

# Assignment 3

Ameer Ashraf Louly – G3

## Question 1:

```

1  module dff_en_pre_tb();
2      reg E, D, clk, PRE, Q_expected;
3      wire Q;
4
5      dff_en_pre DUT(E, D, Q, clk, PRE);
6
7      initial begin
8          clk = 0;
9          forever begin
10             #1 clk = ~clk;
11         end
12     end
13
14     initial begin
15         $monitor("D = %d, Q = %d", D, Q);
16     end
17
18     initial begin
19         // Preset Check
20         PRE = 0;
21         @(negedge clk);
22         if(Q != 1) begin
23             $display("Error - Incorrect PRESET");
24             $exit;
25         end
26         PRE = 1;
27         $display("PRESET WORKS");
28
29         // DFF check
30         PRE = 0;
31         @(negedge clk);
32         PRE = 1;
33         E = 1;
34         repeat(50) begin
35             D = $random;
36             Q_expected = D;
37             @(negedge clk);
38             if(Q != Q_expected) begin
39                 $display("Enable doesn't work");
40                 $exit;
41             end
42         end /* End of Repeat */
43         $display("DFF WORKS");
44
45         // Enable Closed Check
46         E = 1;
47         D = 1;
48         @(negedge clk);
49         E = 0;
50         repeat(50) begin
51             D = $random;
52             @(negedge clk);
53             if(Q != 1) begin
54                 $display("Enable doesn't work");
55                 $exit;
56             end
57         end /* End of Repeat */
58         $display("ENABLE WORKS");
59
60         $display("Design Works as expected");
61         $exit;
62     end
63 endmodule

```

Figure 1 TB Code

```

1  vlib work
2
3  vlog q2.v q2_tb.v
4
5  vsim -voptargs=+acc work.dff_en_pre_tb
6
7  add wave *
8
9  run -all
10
11 #quit -sim
12

```

Figure 2 DO File

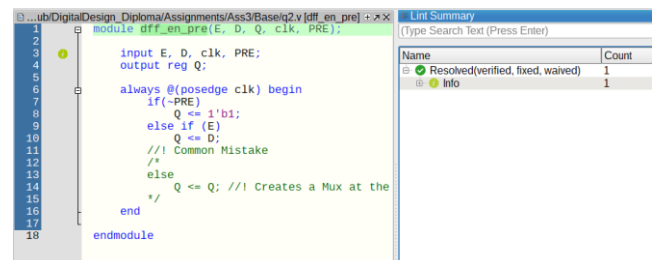


Figure 3 Questa Lint

## Question 2:

Write a testbench for question 5 part C in assignment 2. Test the parameterized asynchronous FlipFlop using 2 testbenches, testbench 1 that overrides the design with FF\_TYPE = "DFF" and the testbench 2 overrides parameter with FF\_TYPE = "TFF".

```

1  module dff_tb();
2      reg d, rstn, clk;
3      wire Q, Q_bar, Q_expected, Q_bar_expected;
4
5      ff #(.FF_TYPE("DFF")) DUT(d, rstn, clk, Q, Q_bar);
6      dff goldenModel(d, rstn, clk, Q_expected, Q_bar_expected);
7
8      initial begin
9          clk = 0;
10         forever
11             #1 clk = ~clk;
12     end
13
14     initial begin
15         rstn = 0;
16         @(negedge clk);
17         if((Q != Q_expected) && (Q_bar != Q_bar_expected)) begin
18             $display("Error - Reset");
19             $exit;
20         end
21         rstn = 1;
22
23         repeat(100) begin
24             d = $random;
25             @(negedge clk);
26             if((Q != Q_expected) && (Q_bar != Q_bar_expected)) begin
27                 $display("Error - Incorrect Value");
28                 $exit;
29             end
30         end
31
32         $display("Design Passed!");
33         $exit;
34     end
35 endmodule

```

Figure 4 DFF TB

```

1  module tff_tb();
2      reg t, rstn, clk;
3      wire Q, Q_bar, Q_expected, Q_bar_expected;
4
5      ff #(.FF_TYPE("TFF")) DUT(t, rstn, clk, Q, Q_bar);
6      tff goldenModel(t, rstn, clk, Q_expected, Q_bar_expected);
7
8      initial begin
9          clk = 0;
10         forever
11             #1 clk = ~clk;
12     end
13
14     initial begin
15         rstn = 0;
16         @(negedge clk);
17         if((Q != Q_expected) && (Q_bar != Q_bar_expected)) begin
18             $display("Error - Reset");
19             $exit;
20         end
21         rstn = 1;
22
23         repeat(100) begin
24             t = $random;
25             @(negedge clk);
26             if((Q != Q_expected) && (Q_bar != Q_bar_expected)) begin
27                 $display("Error - Incorrect Value");
28                 $exit;
29             end
30         end
31
32         $display("Design Passed!");
33         $exit;
34     end
35 endmodule

