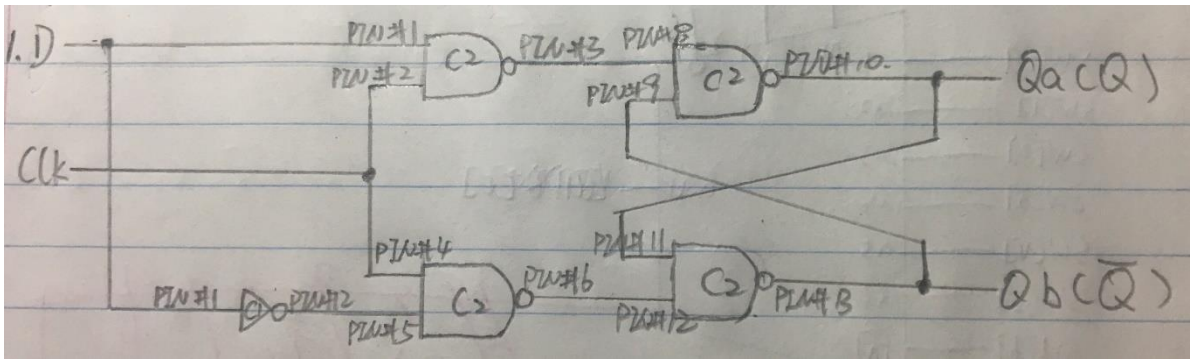
Prelab for Lab4

Xiaoyu Zhou, 1004081147

Part I

1. Draw a schematic of the gated D latch using interconnected 7400-series chips draw a schematic of the gated D latch using interconnected 7400-series chips.



CHIPS USED:

C1 – 74LS04

C2 – 74LS00

CONNCETED TO ALL CHIPS:

PIN#7 – Gnd

PIN#14 – Vcc

4. Are there any input combinations of Clk and D that should NOT be the first you test?

No, we can use all the inputs since PIN#1 and PIN#5 will never be the same because they are D and D'. Thus, PIN#8 and PIN#12, which are the two inputs of the latch, will never be 0s at the same time. So, we don't have to avoid any inputs.

