;
15         in1_tb= $random;
16         out_exp= in0_tb | in1_tb;
17         #10;
18         if(out_tb != out_exp)begin
19             $display("error-ALU in OR");
20         end
21     end
22     $display("No errors found in OR");
23 end
24
25 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
```

[illegible]

- Testbench 1: N=4 & OPCODE = 2

```

1 module Nbit_ALU_tb_randomized_sub();
2 parameter N_width_tb = 4;
3 parameter OPCODE_tb = 2;
4
5 reg [N_width_tb-1:0] in0_tb , in1_tb , out_exp;
6 wire [N_width_tb-1:0] out_tb;
7
8 Nbit_ALU #(.N_width(N_width_tb), .OPCODE(OPCODE_tb)) DUT (out_tb , in0_tb , in1_tb);
9
10 //opcode = 2
11 integer i;
12 initial begin
13     for(i=0; i<999; i=i+1)begin
14         in0_tb= $random;
15         in1_tb= $random;
16         out_exp= in0_tb - in1_tb;
17         #10;
18         if(out_tb != out_exp)begin
19             $display("error-ALU in subtraction");
20         end
21     end
22     $display("No errors found in Subtraction");
23 end
24
25 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

```

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| sim:/Nbit_ALU_tb_randomized_sub/i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| sim:/Nbit_ALU_tb_randomized_sub/in0_tb | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| sim:/Nbit_ALU_tb_randomized_sub/in1_tb | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| sim:/Nbit_ALU_tb_randomized_sub/out_exp | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| sim:/Nbit_ALU_tb_randomized_sub/out_tb | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |

- **Testbench 1: N=4 & OPCODE = 3**

```

1 module Nbit_ALU_tb_randomized_xor();
2 parameter N_width_tb = 4;
3 parameter OPCODE_tb = 3;
4
5 reg [N_width_tb-1:0] in0_tb , in1_tb , out_exp;
6 wire [N_width_tb-1:0] out_tb;
7
8 Nbit_ALU #(.N_width(N_width_tb), .OPCODE(OPCODE_tb)) DUT (out_tb , in0_tb , in1_tb);
9
10 //opcode = 3
11 integer i;
12 initial begin
13     for(i=0; i<999; i=i+1)begin
14         in0_tb= $random;
15         in1_tb= $random;
16         out_exp= in0_tb ^ in1_tb;
17         #10;
18         if(out_tb != out_exp)begin
19             $display("error-ALU in XOR");
20         end
21     end
22     $display("No errors found in XOR");
23 end
24
25 endmodule