```

Figure 5 TFF TB

```

1  vlib work
2
3  vlog ff.v ff_tb.v
4
5  vsim -voptargs=+acc work.tff_tb
6
7  add wave *
8
9  run -all
10
11 #quit -sim
12

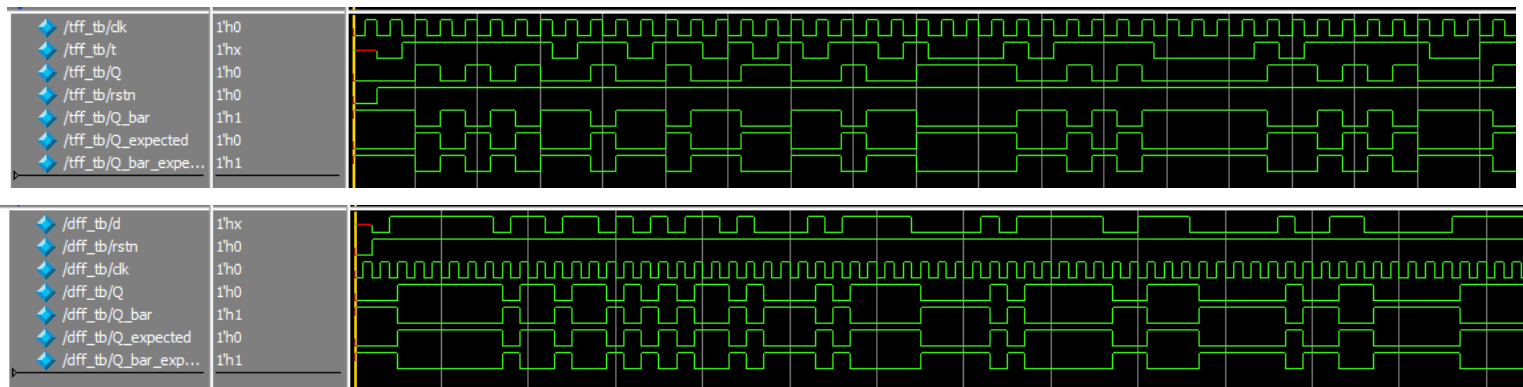
```

```

1  vlib work
2
3  vlog ff.v ff_tb.v
4
5  vsim -voptargs=+acc work.dff_tb
6
7  add wave *
8
9  run -all
10
11 #quit -sim
12

```

Figure 6 DO Files



## Question 3:

Implement BCD up counter (MOD 10 counter), where the counter has 10 states. The counter will divide the clock frequency by 10.

```

1  module bcd_counter(clk, rst, clk_div10_out);
2      input clk, rst;
3      output clk_div10_out;
4      reg [3:0] cnt;
5      assign clk_div10_out = &(~(cnt ^ 4'd10));
6
7      always @(posedge clk or posedge rst) begin
8          if(rst)
9              cnt <= 0;
10         else if(clk_div10_out)
11             cnt <= 0;
12         else
13             cnt <= cnt + 1;
14     end
15 endmodule

```

Figure 7 BCD Counter Code

```

1  module bcd_counter_tb();
2      reg clk, rst;
3      wire clk_div10_out;
4
5      bcd_counter DUT(clk, rst, clk_div10_out);
6
7      initial begin
8          clk = 0;
9          forever begin
10             #50 clk = ~clk;
11         end
12     end
13
14     initial begin
15         rst = 1;
16         @(negedge clk);
17         rst = 0;
18         #10000;
19
20         $exit;
21     end
22 endmodule

```

Figure 8 TB Code

```

1  vlib work
2
3  vlog bcd_counter_tb.v bcd_counter.v
4
5  vsim -voptargs=+acc work.bcd_counter_tb
6
7  add wave *
8
9  run -all
10
11 #quit -sim
12
13

