ELL201 Project Report

Tejasraj Mangla - 2022AM11815
Karan Noor Singh Sarao - 2022AM11220
Yash Kumar - 2022EE31196
Shweta Tiwari - 2022AM11821

1. Idea

The idea behind this project is to create a digital simulation of a classic memory matching game. In a physical memory matching game, players typically arrange a set of cards face down and take turns flipping two cards over at a time. The goal is to find matching pairs of cards by remembering their positions when they are flipped.

In this digital version we implemented using Verilog, the same concept is replicated. The memory game module simulates the state of each card (face up or face down) using binary values, where 1 represents a face-up card and 0 represents a face-down card. The game logic allows players to flip cards and checks for matches when two cards are flipped.

Key components of the project include:
1. Memory Game Module
   - Handles the game logic, including flipping cards, detecting matches, and updating the state of each card.
   - Utilizes registers to store information about the state of each card and to track the IDs of cards being flipped and matched.
   - Responds to inputs such as clock signals, resets, and commands to flip cards.

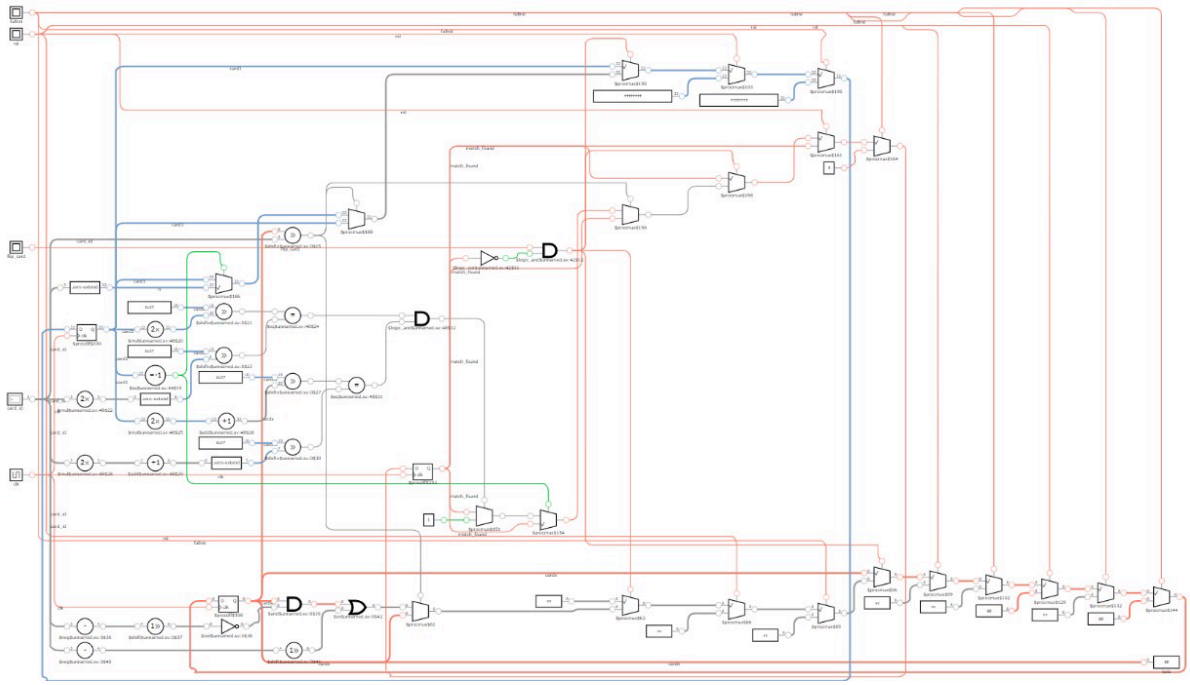
2. Testbench Module
   - Provides a simulated environment to test the functionality of the memory game module.
   - Generates simulated input signals (such as clock pulses and commands to flip cards) and observes the output (LEDs representing the state of each card).
   - Allows for verification of the game's behavior under different scenarios and conditions.

