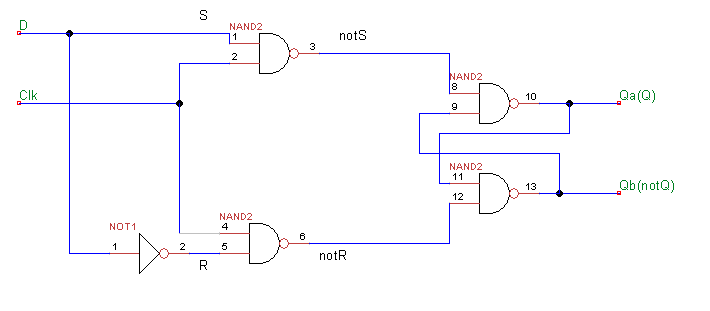
**Lab 4 Pre-Lab Report**

Ziyao Chen

1002453409

**Part 1**





1. Clk shouldn’t be set to 0 as the first test case, otherwise we will get 1 as output for both notS and notR, which will case our Q and notQ to be indeterminate status. So, 00 and 01 can’t be first test cases for Clk and D respectively.

**Part 2**



module part2(SW, KEY, HEX0, HEX4, HEX5, LEDR);

input [9:0] SW;

input [3:0] KEY;

output [6:0] HEX0;

output [6:0] HEX4;

output [6:0] HEX5;

output [9:0] LEDR;

wire [7:0] ALUout;

wire [7:0] register\_out;

register reg1(

.d(ALUout[7:0]),

.clk(KEY[0]),

.reset\_n(SW[9]),

.q(register\_out[7:0])

);

alu a1(

.A(SW[3:0]),

.B(register\_out[3:0]),

.func(SW[7:5]),

.ALUout(ALUout[7:0])

);

SevenSegmentDecoder h0(

.c3(SW[3]),

.c2(SW[2]),

.c1(SW[1]),

.c0(SW[0]),

.ss\_out(HEX0[6:0])

);

SevenSegmentDecoder h4(

.c3(register\_out[3]),

.c2(register\_out[2]),

.c1(register\_out[1]),

.c0(register\_out[0]),

.ss\_out(HEX4[6:0])

);

SevenSegmentDecoder h5(

.c3(register\_out[7]),

.c2(register\_out[6]),

.c1(register\_out[5]),

.c0(register\_out[4]),

.ss\_out(HEX5[6:0])

);

assign LEDR[7:0] = register\_out[7:0];

endmodule

module register(d, clk, reset\_n, q);

input [7:0] d;

input clk, reset\_n;

output [7:0] q;

reg [7:0] q; /\*synthesis keep \*/

always @(posedge clk)

begin

if (reset\_n == 1'b0)

q <= 8'b00000000;

else

q <= d;

end

endmodule

module alu(A,B,func,ALUout);

input [3:0] A;

input [3:0] B;

input [2:0] func;

output [7:0] ALUout;

reg [7:0] ALUout;

wire [3:0] f0s; // output storage for f0 sum

wire f0cout; // output storage for f0 carry out

wire [3:0] f1s; //output storage for f1 sum

wire f1cout; // output storage for f1 carry out

ripplecarryadder f0(A[3:0], 4'b0001, 1'b0, f0s, f0cout); //setup for function0

ripplecarryadder f1(A[3:0], B[3:0], 1'b0, f1s, f1cout); //setup for function1

always @(\*)

begin

case (func[2:0])

3'b000: ALUout[7:0] = {3'b000,f0cout,f0s};

3'b001: ALUout[7:0] = {3'b000, f1cout, f1s};

3'b010: ALUout[7:0] = A+B;

3'b011: ALUout[7:0] = {A|B, A^B};

3'b100: ALUout[7:0] = {7'b0000\_000, A[3]|A[2]|A[1]|A[0]|B[3]|B[2]|B[1]|B[0]};

3'b101: ALUout[7:0] = {A,B};