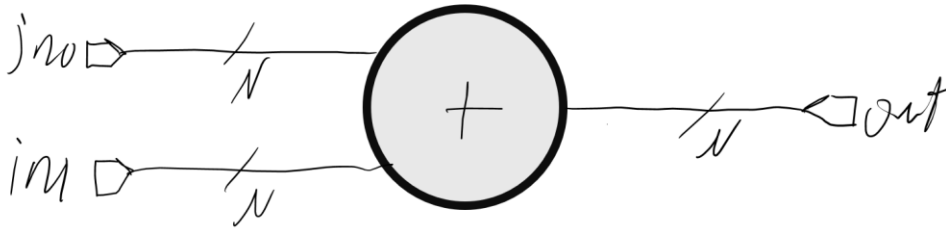
```

```
# 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
VSIM58> run -all
# No errors found in XOR
```

[illegible]

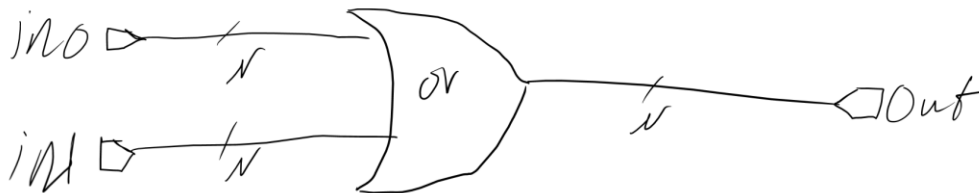
1) synthesized schematics of the design with OPCODE = 0

a) $N=4$, OPCODE=0



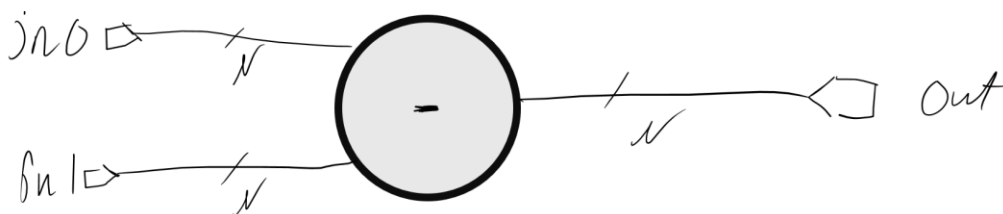
2) synthesized schematics of the design with OPCODE = 1

b) $N=4$, OPCODE=1



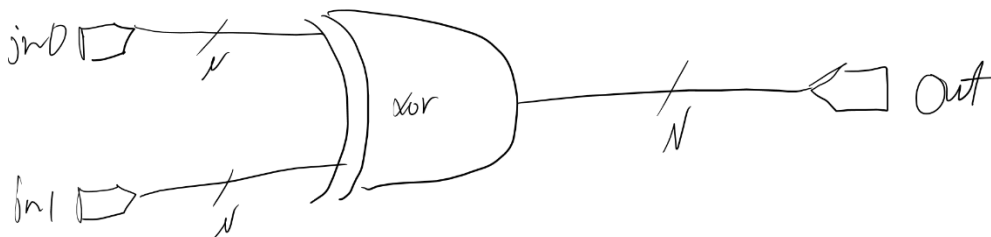
3) synthesized schematics of the design with OPCODE = 2

c) $N=4$, OPCODE=2



4) synthesized schematics of the design with OPCODE = 3

d) $N=4$, OPCODE=3



Q2)

Code:

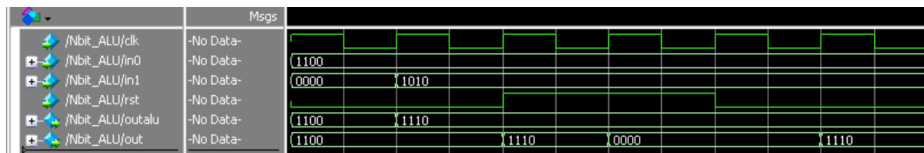
```
1 module Nbit_ALU(out, clk, rst, outalu , in0 , in1);
2   parameter N_width = 4;
3   parameter OPCODE = 0;
4
5   input [N_width-1:0] in0 , in1;
6   input clk , rst;
7   output [N_width-1:0] outalu;
8   output reg [N_width-1:0] out;
9
10  assign outalu = (OPCODE == 0)? in0+in1 : (OPCODE == 1)? in0|in1 : (OPCODE == 2)? in0-in1 : in0^in1;
11
12  always @(posedge clk)begin
13    if (rst)
14      out <= 0;
15    else
16      out <= outalu;
17  end
18 endmodule
```

Simulation:

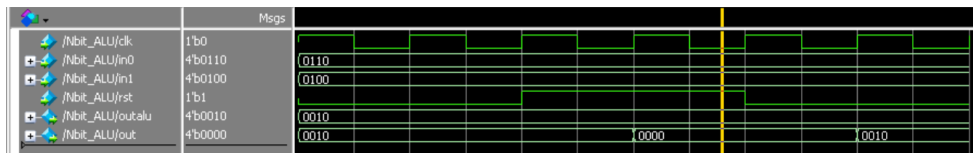
OPCODE = 0:



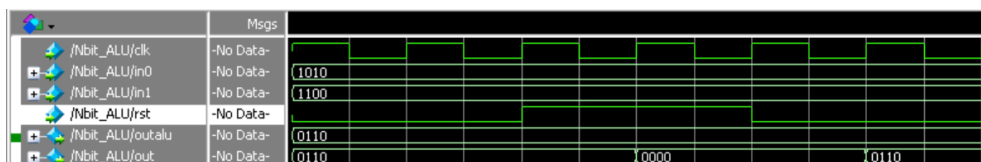
OPCODE = 1:



OPCODE = 2:

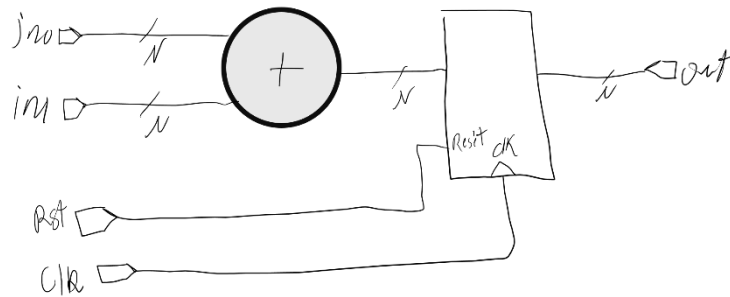


OPCODE = 3:



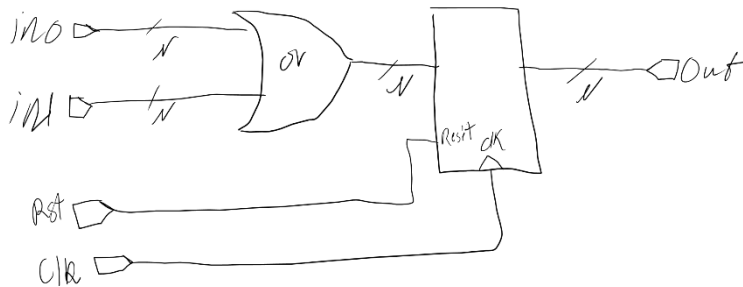
1) synthesized schematics of the design with OPCODE = 0

a) $N=4$, OPCODE=0



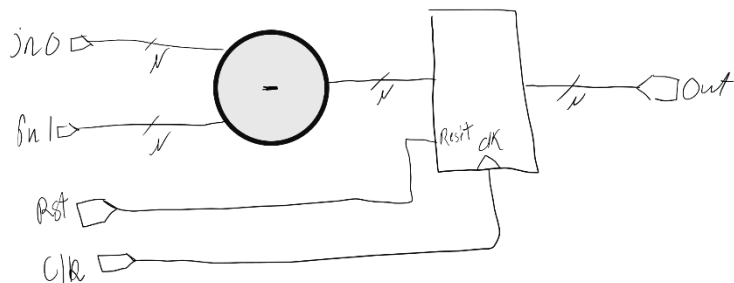
2) synthesized schematics of the design with OPCODE = 1

b) $N=4$, OPCODE=1



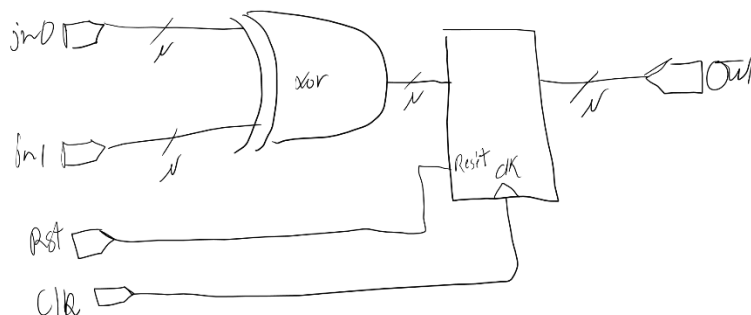
3) synthesized schematics of the design with OPCODE = 2

c) $N=4$, OPCODE=2



4) synthesized schematics of the design with OPCODE = 3

d) $N=4$, OPCODE=3

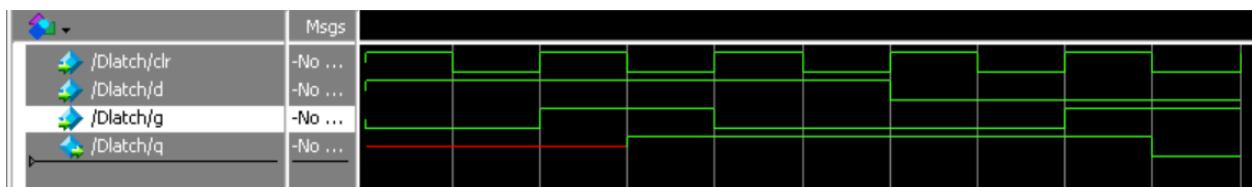


Q3)

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:

```
1 module Dlatsc(q, data, aset, aclr, gate);
2     parameter LAT_WIDTH =4;
3
4     input aset, aclr, gate;
5     input [LAT_WIDTH-1:0] data;
6     output reg [LAT_WIDTH-1:0] q;
7
8     always @(posedge gate or posedge aset or posedge aclr) begin
9         if(aset)begin
10             q <= (q | ~q);
11         end
12         if(aclr)begin
13             q <= 0;
14         end
15         if(!aset && !aclr) begin
16             q <= data;
17         end
18     end
19 end
20
21
22 endmodule
```

Simulation:

| | Msgs | |
|--------------|-----------|--------------------------|
| /Dlatsc/gate | -No Data- | |
| /Dlatsc/aclr | -No Data- | |
| /Dlatsc/aset | -No Data- | |
| /Dlatsc/data | -No Data- | 0101 |
| /Dlatsc/q | -No Data- | 0101 1111 0000 0101 0011 |

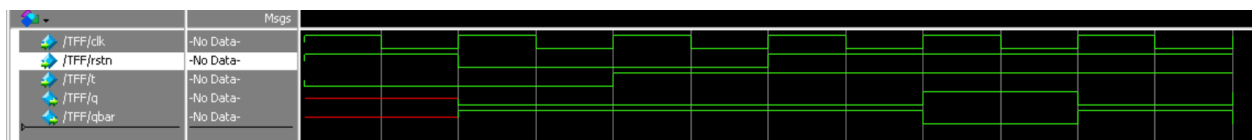
Q5)

a)

Code:

```
1  module TFF(q ,qbar, t, rstn, clk);
2      input t, rstn, clk;
3      output reg q;
4      output qbar;
5
6      assign qbar = ~q;
7
8      always @(posedge clk or negedge rstn) begin
9          if(!rstn)
10             q <= 0;
11         else begin
12             if(t)
13                 q <= ~q;
14         end
15     end
16 endmodule
```

Simulation:



b)

Code:

```
1 module DFF(q ,qbar, d, rstn, clk);
2     input d, rstn, clk;
3     output reg q;
4     output qbar;
5
6     assign qbar = ~q;
7
8     always @(posedge clk or negedge rstn) begin
9         if(!rstn)
10            q <= 0;
11        else begin
12            q <= d;
13        end
14    end
15
16 endmodule
```

Simulation:

| | Msgs | | | | | | | | |
|-----------|-----------|--|--|--|--|--|--|--|--|
| /DFF/clk | -No Data- | | | | | | | | |
| /DFF/rstn | -No Data- | | | | | | | | |
| /DFF/d | -No Data- | | | | | | | | |
| /DFF/q | -No Data- | | | | | | | | |
| /DFF/qbar | -No Data- | | | | | | | | |

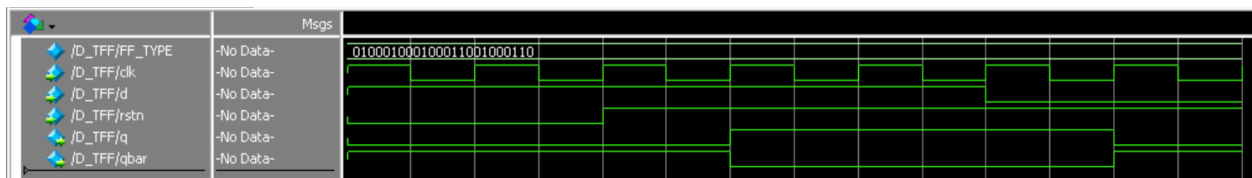
c)

Code:

```
1 module D_TFF(q ,qbar, d, rstn, clk);
2     parameter FF_TYPE = "DFF";
3
4     input d, rstn, clk;
5     output reg q;
6     output qbar;
7
8
9     assign qbar = ~q;
10
11     always @(posedge clk or negedge rstn) begin
12         if(!rstn)
13             q <= 0;
14         else begin
15             if(FF_TYPE == "DFF")begin
16                 q <= d;
17             end
18             else begin
19                 if(d)
20                     q <= ~q;
21             end
22         end
23     end
24
25 endmodule
```

Simulation:

FF_TYPE = "DFF"



FF_TYPE = "TFF"

