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

// Left/right 8-bit rotating register
module rotatingRegister8bit(input [9:0] SW, input [3:0] KEY, output [9:0] LEDR);
    wire[7:0] DATA_IN;
    wire reset,clock,parallelLoadn,rotateRight,LSRight;

    wire[7:0] Q;
    wire m0;

    // Input assignment
    assign {LSRight,rotateRight,parallelLoadn,clock} = KEY;
    assign reset = SW[9];
    assign DATA_IN = SW[7:0];

    // Multiplexer to decide whether it is a right rotating or a right shift
    mux2to1 mux0(.x(Q[0]),.y(0),.s(LSRight),.m(m0));

    // Putting the 8 rotation slices together
    regSlice r0(.clock(clock),.reset(reset),.loadLeft(rotateRight),.loadn(parallelLoadn),
                .D(DATA_IN[0]),.Q(Q[0]),.left(Q[1]),.right(Q[7]));
    regSlice r1(.clock(clock),.reset(reset),.loadLeft(rotateRight),.loadn(parallelLoadn),
                .D(DATA_IN[1]),.Q(Q[1]),.left(Q[2]),.right(Q[0]));
    regSlice r2(.clock(clock),.reset(reset),.loadLeft(rotateRight),.loadn(parallelLoadn),
                .D(DATA_IN[2]),.Q(Q[2]),.left(Q[3]),.right(Q[1]));
    regSlice r3(.clock(clock),.reset(reset),.loadLeft(rotateRight),.loadn(parallelLoadn),
                .D(DATA_IN[3]),.Q(Q[3]),.left(Q[4]),.right(Q[2]));
    regSlice r4(.clock(clock),.reset(reset),.loadLeft(rotateRight),.loadn(parallelLoadn),
                .D(DATA_IN[4]),.Q(Q[4]),.left(Q[5]),.right(Q[3]));
    regSlice r5(.clock(clock),.reset(reset),.loadLeft(rotateRight),.loadn(parallelLoadn),
                .D(DATA_IN[5]),.Q(Q[5]),.left(Q[6]),.right(Q[4]));
    regSlice r6(.clock(clock),.reset(reset),.loadLeft(rotateRight),.loadn(parallelLoadn),
                .D(DATA_IN[6]),.Q(Q[6]),.left(Q[7]),.right(Q[5]));
    // Make sure to load m0 into left to allow LSRight to work
    regSlice r7(.clock(clock),.reset(reset),.loadLeft(rotateRight),.loadn(parallelLoadn),
                .D(DATA_IN[7]),.Q(Q[7]),.left(m0),.right(Q[6]));

    assign LEDR[7:0] = Q; // Output
endmodule

// A single slice of the rotating register
// NOTE: rotating right = loading in the left bit
module regSlice(input left,right,D,loadLeft,loadn,clock,reset, output Q);
    wire m1,m2;

    // Multiplexer to choose whether to load left or right
    mux2to1 mux1(.x(right),.y(left),.s(loadLeft),.m(m1));
    // Multiplexer for whether to actually load digit
    mux2to1 mux2(.x(D),.y(m1),.s(loadn),.m(m2));
    // Flip-flop
    flipflop register(.clock(clock),.reset(reset),.D(m2),.Q(Q));
endmodule

// 2 to 1 multiplexer, processes 1 bit
module mux2to1(input x,y,s, output m);
    reg muxOut;

    // Multiplexer code
    always @(*)
    begin
        if(s==0)
            muxOut = x;
        else
            muxOut = y;
    end

    // Assign to output
    assign m = muxOut;
endmodule
```

```
70
71  // Active-high, synchronus reset positive edge triggered flip flop
72  module flipflop(input clock,reset, input D, output Q);
73      reg q;
74
75      // Triggered every time clock rises -> on button press
76      always @(posedge clock)
77      begin
78          if(reset==1) // Set to 0
79              q <= 0;
80          else // Pass output of ALU to q
81              q <= D;
82      end
83
84      assign Q = q;
85  endmodule
86
```