Overall, our project aims to create a digital version of a memory matching game that can be implemented on hardware platforms such as FPGAs or tested in simulation environments. It demonstrates concepts of digital logic design, state machines, and sequential logic, while providing an entertaining and interactive experience reminiscent of the classic memory game.

2. Background

The card order is preset in input through the register “cardss” and card number to be flipped is given using “card_id” which varies from 0-7 representing 8 cards in this implementation additionally input wire “flip_card” acts as a switch to give input, the opening and closing of cards is demonstrated through output “leds”. The input “rst” is used to close the cards after every turn, while “fullrst” is used after a win to restart.
The game is started by first deciding the “card_id” and then “flip_card” switch is used to flip the card, led turns on represents the opening of first card similarly chose the second card.Both cards are closed using “rst” , the win is decided in the next flip once the other player choses a card and led don’t turn on that represents a win in last round, thus players try to match the cards until a flip doesn’t turn on the led of a chosen card. After a win the game can be restarted using “fullrst”.

3. Circuit Diagram



4. Truth Table

Truth table for a specific case when card order is **0,0,3,2,3,1,1,2**
We can see that once card_id 0 and 1 (both being 0 card number) are matched, the leds stay turned off and the game is over

Terminal

```
Card order: 1001011110110000
    card_id = xxx, leds = 00000000
    card_id = 001, leds = 00000000
    card_id = 001, leds = 00000010
    card_id = 111, leds = 00000010
    card_id = 111, leds = 10000010
card_id = 111, leds = 00000000
    card_id = 010, leds = 00000000
    card_id = 010, leds = 00000100
    card_id = 001, leds = 00000100
    card_id = 001, leds = 00000110
    card_id = 001, leds = 00000000
    card_id = 000, leds = 00000000
    card_id = 000, leds = 00000001
    card_id = 001, leds = 00000001
    card_id = 001, leds = 00000011
    card_id = 001, leds = 00000000
    card_id = 111, leds = 00000000
    card_id = 010, leds = 00000000
main.v:154: $finish called at 160 (1s)
```

Since our game has 8 possible initial inputs and 7 possible subsequent inputs, And it loops if cards don't match. Furthermore, the truth table will change as we change the underlying set of cards in the code (which can be done to change the answers). Therefore, a more than 56-row state table was unfeasible to add here.

Hence, we included a part of the table which can be extrapolated instead.