3'b101: ALUout[7:0] = B << A;

3'b110: ALUout[7:0] = B >> A;

3'b111: ALUout[7:0] = A\*B; endcase

end

endmodule

module ripplecarryadder(A, B, cin, S, cout);

input [3:0] A;

input [3:0] B;

input cin;

output [3:0] S;

output cout;

wire Connection1, Connection2, Connection3;

fulladder f1(A[0], B[0], cin, Connection1, S[0]);

fulladder f2(A[1], B[1], Connection1, Connection2, S[1]);

fulladder f3(A[2], B[2], Connection2, Connection3, S[2]);

fulladder f4(A[3], B[3], Connection3, cout, S[3]);

endmodule

module fulladder(a, b, carryin, carryout, sum);

input a, b, carryin; // adding a and b with carryin from the previous bit's carryout

output carryout, sum; // carryout for the next bit's carryin, and sum for XOR of a and b

assign carryout = (a && b) | (b && carryin) | (a && carryin);

assign sum = a ^ b ^ carryin;

endmodule

module SevenSegmentDecoder(c3,c2,c1,c0,ss\_out);

input c3,c2,c1,c0; //the bcd input

output [6:0] ss\_out; //the seven segements

assign ss\_out[0] = ~c3 & ~c2 & ~c1 & c0 | ~c3 & c2 & ~c1 & ~c0 | c3 & ~c2 & c1 & c0 | c3 & c2 & ~c1 & c0;

assign ss\_out[1] = c3 & c1 & c0 | c2 & c1 & ~c0 | ~c3 & c2 & ~c1 & c0 | c3 & c2 & ~c0;

assign ss\_out[2] = ~c3 & ~c2 & c1 & ~c0 | c3 & c2 & ~c0 | c3 & c2 & c1;

assign ss\_out[3] = ~c3 & ~c2 & ~c1 & c0 | ~c3 & c2 & ~c1 & ~c0 | c2 & c1 & c0 | c3 & ~c2 & c1 & ~c0;

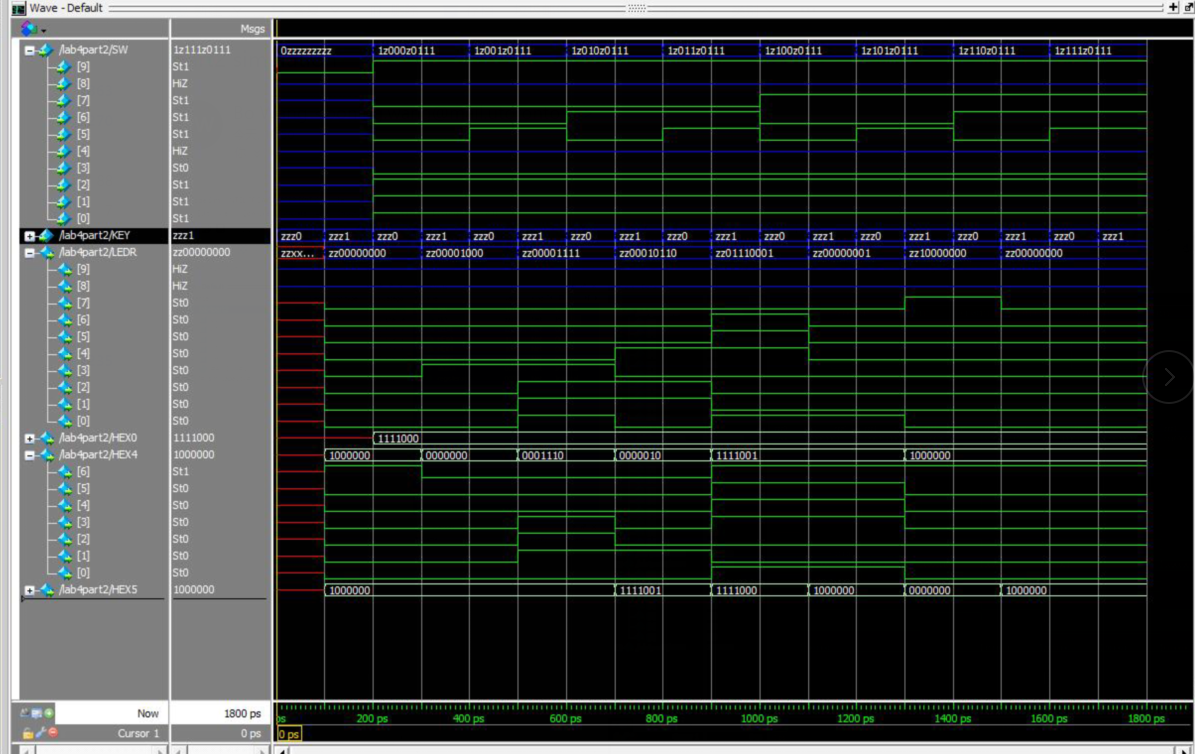
assign ss\_out[4] = ~c3 & c0 | ~c3 & c2 & ~c1 | ~c2 & ~c1 & c0;

