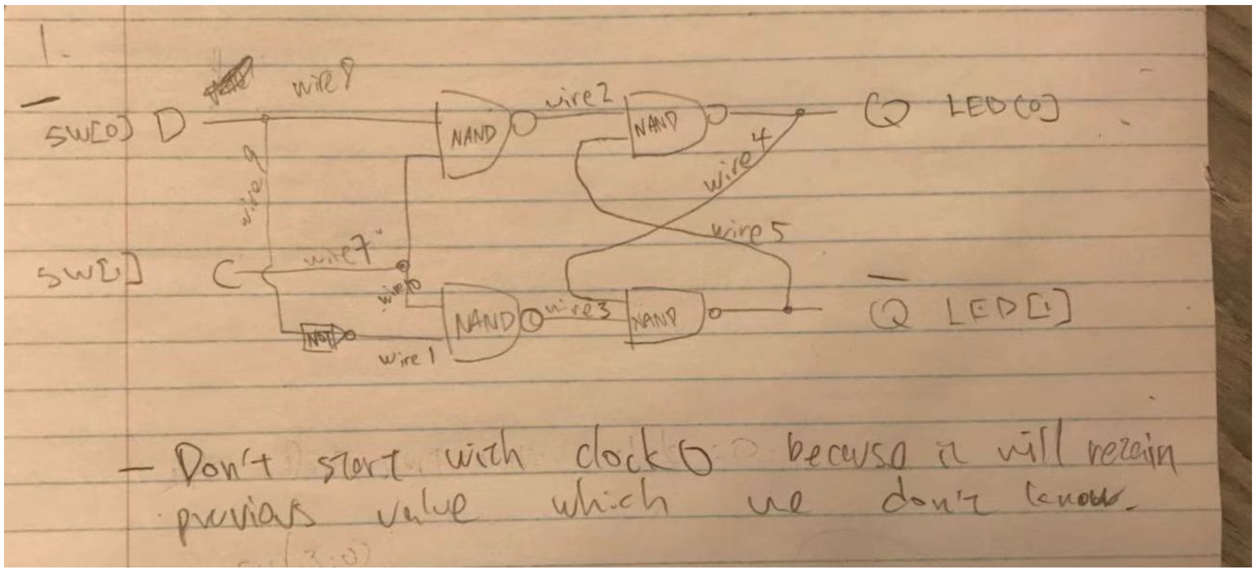
CSC258 prelab 4
Ryan Chang 1004103748

PART1 SCHEMATIC and Question



– Don't start with clock 0 because it will retain previous value which we don't know.