```

Figure 9 DO File

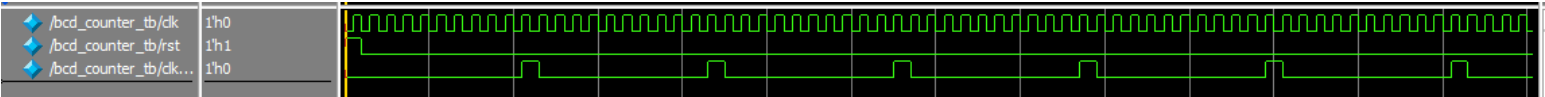


Figure 11 Waveform Output

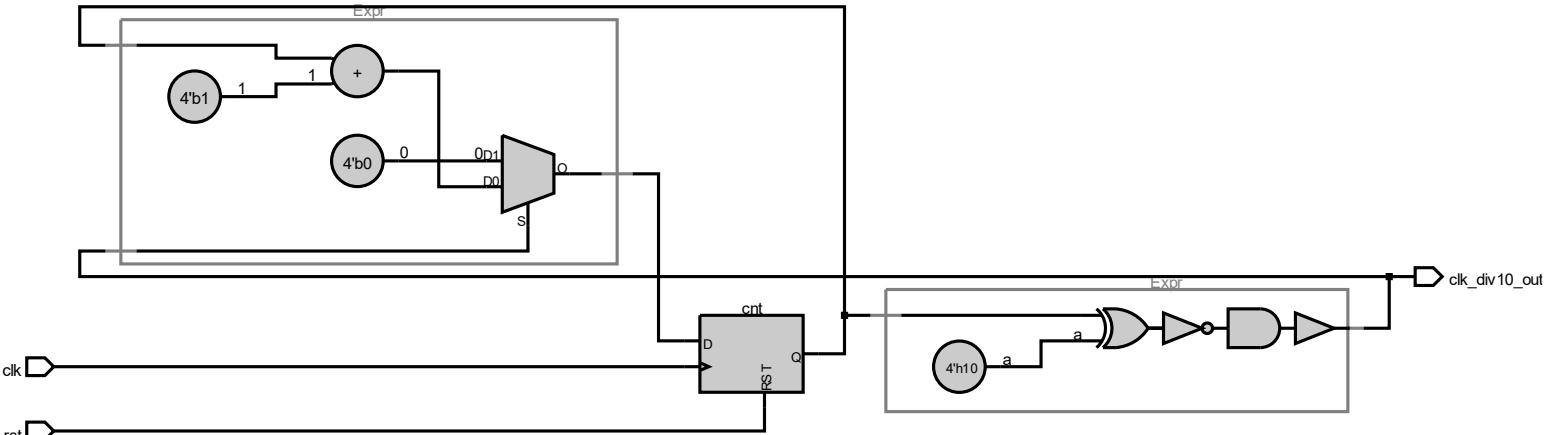


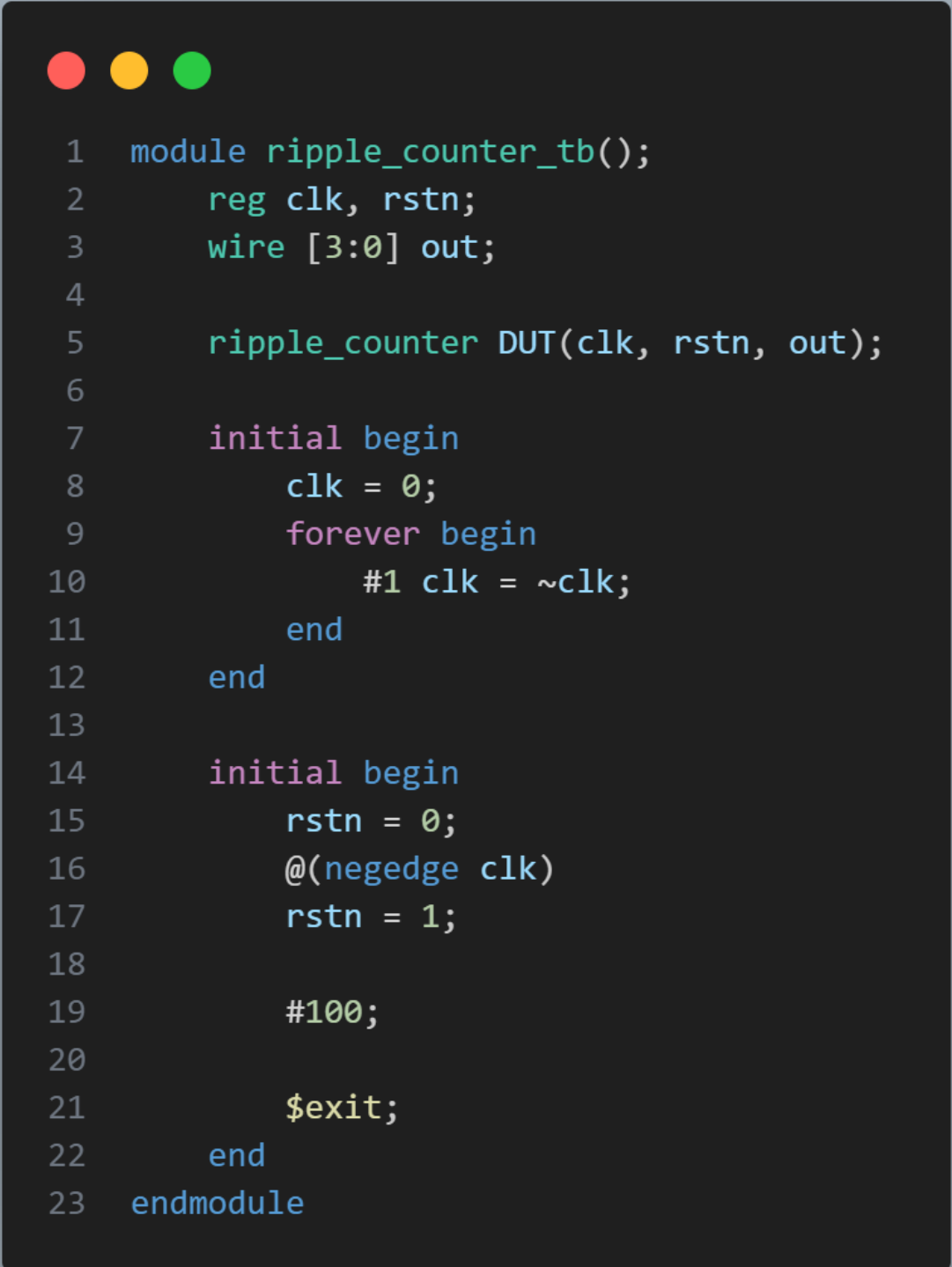
Figure 10 BCD Counter Schematic

## Question 4: Ripple Counter

```
1  module ripple_counter(  
2      clk,  
3      rstn,  
4      out  
5  );  
6      input clk, rstn;  
7      output [3:0] out;  
8  
9      wire [3:0] clk_chain;  
10     assign clk_chain[0] = clk;  
11     genvar i;  
12     generate  
13         for(i = 0; i < 4; i = i +1) begin  
14             wire d_q_bar;  
15             if(i > 0)  
16                 assign clk_chain[i] = out[i - 1];  