Part II

1. Verilog.

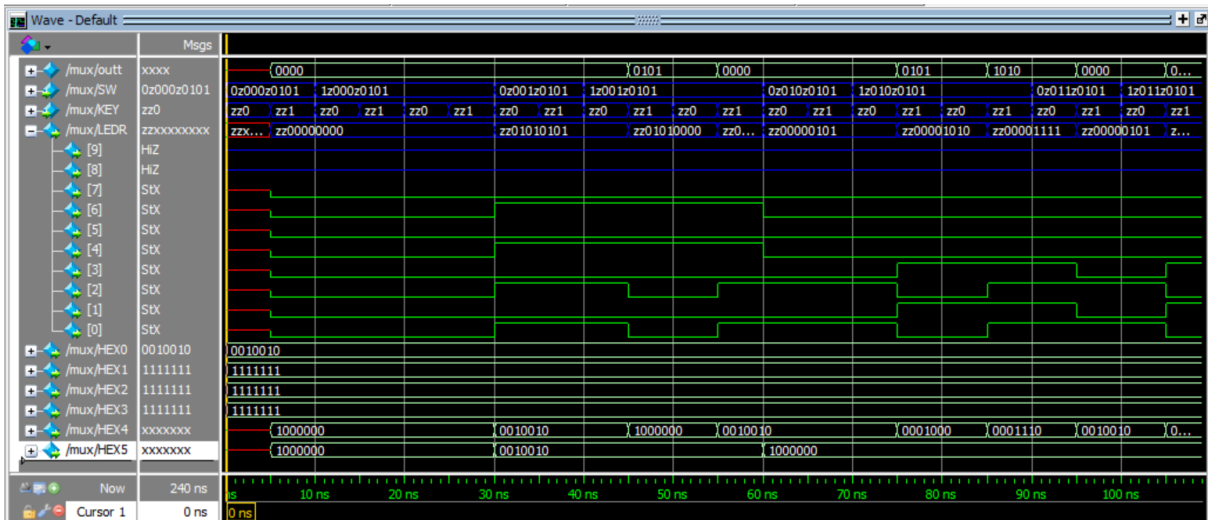
```
module mux(LEDR, SW, KEY, HEX0, HEX1, HEX2, HEX3,                HEX4, HEX5);
```

```
parameter Narrow = 4;
wire [Narrow-1:0] outt;
input [9:0] SW;
    input [2:0] KEY;
output [9:0] LEDR;
    output [6:0] HEX0;
    output [6:0] HEX1;
    output [6:0] HEX2;
    output [6:0] HEX3;
    output [6:0] HEX4;
    output [6:0] HEX5;


    trans u1(
      .A(SW[3:0]),
      .B(outt),
            .in1(SW[7:5]),
      .out1(LEDR[7:0])
      );

    D_flipflop a(
            .d(LEDR[3:0]),
            .clock(KEY[0]),
            .reset_n(SW[9]),
            .q(outt)
            );

    hex u2(
            .in2(LEDR[3]),
            .in3(LEDR[2]),
            .in4(LEDR[1]),
            .in5(LEDR[0]),
            .o2(HEX4[0]),
            .o3(HEX4[1]),
            .o4(HEX4[2]),
            .o5(HEX4[3]),
            .o6(HEX4[4]),
            .o7(HEX4[5]),
            .o8(HEX4[6])
            );

    hex u3(
            .in2(LEDR[7]),
            .in3(LEDR[6]),
            .in4(LEDR[5]),
            .in5(LEDR[4]),
            .o2(HEX5[0]),
            .o3(HEX5[1]),
            .o4(HEX5[2]),
            .o5(HEX5[3]),
            .o6(HEX5[4]),
            .o7(HEX5[5]),
            .o8(HEX5[6])
            );

    hex A(
            .in2(SW[3]),
            .in3(SW[2]),
            .in4(SW[1]),
            .in5(SW[0]),
            .o2(HEX0[0]),
            .o3(HEX0[1]),
            .o4(HEX0[2]),
            .o5(HEX0[3]),
            .o6(HEX0[4]),
            .o7(HEX0[5]),
            .o8(HEX0[6])
            );



    assign HEX1[6] = 1;
    assign HEX1[5] = 1;
    assign HEX1[4] = 1;
    assign HEX1[3] = 1;
    assign HEX1[2] = 1;
    assign HEX1[1] = 1;
    assign HEX1[0] = 1;

    assign HEX2[6] = 1;
    assign HEX2[5] = 1;
    assign HEX2[4] = 1;
    assign HEX2[3] = 1;
    assign HEX2[2] = 1;
    assign HEX2[1] = 1;
    assign HEX2[0] = 1;
        assign HEX3[6] = 1;
        assign HEX3[5] = 1;
        assign HEX3[4] = 1;
        assign HEX3[3] = 1;
        assign HEX3[2] = 1;
        assign HEX3[1] = 1;
        assign HEX3[0] = 1;

endmodule

module D_flipflop(d, clock, reset_n, q);
        parameter Narrow = 4;
        parameter N = 1;
    input [Narrow-1:0] d;
        input [N-1:0] clock;
        input [N-1:0] reset_n;
        output [Narrow-1:0] q;
        reg [Narrow-1:0] q;

        always @(posedge clock)
        begin
            if (reset_n == 1'b0)
                q <= 0;
            else
                q <= d;
        end
endmodule

module trans(A, B, in1, out1);
    parameter Width = 8;
        parameter Narrow = 4;
        parameter W = 3;
    input [Narrow-1:0] A;
        input [Narrow-1:0] B;
        input [W-1:0] in1;
    output [Width-1:0]out1;
        reg [Width-1:0]out1;
        wire c_out1, c_out2;
