Dayananda Sagar College of Engineering
Department of Electronics and Communication Engineering
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru – 560 078.
(An Autonomous Institute affiliated to VTU, Approved by AICTE & ISO 9001:2008 Certified)
Accredited by National Assessment and Accreditation Council (NAAC) with 'A' grade

Open Ended Experiment

Course: ADD Laboratory
Course Code: 21EC62

Semester: 6(A1)
Date: 22/06/2024

A Report on

DICE GAME

Submitted by

USN:1DS21EC001 **NAME:** A Himanshu
USN: 1DS21EC006 **NAME:** Abhinav Sundriyal
USN:1DS21EC016 **NAME:** Aditya Pawaskar
USN:1DS21EC020 **NAME:** Ajay HR

Faculty In-charge: Dr. Madhura R

Signature of Faculty In-charge

Introduction

The Dice Game is a simple game that utilizes basic principles of digital design and state machines. The game is based on rolling a pair of dice and evaluating the outcome according to specific rules. The primary goal is to reach a winning state by rolling certain sums, while avoiding losing conditions. This game serves as an excellent example for understanding state machines, as well as for practicing the design and implementation of digital systems using Hardware Description Languages (HDLs) such as Verilog.

In this project, we designed and implemented a Dice Game using Verilog HDL. The game consists of different states including initial state, rolling state, win state, lose state, and a roll again state. Each state transitions based on specific inputs, such as the sum of the dice roll and control signals. The design is tested and verified using a testbench, ensuring that all possible scenarios are accounted for and that the system behaves as expected.