PART2 VERILOG CODE

```
module mainalu(SW, KEY, LEDR, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5);
        input [9:0] SW;
        input [2:0] KEY;
        output [7:0] 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;

        wire[7:0] Case0;
        wire[7:0] Case1;
        wire[7:0] Case2;
        wire[7:0] Case3;
        wire[7:0] Case4;
        wire[7:0] Case5;
    wire[7:0] Case6;
    wire[7:0] Case7;
```

```
 wire[7:0] final;
	reg[7:0] Out;


	fourbitadder fba1(
	.A(SW[3:0]),
	.B(4'b0001),
	.Cin(1'b0),
	.Sum(Case0[7:0])
	);

	fourbitadder fba2(
	.A(SW[3:0]),
	.B(final[3:0]),
	.Cin(1'b0),
	.Sum(Case1[7:0])
	);

	verisum v1(
	.A(SW[3:0]),
	.B(final[3:0]),
	.Result(Case2[7:0])
	);

	veriXOR v2(
	.A(SW[3:0]),
	.B(final[3:0]),
	.Result(Case3[7:0])
	);

	orplusredor o3(
	.A(SW[3:0]),
	.B(final[3:0]),
	.Result(Case4[7:0])

	);

leftShift ls1(
	.A(SW[3:0]),
	.B(final[3:0]),
	.Result(Case5[7:0])
	);
```

```
rightShift rs1(
      .A(SW[3:0]),
      .B(final[3:0]),
      .Result(Case6[7:0])
      );

multiply m1(
         .A(SW[3:0]),
      .B(final[3:0]),
      .Result(Case7[7:0])
      );



      always @(*)
      begin
      case (SW[7:5])
      3'b000: Out = Case0;
      3'b001: Out = Case1;
      3'b010: Out = Case2;
      3'b011: Out = Case3;
      3'b100: Out = Case4;
      3'b101: Out = Case5;
                        3'b110: Out = Case6;
                        3'b111: Out = Case7;
      default: Out = 8'b00000000;
      endcase
      end


dFlipFlop dff1(
       .D(Out[7:0]),
       .clock(KEY[0]),
       .reset_n(SW[9]),
       .Q(final[7:0])
);
      assign LEDR = final;

      sevenseg u6 (
      .Data(SW[3:0]),
      .Display(HEX0[6:0])
      );
```

```
          sevenseg u8 (
          .Data(final[3:0]),
          .Display(HEX4[6:0])
          );

          sevenseg u9 (
          .Data(final[7:4]),
          .Display(HEX5[6:0])

          );
    assign HEX0[6:0] = 7'b1111111;
          assign HEX1[6:0] = 7'b1111111;
          assign HEX3[6:0] = 7'b1111111;

endmodule


module fourbitadder(A, B, Cin, Sum);
          input [3:0] A, B;
          input Cin;
          output [7:0] Sum;
          wire w1, w2, w3;


          fullAdder fa1(
 .a(A[0]),
 .b(B[0]),
 .cin(Cin),
 .s(Sum[0]),
 .cout(w1)
);

          fullAdder fa2(
 .a(A[1]),
 .b(B[1]),
 .cin(w1),
 .s(Sum[1]),
 .cout(w2)
);

          fullAdder fa3(
 .a(A[2]),
 .b(B[2]),
 .cin(w2),
 .s(Sum[2]),
 .cout(w3)
);

        fullAdder fa4(
 .a(A[3]),
 .b(B[3]),
 .cin(w3),
 .s(Sum[3]),
 .cout(Sum[4])
);
        assign Sum[7:5] = 3'b000;

endmodule

module fullAdder(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 veriXOR(A, B, Result);
 input[3:0] A;
        input[3:0] B;
        output[7:0] Result;

        assign Result[3:0] = A^B;
        assign Result[7:4] = A|B;

endmodule

module verisum(A, B, Result);
        input[3:0] A;
        input[3:0] B;
        output[7:0] Result;

        assign Result[3:0] = A + B;
	assign Result[7:4] = 4'b0000;

endmodule

module orplusredor(A, B, Result);
	input[3:0] A;
	input[3:0] B;
	output[7:0] Result;

	assign Result[0] = | {A,B};
	assign Result[7:1] = 7'b0000000;

endmodule

module concat(A, B, Result);
	input[3:0] A;
	input[3:0] B;
	output[7:0] Result;

	assign Result[7:0] = {A, B};

endmodule

module leftShift(A, B, Result);
	input[3:0] A;
	input[3:0] B;
	output[7:0] Result;

	assign Result[7:0] = B << A;

endmodule

module rightShift(A, B, Result);
	input[3:0] A;
	input[3:0] B;
	output[7:0] Result;

	assign Result[7:0] = B >> A;

endmodule

module multiply(A, B, Result);
	input[3:0] A;
```