        wire [Narrow-1:0] s_out1;
        wire [Narrow-1:0] s_out2;

        adder b1(
            .a(A),
                .b(B),
                .c(1'b0),
                .outc(c_out1),
                .outs(s_out1)
                );

        adder b2(
            .a(A),
                .b(4'b0001),
                .c(1'b0),
                .outc(c_out2),
                .outs(s_out2)
                );

        always @(*)
        begin
            case(in1)
                    3'b000: out1 = A * B;
                    3'b001: out1 = {A | B, A ^ B};
                    3'b010: out1 = {c_out1, s_out1};
                    3'b011: out1 = A + B;
                    3'b100: out1 = {c_out2, s_out2};
                    3'b101: out1 = |A | |B;
                    3'b110: out1 = B << A;
                    3'b111: out1 = B >> A;
                    default: out1 = 8'b0000_0000;
           endcase
        end
endmodule

module adder(a, b, c, outc, outs);
    parameter Narrow = 4;
    input [Narrow-1:0] a;
        input [Narrow-1:0] b;
        input c;
    output outc;
        output [Narrow-1:0] outs;
        wire connect1, connect2, connect3;

    full_adder e(
          .A(a[0]),
                .B(b[0]),
                .cin(c),
                .S(outs[0]),
```

```
                .cout(connect1)
            );

        full_adder f(
            .A(a[1]),
                .B(b[1]),
                .cin(connect1),
                .S(outs[1]),
                .cout(connect2)
            );

         full_adder g(
            .A(a[2]),
                .B(b[2]),
                .cin(connect2),
                .S(outs[2]),
                .cout(connect3)
            );

           full_adder h(
            .A(a[3]),
                .B(b[3]),
                .cin(connect3),
                .S(outs[3]),
                .cout(outc)
            );
endmodule

module full_adder(A, B, cin, S, cout);
        input A;
        input B;
            input cin;
        output S;
            output cout;

        assign cout = A&B | A&cin | B&cin;
            assign S = A ^ B ^ cin;

endmodule

module hex(in2, in3, in4, in5, o2, o3, o4, o5, o6, o7, o8);
        input in2;
            input in3;
            input in4;
            input in5;
        output o2;
            output o3;
            output o4;
            output o5;
            output o6;
            output o7;
            output o8;

        hex0 u1(
            .c3(in2),
            .c2(in3),
                .c1(in4),
            .c0(in5),
            .out1(o2)
            );

        hex1 u2(
            .c3(in2),
            .c2(in3),
                .c1(in4),
            .c0(in5),
            .out2(o3)
            );

           hex2 u3(
            .c3(in2),
            .c2(in3),
                .c1(in4),