assign ss\_out[5] = ~c3 & ~c2 & c0 | ~c3 & ~c2 & c1 | ~c3 & c1 & c0 | c3 & c2 & ~c1 & c0;

assign ss\_out[6] = ~c3 & ~c2 & ~c1 | ~c3 & c2 & c1 & c0 | c3 & c2 & ~c1 & ~c0;

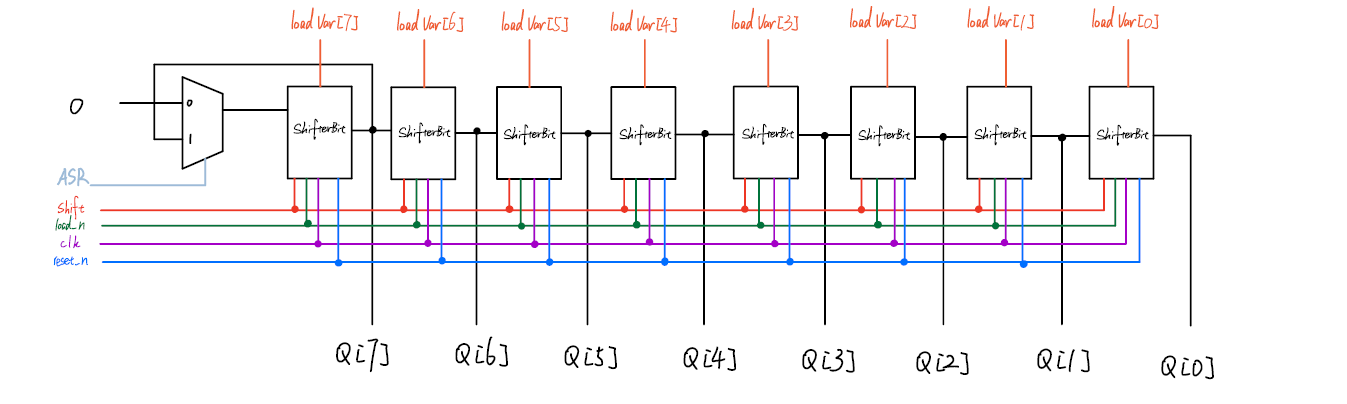
endmodule





**Part 3**

1. Nothing will happen since shift is 0, which implies that the output of its corresponding mux2to1 is 0, then nothing will happen.