Block Diagram

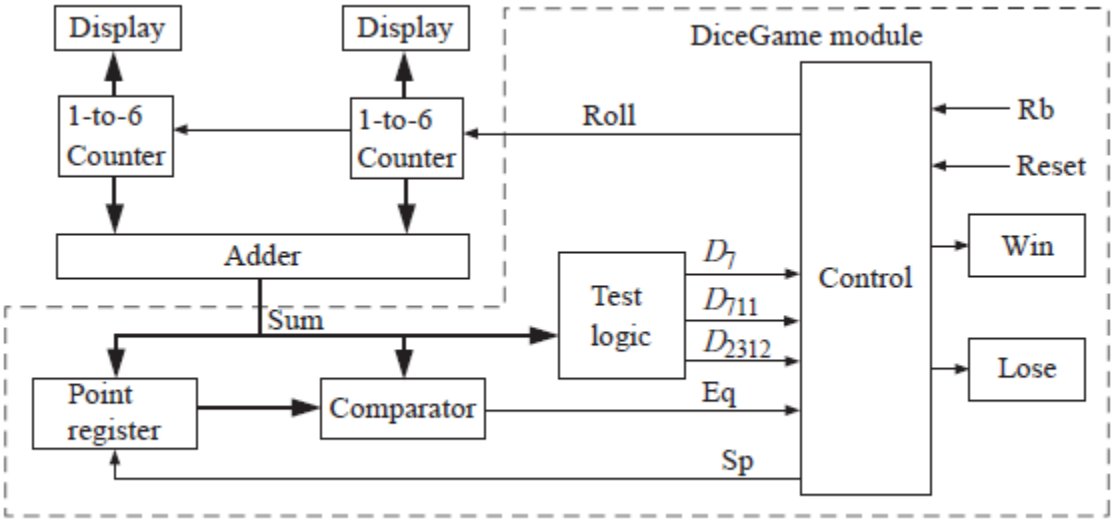


Fig1. Dice Game

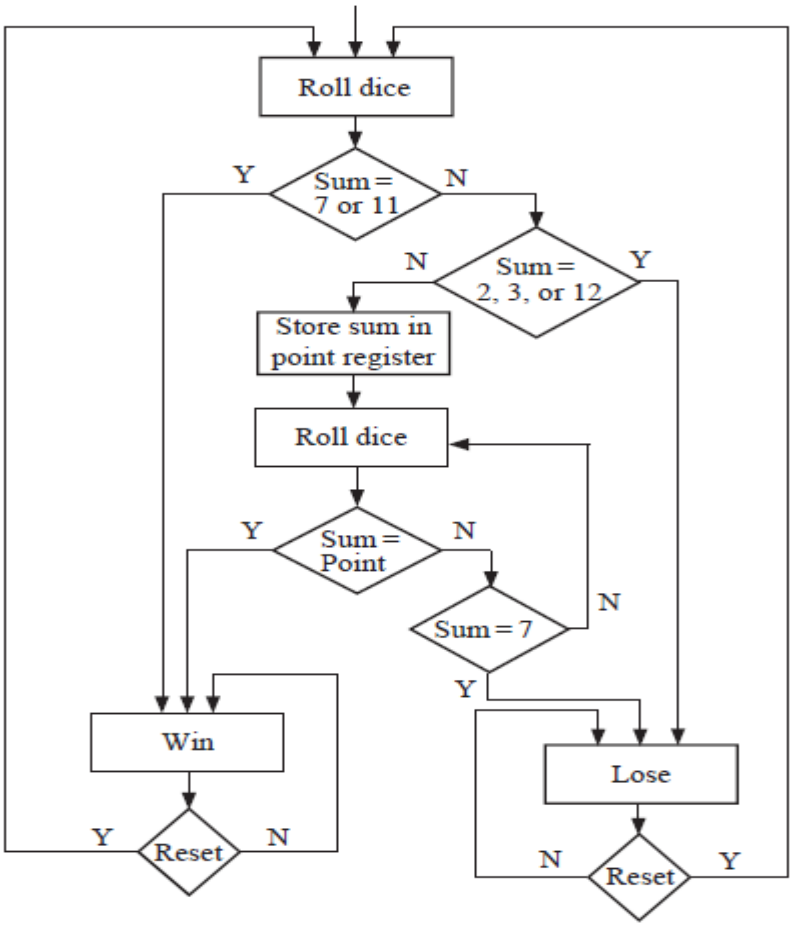


Fig2. Flowchart

The Dice Game is designed using a finite state machine (FSM) approach. The FSM has five primary states:

1. **WAIT**: The initial state where the system waits for the player to press the roll button (Rb).
2. **FIRST_ROLL**: This state represents the first roll of the dice. The outcome of this roll determines the next state.
3. **WIN_STATE**: The player reaches this state if they roll a winning sum (7 or 11) on the first roll or match the point in subsequent rolls.
4. **LOSE_STATE**: The player reaches this state if they roll a losing sum (2, 3, or 12) on the first roll or roll a 7 in subsequent rolls.
5. **ROLL_AGAIN**: If the player rolls any other number on the first roll, this state is activated. The rolled number is set as the point, and the player continues to roll until they either match the point (win) or roll a 7 (lose).

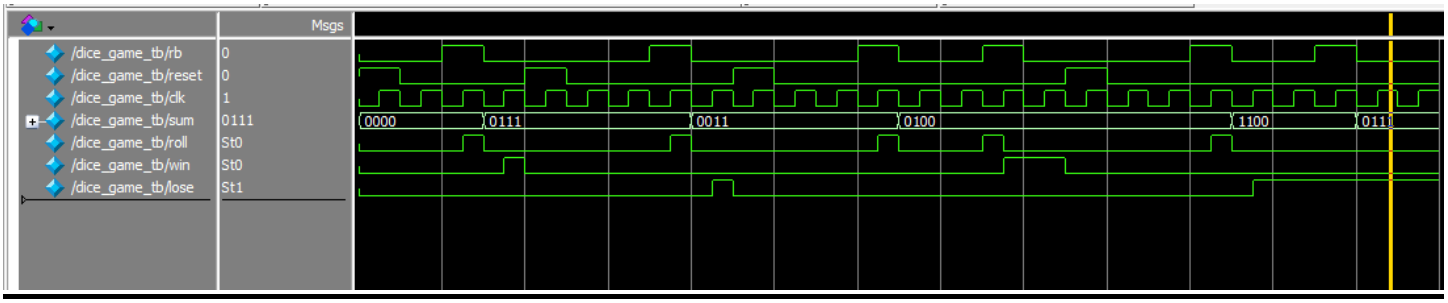
RTL Code

```
module dice_game (
    input rb,
    input reset,
    input clk,
    input [3:0] sum,
    output reg roll,
    output reg win,
    output reg lose
);

    // State declarations
    reg [2:0] state, nextstate;
    reg [3:0] point;
    reg sp;

    parameter WAIT = 3'b000;
    parameter FIRST_ROLL = 3'b001;
    parameter WIN_STATE = 3'b010;
    parameter LOSE_STATE = 3'b011;
    parameter ROLL_AGAIN = 3'b100;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            state <= WAIT;
        end else begin
            state <= nextstate;
        end
    end

    always @(rb or reset or sum or state) begin
        // Default outputs
        sp <= 1'b0;
        roll <= 1'b0;
        win <= 1'b0;
        lose <= 1'b0;

        case (state)
            WAIT: begin
                if (rb) begin
                    nextstate <= FIRST_ROLL;
                end
                else begin
                    nextstate <= WAIT;
                end
            end
            FIRST_ROLL: begin
                if (rb) begin
                    roll <= 1'b1;
                end
                else if (sum == 7 || sum == 11) begin
                    nextstate <= WIN_STATE;
                end
                else if (sum == 2 || sum == 3 || sum == 12) begin
                    nextstate <= LOSE_STATE;
                end
                else begin
                    sp <= 1'b1;
                    nextstate <= ROLL_AGAIN;
                end
            end
            WIN_STATE: begin
                win <= 1'b1;
                if (reset) begin
                    nextstate <= WAIT;
                end
            end
            LOSE_STATE: begin
                lose <= 1'b1;
                if (reset) begin
                    nextstate <= WAIT;
                end
            end
            ROLL_AGAIN: begin
                if (rb) begin
                    roll <= 1'b1;

                    if (sum == point) begin
                        nextstate <= WIN_STATE;
                    end
                    else if (sum == 7) begin
                        nextstate <= LOSE_STATE;
                    end
                    else begin
                        nextstate <= ROLL_AGAIN;
                    end
                end
            end
        endcase
    end

    always @(posedge clk) begin
        if (sp) begin
            point <= sum;
        end
    end

endmodule