            .c0(in5),
            .out3(o4)
            );

        hex3 u4(
            .c3(in2),
            .c2(in3),
                .c1(in4),
            .c0(in5),
            .out4(o5)
            );

        hex4 u5(
            .c3(in2),
                .c2(in3),
                    .c1(in4),
                .c0(in5),
                .out5(o6)
                );

        hex5 u6(
            .c3(in2),
            .c2(in3),
                    .c1(in4),
                .c0(in5),
                .out6(o7)
                );

        hex6 u7(
            .c3(in2),
            .c2(in3),
                    .c1(in4),
                .c0(in5),
                .out7(o8)
                );
endmodule

module hex0(c3, c2, c1, c0, out1);
        input c3; //selected when s is 0
        input c2; //selected when s is 1
        input c1; //select signal
            input c0;
        output out1; //output

        assign out1 = ~c3 & c2 & ~c1 & ~c0 | c3 & c2 & ~c1 & c0 |
c3 & ~c2 & c1 & c0 | ~c3 & ~c2 & ~c1 & c0;

endmodule

module hex1(c3, c2, c1, c0, out2);
        input c3; //selected when s is 0
        input c2; //selected when s is 1
        input c1; //select signal
            input c0;
        output out2; //output

        assign out2 = c2 & c1 & ~c0 | c3 & c1 & c0 | ~c3 & c2 & ~c1
& c0 | c3 & c2 & ~c0;

endmodule

module hex2(c3, c2, c1, c0, out3);
        input c3; //selected when s is 0
        input c2; //selected when s is 1
        input c1; //select signal
            input c0;
        output out3; //output

        assign out3 = ~c3 & ~c2 & c1 & ~c0 | c3 & c2 & c1 | c3 & c2
& ~c0;

endmodule

module hex3(c3, c2, c1, c0, out4);
        input c3; //selected when s is 0
        input c2; //selected when s is 1
        input c1; //select signal
            input c0;
        output out4; //output

        assign out4 = ~c3 & c2 & ~c1 & ~c0 | ~c2 & ~c1 & c0 | c2 &
c1 & c0 | c3 & ~c2 & c1 & ~c0;

endmodule

module hex4(c3, c2, c1, c0, out5);
        input c3; //selected when s is 0
        input c2; //selected when s is 1
        input c1; //select signal
            input c0;
        output out5; //output

        assign out5 = ~c3 & c2 & ~c1 | ~c2 & ~c1 & c0 | ~c3 & c0;

endmodule

module hex5(c3, c2, c1, c0, out6);
        input c3; //selected when s is 0
        input c2; //selected when s is 1
        input c1; //select signal
            input c0;
```

```
    output out6; //output

    assign out6 = ~c3 & ~c2 & c0 | ~c3 & ~c2 & c1 | ~c3 & c1 &
c0 | c3 & c2 & ~c1 & c0;

endmodule

module hex6(c3, c2, c1, c0, out7);
    input c3; //selected when s is 0
    input c2; //selected when s is 1
    input c1; //select signal
      input c0;
    output out7; //output

    assign out7 = ~c3 & ~c2 & ~c1 | c3 & c2 & ~c1 & ~c0 | ~c3 &
c2 & c1 & c0;

endmodule
```