17  
18             dff u_dff(d_q_bar, rstn, clk_chain[i], out[i], d_q_bar);  
19         end  
20     endgenerate  
21 endmodule
```

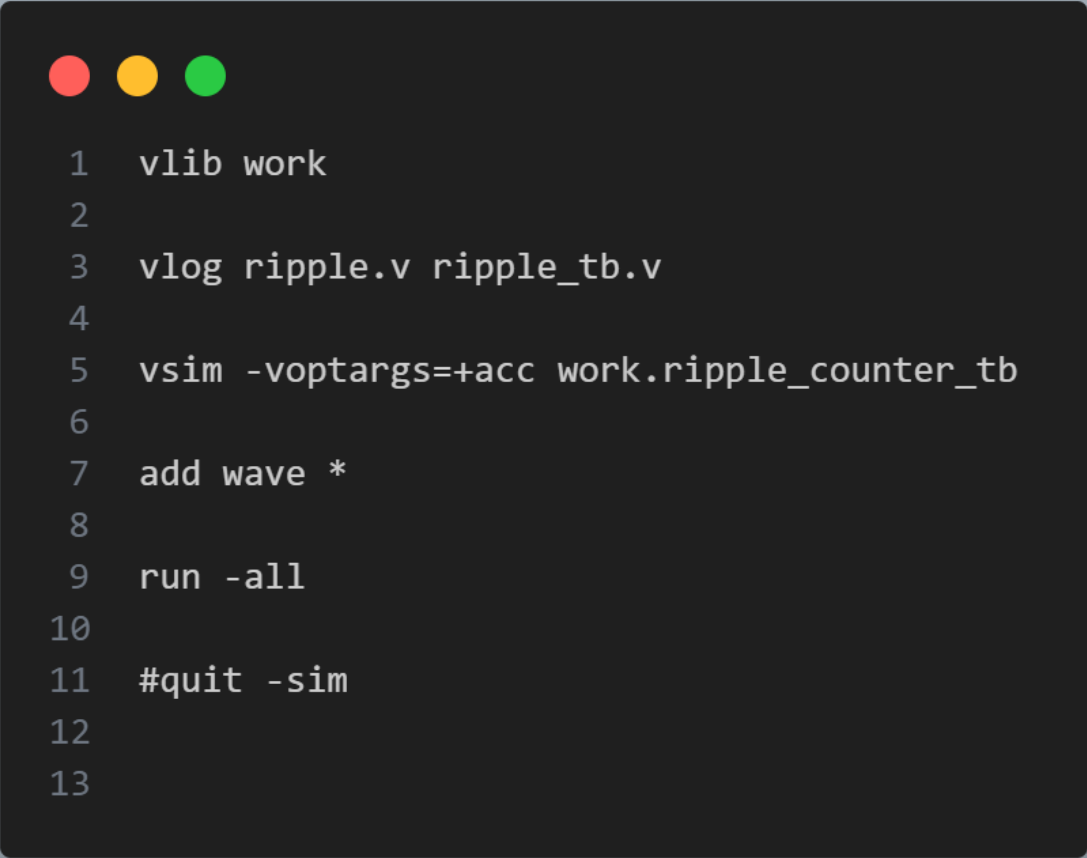
Figure 12 Ripple Counter Code





```
1  module ripple_counter_tb();
2      reg clk, rstn;
3      wire [3:0] out;
4
5      ripple_counter DUT(clk, rstn, out);
6
7      initial begin
8          clk = 0;
9          forever begin
10             #1 clk = ~clk;
11         end
12     end
13
14     initial begin
15         rstn = 0;
16         @(negedge clk)
17             rstn = 1;
18
19         #100;
20
21         $exit;
22     end
23 endmodule
```