```

Testbench

```
module dice_game_tb;

    reg rb;
    reg reset;
    reg clk;
    reg [3:0] sum;
    wire roll;
    wire win;
    wire lose;

    dice_game uut (
        .rb(rb),
        .reset(reset),
        .clk(clk),
        .sum(sum),
        .roll(roll),
        .win(win),
        .lose(lose)
    );

    always #5 clk = ~clk;

    initial begin
        $monitor("At time %t: rb = %b, reset = %b, clk = %b, sum = %d, roll = %b, win = %b, lose = %b",$time, rb, reset, clk, sum, roll, win, lose);

        //Initialize
        clk = 0;
        rb = 0;
        reset = 1;
        sum = 4'd0;

        #10 reset = 0;

        //case 1
        #10 rb = 1;
        #10 rb = 0; sum = 4'd7; //win

        #10 reset=1;
        #10 reset=0;

        //case 2
        #20 rb = 1;
        #10 rb = 0; sum = 4'd3; //lose

        #10 reset=1;
        #10 reset=0;

        // Test case 3
        #20 rb = 1;
        #10 rb = 0; sum = 4'd4; //go to ROLL_AGAIN state
        #20 rb = 1;
        #10 rb = 0; sum = 4'd4; //win

        #10 reset=1;
        #10 reset=0;

        // Test case 4
        #20 rb = 1;
        #10 rb = 0; sum = 4'd12; //go to ROLL_AGAIN state
        #20 rb = 1;
        #10 rb = 0; sum = 4'd7; //lose

        #20 $finish;
    end
endmodule
```

