Assignment 3

Ameer Ashraf Louly – G3

Question 1:

```
module dff_en_pre_tb();
    reg E, D, clk, PRE, Q_expected;
    wire Q;

    dff_en_pre DUT(E, D, Q, clk, PRE);

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

    initial begin
        $monitor("D = %d, Q = %d", D, Q);
    end

    initial begin
        // Preset Check
        PRE = 0;
        @(negedge clk);
        if(Q != 1) begin
            $display("Error - Incorrect PRESET");
            $exit;
        end
        PRE = 1;
        $display("PRESET WORKS");

        // DFF check
        PRE = 0;
        @(negedge clk);
        PRE = 1;
        E = 1;
        repeat(50) begin
            D = $random;
            Q_expected = D;
            @(negedge clk);
            if(Q != Q_expected) begin
                $display("Enable doesn't work");
                $exit;
            end
        end /* End of Repeat */
        $display("DFF WORKS");

        // Enable Closed Check
        E = 1;
        D = 1;
        @(negedge clk);
        E = 0;
        repeat(50) begin
            D = $random;
            @(negedge clk);
            if(Q != 1) begin
                $display("Enable doesn't work");
                $exit;
            end
        end /* End of Repeat */
        $display("ENABLE WORKS");

        $display("Design Works as expected");
        $exit;
    end
endmodule
```

Figure 1 TB Code

```
vlib work

vlog q2.v q2_tb.v

vsim -voptargs=+acc work.dff_en_pre_tb

add wave *

run -all

#quit -sim
```

Figure 2 DO File

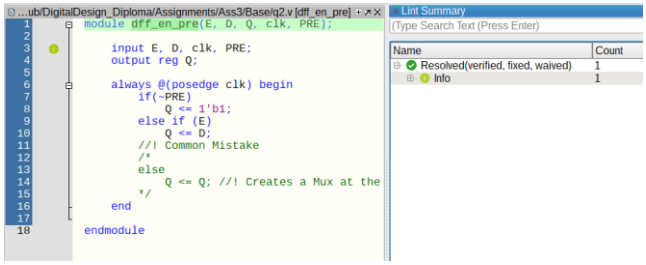


Figure 3 Questa Lint

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”.

```
module dff_tb();
    reg d, rstn, clk;
    wire Q, Q_bar, Q_expected,Q_bar_expected;

    ff #(.FF_TYPE("DFF")) DUT(d, rstn, clk, Q, Q_bar);
    dff goldenModel(d, rstn, clk, Q_expected,Q_bar_expected);

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

    initial begin
        rstn = 0;
        @(negedge clk);
        if((Q != Q_expected) && (Q_bar != Q_bar_expected)) begin
            $display("Error - Reset");
            $exit;
        end
        rstn = 1;

        repeat(100) begin
            d = $random;
            @(negedge clk);
            if((Q != Q_expected) && (Q_bar != Q_bar_expected)) begin
            $display("Error - Incorrect Value");
            $exit;
            end
        end

        $display("Design Passed!");
        $exit;
    end
endmodule
```

