**Bilkent University**

**CS223-01 COMPUTER SCIENCE**

**BITWISE GAME**

**Muhammet Said Demir**

**21602021**

**18.12.2017**

**About this project:** In this project, the player will follow the movements of the motor and try to copy it through keyboard.

The game actually contains multiple modules which includes the ready modules that teachers gave us, so I don’t write detailed information about them.

First, I had to use a random generator in order to get 4 bits random number. In this part, I got help from several websites, but in the final plan, I created 16 bits random number but only used the last 4 bits for simplicity.

Then, I’m using this 4 bits random number to generate the motor movement with implementing the ready modules. 2 actions, which corresponds to 2 bits each.

After that, I get a value from the user through keyboard with using ready modules. And with that in mind, I compare it to the number that had been created by random generator.

If they are the same, then I increment the total score of the player which I’m displaying with using ready seven\_segment module. If not, decrease it. Our points can be only between 9 and 0. User can also reset the score with a switch.

Further more in to details: I created my random number with using this method:

module lfsr\_counter( input clock,reset, output reg [3:0] result);

logic feedback;

reg [15:0]out;

assign feedback = ~( out[15] ^ out [10] ^ out [9] ^ out[5]);

always @(posedge clock, posedge reset)

begin

if(reset)

out = 16'b0;

else

out = {out[14:0],feedback};

end

assign result = out[3:0];

endmodule

The output is the random 4 bit number. It uses the logic of XNOR gate to help us to get semi-random number which is hard to trace.

Then I cut this number to half to get directions and rotations of two 2 bits numbers:

direction[0] <= random[1];

direction[1] <= random[3];

rotation\_duration[0] <= random[0];

rotation\_duration[1] <= random[2];

So, 0000 is, ShortLeft-ShortLeft and 1111 is LongRight-LongRight. 1001 is, ShortRight-LongLeft

And after calling the motor generator, the player can see the pattern that s/he has to follow. After that, I changed my keymap order in this way:

      4'b0000: out<= 4'b0010;

      4'b0001: out<= 4'b0000;

      4'b0010: out<= 4'b0001;

      4'b0011: out<= 4'b1101;

      4'b0100: out<= 4'b0100;

      4'b0101: out<= 4'b0101;

      4'b0110: out<= 4'b0110;

      4'b0111: out<= 4'b1000;

      4'b1000: out<= 4'b1001;

      4'b1001: out<= 4'b1010;

      4'b1010: out<= 4'b0011;

      4'b1011: out<= 4'b0111;

      4'b1100: out<= 4'b1011;

      4'b1101: out<= 4'b1111;

      4'b1110: out<= 4'b1110;

      4'b1111: out<= 4'b1100;

|  |  |  |  |
| --- | --- | --- | --- |
| C | E | F | B |
| 7 | 3 | A | 9 |
| 8 | 6 | 5 | 4 |
| D | 1 | 0 | 2 |

Then, I get value from the user and hold it in a register value.

After all of it, I use my comparison methods to compare the values and change the score in this order. My full code is:

module project(input logic clk, reset,

    output a, b, c, d, e, f, g, dp,

    output [3:0] an,

output [3:0] phases,

    input start,

    output logic [3:0] keyb\_row,

    input logic [3:0] keyb\_col

);

    logic [3:0] random,key\_value,score;

    logic [1:0] direction,rotation\_duration;

    logic [3:0] temp,temp1, temp2;

    logic  key\_valid;

    logic [1:0] state;

    lfsr\_counter generator(clk,reset,temp);

    keypad4X4 keyboard(clk,keyb\_row,keyb\_col,temp1,key\_valid);

    mappingTable map(temp1,temp2);

    always @(posedge clk)

       if(reset)

            score<=4'd0;

       else

            if(state == 2'b00)

            begin

                random <= temp;

                state <= 2'b01;

            end

            else if(state == 2'b01)

            begin

                direction[0] <= random[1];

                direction[1] <= random[3];

                rotation\_duration[0] <= random[0];

                rotation\_duration[1] <= random[2];

                if(key\_valid==0)

                    state <=2'b10;

            end

            else if(state == 2'b10)

            begin

                if(key\_valid == 1)

                begin

                     key\_value <= temp2;

                     state <= 2'b11;

                 end

             end

             else if(state == 2'b11)

             begin

                if(random == key\_value & score != 4'd9)

                    score <= score + 4'd1;

                else if(random == key\_value & score == 4'd9)

                    score <= 4'd0;

                else if(score != 4'd0)

                    score <= score - 4'd1;

                else

                    score <= score;

                state <= 2'b00;

             end

    steppermotor\_wrapper motor(clk,direction,rotation\_duration,phases,start | state == 2'b01);

    SevSeg\_4digit display(clk, 4'b0, 4'b0, 4'b0, score, a,b,c,d,e,f,g,dp,an);

endmodule

module lfsr\_counter(  input clock,reset, output reg [3:0] result);

    logic feedback;

    reg [15:0]out;

    assign feedback = ~( out[15] ^ out [10] ^ out [9] ^ out[5]);

    always @(posedge clock, posedge reset)

    begin

    if(reset)

    out = 16'b0;

    else

    out = {out[14:0],feedback};

    end

    assign result = out[3:0];

endmodule

module mappingTable(input logic [3:0] key\_value, output logic [3:0]out) ;

    always\_comb

      case(key\_value)

      4'b0000: out<= 4'b0010;

      4'b0001: out<= 4'b0000;

      4'b0010: out<= 4'b0001;

      4'b0011: out<= 4'b1101;

      4'b0100: out<= 4'b0100;

      4'b0101: out<= 4'b0101;

      4'b0110: out<= 4'b0110;

      4'b0111: out<= 4'b1000;

      4'b1000: out<= 4'b1001;

      4'b1001: out<= 4'b1010;

      4'b1010: out<= 4'b0011;

      4'b1011: out<= 4'b0111;

      4'b1100: out<= 4'b1011;

      4'b1101: out<= 4'b1111;

      4'b1110: out<= 4'b1110;

      4'b1111: out<= 4'b1100;

      endcase

endmodule