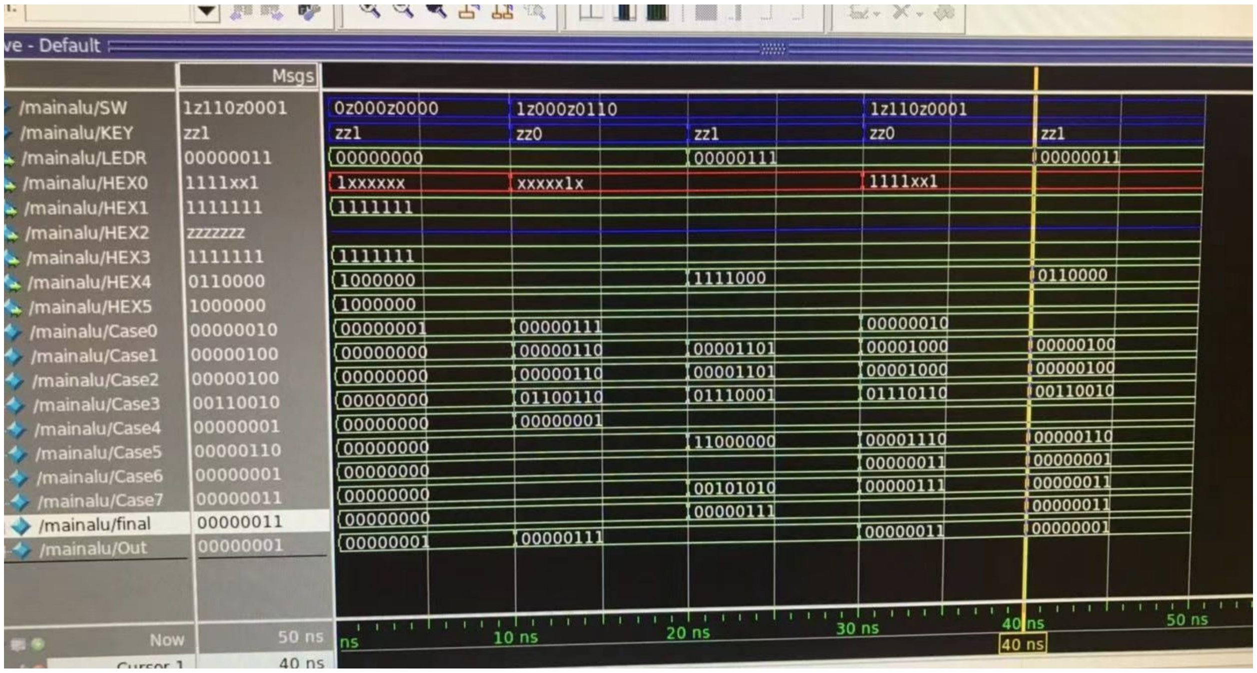
```
            input[3:0] B;
            output[7:0] Result;

            assign Result[7:0] = A*B;

endmodule

module dFlipFlop (D, clock, reset_n, Q);
    input[7:0] D;
    input clock;
    input reset_n;

    output reg[7:0] Q;

    always @(posedge clock)
    begin
            if (reset_n == 1'b0)
                    Q <= 0;
            else
                    Q <= D;
    end

endmodule

module sevenseg (Display, Data);

            input[3:0] Data;

            output[6:0] Display;

            assign Display[0] = ~Data[3] & ~Data[2] & ~Data[1] &  Data[0] |~Data[3] &  Data[2] &
~Data[1] & ~Data[0] | Data[3] &  Data[2] & ~Data[1] &  Data[0] |Data[3] & ~Data[2] &  Data[1] &
Data[0];

            assign Display[1] = ~Data[3] &  Data[2] & ~Data[1] &  Data[0] |Data[3] &  Data[2] &
~Data[1] & ~Data[0] |Data[3] &  Data[1] &  Data[0] |  Data[2] &  Data[1] & ~Data[0];

            assign Display[2] =  Data[3] &  Data[2] & ~Data[1] & ~Data[0] |~Data[3] & ~Data[2] &
Data[1] & ~Data[0] | Data[3] &  Data[2] &  Data[1];

            assign Display[3] = ~Data[3] &  Data[2] & ~Data[1] & ~Data[0] | ~Data[2] & ~Data[1] &
Data[0] |Data[2] &  Data[1] &  Data[0] | Data[3] & ~Data[2] &  Data[1] & ~Data[0];
```

```
        assign Display[4] = ~Data[3] &  Data[0] | ~Data[3] &  Data[2] & ~Data[1]| ~Data[2] &
~Data[1] &  Data[0];

        assign Display[5] = ~Data[3] & ~Data[2]  &  Data[0] |~Data[3] & ~Data[2] &  Data[1]
|~Data[3] & Data[1] &  Data[0] |Data[3] &  Data[2] & ~Data[1] &  Data[0];

        assign Display[6] = ~Data[3] & ~Data[2]  & ~Data[1] |~Data[3] &  Data[2] &  Data[1] &
Data[0] | Data[3] &  Data[2] & ~Data[1] & ~Data[0];

endmodule
```