2. Simulation.



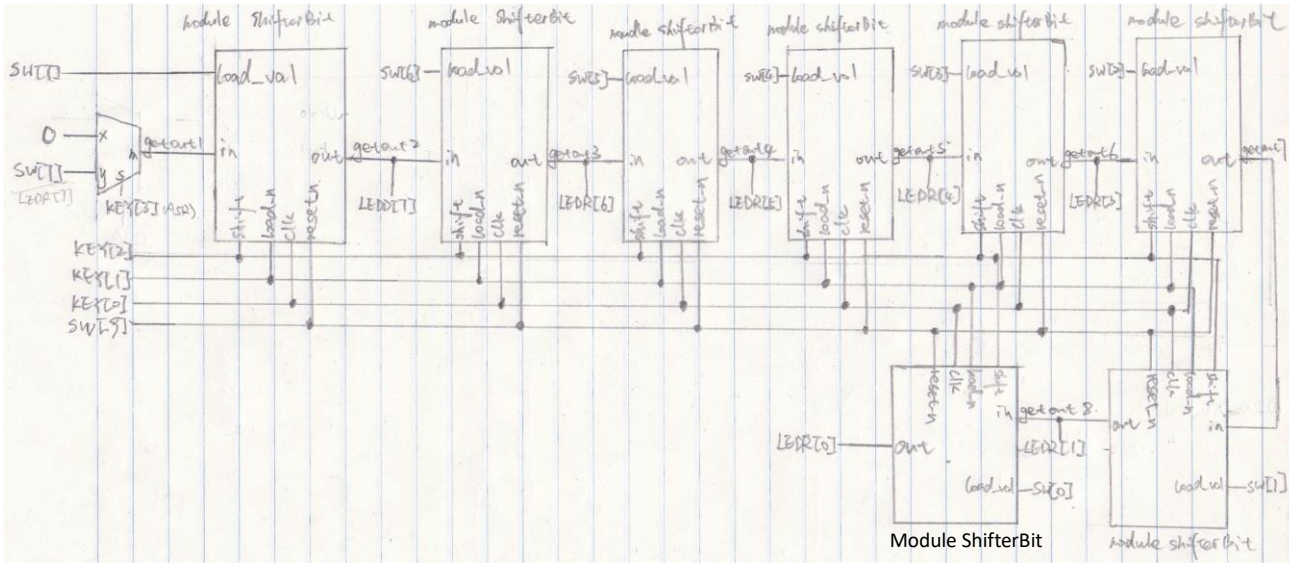
(this is just a part of all the simulation)