*Figure 13 Ripple Counter TB*



```
1  vlib work
2
3  vlog ripple.v ripple_tb.v
4
5  vsim -voptargs=+acc work.ripple_counter_tb
6
7  add wave *
8
9  run -all
10
11 #quit -sim
12
13
```

Figure 14 Do File

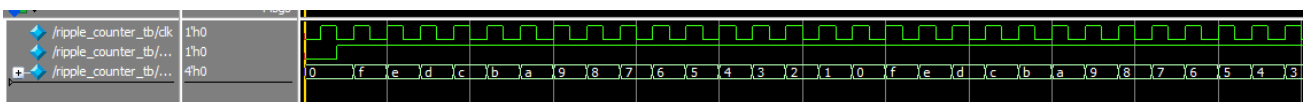


Figure 15 Waveform

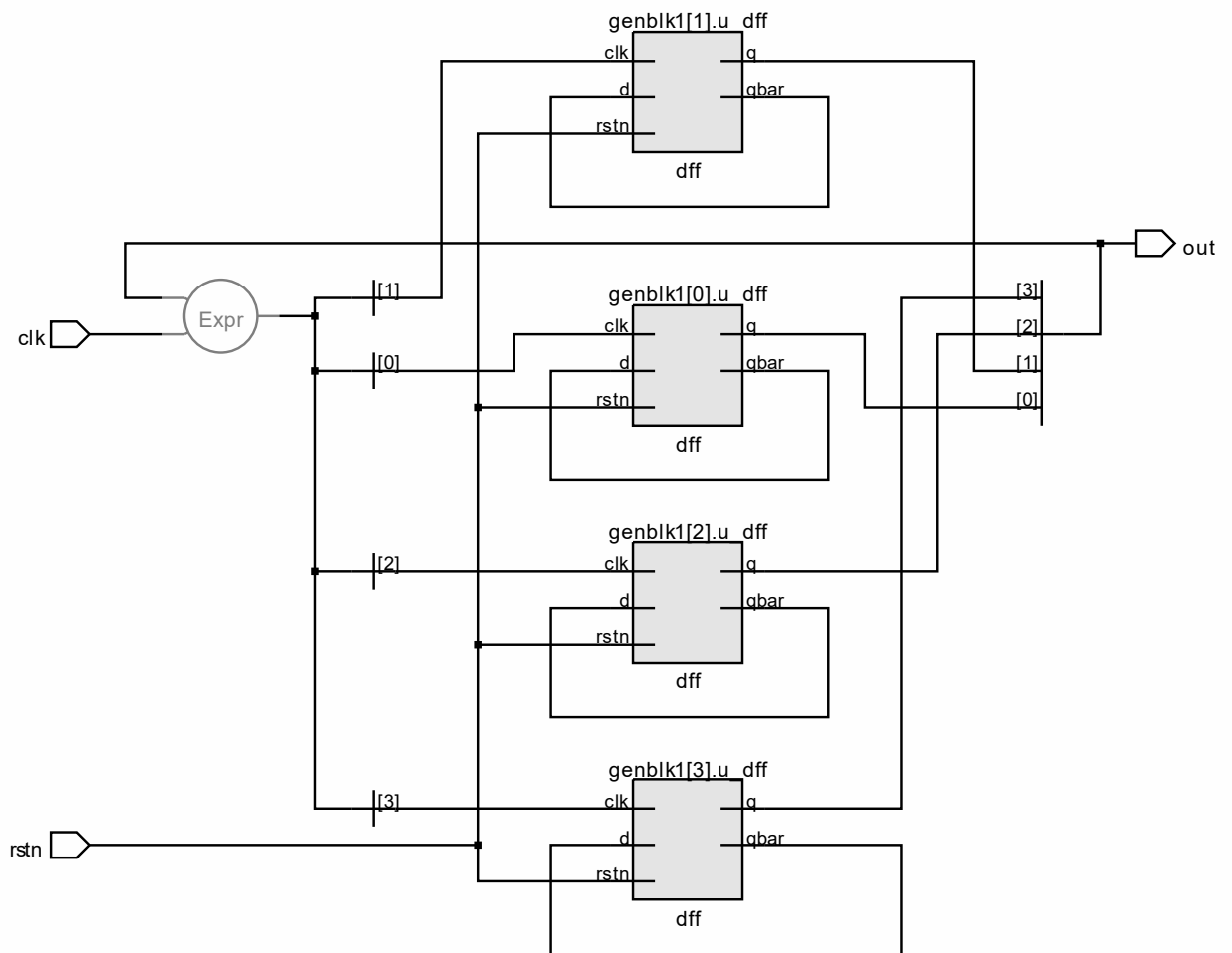


Figure 16 Ripple Counter Schematic

## Question 5:

```

1  module shift_reg(sclr, sset, shiftin, load, clk, enable, aclr, aset, data, shiftout, q);
2      parameter LOAD_AVALUE = 1;
3      parameter SHIFT_DIRECTION = "LEFT";
4      parameter LOAD_SVALUE = 1;
5      parameter SHIFT_WIDTH = 8;
6
7      input sclr, sset, shiftin, load, clk, enable, aclr, aset;
8      input [SHIFT_WIDTH - 1 : 0] data;
9      output reg shiftout;
10     output reg [SHIFT_WIDTH - 1 : 0] q;
11
12     always @(posedge clk or posedge aclr or posedge aset) begin
13         if(aclr)
14             q <= 0;
15         else if(aset)
16             q <= LOAD_AVALUE;
17         else if(enable) begin
18             if(sclr)
19                 q <= 0;
20             else if(sset)
21                 q <= LOAD_SVALUE;
22             else if(load)
23                 q <= data;
24             else if(SHIFT_DIRECTION == "LEFT")
25                 {shiftout, q} = {q, shiftin};
26             else if(SHIFT_DIRECTION == "RIGHT")
27                 {q, shiftout} = {shiftin, q};
28         end /* End of Enable IF STATEMENT */
29     end /* End of Always Block */
30 endmodule

```

Figure 17 SLE Shift Register Code

```

module shift_reg_tb();
    parameter LOAD_AVALUE = 2;
    parameter SHIFT_DIRECTION = "LEFT";
    parameter LOAD_SVALUE = 4;
    parameter SHIFT_WIDTH = 8;

    reg sclr, sset, shiftin, load, clk, enable, aclr, aset;
    reg [SHIFT_WIDTH - 1 : 0] data, q_expected;
    wire shiftout;
    wire [SHIFT_WIDTH - 1 : 0] q;

    shift_reg #(LOAD_AVALUE(LOAD_AVALUE),
                .LOAD_SVALUE(LOAD_SVALUE),
                .SHIFT_DIRECTION(SHIFT_DIRECTION),
                .SHIFT_WIDTH(SHIFT_WIDTH))
        DUT (
            .sclr(sclr),
            .sset(sset),
            .shiftin(shiftin),
            .load(load),