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

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

output out;

wire connection1; // store the output of first mux2to1

wire connection2; // store the output of second mux2to1

wire connection3; // store the output of register

mux2to1 m1(

.x(connection3),

.y(in),

.s(shift),

.m(connection1)

);

mux2to1 m2(

.x(load\_val),

.y(connection1),

.s(load\_n),

.m(connection2)

);

register r(

.d(connection2),

.clk(clk),

.reset\_n(reset\_n),

.q(connection3)

);

assign out = connection3;

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;

// OR

// assign m = s ? y : x;

endmodule

module register(d, clk, reset\_n, q);

input d;

input clk, reset\_n;

output q;

reg q;

always @(posedge clk)

begin

if (reset\_n == 1'b0)

q <= 1'b0;

else

q <= d;

end

endmodule



module shifter(LoadVal, Load\_n, ShiftRight, ASR, clk, reset\_n, Q);

input [7:0] LoadVal;

input Load\_n, ShiftRight, ASR, clk, reset\_n;

output [7:0] Q;

wire asr\_out; // store the output of mux2to1 asr, 0 leads Q[7] = 0, 1 leads Q[7] = Q[7];

wire [7:0] shifterbit\_out; // store the output of each shifterbit

mux2to1 asr(

.x(0),

.y(shifterbit\_out[7]),

.s(ASR),

.m(asr\_out)

);

shifterbit s7(

.shift(ShiftRight),

.load\_n(Load\_n),

.clk(clk),

.reset\_n(reset\_n),

.in(asr\_out),

.load\_val(LoadVal[7]),

.out(shifterbit\_out[7])

);

shifterbit s6(

.shift(ShiftRight),

.load\_n(Load\_n),

.clk(clk),

.reset\_n(reset\_n),

.in(shifterbit\_out[7]),

.load\_val(LoadVal[6]),

.out(shifterbit\_out[6])

);

shifterbit s5(

.shift(ShiftRight),

.load\_n(Load\_n),

.clk(clk),

.reset\_n(reset\_n),

.in(shifterbit\_out[6]),

.load\_val(LoadVal[5]),

.out(shifterbit\_out[5])

);

shifterbit s4(

.shift(ShiftRight),

.load\_n(Load\_n),

.clk(clk),

.reset\_n(reset\_n),

.in(shifterbit\_out[5]),

.load\_val(LoadVal[4]),

.out(shifterbit\_out[4])

);

shifterbit s3(

.shift(ShiftRight),

.load\_n(Load\_n),

.clk(clk),

.reset\_n(reset\_n),

.in(shifterbit\_out[4]),

.load\_val(LoadVal[3]),

.out(shifterbit\_out[3])

);

shifterbit s2(

.shift(ShiftRight),

.load\_n(Load\_n),

.clk(clk),

.reset\_n(reset\_n),

.in(shifterbit\_out[3]),

.load\_val(LoadVal[2]),

.out(shifterbit\_out[2])

);

shifterbit s1(

.shift(ShiftRight),

.load\_n(Load\_n),

.clk(clk),

.reset\_n(reset\_n),

.in(shifterbit\_out[2]),

.load\_val(LoadVal[1]),

.out(shifterbit\_out[1])

);

shifterbit s0(

.shift(ShiftRight),

.load\_n(Load\_n),

.clk(clk),

.reset\_n(reset\_n),

.in(shifterbit\_out[1]),

.load\_val(LoadVal[0]),

.out(shifterbit\_out[0])

);

assign Q[7:0] = shifterbit\_out[7:0];

endmodule

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

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

output out;

wire connection1; // store the output of first mux2to1

wire connection2; // store the output of second mux2to1

wire connection3; // store the output of register

mux2to1 m1(

.x(connection3),

.y(in),

.s(shift),

.m(connection1)

);

mux2to1 m2(

.x(load\_val),

.y(connection1),

.s(load\_n),

.m(connection2)

);

register r(

.d(connection2),

.clk(clk),

.reset\_n(reset\_n),

.q(connection3)

);

assign out = connection3;

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;

// OR

// assign m = s ? y : x;

endmodule

module register(d, clk, reset\_n, q);

input d;

input clk, reset\_n;

output q;

reg q;

always @(posedge clk)

begin

if (reset\_n == 1'b0)

q <= 1'b0;

else

q <= d;

end

endmodule