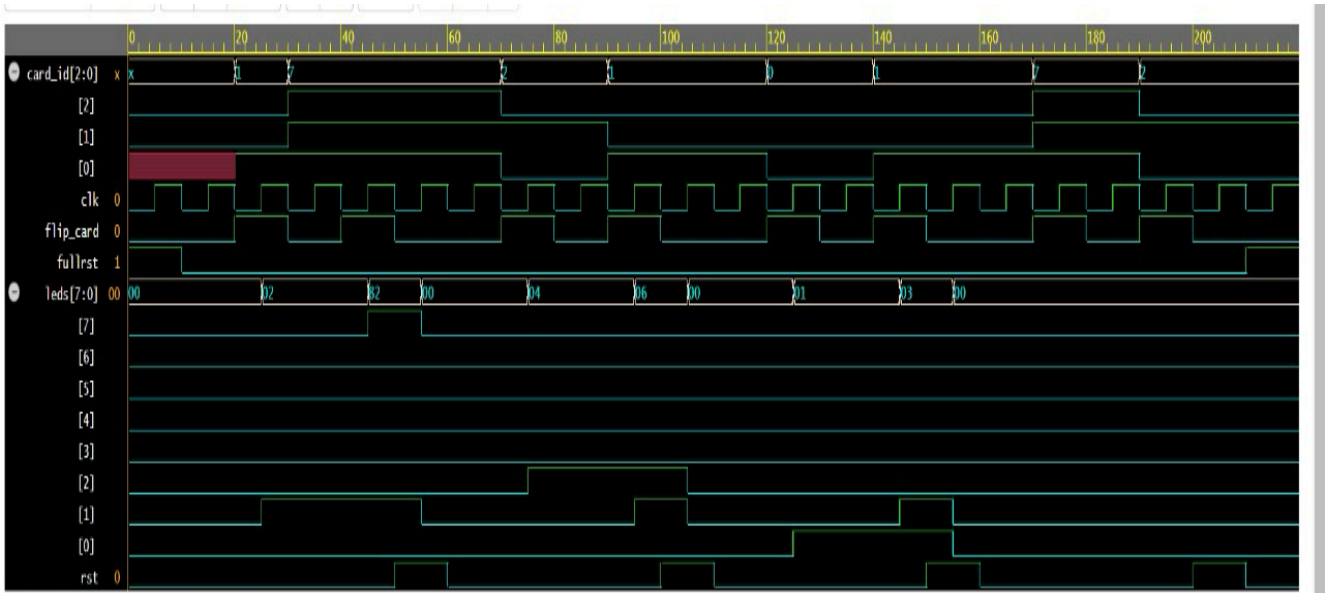
5. Verilog code

```
module memory_game (
    input wire clk,          // Clock input
    input fullrst,
    input wire rst,          // Reset input
    input wire flip_card,
    input wire [2:0] card_id,
    output wire [7:0] leds
);
 reg [15:0] cardss;
reg [7:0] cards;            // State of each card (1 for face up, 0 for face down)
integer card1, card2;     // IDs of the two cards being flipped
reg [1:0] match_id;         // IDs of the two matched cards
reg match_found;            // Flag indicating if a match has been found
integer n;                  // Integer to store decimal form of card_id

// Initialize variables
initial begin
    cardss = 16'b1001011110110000;
    cards = 8'b00000000;
    card1 = -1;         // Invalid card ID
    card2 = -1;         // Invalid card ID
    match_id = 2'b00;       // No match initially
    match_found = 0;
end

// Convert card_id to decimal form
always @* begin
    n = card_id;
end

always @(posedge clk) begin
    if (fullrst) begin
        cards <= 8'b00000000; // Reset cards to face down
        card1 <= -1;    // Reset card IDs
        card2 <= -1;
        match_id <= 2'b00;    // Reset match ID
        match_found <= 0;     // Reset match found flag
    end else if (rst) begin
        cards <= 8'b00000000; // Reset cards to face down
        card1 <= -1;      // Reset card IDs
        card2 <= -1;
    end else if (flip_card && !match_found) begin
        if (cards[card_id] == 1'b0) begin // Flip the card if it's face down
            if (card1 == -1) begin    // First card flip
                card1 <= n;
            end else begin                // Second card flip
                card2 = n;
                if (cardss[2*card1]==cardss[2*card2] &&
cardss[2*card1+1]==cardss[2*card2+1]) begin
                    // Cards match
                    match_id <= {cardss[card1+1],cardss[card1]};
                    match_found <= 1; // Set match found flag
                end
```

```
            end
            cards[card_id] <= 1'b1; // Update card state to face up
        end
    end
end

// Assign LEDs to represent the state of each card
assign leds = cards;

endmodule

module memory_game_tb;

    reg clk;
    reg fullrst;
    reg rst;
    reg flip_card;
    reg [2:0] card_id;
    wire [7:0] leds;

    // Instantiate the memory_game module
    memory_game dut (
        .clk(clk),
        .fullrst(fullrst),
        .rst(rst),
        .flip_card(flip_card),
        .card_id(card_id),
        .leds(leds)
    );

    // Clock generation
    always begin
        #5 clk = ~clk;
    end

    // Testbench
    initial begin
        // Initialize inputs
        clk = 0;
        fullrst = 0;
        rst = 0;
        flip_card = 0;

      $dumpfile("dump.vcd");
      $dumpvars(1);
        // Apply reset
        fullrst = 1;
        #10 fullrst = 0;

        // First two flips
        #10 flip_card = 1;card_id = 3'b001;
        #10 flip_card=0;
        card_id = 3'b111;
        #10 flip_card = 1;
        #10 begin
```

```
            flip_card = 0;
            // Display LED output
             $display("LEDs: %b Card ID: %b", leds, card_id);
            rst = 1;
            #10 rst = 0;
        end
        #10 flip_card = 1;card_id = 3'b010;
        #10 flip_card = 0;
        #10 flip_card = 1;card_id = 3'b001;
        #10 begin
            flip_card = 0;
            // Display LED output
            $display("LEDs: %b Card ID: %b", leds, card_id);
            rst=1;
            #10 rst=0;
        end
        #10 flip_card = 1;card_id = 3'b000;
        #10 flip_card = 0;
        #10 flip_card = 1;card_id = 3'b001;
        #10 begin
            flip_card = 0;
            // Display LED output
             $display("LEDs: %b Card ID: %b", leds, card_id);
            rst=1;
            #10 rst=0;
        end

        #10 flip_card = 1;card_id = 3'b111;
        #10 flip_card = 0;
        #10 flip_card = 1;card_id = 3'b010;
        #10 begin
            flip_card = 0;
            // Display LED output
             $display("LEDs: %b Card ID: %b", leds, card_id);
            rst=1;
            #10 rst=0;
        end

        // Full reset
        fullrst = 1;
        #10 begin
            fullrst = 0;
            // Display LED output
             $display("LEDs: %b Card ID: %b", leds, card_id);
        end

        // End simulation
        $finish;
    end

endmodule
```

