Part I

1)

module alu(SW, KEY, LEDR, HEX0, HEX4, HEX5);

input [9:0] SW;

input [3:0] KEY;

output [7:0] LEDR;

output [6:0] HEX0;

output [6:0] HEX4;

output [6:0] HEX5;

reg [7:0] q;

reg [7:0] out;

wire [3:0] A;

wire [3:0] B;

assign A = SW[3:0];

assign B = q[3:0];

wire [3:0] result0;

wire [7:0] final\_result0;

wire cout0;

ripple\_carry\_adder\_new rca0(.input1(A[3:0]),

input2(4'b0001),

.sum(result0[3:0]),

.cout(cout0)

);

add\_zeros az0(.sum(result0[3:0]),

.cout(cout0),

.result(final\_result0[7:0])

);

wire [3:0] result1;

wire [7:0] final\_result1;

wire cout1;

ripple\_carry\_adder\_new rca1(.input1(A[3:0]),

input2(B[3:0]),

.sum(result1[3:0]),

.cout(cout1)

);

add\_zeros az1(.sum(result1[3:0]),

.cout(cout1),

.result(final\_result1[7:0])

);

wire [7:0] final\_result2;

assign final\_result2 = {4'b0000, A+B};

wire [7:0] final\_result3;

assign final\_result3 = {A|B, A^B};

wire [7:0] final\_result4;

assign final\_result4 = {7'b0000000, (|A) | (|B)};

wire [7:0] final\_result5;

assign final\_result5 = {4'b0000, B << A};

wire [7:0] final\_result6;

assign final\_result6 = {4'b0000, B >> A};

wire [7:0] final\_result7;

assign final\_result7 = {4'b0000, A\*B};

always @(\*)

begin

case (SW[7:5])

3'b000: out = final\_result0;

3'b001: out = final\_result1;

3'b010: out = final\_result2;

3'b011: out = final\_result3;

3'b100: out = final\_result4;

3'b101: out = final\_result5;

3'b110: out = final\_result6;

3'b111: out = final\_result7;

default: out = 8'b00000000;

endcase

end

assign LEDR = out;

always @(posedge KEY[0])

begin

if (SW[9] == 1'b0)

q <= 0;

else

q <= out;

//q <= final\_result0;

end

muxseg mux0(

.u(A[0]),

.w(A[1]),

.v(A[2]),

.x(A[3]),

.m0(HEX0[0]),

.m1(HEX0[1]),

.m2(HEX0[2]),

.m3(HEX0[3]),

.m4(HEX0[4]),

.m5(HEX0[5]),

.m6(HEX0[6])

);

muxseg mux4(

.u(q[0]),

.w(q[1]),

.v(q[2]),

.x(q[3]),

.m0(HEX4[0]),

.m1(HEX4[1]),

.m2(HEX4[2]),

.m3(HEX4[3]),

.m4(HEX4[4]),

.m5(HEX4[5]),

.m6(HEX4[6])

);

muxseg mux5(

.u(q[4]),

.w(q[5]),

.v(q[6]),

.x(q[7]),

.m0(HEX5[0]),

.m1(HEX5[1]),

.m2(HEX5[2]),

.m3(HEX5[3]),

.m4(HEX5[4]),

.m5(HEX5[5]),

.m6(HEX5[6])

);

endmodule

module add\_zeros (sum, cout, result);

input [3:0] sum;

input cout;

output [7:0] result;

wire [2:0] CONSTANT;

assign CONSTANT = 3'b000;

assign result[7:5] = CONSTANT;

assign result[4] = cout;

assign result[3:0] = sum;

endmodule

module ripple\_carry\_adder\_new(input1, input2, sum, cout);

input [3:0] input1;

input [3:0] input2;

output [3:0] sum;

output cout;

wire con0, con1, con2;

full\_adder\_new f\_d1(.sum(sum[0]),

.cout(con0),

.a(input1[0]),

.b(input2[0]),

.cin(1'b0)

);

full\_adder\_new f\_d2(.sum(sum[1]),

.cout(con1),

.a(input1[1]),

.b(input2[1]),

.cin(con0)

);

full\_adder\_new f\_d3(.sum(sum[2]),

.cout(con2),

.a(input1[2]),

.b(input2[2]),

.cin(con1)

);

full\_adder\_new f\_d4(.sum(sum[3]),

.cout(cout),

.a(input1[3]),

.b(input2[3]),

.cin(con2)