            .clk(clk),
            .enable(enable),
            .aclr(aclr),
            .aset(aset),
            .shiftout(shiftout),
            .q(q)
        );

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

    initial begin
        // Check for aclr
        aset = 1;
        aclr = 1;
        repeat(50) begin
            sclr = $random;
            sset = $random;
            shiftin = $random;
            load = $random;
            enable = $random;
            data = $random;
            @(negedge clk);
            if(q != 0) begin

```

```

        $display("Error - aclr");
        $exit;
    end
end

// check for aset
aclr = 0;
aset = 1;
repeat(50) begin
    sclr = $random;
    sset = $random;
    shiftin = $random;
    load = $random;
    enable = $random;
    data = $random;
    @(negedge clk);
    if(q != LOAD_AVALUE) begin
        $display("Error - aset");
        $exit;
    end
end

// Test for sclr
aclr = 0;
aset = 0;
sclr = 1;
sset = 1;
repeat(50) begin
    shiftin = $random;
    load = $random;
    enable = $random;
    data = $random;
    @(negedge clk);
    if(q != 0) begin
        $display("Error - sclr");
        $exit;
    end
end

// Test for sset
aclr = 0;
aset = 0;
sclr = 0;
sset = 1;
enable = 1;
repeat(50) begin
    shiftin = $random;
    load = $random;

```

```

        data = $random;
        @(negedge clk);
        if(q != LOAD_SVALUE) begin
            $display("Error - sset");
            $exit;
        end
    end
end

// Test for load
aclr = 0;
aset = 0;
sclr = 0;
sset = 0;
load = 1;
enable = 1;
repeat(50) begin
    shiftin = $random;
    data = $random;
    q_expected = data;
    @(negedge clk);
    if(q != q_expected) begin
        $display("Error - load");
        $exit;
    end
end

// test for shifting
// Clearing Input for signifcant amount of time
aclr = 1;
repeat(50) @(negedge clk);
aclr = 0;
// Setting value
aset = 1;
@(negedge clk);
aset = 0;
// Deactivating Control Signals
aclr = 0;
aset = 0;
sclr = 0;
sset = 0;
load = 0;
shiftin = 0;
enable = 1;
$display("Starting Shift Test");
repeat(10) @(negedge clk);
$exit;
end
endmodule