PART 2 VSIM CODE

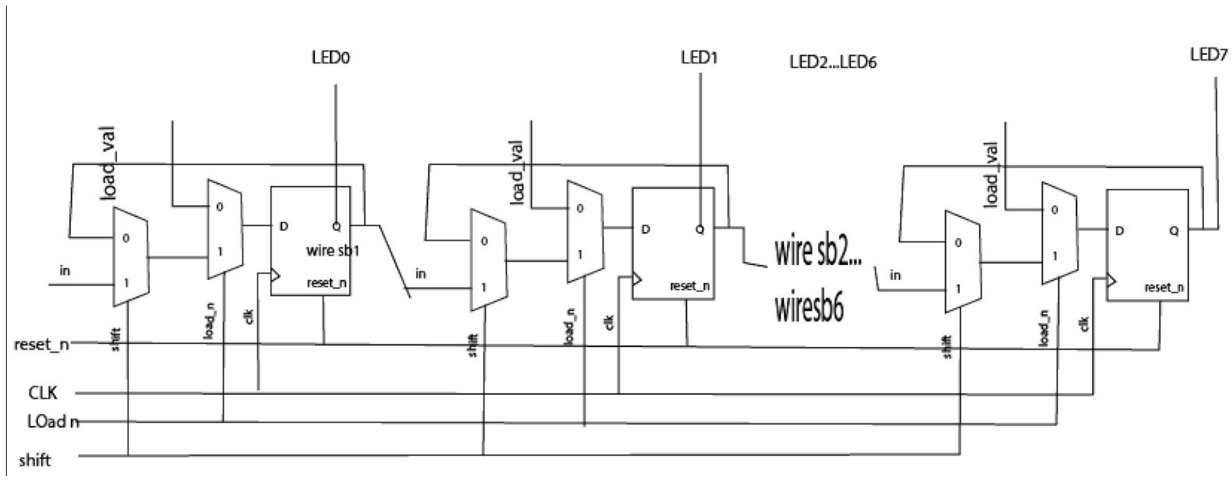
```
# Set the working dir, where all compiled Verilog goes.
vlib work

# Compile all verilog modules in mux.v to working dir;
# could also have multiple verilog files.
vlog -timescale 1ns/1ns regalu.v

# Load simulation using mux as the top level simulation module.
vsim mainalu

# Log all signals and add some signals to waveform window.
log {/*}
# add wave {/*} would add all items in top level simulation module.
add wave {/*}
#SW[9] is reset_n and its active LOW! u silly shit
#this resets final to 0
force {SW[3:0]} 2#0000
force {SW[9]} 0
force {SW[7:5]} 2#000
force {KEY[0]} 1
run 10ns

#testing case 0 where A = 0110
#input values and let clock =0
force {SW[3:0]} 2#0110
force {SW[9]} 1
force {SW[7:5]} 2#000
force {KEY[0]} 0
run 10ns

#clock = 1, so expect LEDR = 00000111
force {KEY[0]} 1
run 10ns

#testing case 6 where A = 0011 or 3 and B = 0111 per above
#input values and let clock =0
force {SW[3:0]} 2#0011
force {SW[9]} 1
force {SW[7:5]} 2#110
force {KEY[0]} 0
run 10ns

#expect 000 because 0111 >> 0011 = 000
force {KEY[0]} 1
run 10ns
```

PART 2 VSIM SCREENSHOT



```
                    Msgs
/mainalu/SW        1z110z0001
/mainalu/KEY       zz1
/mainalu/LEDR      00000011
/mainalu/HEX0      1111xx1
/mainalu/HEX1      1111111
/mainalu/HEX2      zzzzzzz
/mainalu/HEX3      1111111
/mainalu/HEX4      0110000
/mainalu/HEX5      1000000
/mainalu/Case0     00000010
/mainalu/Case1     00000100
/mainalu/Case2     00000100
/mainalu/Case3     00110010
/mainalu/Case4     00000001
/mainalu/Case5     00000110
/mainalu/Case6     00000001
/mainalu/Case7     00000011
/mainalu/final     00000011
/mainalu/Out       00000001

Now                50 ns
Cursor 1           40 ns
```

PART3 Q1
When load_n = 1 and ShiftRight = 0 the shifter bits takes the previous output as input.
PART3 SCHEDMATIC