Figure 4 DFF TB

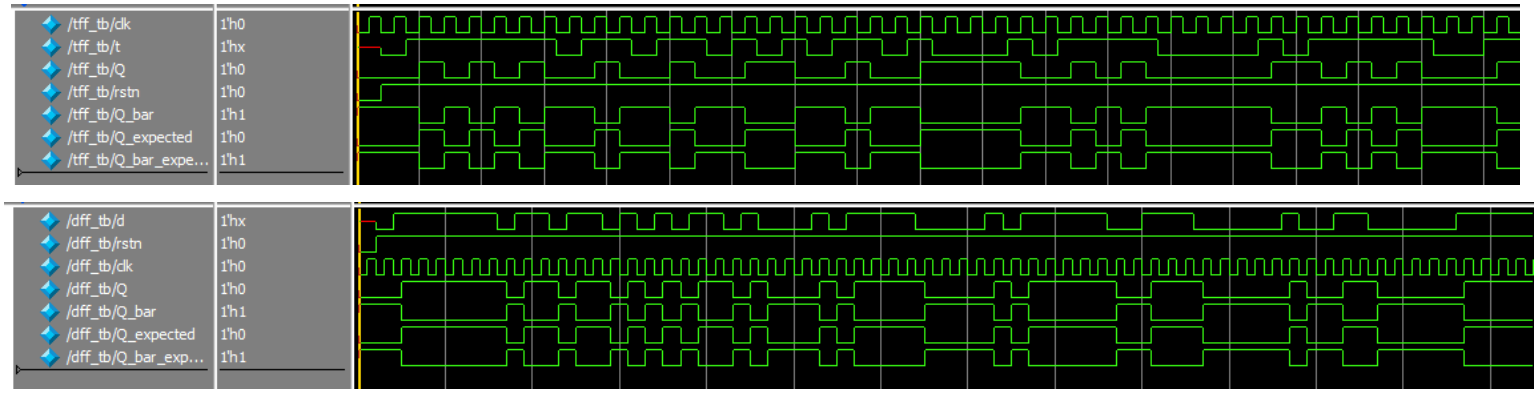
```
module tff_tb();
    reg t, rstn, clk;
    wire Q, Q_bar, Q_expected,Q_bar_expected;

    ff #(.FF_TYPE("TFF")) DUT(t, rstn, clk, Q, Q_bar);
    tff goldenModel(t, rstn, clk, Q_expected,Q_bar_expected);

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

    initial begin
        rstn = 0;
        @(negedge clk);
        if((Q != Q_expected) && (Q_bar != Q_bar_expected)) begin
            $display("Error - Reset");
            $exit;
        end
        rstn = 1;

        repeat(100) begin
            t = $random;
            @(negedge clk);
            if((Q != Q_expected) && (Q_bar != Q_bar_expected)) begin
            $display("Error - Incorrect Value");
            $exit;
            end
        end

        $display("Design Passed!");
        $exit;
    end
endmodule
```

Figure 5 TFF TB

```
vlib work

vlog ff.v ff_tb.v

vsim -voptargs=+acc work.tff_tb

add wave *

run -all

#quit -sim
```

```
vlib work

vlog ff.v ff_tb.v

vsim -voptargs=+acc work.dff_tb

add wave *

run -all

#quit -sim
```

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.

```
module bcd_counter(clk, rst, clk_div10_out);
    input clk, rst;
    output clk_div10_out;
    reg [3:0] cnt;
    assign clk_div10_out = &(~(cnt ^ 4'd10));

    always @(posedge clk or posedge rst) begin
        if(rst)
            cnt <= 0;
        else if(clk_div10_out)
            cnt <= 0;
        else
            cnt <= cnt + 1;
    end
endmodule
```

Figure 7 BCD Counter Code

```
module bcd_counter_tb();
    reg clk, rst;
    wire clk_div10_out;

    bcd_counter DUT(clk, rst, clk_div10_out);

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

    initial begin
        rst = 1;
        @(negedge clk);
        rst = 0;
        #10000;

        $exit;
    end
endmodule
```

Figure 8 TB Code

```
vlib work

vlog bcd_counter_tb.v bcd_counter.v

vsim -voptargs=+acc work.bcd_counter_tb

add wave *

run -all

#quit -sim
```