```

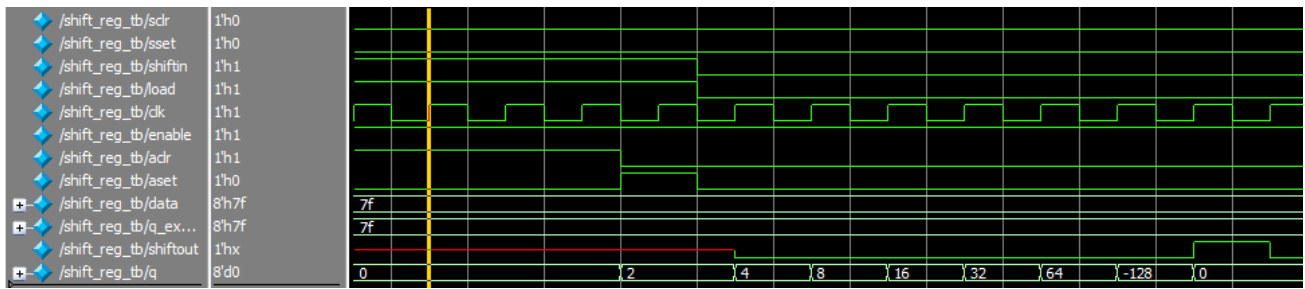


Figure 18 Shift Functionality Working

```

1  vlib work
2
3  vlog shift_reg.v shift_reg_tb.v
4
5  vsim -voptargs=+acc work.shift_reg_tb
6
7  add wave *
8
9  run -all
10
11 #quit -sim
12
13

```

Figure 19 Do file



## Question 6:

Implement the following SLE (sequential logic element). This design will act as flipflop or latch based on the LAT signal as demonstrated in the truth table.

```

1  module sle(D, clk, en, ALn, ADn, SLn, SD, LAT, Q);
2      input D, clk, en, ALn, ADn, SLn, SD, LAT;
3      output Q;
4      reg QLat;
5      reg Qff;
6
7
8      always @(*) begin
9          if(LAT) begin
10             if(clk) begin
11                 if(en) begin
12                     if(~SLn)
13                         QLat = SD;
14                     else
15                         QLat = D;
16                 end
17             end
18         end
19     end
20
21     assign Q = (LAT) ? QLat : Qff;
22
23     always @(posedge clk or negedge ALn or posedge LAT) begin
24         if(~ALn)
25             Qff <= ~ADn;
26         else if(~LAT) begin
27             if(en) begin
28                 if(~SLn)
29                     Qff <= SD;
30                 else
31                     Qff <= D;
32             end
33         end
34     end
35 endmodule

```

Figure 20 SLE Latch/FF Code

The screenshot shows a Verilog code editor with the following code:

```

1  module sle(D, clk, en, ALn, ADn, SLn, SD, LAT, Q)
2      input D, clk, en, ALn, ADn, SLn, SD, LAT;
3      output Q;
4      reg QLat;
5      reg Qff;
6
7
8      always @(*) begin
9          if(LAT) begin
10             if(clk) begin
11                 if(en) begin
12                     if(~SLn)
13                         QLat = SD;
14                     else
15                         QLat = D;
16                 end
17             end
18         end
19     end
20
21     assign Q = (LAT) ? QLat : Qff;
22

```

The Lint Summary window on the right shows the following results:

Name	Count
Resolved(verified, fixed, waived)	2
Warning	1
reset_set_non_const_assign	1
Info	1
multi_ports_in_single_line	1