PART 3 Verilog Code
```
module ShiftRegister(SW, KEY, LEDR);
        input [9:0] SW;
        input [9:0] KEY;
        output [9:0] LEDR;
        wire w1, w2, w3;

        Shifter s1(
        .loadVal(SW[7:0]),
        .load_n(KEY[1]),
        .ShiftRight(KEY[2]),
        .ASR(KEY[3]),
        .clk(KEY[0]),
        .reset_n(SW[9]),
        .Q(LEDR[7:0])
        );



endmodule

module Shifter(loadVal, load_n, ShiftRight, ASR, clk, reset_n, Q);
        input [7:0] loadVal;
        input load_n;
        input ShiftRight;
        input ASR;
        input clk;
        input reset_n;
        output wire [7:0] Q;

        reg leftMost;


        always @(ASR)   //shuld it be @(*)????
        begin
        if (ASR == 0)
        leftMost <= 0;
        else
        leftMost <= loadVal[7]; //should it be Q[0]
        end


        ShifterBit u0(
        .load_val(loadVal[7]),
        .in(leftMost),
        .shift(ShiftRight),
        .load_n(load_n),
        .clk(clk),
        .reset_n(reset_n),
        .out(Q[7])
        );

        ShifterBit u1(
        .load_val(loadVal[6]),
        .in(Q[7]),
        .shift(ShiftRight),
        .load_n(load_n),
        .clk(clk),
        .reset_n(reset_n),
        .out(Q[6])
        );

        ShifterBit u2(
        .load_val(loadVal[5]),
        .in(Q[6]),
        .shift(ShiftRight),
        .load_n(load_n),
        .clk(clk),
        .reset_n(reset_n),
        .out(Q[5])
```

```
);

ShifterBit u3(
.load_val(loadVal[4]),
.in(Q[5]),
.shift(ShiftRight),
.load_n(load_n),
.clk(clk),
.reset_n(reset_n),
.out(Q[4])
);

ShifterBit u4(
.load_val(loadVal[3]),
.in(Q[4]),
.shift(ShiftRight),
.load_n(load_n),
.clk(clk),
.reset_n(reset_n),
.out(Q[3])
);

ShifterBit u5(
.load_val(loadVal[2]),
.in(Q[3]),
.shift(ShiftRight),
.load_n(load_n),
.clk(clk),
.reset_n(reset_n),
.out(Q[2])
);

ShifterBit u6(
.load_val(loadVal[1]),
.in(Q[2]),
.shift(ShiftRight),
.load_n(load_n),
.clk(clk),
.reset_n(reset_n),
.out(Q[1])
);

ShifterBit u7(
        .load_val(loadVal[0]),
        .in(Q[1]),
        .shift(ShiftRight),
        .load_n(load_n),
        .clk(clk),
        .reset_n(reset_n),
        .out(Q[0])
        );

endmodule

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

        wire m1m2;
        wire m2dff1;

        mux2to1 M1(
        .x(out),
        .y(in),
        .s(shift),
        .m(m1m2)
        );

        mux2to1 M2(
        .x(load_val),
        .y(m1m2),
        .s(load_n),
        .m(m2dff1)
        );

        dFlipFlop DFF1(
        .D(m2dff1),
        .clock(clk),
        .reset_n(reset_n),
        .Q(out)
        );
```