Figure 9 DO File

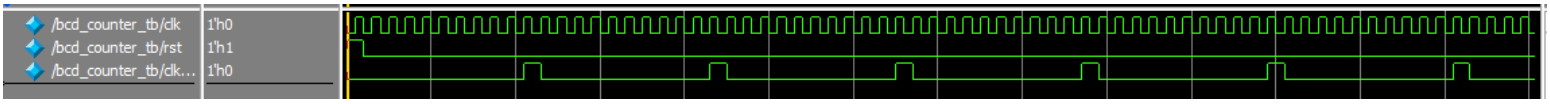


Figure 11 Waveform Output

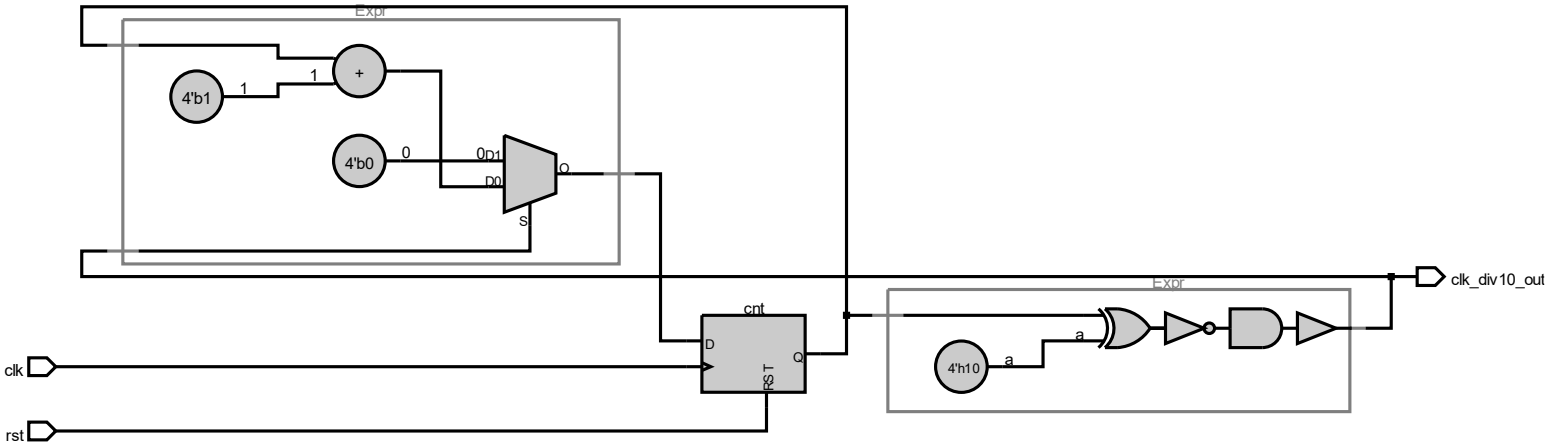


Figure 10 BCD Counter Schematic

Question 4: Ripple Counter

```
module ripple_counter(
    clk,
    rstn,
    out
);
    input clk, rstn;
    output [3:0] out;

    wire [3:0] clk_chain;
    assign clk_chain[0] = clk;
    genvar i;
    generate
        for(i = 0; i < 4; i = i +1) begin
            wire d_q_bar;
            if(i > 0)
                assign clk_chain[i] = out[i - 1];

            dff u_dff(d_q_bar, rstn, clk_chain[i], out[i], d_q_bar);
        end
    endgenerate
endmodule
```

Figure 12 Ripple Counter Code

```
module ripple_counter_tb();
    reg clk, rstn;
    wire [3:0] out;

    ripple_counter DUT(clk, rstn, out);

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

    initial begin
        rstn = 0;
        @(negedge clk)
        rstn = 1;

        #100;

        $exit;
    end
endmodule
```

Figure 13 Ripple Counter TB

```
vlib work

vlog ripple.v ripple_tb.v

vsim -voptargs=+acc work.ripple_counter_tb

add wave *

run -all

#quit -sim
```