);

endmodule

module full\_adder\_new(sum, cout, a, b, cin);

output sum, cout;

input [3:0] a;

input [3:0] b;

input cin;

assign sum = a^b^cin;

assign cout = (a&b) | (cin&(a^b));

endmodule

module muxseg0(c0, c1, c2, c3, m);

input c0; //c0

input c1;

input c2;

input c3;

output m;

assign m = (~c3 & ~c1 & (c2&~c0|~c2 & c0)) | (c3 & c0 &(c2 & ~c1 |~c2 &c1 ));

endmodule

module muxseg1(c0, c1, c2, c3, m);

input c0;

input c1;

input c2;

input c3;

output m;

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

endmodule

module muxseg2(c0, c1, c2, c3, m);

input c0;

input c1;

input c2;

input c3;

output m;

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

endmodule

module muxseg3(c0, c1, c2, c3, m);

input c0;

input c1;

input c2;

input c3;

output m;

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

endmodule

module muxseg4(c0, c1, c2, c3, m);

input c0;

input c1;

input c2;

input c3;

output m;

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

endmodule

module muxseg5(c0, c1, c2, c3, m);

input c0;

input c1;

input c2;

input c3;

output m;

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

endmodule

module muxseg6(c0, c1, c2, c3, m);

input c0;

input c1;

input c2;

input c3;

output m;

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

endmodule

module muxseg(u, w, v, x, m0, m1, m2 , m3, m4, m5, m6 );

input u; input v;

input w;

input x;

output m0;

output m1;

output m2;

output m3;

output m4;

output m5;

output m6;

muxseg0 mux0(

.c0(u),

.c1(w),

.c2(v),

.c3(x),

.m(m0)

);

muxseg1 mux1(

.c0(u),

.c1(w),

.c2(v),

.c3(x),

.m(m1)

);

muxseg2 mux2(

.c0(u),

.c1(w),

.c2(v),

.c3(x),

.m(m2)

);

muxseg3 mux3(

.c0(u),

.c1(w),

.c2(v),

.c3(x),

.m(m3)

);

muxseg4 mux4(

.c0(u),

.c1(w),

.c2(v),

.c3(x),

.m(m4)

);

muxseg5 mux5(

.c0(u),

.c1(w),

.c2(v),

.c3(x),

.m(m5)

);