6. TestBench Waveforms



CPLD Mapping:

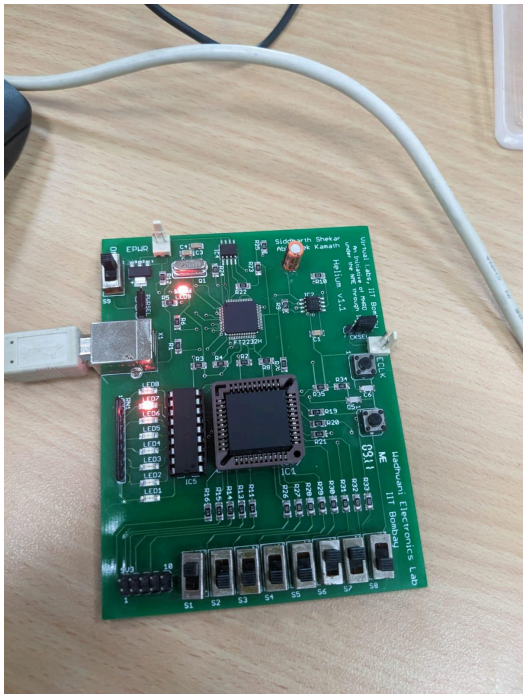
LEDs correspond to their respective cards
0,1,2,3,4,5,6,7 to ***0,0,3,2,3,1,1,2***

Switches S1, S2, S3 are flip_card (choose input card), rst (save both choices), and full_rst (restart), respectively;