Figure 22 Questa Lint

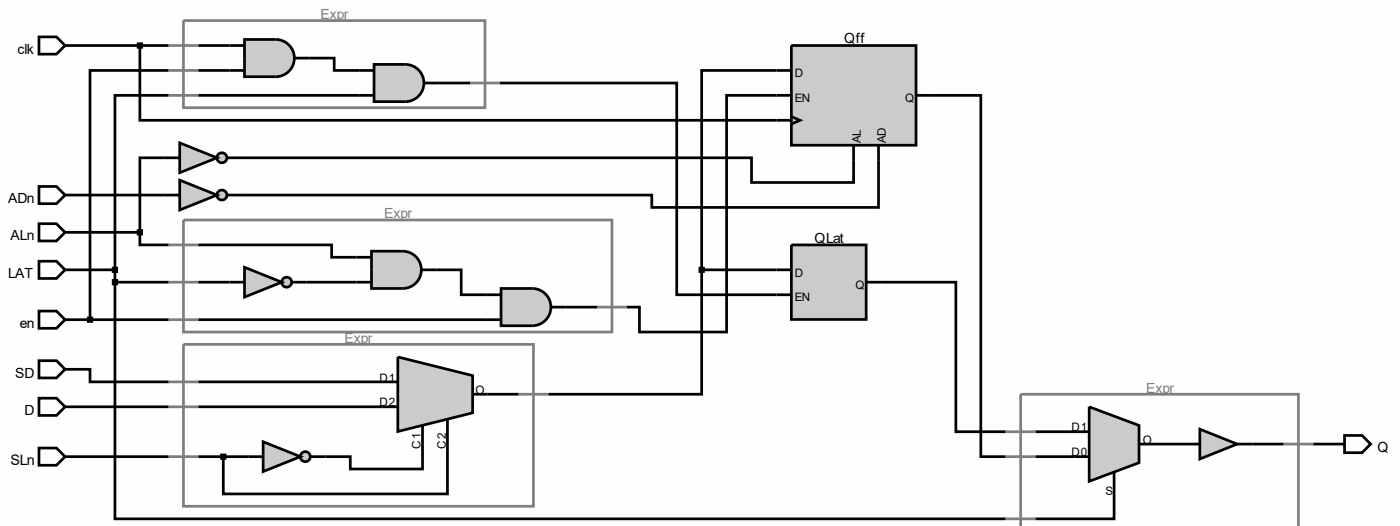


Figure 21 Schematic

```

module sle_tb();
    reg D, clk, en, ALn, ADn, SLn, SD, LAT;
    wire Q;

    sle DUT(D, clk, en, ALn, ADn, SLn, SD, LAT, Q);

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

    initial begin
        LAT = 0;
    end

```

```

ALn = 0;
repeat(10) begin
    ADn = $random;
    D = $random;
    @(negedge clk);
    if(Q != ~ADn) begin
        $display("Error Async Load");
        $exit;
    end
end

ALn = 1;
SLn = 0;
en = 1;
repeat(10)begin
    SD = $random;
    D = $random;
    @(negedge clk);
    if(Q != SD) begin
        $display("Error Sync Load");
        $exit;
    end
end

ALn = 1;
SLn = 1;
en = 1;
repeat(10)begin
    SD = $random;
    D = $random;
    @(negedge clk);
    if(Q != D) begin
        $display("Error Latch Operation");
        $exit;
    end
end

LAT = 1;

ALn = 1;
SLn = 0;
en = 1;
repeat(20)begin
    SD = $random;
    D = $random;
    #1;
end

```

```

    ALn = 1;
    SLn = 1;
    en = 1;
    repeat(20)begin
        SD = $random;
        D = $random;
        #1;
    end

    $display("Design Works");
    $exit;
end
endmodule

```

Figure 23 Testbench Code

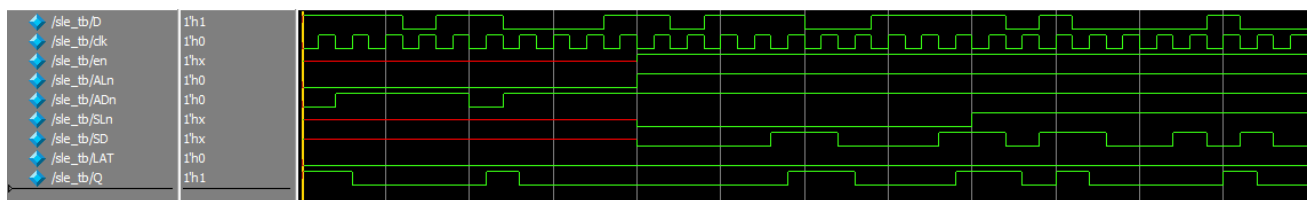


Figure 24 Tests when LAT = 0

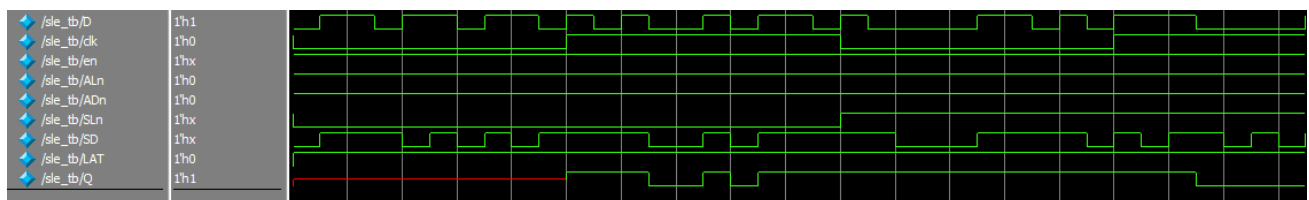


Figure 25 Checks when LAT = 1