muxseg6 mux6(

.c0(u),

.c1(w),

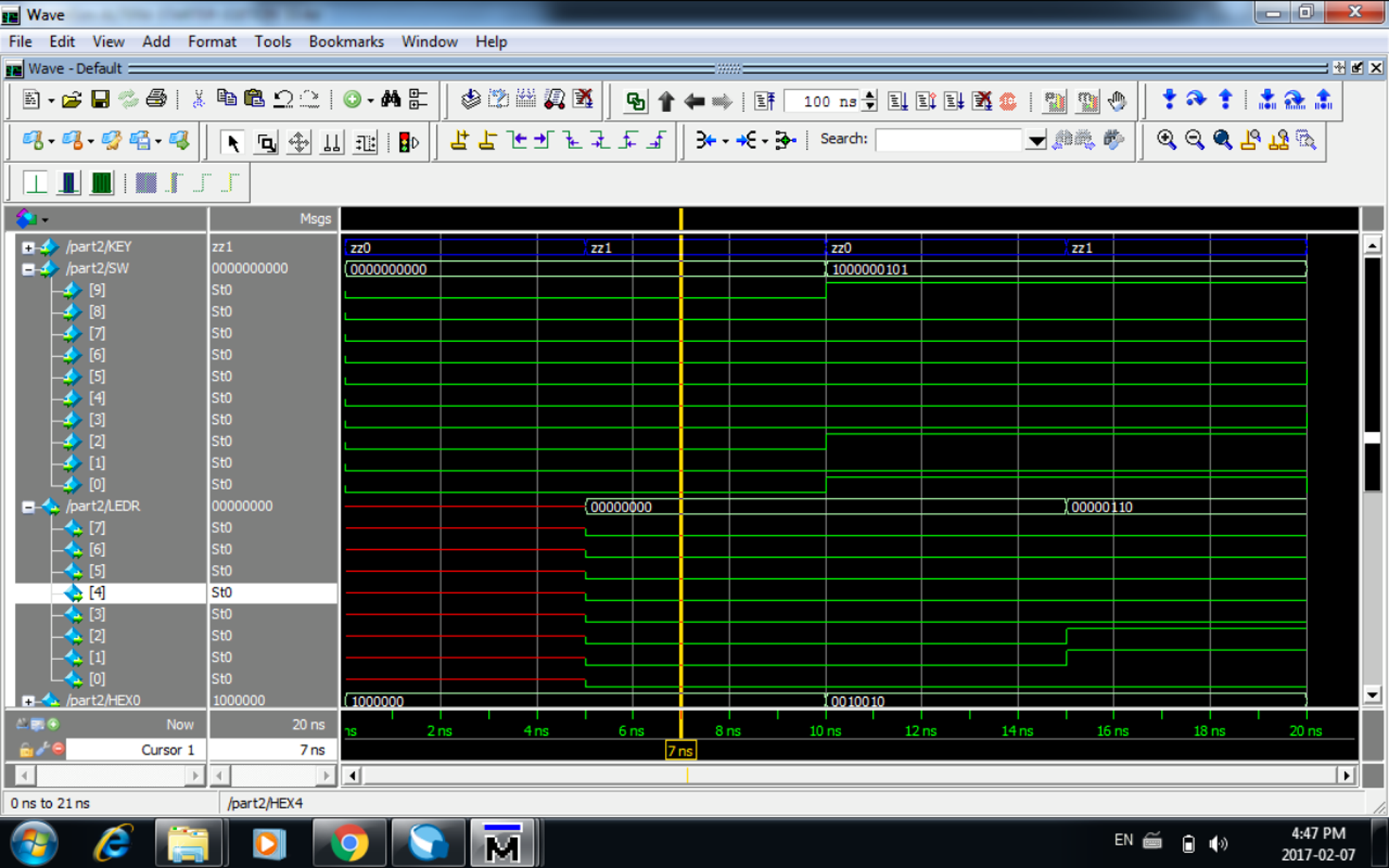
.c2(v),

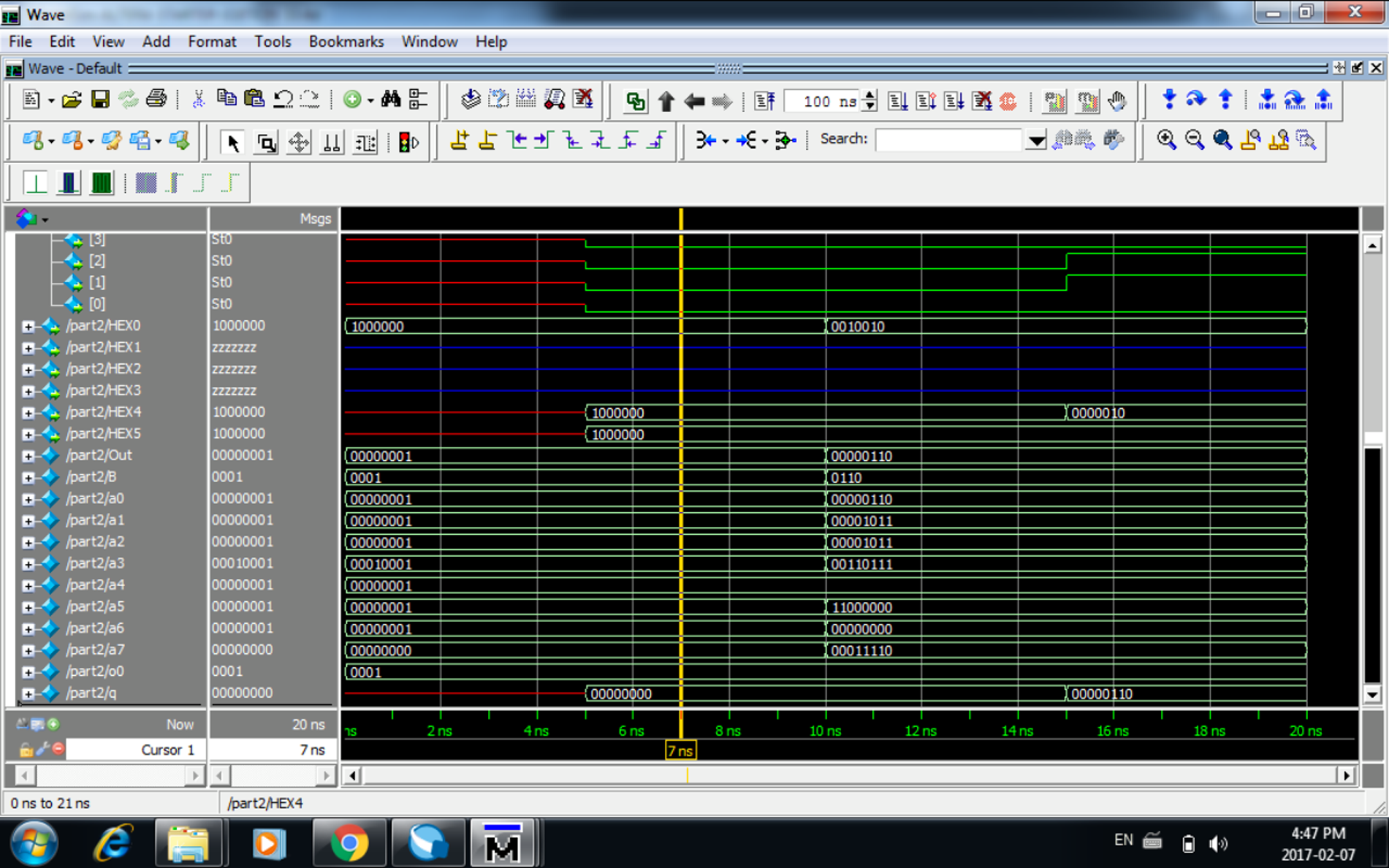
.c3(x),

.m(m6)

);

endmodule





4)

module shifter (SW, KEY, LEDR);

input [9:0] SW;

input [3:0] KEY;

output [7:0] LEDR;

wire [7:0] Q;

reg sb7\_in;

shifter\_bit sb0(

.in(Q[1]),

.load\_val(SW[0]),

.shift(KEY[2]),

.load\_n(KEY[1]),

.clk(KEY[0]),

.reset\_n(SW[9]),

.out(Q[0])

);

shifter\_bit sb1(

.in(Q[2]),

.load\_val(SW[1]),

.shift(KEY[2]),

.load\_n(KEY[1]),

.clk(KEY[0]),

.reset\_n(SW[9]),

.out(Q[1])

);

shifter\_bit sb2(

.in(Q[3]),

.load\_val(SW[2]),

.shift(KEY[2]),

.load\_n(KEY[1]),

.clk(KEY[0]),

.reset\_n(SW[9]),

.out(Q[2])

);

shifter\_bit sb3(

.in(Q[4]),

.load\_val(SW[3]),

.shift(KEY[2]),

.load\_n(KEY[1]),

.clk(KEY[0]),

.reset\_n(SW[9]),

.out(Q[3])

);

shifter\_bit sb4(

.in(Q[5]),

.load\_val(SW[4]),

.shift(KEY[2]),

.load\_n(KEY[1]),

.clk(KEY[0]),

.reset\_n(SW[9]),

.out(Q[4])

);

shifter\_bit sb5(

.in(Q[6]),

.load\_val(SW[5]),

.shift(KEY[2]),