Simulation Results




```
Transcript
VSIM 40> run -all
# At time                     0: rb = 0, reset = 1, clk = 0, sum =  0, roll = 0, win = 0, lose = 0
# At time                  5000: rb = 0, reset = 1, clk = 1, sum =  0, roll = 0, win = 0, lose = 0
# At time                 10000: rb = 0, reset = 0, clk = 0, sum =  0, roll = 0, win = 0, lose = 0
# At time                 15000: rb = 0, reset = 0, clk = 1, sum =  0, roll = 0, win = 0, lose = 0
# At time                 20000: rb = 1, reset = 0, clk = 0, sum =  0, roll = 0, win = 0, lose = 0
# At time                 25000: rb = 1, reset = 0, clk = 1, sum =  0, roll = 1, win = 0, lose = 0
# At time                 30000: rb = 0, reset = 0, clk = 0, sum =  7, roll = 0, win = 0, lose = 0
# At time                 35000: rb = 0, reset = 0, clk = 1, sum =  7, roll = 0, win = 1, lose = 0
# At time                 40000: rb = 0, reset = 1, clk = 0, sum =  7, roll = 0, win = 0, lose = 0
# At time                 45000: rb = 0, reset = 1, clk = 1, sum =  7, roll = 0, win = 0, lose = 0
# At time                 50000: rb = 0, reset = 0, clk = 0, sum =  7, roll = 0, win = 0, lose = 0
# At time                 55000: rb = 0, reset = 0, clk = 1, sum =  7, roll = 0, win = 0, lose = 0
# At time                 60000: rb = 0, reset = 0, clk = 0, sum =  7, roll = 0, win = 0, lose = 0
# At time                 65000: rb = 0, reset = 0, clk = 1, sum =  7, roll = 0, win = 0, lose = 0
# At time                 70000: rb = 1, reset = 0, clk = 0, sum =  7, roll = 0, win = 0, lose = 0
# At time                 75000: rb = 1, reset = 0, clk = 1, sum =  7, roll = 1, win = 0, lose = 0
# At time                 80000: rb = 0, reset = 0, clk = 0, sum =  3, roll = 0, win = 0, lose = 0
# At time                 85000: rb = 0, reset = 0, clk = 1, sum =  3, roll = 0, win = 0, lose = 1
# At time                 90000: rb = 0, reset = 1, clk = 0, sum =  3, roll = 0, win = 0, lose = 0
# At time                 95000: rb = 0, reset = 1, clk = 1, sum =  3, roll = 0, win = 0, lose = 0
# At time                100000: rb = 0, reset = 0, clk = 0, sum =  3, roll = 0, win = 0, lose = 0
# At time                105000: rb = 0, reset = 0, clk = 1, sum =  3, roll = 0, win = 0, lose = 0

# At time                185000: rb = 0, reset = 0, clk = 1, sum =  4, roll = 0, win = 0, lose = 0
# At time                190000: rb = 0, reset = 0, clk = 0, sum =  4, roll = 0, win = 0, lose = 0
# At time                195000: rb = 0, reset = 0, clk = 1, sum =  4, roll = 0, win = 0, lose = 0
# At time                200000: rb = 1, reset = 0, clk = 0, sum =  4, roll = 0, win = 0, lose = 0
# At time                205000: rb = 1, reset = 0, clk = 1, sum =  4, roll = 1, win = 0, lose = 0
# At time                210000: rb = 0, reset = 0, clk = 0, sum = 12, roll = 0, win = 0, lose = 0
# At time                215000: rb = 0, reset = 0, clk = 1, sum = 12, roll = 0, win = 0, lose = 1
# At time                220000: rb = 0, reset = 0, clk = 0, sum = 12, roll = 0, win = 0, lose = 1
# At time                225000: rb = 0, reset = 0, clk = 1, sum = 12, roll = 0, win = 0, lose = 1
# At time                230000: rb = 1, reset = 0, clk = 0, sum = 12, roll = 0, win = 0, lose = 1
# At time                235000: rb = 1, reset = 0, clk = 1, sum = 12, roll = 0, win = 0, lose = 1
# At time                240000: rb = 0, reset = 0, clk = 0, sum =  7, roll = 0, win = 0, lose = 1
# At time                245000: rb = 0, reset = 0, clk = 1, sum =  7, roll = 0, win = 0, lose = 1
# At time                250000: rb = 0, reset = 0, clk = 0, sum =  7, roll = 0, win = 0, lose = 1
# At time                255000: rb = 0, reset = 0, clk = 1, sum =  7, roll = 0, win = 0, lose = 1
# ** Note: $finish    : D:/VS_Code/Verilog_folder/College_lab/6th sem/dice_game_tb.v(65)
#    Time: 260 ns  Iteration: 0  Instance: /dice_game_tb
```

1. **Initialization Phase**:
    a. At time 0: The reset is high, initializing the system. rb is low, and clk is 0.
    b. At time 10: The reset is deasserted, and the system is ready for operation.
2. **Case 1: Immediate Win**:
    a. At time 20: rb is asserted (1) to indicate the start of a new game.
    b. At time 30: rb is deasserted, and sum is set to 7. The system enters FIRST_ROLL state, detects sum of 7, and moves to WIN_STATE.
    c. win signal goes high, indicating a win.

3. **Case 2: Immediate Lose**:
    a. At time 70: rb is asserted again for a new game.
    b. At time 80: rb is deasserted, and sum is set to 3. The system enters FIRST_ROLL state, detects sum of 3, and moves to LOSE_STATE.
    c. Lose signal goes high, indicating a loss.
4. **Case 3: Roll Again and Win**:
    a. At time 120: rb is asserted for a new game.
    b. At time 130: rb is deasserted, and sum is set to 4. The system enters FIRST_ROLL state, detects sum of 4, and moves to ROLL_AGAIN state, storing sum as point.
    c. At time 150: rb is asserted again.
    d. At time 160: rb is deasserted, and sum is set to 4 again, which matches point. The system moves to WIN_STATE.
    e. win signal goes high, indicating a win.
5. **Case 4: Roll Again and Lose**:
    a. At time 200: rb is asserted for a new game.
    b. At time 210: rb is deasserted, and sum is set to 12. The system enters FIRST_ROLL state, detects sum of 12, and moves to ROLL_AGAIN state, storing sum as point.
    c. At time 230: rb is asserted again.
    d. At time 240: rb is deasserted, and sum is set to 7, which is a losing condition in ROLL_AGAIN state. The system moves to LOSE_STATE.
    e. Lose signal goes high, indicating a loss.

Results

The Dice Game Verilog module successfully simulates the classic game of Craps, implementing the game's state transitions and win/lose conditions accurately. The testbench effectively verifies all critical scenarios, demonstrating that the module behaves as expected under various conditions.