Figure 14 Do File

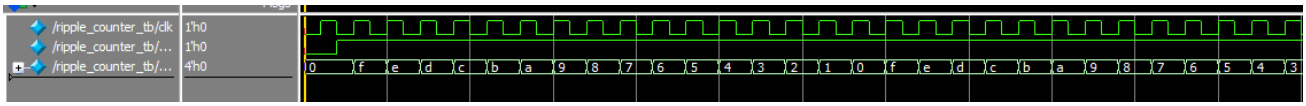


Figure 15 Waveform

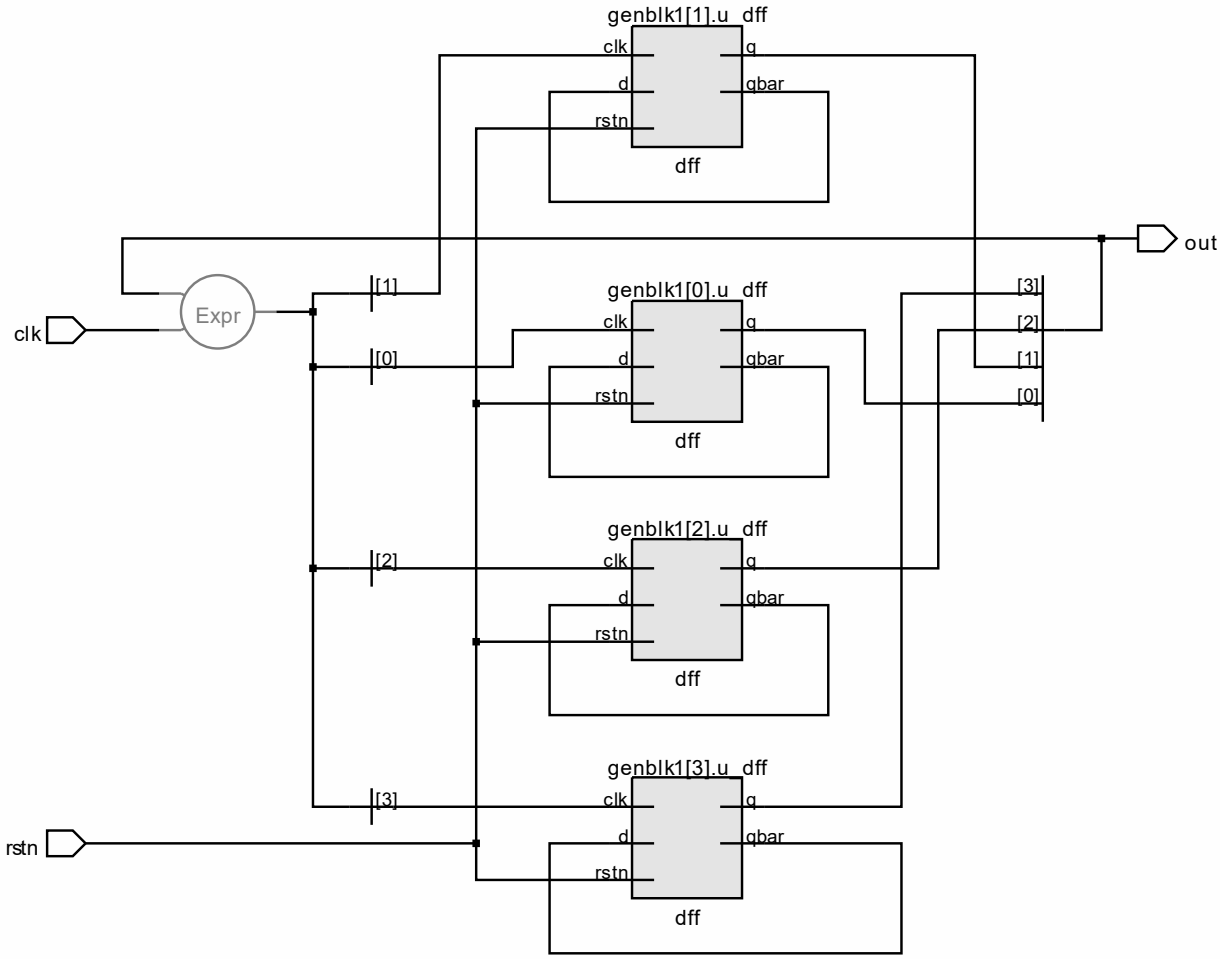


Figure 16 Ripple Counter Schematic

Question 5:

```
module shift_reg(sclr, sset, shiftin, load, clk, enable, aclr, aset, data, shiftout, q);
    parameter LOAD_AVALUE = 1;
    parameter SHIFT_DIRECTION = "LEFT";
    parameter LOAD_SVALUE = 1;
    parameter SHIFT_WIDTH = 8;

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

    always @(posedge clk or posedge aclr or posedge aset) begin
        if(aclr)
            q <= 0;
        else if(aset)
            q <= LOAD_AVALUE;
        else if(enable) begin
            if(sclr)
                q <= 0;
            else if(sset)
                q <= LOAD_SVALUE;
            else if(load)
                q <= data;
            else if(SHIFT_DIRECTION == "LEFT")
                {shiftout, q} = {q, shiftin};
            else if(SHIFT_DIRECTION == "RIGHT")
                {q, shiftout} = {shiftin, q};
        end /* End of Enable IF STATEMENT */
    end /* End of Always Block */
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

        // 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
```

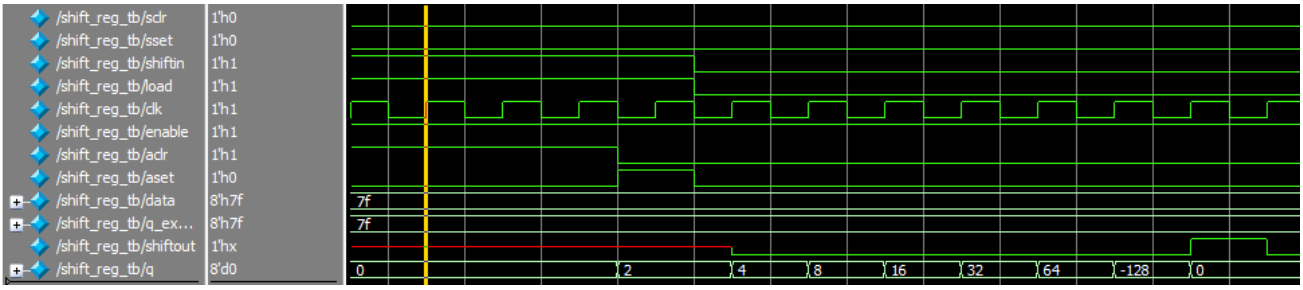


Figure 18 Shift Functionality Working

```
vlib work

vlog shift_reg.v shift_reg_tb.v

vsim -voptargs=+acc work.shift_reg_tb

add wave *

run -all

#quit -sim
```

