LED CATCHER

A GAME OF REFLEXES

TEAM MEMBERS:

1. 221CS109
   ANIKET MAITRA
   @am.221cs109@nitk.edu.in
   9740034505
2. 221CS124
   GOLLAPALLI HARSHITHA
   @gollapalliharshitha.221cs124@nitk.edu.in
   8660519201
3. 221CS157
   DIVYANSHU MANOJBHAI SURTI
   @surtidivyanshumanojbhai.221cs157@nitk.edu.in
   9979984341

ABSTRACT:

In this project we aim to design a circuit in which we set up 3 columns of LED's, then we generate light signals in those LED's in a random manner and keep three special LEDs marked in the bottom.

The game is all about when any of those special LED's light up then we must press a corresponding push-button switch. This switch must be pressed within a very short period of time after the LED lights up.

If it is pressed within the limited period of time then the game continues, else it stops.

- MOTIVATION AND BACKROUND:
  The basic idea of the game is to keep the player continuously engaged and test his/her reflex action till he/she makes a mistake.
  This game can really help to sharpen the reflex action of a person.
  Another motivation for creating this game is keeping Formula-1 drivers and table tennis players in mind. They need to have extremely sharp reflexes to excel in their respective sport.

- CONTRIBUTION:

  1. Aniket Maitra
  Came up with the idea of the game using LEDs and the idea of setting a time frame within which the user must press the button and overall design of the hardware model. Created Logisim circuit for the same.

  2. Surti Divyanshu
  Came up with the idea of terminating the game by lighting up all LEDs when the game gets over and hardware requirements for the project.

  3. Harshitha Gollapalli
  Helped to fix errors in the brainstorming session and recommended better set of hardware components.

BRIEF DESCRIPTION:

In order to construct this LED chaser game, I have used the following components:

1. JK flip flop
2. Clock
3. AND gates
4. Hex display decoder
5. Two bit counter
6. Two bit Splitter
7. Push Button
8. LEDs

- Usage of JK flip flop:

On the first clock pulse, the J input is 1 and the K input is 1. This causes the flip-flop to toggle its state, so the Q output goes from 0 to 1.

On the next clock pulse, the Q output is now 1, so it sets the J input to 1, and the complement of Q (Q') is 0, setting the K input to 0. With these inputs, the flip-flop toggles again, and the Q output goes from 1 to 0.

This process repeats with every clock pulse, resulting in the generation of alternate 0s and 1s at the Q output.

This results in the sequential glowing of LEDs in a pattern. One JK flip flop influences the input of the other JK flip flops in the same sequence for a particular column with synchronized clock pulse. So I have repeated the same combination of JK flip flops for all three columns but in random manner which is totally dependent on the reset button of the JK flip flop. Whenever I want to start the game, I have to toggle the reset button at different clock time to randomize the pattern.

- Usage of AND gate:

To change the sequence in the pattern, I have used AND gates.

AND gate is also used to test whether the user has pushed the correct button in the correct column when the last LED glows.

- Usage of Bit Counter:

This is totally dependent on the AND gate. Whenever the user enters the correct combination for the correct led the output of the particular column AND gate becomes high.

Whenever the output of AND gate is high the input of counter will increase by one according to the clock which is applied on it.

To display the output of that counter we use another hex display decoder and a reset button to reset the counter whenever I want to stop the game.

- Usage of Bit Splitter:

As I can see the counter data output is 8 bit and the hex display decoder input is 4 bit that's why we use bit splitter to display the number.I have used bit splitter to fan in 8 bit input and fan out into 2 bit output for input of each decoder.

- Usage of push button:

Push button is used to take the input from the user to play the game.

- Usage of LED:

LEDs are the main component of the game; without it we cannot make the user understand the pattern of glowing of LEDs.

- Usage of Clock:

Clock controls the basic sequence changing from 0 to 1 and vice versa on a particular tick frequency given by a user.