.load\_n(KEY[1]),

.clk(KEY[0]),

.reset\_n(SW[9]),

.out(Q[5])

);

shifter\_bit sb6(

.in(Q[7]),

.load\_val(SW[6]),

.shift(KEY[2]),

.load\_n(KEY[1]),

.clk(KEY[0]),

.reset\_n(SW[9]),

.out(Q[6])

);

shifter\_bit sb7(

.in(sb7\_in),

.load\_val(SW[7]),

.shift(KEY[2]),

.load\_n(KEY[1]),

.clk(KEY[0]),

.reset\_n(SW[9]),

.out(Q[7])

);

always @(\*)

begin

if (KEY[3]== 1'b0)

sb7\_in <= 0;

else

sb7\_in <= SW[7];

end

assign LEDR = Q;

endmodule

3)

module shifter\_bit(in, load\_val, shift, load\_n, clk, reset\_n, out);

input in, load\_val, shift, load\_n, clk, reset\_n;

output out;

wire m1\_out;

wire m2\_out;

mux2to1 m1(

.x(out),

.y(in),

.s(shift),

.m(m1\_out)

);

mux2to1 m2(

.x(load\_val),

.y(m1\_out),

.s(load\_n),

.m(m2\_out)

);

reg q;

always @(posedge clk)

begin

if (reset\_n == 1'b0)

q <= 0;

else

q <= m2\_out;

end

assign out = q;

endmodule

module mux2to1 (x, y, s, m);

input x;

input y;

input s;

output m;

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

endmodule

5)