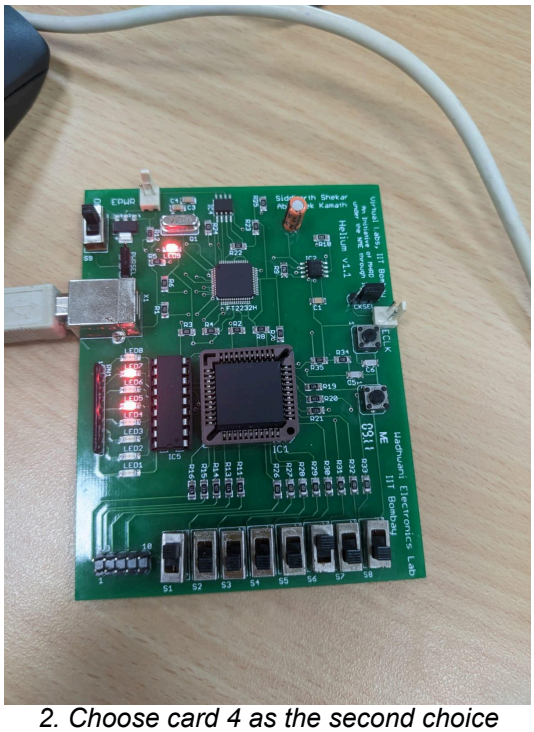
Switches S6, S7, S8 are 4, 2 and 1 whose sum is used to choose the actual card (binary inputs)

7. CPLD Demonstration

CASE 1: No-Match (Led stays on after round)



1. Choose card 6 as first choice



2. Choose card 4 as the second choice

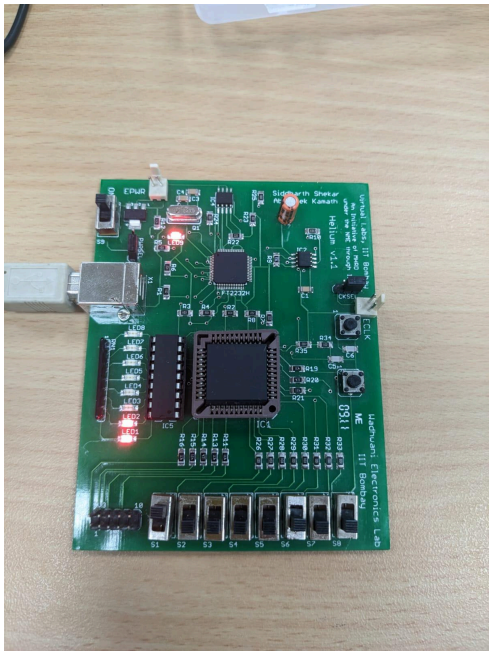
3. Reset to save your choices

4. Choose card 0 to check if cards 4 & 6 matched

Since card 4 is '2' and card 6 is '1', card 0's LED turns on, indicating a match wasn't found.

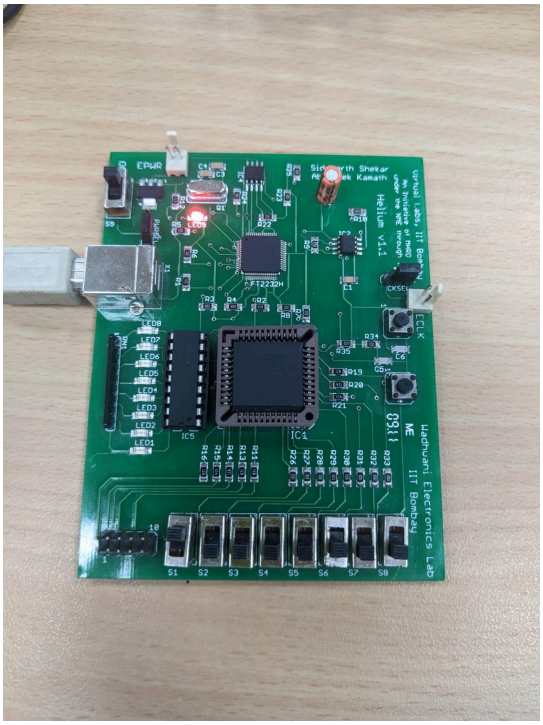
CASE 2: Match (Led stays off to indicate game over)

1. *Choose card 1 as first choice*



2. Choose card 0 as the second choice

3. Reset to save your choices



4. Choose card 0 to check if cards 0 & 1 matched

Since card 0 is '0' and card 1 is '0', card 0's LED remains off, indicating a match was found and the game is over.