Instructions/Rules for the game:

First enable the simulation in Logism by pressing ctrl+E.

Second set the reset button high to start the game.

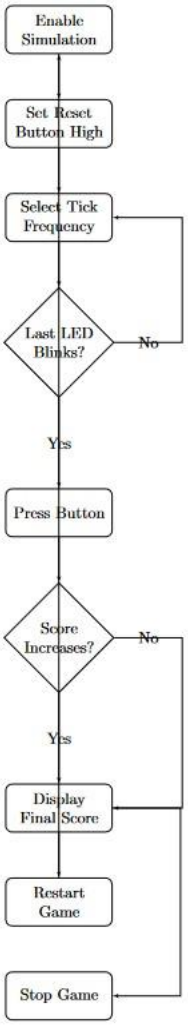
Now according to the convenience of the user select that tick frequency.

The rules of the game are that whenever the last LED in any column blinks we must press the button corresponding to that column. If we do so in a specific correct time limit then the counter increases our score otherwise the counter does not increase our score. To stop the game press ctrl+R. This stops the simulation.

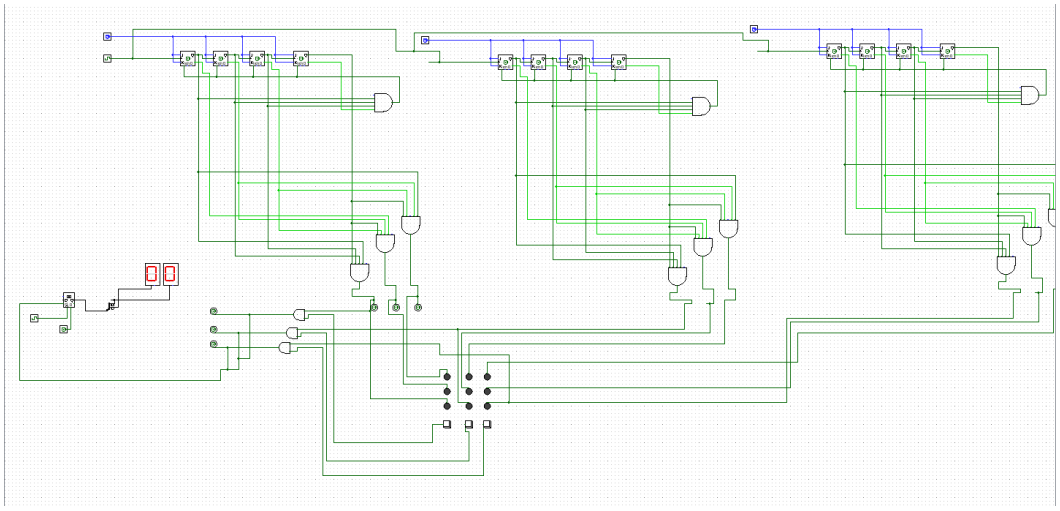
To play again start from the beginning of instructions/rules.

WORKING:

a	b	c	d	x	y	z	u	v	w
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0



Circuit Diagram:



VERILOG CODE:

```
module LED_Game (
    input wire clk,              // Clock input
    input wire reset,            // Reset input
    input wire start_game,       // Start game signal
    input wire [2:0] button,     // Push-button inputs (3 buttons)
    output wire [3:0] led,       // LED outputs (4 columns)
    output wire game_over        // Game over signal
);

// Parameters
parameter MAX_COUNT = 25000000; // Set this to control the time limit (e.g., 1 second at 25 MHz)

// Internal registers
reg [3:0] special_led;  // Stores the index of the currently lit special LED (last row)
reg [3:0] led_pattern;  // Stores the LED pattern for each column
reg [31:0] count;        // Counter for timing

// Game state
reg game_started;  // Indicates whether the game is in progress
reg game_active;   // Indicates whether the game is active
reg game_won;      // Indicates if the player won the game

// State machine states
localparam IDLE = 2'b00;
localparam PLAYING = 2'b01;
localparam GAME_OVER = 2'b10;

// State register
reg [1:0] state;

always @(posedge clk or posedge reset) begin
    if (reset) begin
        // Initialize the game
```

```
        // Initialize the game
        game_started <= 0;
        game_active <= 0;
        game_won <= 0;
        special_led <= 4'b0000;
        led_pattern <= 4'b0000;
        count <= 0;
        state <= IDLE;
    end else begin
        case(state)
            IDLE: begin
                if (start_game) begin
                    // Start the game
                    game_started <= 1;
                    game_active <= 1;
                    state <= PLAYING;
                end
            end
            PLAYING: begin
                if (game_active) begin
                    if (count >= MAX_COUNT) begin
                        // Time's up - game over
                        game_active <= 0;
                        game_won <= 0;
                        state <= GAME_OVER;
                    end else if (button == special_led) begin
                        // Correct button pressed - move to the next LED
                        special_led <= special_led + 1;
                        count <= 0;
                        if (special_led == 4'b1111) begin
                            // Player won the game
                            game_active <= 0;
                            game_won <= 1;
```

```
                        state <= GAME_OVER;
                    end
                end
            end
            GAME_OVER: begin
                if (!game_started || start_game) begin
                    // Reset the game
                    game_started <= 0;
                    game_active <= 0;
                    special_led <= 4'b0000;
                    led_pattern <= 4'b0000;
                    count <= 0;
                    state <= IDLE;
                end
            end
        endcase
    end
end

// LED and Button Logic
assign led = led_pattern;
assign game_over = (state == GAME_OVER);

always @(posedge clk) begin
    if (game_active) begin
        // Generate random LED pattern for columns
        led_pattern <= $random;
    end
end

always @(posedge clk) begin
    if (game_active) begin
        // Increment the count for timing
        count <= count + 1;
        // Increment the count for timing
        count <= count + 1;
    end
end

endmodule
```

```
module testbench_LED_Game;

  // Inputs
  reg clk;
  reg reset;
  reg start_game;
  reg [2:0] button;

  // Outputs
  wire [3:0] led;
  wire game_over;

  // Instantiate the LED_Game module
  LED_Game uut (
    .clk(clk),
    .reset(reset),
    .start_game(start_game),
    .button(button),
    .led(led),
    .game_over(game_over)
  );

  // Clock generation
  always begin
    #10 clk = ~clk; // Toggle the clock every 5 time units (adjust as needed)
  end

  // Initializations
  initial begin

  // Specify the VCD file
  $dumpfile("project.vcd");
```

```
  // Dump the signals you want to monitor
  $dumpvars(0, testbench_LED_Game);

    // Initialize inputs
    clk = 0;
    reset = 1;
    start_game = 0;
    button = 0;

    // Reset the module
    reset = 0;
    #10 reset = 1;

    // Start the game
    start_game = 1;
    #10 start_game = 0;

    // Simulate game
    button = 0;
    #100 button = 1;
    #50 button = 2;
    #60 button = 2;
    #30 button = 3;
    #40 button = 3;

    // Finish the simulation
    $finish;
  end

  // Monitor game_over
  always @(game_over) begin
    if (game_over)
      $display("Game over!");
    else
    else
      $display("Game in progress...");
  end

  // Dump VCD output
  initial begin
    $dumpfile("simulation_results.vcd");
    $dumpvars(0, testbench_LED_Game);
  end

endmodule
```

REFERENCES:

1. https://hackaday.io/project/177594-4017-decade-logic-clock
2. https://www.instructables.com/Digital-Clock-But-Without-a-Microcontroller-Hardco/
3. https://en.wikipedia.org/wiki/555_timer_IC
4. https://www.iqsdirectory.com/articles/electric-switch/push-button-switches.html
5. https://computers.tutsplus.com/how-to-use-a-breadboard-and-build-a-led-circuit-mac-54746t