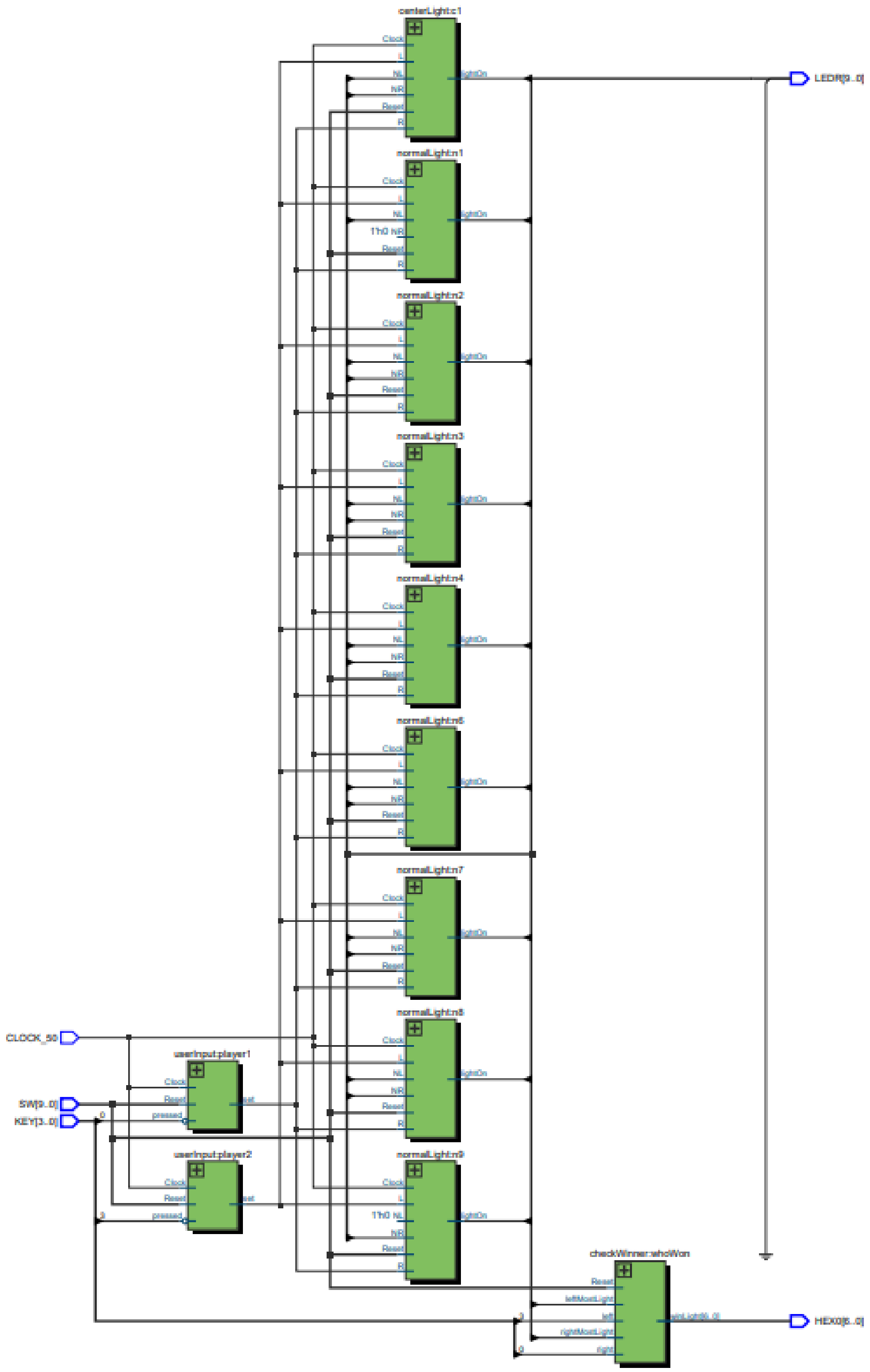
Top Level Block Diagram:



Verilog Code:

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


	assign LEDR[0] = 1'b0;		//skip LEDR[0]
	//connecting wires
	wire right,left;		//used to connect FSMs on left to right

	// buttons for the two different players
	userInput player1 (.Clock(CLOCK_50) , .Reset(SW[9]), .pressed(~KEY[0]), .set(right));
//player 1 presses KEY[0]
	userInput player2 (.Clock(CLOCK_50) , .Reset(SW[9]), .pressed(~KEY[3]), .set(left));
//player 2 presses KEY[3]

	//player1 set value is true when rightmost input is true (player 1 = right)
	//player2 set value is true when leftmost input is true (player 2 = left)


	// instantiations of normal lights and center lights (skips LEDR[0])
	normalLight n1(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[2]),
.NR(1'b0), .lightOn(LEDR[1]));
	normalLight n2(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[3]),
.NR(LEDR[1]), .lightOn(LEDR[2]));
	normalLight n3(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[4]),
.NR(LEDR[2]), .lightOn(LEDR[3]));
	normalLight n4(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[5]),
.NR(LEDR[3]), .lightOn(LEDR[4]));

	centerLight c1(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[6]),
.NR(LEDR[4]), .lightOn(LEDR[5]));

	normalLight n6(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[7]),
.NR(LEDR[5]), .lightOn(LEDR[6]));
	normalLight n7(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[8]),
.NR(LEDR[6]), .lightOn(LEDR[7]));
```

```
	normalLight n8(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(LEDR[9]),
.NR(LEDR[7]), .lightOn(LEDR[8]));
	normalLight n9(.Clock(CLOCK_50), .Reset(SW[9]), .L(left), .R(right), .NL(1'b0),
.NR(LEDR[8]), .lightOn(LEDR[9]));

