Verilog Code:

```
module DE1_SoC(CLOCK_50, SW, LEDR, KEY, HEX0, HEX5);
	input CLOCK_50;
	input [9:0] SW;
	input [3:0] KEY;
	output logic [9:0] LEDR;
	output logic [6:0] HEX0, HEX5;


	assign LEDR[0] = 1'b0;		//skip LEDR[0]

	wire right,left;
	wire winR, winL;
	wire [2:0] counterL, counterR;
	wire [9:0] randomConnector;
	wire computerSignal;

	//clock divider
	wire[31:0] div_clk;
	parameter whichClock = 15;
	clock_divider cdiv (.clock(CLOCK_50), .reset(SW[9]), .divided_clocks(div_clk));

	logic clkSelect;
	// Uncomment ONE of the following two lines depending on intention

	//assign clkSelect = CLOCK_50;		// for simulation
	assign clkSelect = div_clk[whichClock]; // for board

	LFSR generateRandomNumber (.Clock(clkSelect), .Reset(SW[9]),
.LFSROut(randomConnector));

	compare10 comparator (.A10(SW[8:0]), .B10(randomConnector[9:0]),
.Alarger(computerSignal));

	userInput humanPlayer (.Clock(clkSelect) , .Reset(SW[9]), .pressed(~KEY[0]),
.set(right));
	userInput computer (.Clock(clkSelect) , .Reset(SW[9]), .pressed(computerSignal),
.set(left));


	normalLight r4(.Clock(clkSelect), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[2]),
.NR(1'b0), .lightOn(LEDR[1]));
```

```
	normalLight r3(.Clock(clkSelect), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[3]),
.NR(LEDR[1]), .lightOn(LEDR[2]));
	normalLight r2(.Clock(clkSelect), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[4]),
.NR(LEDR[2]), .lightOn(LEDR[3]));
	normalLight r1(.Clock(clkSelect), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[5]),
.NR(LEDR[3]), .lightOn(LEDR[4]));

	centerLight c1(.Clock(clkSelect), .Reset(SW[9] | winR | winL), .L(left), .R(right),
.NL(LEDR[6]), .NR(LEDR[4]), .lightOn(LEDR[5]));

	normalLight l1(.Clock(clkSelect), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[7]),
.NR(LEDR[5]), .lightOn(LEDR[6]));
	normalLight l2(.Clock(clkSelect), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[8]),
.NR(LEDR[6]), .lightOn(LEDR[7]));
	normalLight l3(.Clock(clkSelect), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[9]),
.NR(LEDR[7]), .lightOn(LEDR[8]));
	normalLight l4(.Clock(clkSelect), .Reset(SW[9]), .L(left), .R(right), .NL(1'b0),
.NR(LEDR[8]), .lightOn(LEDR[9]));

	checkWinner whoWon(LEDR[9], LEDR[1], left, right, winL, winR); //leftMostLight:
LEDR[9], rightMostLight: LEDR[1]

	winCounter leftCount(.Clock(clkSelect), .Reset(SW[9] | (counterL == 3'b111) | (counterR
== 3'b111)), .Signal(winL), .out(counterL));
	winCounter rightCount(.Clock(clkSelect), .Reset(SW[9]| (counterL == 3'b111) | (counterR
== 3'b111)), .Signal(winR), .out(counterR));

	seg7display display (.Clock(clkSelect), .Reset(SW[9]), .player1Counter(counterL),
.player2Counter(counterR), .player1Light(HEX5), .player2Light(HEX0));

endmodule


// divided_clocks[0] = 25MHz, [1] = 12.5MHz, [23] = 3Hz, [24] 1.5Hz,
// [25] = 0.75 Hz
module clock_divider (clock, reset, divided_clocks);
	input logic reset, clock;
	output logic [31:0] divided_clocks;



	always_ff @(posedge clock) begin
		divided_clocks <= divided_clocks + 1;
	end
```

```
endmodule

module winCounter(Clock, Reset, Signal, out);
	input Clock, Reset;
	input Signal;
	logic [2:0] PS, NS;
	output logic [2:0] out;

	parameter [2:0]		zero =  3'b000,
						one =   3'b001,
						two =   3'b010,
						three =		3'b011,
						four =  3'b100,
						five =  3'b101,
						six =		3'b110,
						seven =		3'b111;

	always_comb begin
		case(PS)
		zero: if(Signal)		NS = one;
				else			NS = zero;
		one: if(Signal)	NS = two;
				else			NS = one;
		two: if(Signal)	NS = three;
				else			NS = two;
		three: if(Signal) NS = four;
				else			NS = three;
		four: if(Signal) NS = five;
				else			NS = four;
		five: if(Signal)	NS = six;
				else			NS = five;
		six: if(Signal)	NS = seven;
				else			NS = six;
		seven: if(Signal)		NS = zero;
				else			NS = seven;
		endcase
	end

	always @(posedge Clock) begin
		if (Reset) begin
				PS <= zero;
		end
		else
				PS <= NS;
				out <= NS;
```

```
        end

endmodule

module counter_testbench();
        logic clk, rst, test;
        wire[2:0] Out;

        counter dut (.Clock(clk), .Reset(rst), .Signal(test), .out(Out));

        parameter CLOCK_PERIOD = 100;

        initial clk = 1;
        always begin
                        #(CLOCK_PERIOD/2);
                clk = ~clk;
        end
        // Set up the inputs to the design. Each line is a clock cycle.
        initial begin

                rst <= 1;       @(posedge clk);
                                        @(posedge clk);
                                        @(posedge clk);
                                        @(posedge clk);


                rst <= 0;       @(posedge clk);
                                        @(posedge clk);
                                        @(posedge clk);
                                        @(posedge clk);


                test <= 1;      @(posedge clk);
                                        @(posedge clk);
                                        @(posedge clk);
                                        @(posedge clk);
                                        @(posedge clk);
                                        @(posedge clk);

                rst <= 1;       @(posedge clk);
                                        @(posedge clk);
                                        @(posedge clk);
                rst <= 0;       @(posedge clk);
                                        @(posedge clk);
```

```
		test <= 0;		@(posedge clk);
							@(posedge clk);
							@(posedge clk);
							@(posedge clk);
							@(posedge clk);
							@(posedge clk);
							@(posedge clk);

		rst <= 1;		@(posedge clk);
							@(posedge clk);
							@(posedge clk);
							@(posedge clk);


		rst <= 0;		@(posedge clk);
							@(posedge clk);
							@(posedge clk);
							@(posedge clk);
							@(posedge clk);

		$stop;
	end
endmodule

module LFSR (Clock, Reset, LFSROut);
	input Clock, Reset;
	output [9:0] LFSROut;
	logic [9:0] PS, NS;
	reg temp;

	always @(*) begin
		temp = ~(PS[6] ^ PS[8]);
		NS = {PS[8:0], temp};
	end

	assign LFSROut = PS;

	always @(posedge Clock)
		if(Reset)
			PS <= 10'b0000000000;
		else
			PS <= NS;
endmodule
```

```
module LFSR_testbench();
	logic clk, rst;
	wire [8:0] Out;

	LFSR dut (.Clock(clk), .Reset(rst), .LFSROut(Out));


	parameter CLOCK_PERIOD = 100;

	initial clk = 1;
	always begin
			#(CLOCK_PERIOD/2);
		clk = ~clk;
	end

	initial begin

		rst <= 1;	@(posedge clk);
		rst <= 0;	@(posedge clk);
					@(posedge clk);
					@(posedge clk);
					@(posedge clk);

					@(posedge clk);
					@(posedge clk);
					@(posedge clk);
					@(posedge clk);

		rst <= 1;	@(posedge clk);
		rst <= 0;	@(posedge clk);
					@(posedge clk);
					@(posedge clk);
					@(posedge clk);

		$stop;
	end
endmodule

module compare1(A,B, Equal, Alarger);
	input logic A, B;
	output logic Equal, Alarger;

	assign Equal = (A&B) | (~A&~B);
	assign Alarger = (A&~B);
```

```
endmodule

module compare10(A10, B10, Alarger);
	input logic [9:0] A10, B10;
	output logic Alarger;

	wire e0, e1, e2, e3, e4, e5, e6, e7, e8, e9;
	wire AC0, AC1, AC2, AC3, AC4, AC5, AC6, AC7, AC8, AC9;

	compare1 digit0 (A10[0], B10[0], e0, AC0);
	compare1 digit1 (A10[1], B10[1], e1, AC1);
	compare1 digit2 (A10[2], B10[2], e2, AC2);
	compare1 digit3 (A10[3], B10[3], e3, AC3);
	compare1 digit4 (A10[4], B10[4], e4, AC4);
	compare1 digit5 (A10[5], B10[5], e5, AC5);
	compare1 digit6 (A10[6], B10[6], e6, AC6);
	compare1 digit7 (A10[7], B10[7], e7, AC7);
	compare1 digit8 (A10[8], B10[8], e8, AC8);
	compare1 digit9 (A10[9], B10[9], e9, AC9);

	assign Alarger = (	AC9
						| (AC8 & e9)
						| (AC7 & e9 & e8)
						| (AC6 & e9 & e8 & e7)
						| (AC5 & e9 & e8 & e7 & e6)
						| (AC4 & e9 & e8 & e7 & e6 & e5)
						| (AC3 & e9 & e8 & e7 & e6 & e5 & e4)
						| (AC2 & e9 & e8 & e7 & e6 & e5 & e4 & e3)
						| (AC1 & e9 & e8 & e7 & e6 & e5 & e4 & e3
& e2)
						| (AC1 & e9 & e8 & e7 & e6 & e5 & e4 & e3
& e2 & e1)
						);

endmodule

module compare10_testbench();
	logic clk;
	logic [9:0] testA10, testB10;
	logic Alarger, Blarger;
	logic Equal;

	compare10 dut (.A10(testA10), .B10(testB10), .Equal(Equal), .Alarger(Alarger),
.Blarger(Blarger));
```

```
	parameter CLOCK_PERIOD = 100;

	initial clk = 1;
	always begin
			#(CLOCK_PERIOD/2);
		clk = ~clk;
	end
	// Set up the inputs to the design. Each line is a clock cycle.
	initial begin

		testA10 <= 10'b0000000000; testB10 <= 10'b0000000001; @(posedge clk);
//B bigger than A

							@(posedge clk);
		testA10 <= 10'b0000000010; testB10 <= 10'b0000000000; @(posedge clk); //A
bigger than B

							@(posedge clk);
		testA10 <= 10'b0000000000; testB10 <= 10'b0000000000; @(posedge clk);
//equal

							@(posedge clk);
		testA10 <= 10'b0001000000; testB10 <= 10'b0000000001; @(posedge clk); //A
bigger than B

							@(posedge clk);


		testA10 <= 10'b1100000000; testB10 <= 10'b000011000;		@(posedge
clk); // A bigger than B

							@(posedge clk);
		testA10 <= 10'b0000000000; testB10 <= 10'b0000100101; @(posedge clk); //B
bigger than A

							@(posedge clk);


		$stop;
	end
endmodule


module seg7display(Clock, Reset, player1Counter, player2Counter, player1Light, player2Light);
	input Clock, Reset;
```

```
	input [2:0] player1Counter, player2Counter;
	logic [6:0] PS, NS;
	output logic [6:0] player1Light, player2Light;

	//                                                          6543210
	parameter [6:0]		zero =  7'b1000000,	//hex displays for counting (active low)
                                                one =   7'b1111001,
                                                two =   7'b0100100,
                                                three =         7'b0110000,
                                                four =  7'b0011001,
                                                five =  7'b0010010,
                                                six =   7'b0000010,
                                                seven =         7'b1111000;

	always @(*) begin
		case(player1Counter)
			3'b000:		player1Light = zero;
			3'b001:player1Light = one;
			3'b010:player1Light = two;
			3'b011:player1Light = three;
			3'b100:player1Light = four;
			3'b101:player1Light = five;
			3'b110:player1Light = six;
			3'b111:player1Light = seven;
		endcase

		case(player2Counter)
			3'b000: player2Light = zero;
			3'b001: player2Light = one;
			3'b010: player2Light = two;
			3'b011: player2Light = three;
			3'b100: player2Light = four;
			3'b101: player2Light = five;
			3'b110: player2Light = six;
			3'b111: player2Light = seven;
		endcase
	end


endmodule

module centerLight (Clock, Reset, L, R, NL, NR, lightOn);
	input Clock, Reset;
	input	L, R, NL, NR;
	output logic lightOn;
```

```
	logic PS, NS;
	parameter off = 1'b0, on = 1'b1;

	always @(*)
		case(PS)
			off:	if (NL & R)		NS = on;
				else if (NR & L)	NS = on;
				else				NS = off;
			on:	if (R ^ L) NS = off;
				else				NS = on;
		default: NS = 1'bx;
	endcase

	always @(*)
		case(PS)
			off: lightOn = off;
			on: lightOn = on;
			default: lightOn = 1'bx;
	endcase

	always @(posedge Clock)
		if (Reset)
			PS <= on; // reset should turn the center light on
		else
			PS <= NS;

endmodule		//centerLight


module centerLight_testbench();

	logic Clk, Reset;
	logic [9:0] LEDR;
	logic [9:0] SW;
	logic NL, NR, L, R;

	centerLight dut(.Clock(Clk), .Reset(Reset), .L(L), .R(R), .NL(NL), .NR(NR),
.lightOn(LEDR[5]));


	parameter CLOCK_PERIOD = 100;

	initial Clk = 1;
	always begin
```

```
                #(CLOCK_PERIOD/2);
        Clk = ~Clk;
    end
    // Set up the inputs to the design. Each line is a clock cycle.
    initial begin

        Reset <= 1; @(posedge Clk);
                                @(posedge Clk);

        Reset <= 0;    @(posedge Clk);
                                @(posedge Clk);

        L <= 1;        @(posedge Clk);
                                @(posedge Clk);

        NR <= 1;            @(posedge Clk);
                                @(posedge Clk);

        NR <= 0;            @(posedge Clk);
                                @(posedge Clk);

        L <= 0;        @(posedge Clk);
                                @(posedge Clk);

        R <= 1;             @(posedge Clk);
                                @(posedge Clk);

        NL <= 1;       @(posedge Clk);
                                @(posedge Clk);

        NL <= 0;            @(posedge Clk);
                                @(posedge Clk);

        R <= 0;             @(posedge Clk);
                                @(posedge Clk);

        $stop;
    end
endmodule

module normalLight (Clock, Reset, L, R, NL, NR, lightOn);
    input Clock, Reset;


    input L, R, NL, NR;
```

```
	logic PS, NS;
	parameter off = 1'b0, on = 1'b1;
	// when lightOn is true, the normal light should be on.
	output reg lightOn;

	// while
	always @(*)
	case(PS)
		off:	if (NL & R) NS = on;
				else if (NR & L) NS = on;
				else NS = off;
		on:	if (R ^ L) NS = off;
				else NS = on;
		default: NS = 1'bx;
	endcase


	always @(*)
		case(PS)
			off: lightOn = 0;
			on: lightOn = 1;
			default: NS = 1'bx;
	endcase


	always @(posedge Clock)
		if (Reset)
			PS <= off; // normal light should be turned off when reset
		else
			PS <= NS;

	endmodule

module normalLight_testbench();
	logic clk, Reset;
	logic LEDR[3];
	logic L, R, NL, NR;

	normalLight dut (.Clock(clk), .Reset(Reset), .L(L), .R(R), .NL(NL), .NR(NR),
.lightOn(LEDR[3]));


	parameter CLOCK_PERIOD = 100;
		initial begin
			clk <= 0;
```

```
                forever #(CLOCK_PERIOD/2) clk <= ~clk;
        end

    initial begin
        Reset <= 1;

        @(posedge clk);

        @(posedge clk);


        Reset <= 0;
@(posedge clk);

        @(posedge clk);


                                                                L<=1;
    @(posedge clk);

        @(posedge clk);


                                                                    NR <= 1;
    @(posedge clk);

        @(posedge clk);


                                                                NR <= 0;
@(posedge clk);

        @(posedge clk);


        L <= 0;
@(posedge clk);

        @(posedge clk);


                        R <= 1;
@(posedge clk);

        @(posedge clk);
```

```
                                                NL <= 1;                @(posedge
clk);

                @(posedge clk);


                        NL <= 0;
        @(posedge clk);

                @(posedge clk);


                        R <= 0;
        @(posedge clk);


                @(posedge clk);
                $stop; //End the simulation.
        end
endmodule

module userInput(Clock, Reset, pressed, set);
        input Clock, Reset;
        input pressed;
        output reg set;
        logic [1:0] PS, NS;
        parameter [1:0] on = 2'b00, hold = 2'b01, off = 2'b10;

        always @(*)
        case(PS)
                on:     if (pressed) NS = hold;
                                else NS = off;
                hold:   if (pressed) NS = hold;
                                else NS = off;
                off:    if (pressed) NS = on;
                                else NS = off;
                default: NS = 2'bxx;
        endcase

        always @(*)
        case(PS)
                on: set = 1;
                hold: set = 0;
```

```
		off: set = 0;
		default: set = 1'bx;
	endcase

	always_ff @(posedge Clock)
		if (Reset)
			PS <= off;
		else
			PS <= NS;

endmodule

module checkWinner(leftMostLight, rightMostLight, left, right, winL, winR);
	input leftMostLight, rightMostLight, left, right;
	output winL, winR;

	assign winL = leftMostLight & left & ~right;
	assign winR = rightMostLight & right & ~left;
endmodule

module DE1_SoC_testbench(); //CLOCK_50, KEY, SW, LEDR, HEX0
	logic CLOCK_50;
	logic [9:0] SW;
	logic [3:0] KEY;
	logic [9:0] LEDR;
	logic [6:0] HEX0, HEX5;


	DE1_SoC dut (CLOCK_50, SW, LEDR, KEY, HEX0, HEX5);

	initial CLOCK_50 = 1;
	parameter CLOCK_PERIOD = 100;

	always begin
			#(CLOCK_PERIOD/2);
		CLOCK_50 = ~CLOCK_50;
	end



	initial begin			//only toggles with the player 1&2 keys
				SW[9] <= 1; @(posedge CLOCK_50);
							@(posedge CLOCK_50);
				SW[9] <= 0;	@(posedge CLOCK_50);
							@(posedge CLOCK_50);
```

```
		SW[8:0] <= 9'b111000000; @(posedge CLOCK_50);


		KEY[0] <= 1;		@(posedge CLOCK_50); //checking scoreBoard for
counter1
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50);	//1

		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50);	//2

		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50); //3

		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
		KEY[0] <= 1;		@(posedge CLOCK_50);
		KEY[0] <= 0;		@(posedge CLOCK_50);
```

```
KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50); //4

KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50); //5

KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50); //6

KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);
KEY[0] <= 1;		@(posedge CLOCK_50);
KEY[0] <= 0;		@(posedge CLOCK_50);	// should restart game here!
KEY[0] <= 1;		@(posedge CLOCK_50); //7

SW[9] <= 1;			@(posedge CLOCK_50); //reset
```

```
                                                @(posedge CLOCK_50);
                SW[9] <= 0;                 @(posedge CLOCK_50);
                                                @(posedge CLOCK_50);

                KEY[3] <= 1;            @(posedge CLOCK_50); //checking scoreBoard for
counter2
                KEY[3] <= 0;            @(posedge CLOCK_50);
                KEY[3] <= 1;            @(posedge CLOCK_50);
                KEY[3] <= 0;            @(posedge CLOCK_50);
                KEY[3] <= 1;            @(posedge CLOCK_50);
                KEY[3] <= 0;            @(posedge CLOCK_50);
                KEY[3] <= 1;            @(posedge CLOCK_50);
                KEY[3] <= 0;            @(posedge CLOCK_50);
                KEY[3] <= 1;            @(posedge CLOCK_50);
                KEY[3] <= 0;            @(posedge CLOCK_50);
                KEY[3] <= 1;            @(posedge CLOCK_50);


                SW[9] <= 1;                 @(posedge CLOCK_50); //reset
                                                @(posedge CLOCK_50);
                SW[9] <= 0;                 @(posedge CLOCK_50);
                                                @(posedge CLOCK_50);

                $stop;
        end
endmodule
```

Resource Utilization by Entity:

Analysis & Synthesis Resource Utilization by Entity

<<Filter>>

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins	Full Hierarchy Name	Entity Name	Library Name
1	▾ |DE1_SoC	71 (2)	51 (0)	0	0	39	0	|DE1_SoC	DE1_SoC	work
1	|LFSR:generateRandomNumber|	1 (1)	10 (10)	0	0	0	0	|DE1_SoC|LFSR:generateRandomNumber	LFSR	work
2	|centerLight:c1|	2 (2)	1 (1)	0	0	0	0	|DE1_SoC|centerLight:c1	centerLight	work
3	|checkWinner:whoWon|	2 (2)	0 (0)	0	0	0	0	|DE1_SoC|checkWinner:whoWon	checkWinner	work
4	|clock_divider:cdiv|	16 (16)	16 (16)	0	0	0	0	|DE1_SoC|clock_divider:cdiv	clock_divider	work
5	|compare10:comparator|	6 (6)	0 (0)	0	0	0	0	|DE1_SoC|compare10:comparator	compare10	work
6	|normalLight:l1|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:l1	normalLight	work
7	|normalLight:l2|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:l2	normalLight	work
8	|normalLight:l3|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:l3	normalLight	work
9	|normalLight:l4|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:l4	normalLight	work
10	|normalLight:r1|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:r1	normalLight	work
11	|normalLight:r2|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:r2	normalLight	work
12	|normalLight:r3|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:r3	normalLight	work
13	|normalLight:r4|	2 (2)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:r4	normalLight	work
14	|seg7display:display|	14 (14)	0 (0)	0	0	0	0	|DE1_SoC|seg7display:display	seg7display	work
15	|userInput:computer|	4 (4)	2 (2)	0	0	0	0	|DE1_SoC|userInput:computer	userInput	work
16	|userInput:humanPlayer|	3 (3)	2 (2)	0	0	0	0	|DE1_SoC|userInput:humanPlayer	userInput	work
17	|winCounter:leftCount|	6 (6)	6 (6)	0	0	0	0	|DE1_SoC|winCounter:leftCount	winCounter	work
18	|winCounter:rightCount|	6 (6)	6 (6)	0	0	0	0	|DE1_SoC|winCounter:rightCount	winCounter	work

The computed size of my design is 262.