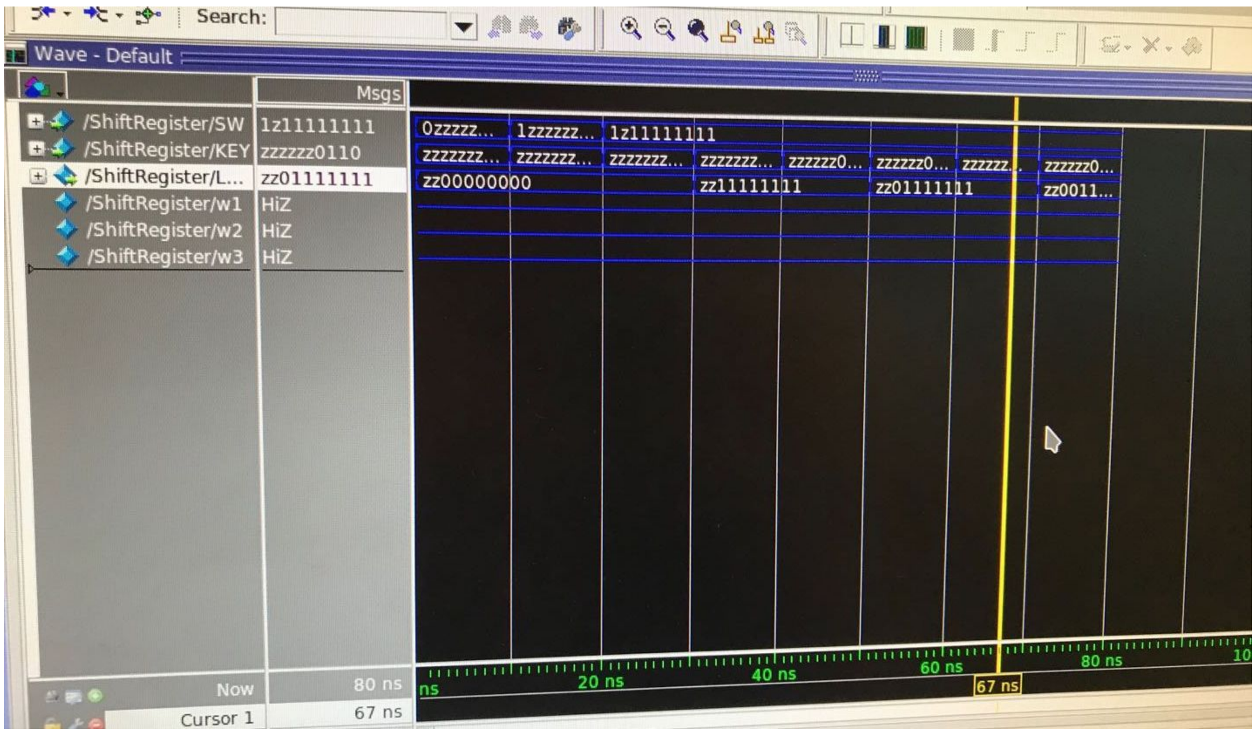
```
endmodule

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 dFlipFlop(D, clock, reset_n, Q);
        input D;
        input clock;
        input reset_n;

        output reg Q;

        always @(posedge clock)
        begin
        if (reset_n == 1'b0)
        Q <= 0;
        else
        Q <= D;
        end

endmodule
```

Note to self:
@(*) responds to any change in input
While @(pos edge) or wtvr responds only to that.

PART 3 VSIM CODE:

```
# Set the working dir, where all compiled Verilog goes.
vlib work

# Compile all verilog modules in mux.v to working dir;
# could also have multiple verilog files.
vlog -timescale 1ns/1ns ShiftRegister.v

# Load simulation using mux as the top level simulation module.
vsim ShiftRegister

# Log all signals and add some signals to waveform window.
log {/*}
# add wave {/*} would add all items in top level simulation module.
add wave {/*}

#SW[9] is reset_n and its active LOW! u silly shit
# Set input values using the force command, signal names need to be in {} brackets.

#first 20s RESETS THE SHIFTER
force {KEY[0]} 1
force {SW[9]} 0
run 10ns

force {SW[9]} 1
force {KEY[0]} 0
run 10ns

#LOAD 11111111
force {KEY[0]} 0
force {KEY[2]} 0
force {SW[7:0]} 2#11111111
force {KEY[1]} 0
run 10ns

force {KEY[0]} 1
run 10ns

#logical shift right   so expect 11111111 > 1'b1 = 01111111
#clock = 0
force {KEY[0]} 0
#ASR = 0 ie logical shift right
force {KEY[3]} 0
```

```
#turn load n off
force {KEY[1]} 1
#turn shiftRigt on (selects input)
force {KEY[2]} 1


run 10ns

force {KEY[0]} 1
run 10ns

#logical shift right   so expect 01111111 > 1'b1 = 00111111
#clock = 0
force {KEY[0]} 0
#ASR = 0 ie logical shift right
force {KEY[3]} 0
#turn load n off
force {KEY[1]} 1
#turn shiftRigt on (selects input)
force {KEY[2]} 1
run 10ns
force {KEY[0]} 1
run 10ns
```
PART 3 VSIM SCREENSHOT



```
/ShiftRegister/SW     1z11111111    0zzzzz...  1zzzzzz...  1z11111111
/ShiftRegister/KEY    zzzzzz0110    zzzzzzz...  zzzzzzz...  zzzzzzz...  zzzzzzz...  zzzzzz0...  zzzzzz0...  zzzzzz...  zzzzzz0...
/ShiftRegister/L...   zz01111111    zz00000000  zz11111111  zz01111111  zz0011...
/ShiftRegister/w1     HiZ
/ShiftRegister/w2     HiZ
/ShiftRegister/w3     HiZ

Now       80 ns
Cursor 1  67 ns     67 ns
```