	//checksWinner to display on hex display
	checkWinner whoWon(.rightMostLight(LEDR[9]), .leftMostLight(LEDR[1]), .left(KEY[3]),
.right(KEY[0]), .Reset(SW[9]), .winLight(HEX0)); //leftMostLight: LEDR[9], rightMostLight:
LEDR[1]



endmodule


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


	initial
		divided_clocks <= 0;

	always @(posedge clock)
		divided_clocks <= divided_clocks + 1;

endmodule



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


	// L is true when left key is pressed
	// R is true when the right key is pressed
	// NL is true when the light on the left is on
	// NR is true when the light on the right is on
	//	lightOn is true -- centerLight is on
	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

module centerLight_testbench();

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

	//instantiating centerLight module for test
	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           //centerLight_testbench

    module normalLight (Clock, Reset, L, R, NL, NR, lightOn);
        input Clock, Reset;
        // L is true when left key is pressed
        // R is true when the right key is pressed
        // NL is true when the light on the left is on
```

```
	// NR is true when the light on the right is on
	//	lightOn is true -- normalLight is on

	//**FSM Logic Block for normalLight**
	//see notes for centerLight, they are mostly the same

	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 //normalLight

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

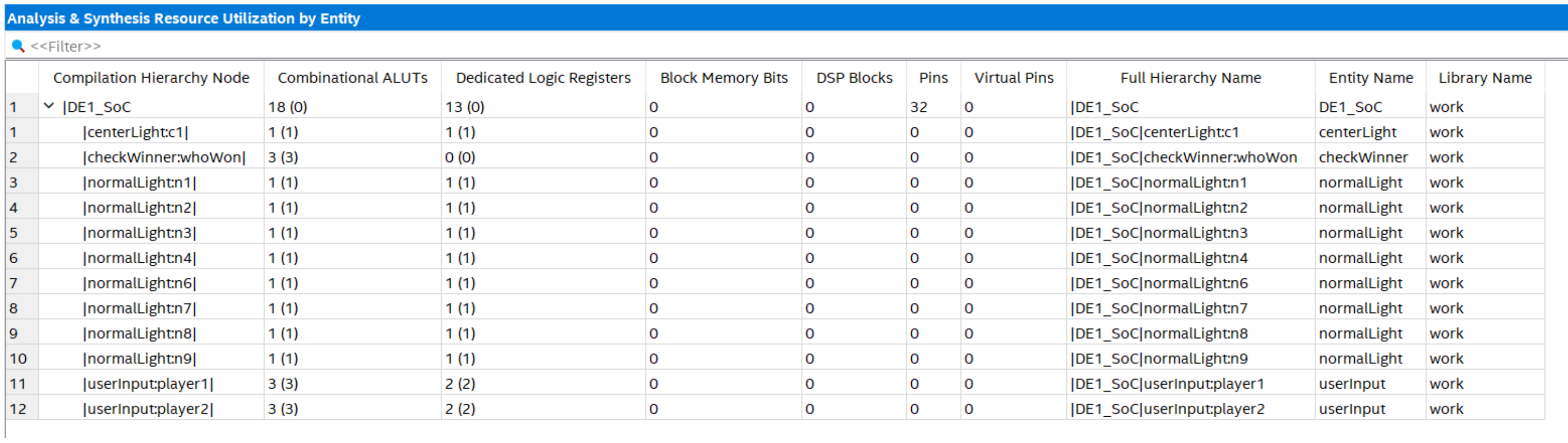
```
          normalLight dut (.Clock(clk), .Reset(Reset), .L(L), .R(R), .NL(NL), .NR(NR),
.lightOn(LEDR[3]));

          //Set up the clock.
          parameter CLOCK_PERIOD = 100;
                    initial begin
                              clk <= 0;
                              forever #(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 the simulation.
		end
endmodule //normalLight_testbench


//metastability (user input)
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, Reset, winLight);
    input leftMostLight, rightMostLight, left, right, Reset;
    parameter b1 = 7'b1111001, b2 = 7'b0100100, b0 = 7'b1000000;
    logic winL, winR;
    output logic [6:0] winLight;


    assign winL = leftMostLight & left & ~right; //set up win condition for player 1
    assign winR = rightMostLight & right & ~left;//               and player 2
    always @(*)
    begin
        if (winL) // if player 2 wins
            winLight = b2;
        else if (winR) // if play 1 wins
            winLight = b1;
        else if (Reset | ~winL & ~winR) //game reset
            winLight = b0;
    end

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;


    //instantiating main module
    DE1_SoC dut (CLOCK_50, SW, LEDR, KEY, HEX0);

    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 player 1 win
        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


        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 player 2 win
            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	18 (0)	13 (0)	0	0	32	0	|DE1_SoC	DE1_SoC	work
1	|centerLight:c1|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|centerLight:c1	centerLight	work
2	|checkWinner:whoWon|	3 (3)	0 (0)	0	0	0	0	|DE1_SoC|checkWinner:whoWon	checkWinner	work
3	|normalLight:n1|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:n1	normalLight	work
4	|normalLight:n2|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:n2	normalLight	work
5	|normalLight:n3|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:n3	normalLight	work
6	|normalLight:n4|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:n4	normalLight	work
7	|normalLight:n6|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:n6	normalLight	work
8	|normalLight:n7|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:n7	normalLight	work
9	|normalLight:n8|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:n8	normalLight	work
10	|normalLight:n9|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|normalLight:n9	normalLight	work
11	|userInput:player1|	3 (3)	2 (2)	0	0	0	0	|DE1_SoC|userInput:player1	userInput	work
12	|userInput:player2|	3 (3)	2 (2)	0	0	0	0	|DE1_SoC|userInput:player2	userInput	work