Part III

1. What is the behaviour of the 8-bit shift register shown in Figure 6 when *Load_n* = 1 and *ShiftRight* = 0?

   The output of last time will be the new input, so there would be a recursion that the inout and the output will keep same until *Load_n* or *ShiftRight* change.

2. Draw a schematic.

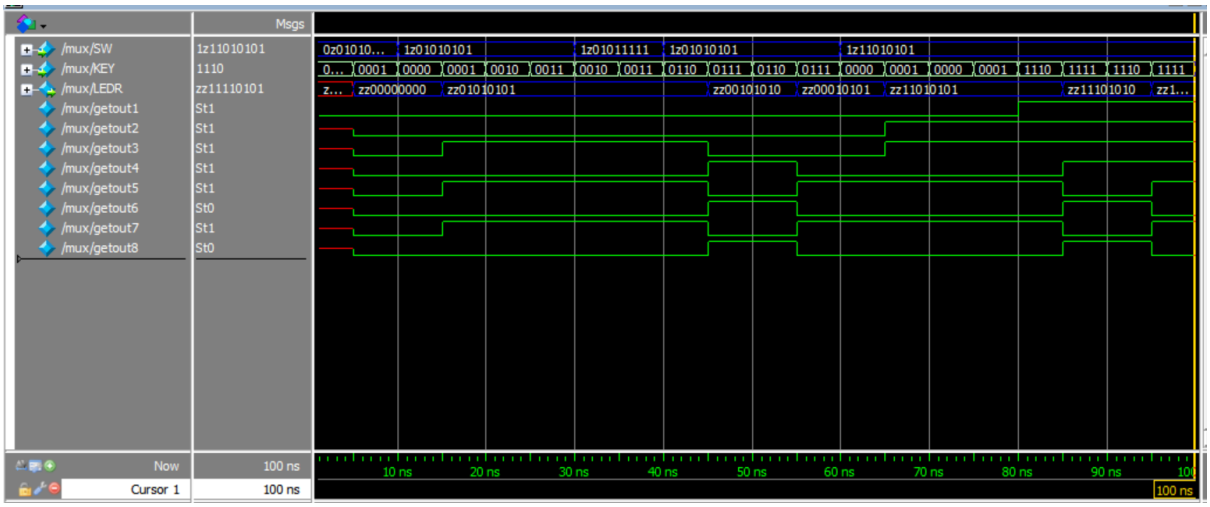


3.4. Write a Verilog.

```
module mux(LEDR, SW, KEY);
	input [9:0] SW;
		input [3:0] KEY;
		output [9:0] LEDR;
		wire getout1, getout2, getout3, getout4, getout5, getout6, getout7,getout8;

		mux2to1 M1(
			.x(1'b0),
				.y(LEDR[7]),
				.s(KEY[3]),
				.m(getout1)
				);

		ShifterBit B1(
			.load_val(SW[7]),
				.in(getout1),
				.shift(KEY[2]),
				.load_n(KEY[1]),
				.clk(KEY[0]),
				.reset_n(SW[9]),
				.out(getout2)
				);

		assign LEDR[7] = getout2;

		ShifterBit B2(
			.load_val(SW[6]),
				.in(getout2),
				.shift(KEY[2]),
				.load_n(KEY[1]),
				.clk(KEY[0]),
				.reset_n(SW[9]),
				.out(getout3)
				);

		assign LEDR[6] = getout3;

		ShifterBit B3(
			.load_val(SW[5]),
				.in(getout3),
				.shift(KEY[2]),
				.load_n(KEY[1]),
				.clk(KEY[0]),
				.reset_n(SW[9]),
				.out(getout4)
				);

		assign LEDR[5] = getout4;

		ShifterBit B4(
			.load_val(SW[4]),
				.in(getout4),
				.shift(KEY[2]),
				.load_n(KEY[1]),
				.clk(KEY[0]),
				.reset_n(SW[9]),
				.out(getout5)
				);

		assign LEDR[4] = getout5;

		ShifterBit B5(
			.load_val(SW[3]),
				.in(getout5),
				.shift(KEY[2]),
				.load_n(KEY[1]),
				.clk(KEY[0]),
				.reset_n(SW[9]),
				.out(getout6)
				);

		assign LEDR[3] = getout6;

		ShifterBit B6(
			.load_val(SW[2]),
				.in(getout6),
				.shift(KEY[2]),
				.load_n(KEY[1]),
				.clk(KEY[0]),
				.reset_n(SW[9]),
				.out(getout7)
				);
```

```
        assign LEDR[2] = getout7;

        ShifterBit B7(
            .load_val(SW[1]),
                .in(getout7),
                .shift(KEY[2]),
                .load_n(KEY[1]),
                .clk(KEY[0]),
                .reset_n(SW[9]),
                .out(getout8)
                );

        assign LEDR[1] = getout8;

        ShifterBit B8(
            .load_val(SW[0]),
                .in(getout8),
                .shift(KEY[2]),
                .load_n(KEY[1]),
                .clk(KEY[0]),
                .reset_n(SW[9]),
                .out(LEDR[0])
                );
endmodule

module ShifterBit(load_val, in, shift, load_n, clk, reset_n, out);
    input load_val, in, shift, load_n, clk, reset_n;
        output out;
        wire connect1, connect2;

        mux2to1 M2(
            .x(out),
                .y(in),
                .s(shift),
                .m(connect1)
                );

        mux2to1 M3(
            .x(load_val),
                .y(connect1),
                .s(load_n),
                .m(connect2)
                );

        D_flipflop D1(
            .d(connect2),
                .clock(clk),
                .reset_n(reset_n),
                .q(out)
                );
endmodule


module mux2to1(x, y, s, m);
    input x; //selected when s is 0
    input y; //selected when s is 1
    input s; //select signal
    output m; //output

    assign m = s & y | ~s & x;
endmodule

module D_flipflop(d, clock, reset_n, q);
        parameter N = 1;
    input [N-1:0] d;
        input [N-1:0] clock;
        input [N-1:0] reset_n;
        output [N-1:0] q;
        reg [N-1:0] q;

        always @(posedge clock)
        begin
            if (reset_n == 1'b0)
                q <= 0;
            else
                q <= d;
        end
endmodule
```

5. Simulation.


Msgs
/mux/SW
/mux/KEY
/mux/LEDR
/mux/getout1
/mux/getout2
/mux/getout3
/mux/getout4
/mux/getout5
/mux/getout6
/mux/getout7
/mux/getout8
1z11010101
1110
zz11110101
St1
St1
St1
St1
St1
St0
St1
St0
0z01010...
1z01010101
1z01011111
1z01010101
1z11010101
0...
0001
0000
0001
0010
0011
0010
0011
0110
0111
0110
0111
0000
0001
0000
0001
1110
1111
1110
1111
z...
zz00000000
zz01010101
zz00101010
zz00010101
zz11010101
zz11101010
zz1...
Now
100 ns
Cursor 1
100 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns