```verilog
`timescale 1ns / 1ns // `timescale time_unit/time_precision


/* top-level module */
module lab4Part3 (input [9:0] SW, input [3:0] KEY, output [9:0] LEDR);

            assign LEDR[9:8] = 2'b00; // LEDR[9] and LEDR[8] are off
            wire [7:0] Q_reg; // carries the register output

            /*
             * SW[7:0] is the data input bus
             * SW[9] is the reset
             * SW[8] is a non-input for any function
             * KEY[0] is the clock signal
             * KEY[1] is the parallel load input mode
             * KEY[2] is the rotate right mode
             * KEY[3] is the logical shift right mode
             * LEDR[7:0] is the register output bus
             */

             universal_ShiftReg U1 (.DATA_IN(SW[7:0]), .LSRight(~KEY[3]), .rotateRight(~KEY[2]),
                                    .ParallelLoad_n(~KEY[1]), .clock(~KEY[0]),
                                    .reset(SW[9]), .Q_out(Q_reg));
             assign LEDR[7:0] = Q_reg;

endmodule // lab4Part3


/* universal shift register */
module universal_ShiftReg (input [7:0] DATA_IN, input LSRight, rotateRight,
                           ParallelLoad_n, clock, reset, output [7:0] Q_out);

            wire [7:0] Q; // carries the output of the flip-flops and subsequent connections
            assign Q_out = Q; // assigning outputs of flip-flops to the register output

            // instantiation of all 8 flip-flops for the register
            flipFlop F7 (.clock(clock), .reset(reset), .load_n(ParallelLoad_n),
                         .loadLeft(rotateRight), .left(Q[0]), .right(Q[6]), .data(DATA_IN[7]),
                         .LSRight(LSRight), .Q(Q[7])); // flip-flop for bit 7 (MSB)

            flipFlop F6 (.clock(clock), .reset(reset), .load_n(ParallelLoad_n),
                         .loadLeft(rotateRight), .left(Q[7]), .right(Q[5]), .data(DATA_IN[6]),
                         .LSRight(1'b0), .Q(Q[6])); // flip-flop for bit 6

            flipFlop F5 (.clock(clock), .reset(reset), .load_n(ParallelLoad_n),
                         .loadLeft(rotateRight), .left(Q[6]), .right(Q[4]), .data(DATA_IN[5]),
                         .LSRight(1'b0), .Q(Q[5])); // flip-flop for bit 5

            flipFlop F4 (.clock(clock), .reset(reset), .load_n(ParallelLoad_n),
                         .loadLeft(rotateRight), .left(Q[5]), .right(Q[3]), .data(DATA_IN[4]),
                         .LSRight(1'b0), .Q(Q[4])); // flip-flop for bit 4

            flipFlop F3 (.clock(clock), .reset(reset), .load_n(ParallelLoad_n),
                         .loadLeft(rotateRight), .left(Q[4]), .right(Q[2]), .data(DATA_IN[3]),
                         .LSRight(1'b0), .Q(Q[3])); // flip-flop for bit 3

            flipFlop F2 (.clock(clock), .reset(reset), .load_n(ParallelLoad_n),
                         .loadLeft(rotateRight), .left(Q[3]), .right(Q[1]), .data(DATA_IN[2]),
                         .LSRight(1'b0), .Q(Q[2])); // flip-flop for bit 2

            flipFlop F1 (.clock(clock), .reset(reset), .load_n(ParallelLoad_n),
                         .loadLeft(rotateRight), .left(Q[2]), .right(Q[0]), .data(DATA_IN[1]),
                         .LSRight(1'b0), .Q(Q[1])); // flip-flop for bit 1

            flipFlop F0 (.clock(clock), .reset(reset), .load_n(ParallelLoad_n),
                         .loadLeft(rotateRight), .left(Q[1]), .right(Q[7]), .data(DATA_IN[0]),
                         .LSRight(1'b0), .Q(Q[0])); // flip-flop for bit 0 (LSB)
```

```verilog

endmodule // universal_ShiftReg


/* flip-flop with multiplexers to select input */
module flipFlop (input clock, reset, load_n, data, loadLeft, right, left,
               LSRight, output reg Q);

            wire R, D;

            /*
             * loadleft = 1 will select left, and right otherwise
             * load_n = 0 will select data, and output of rotation select otherwise
            */

            mux2to1 M1 (.x(right), .y(left), .sel(loadLeft), .f(R)); // left-right rotate option select
            mux2to1 M2 (.x(data), .y(R), .sel(load_n), .f(D)); // parallel load and rotate select

            always @(posedge clock) // triggered on rising edge of the clock signal
            begin

               if (reset) // active-high, clock synchronous reset to 0
                  Q <= 1'b0;
               else if (LSRight == 1'b1 && loadLeft == 1'b1) // override D during logical shift right
                  Q <= 1'b0;
               else
                  Q <= D; // if reset and LSRight are 0, flip-flop tracks D
            end

endmodule // flipFlop


/* 2 to 1 multiplexer */
module mux2to1(input x, y, sel, output f);

            assign f = sel ? y : x; // f = y when sel = 1, x otherwise

endmodule // mux2to1
```