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.

```
module sle(D, clk, en, ALn, ADn, SLn, SD, LAT, Q);
    input D, clk, en, ALn, ADn, SLn, SD, LAT;
    output Q;
    reg QLat;
    reg Qff;


    always @(*) begin
        if(LAT) begin
            if(clk) begin
                if(en) begin
                    if(~SLn)
                        QLat = SD;
                    else
                        QLat = D;
                end
            end
        end
    end

    assign Q = (LAT) ? QLat : Qff;

    always @(posedge clk or negedge ALn or posedge LAT) begin
        if(~ALn)
            Qff <= ~ADn;
        else if(~LAT) begin
            if(en) begin
                if(~SLn)
                    Qff <= SD;
                else
                    Qff <= D;
            end
        end
    end
endmodule
```

Figure 20 SLE Latch/FF Code

D:/Github/DigitalDesign_Diploma/Assignments/Ass3/Base/SLE.v [sle]

```
module sle(D, clk, en, ALn, ADn, SLn, SD, LAT, Q
    input D, clk, en, ALn, ADn, SLn, SD, LAT;
    output Q;
    reg QLat;
    reg Qff;


    always @(*) begin
        if(LAT) begin
            if(clk) begin
                if(en) begin
                    if(~SLn)
                        QLat = SD;
                    else
                        QLat = D;
                end
            end
        end
    end

    assign Q = (LAT) ? QLat : Qff;
```

Lint Summary

(Type Search Text (Press Enter))

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

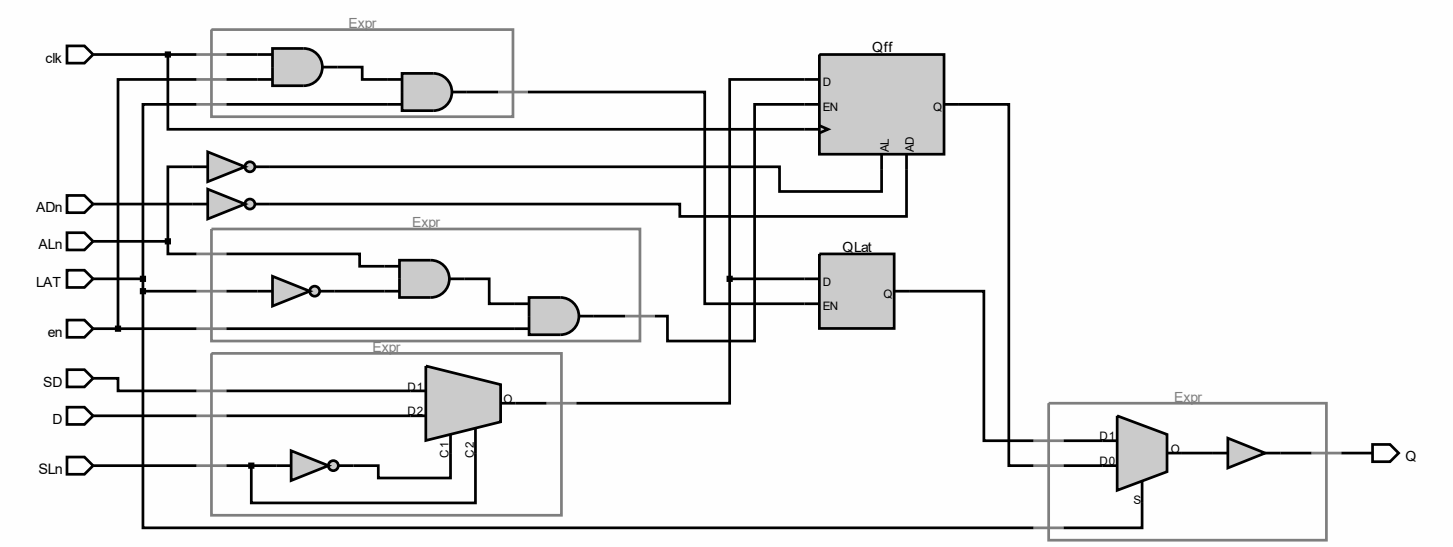


Figure 21 Schematic

```
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;
```

```
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