Final Project: Memory Game on Basys3

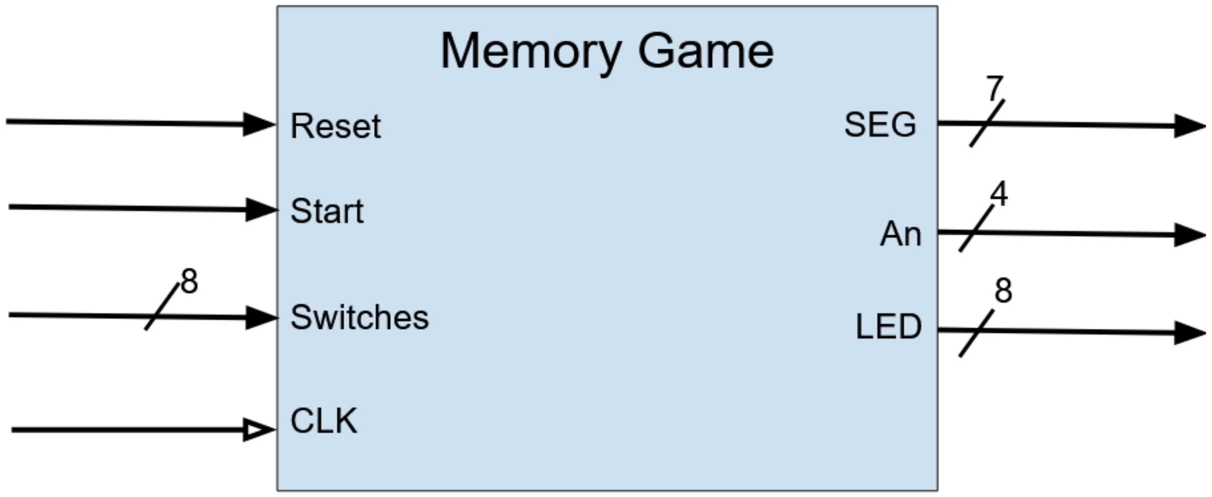
Cal Poly San Luis Obispo

CPE 133-03: Digital Design

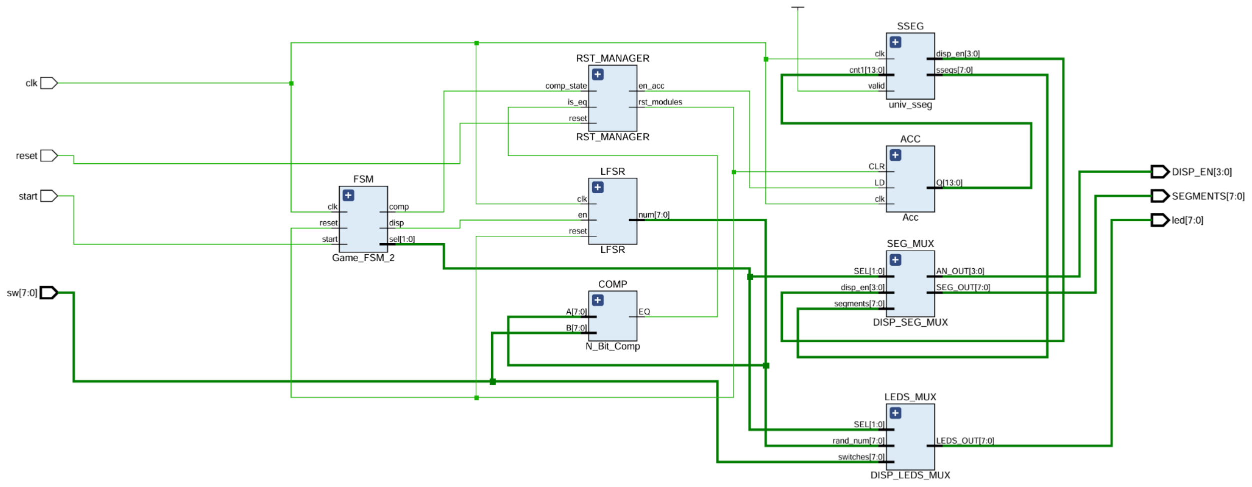
7 June 2024

Diagrams

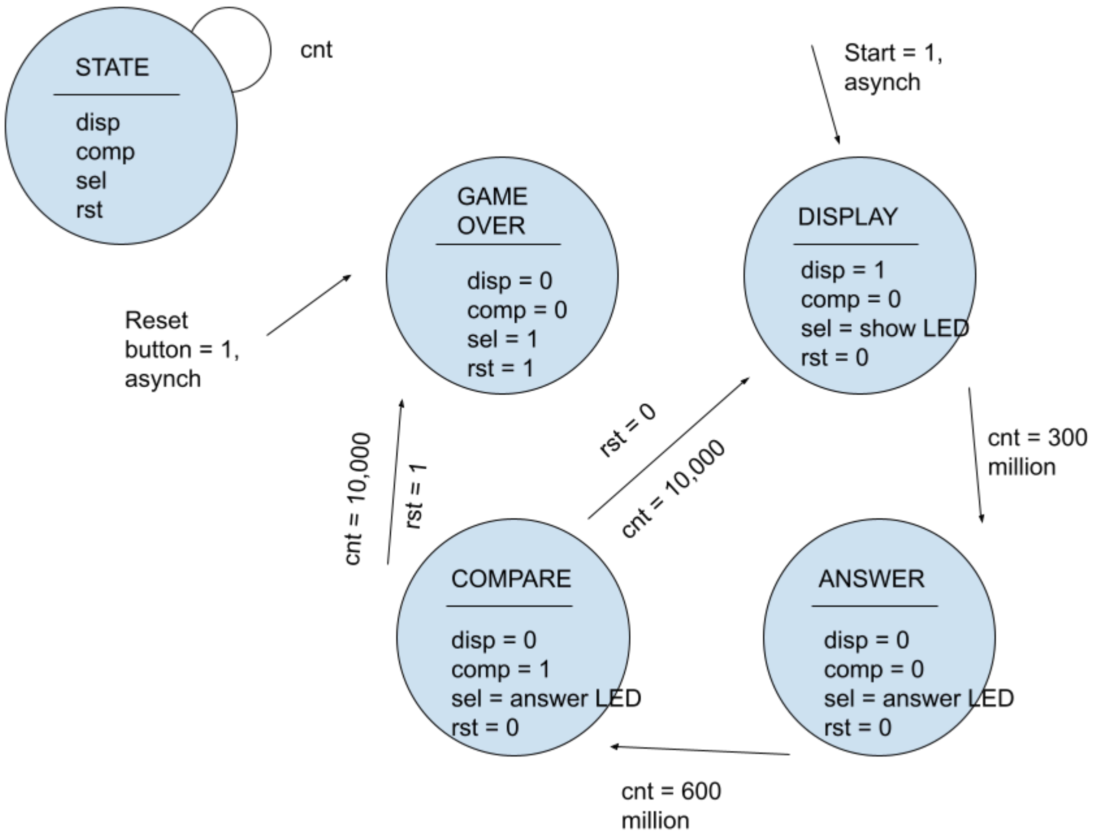
High-Level Black Box Diagram



Low Level Diagram



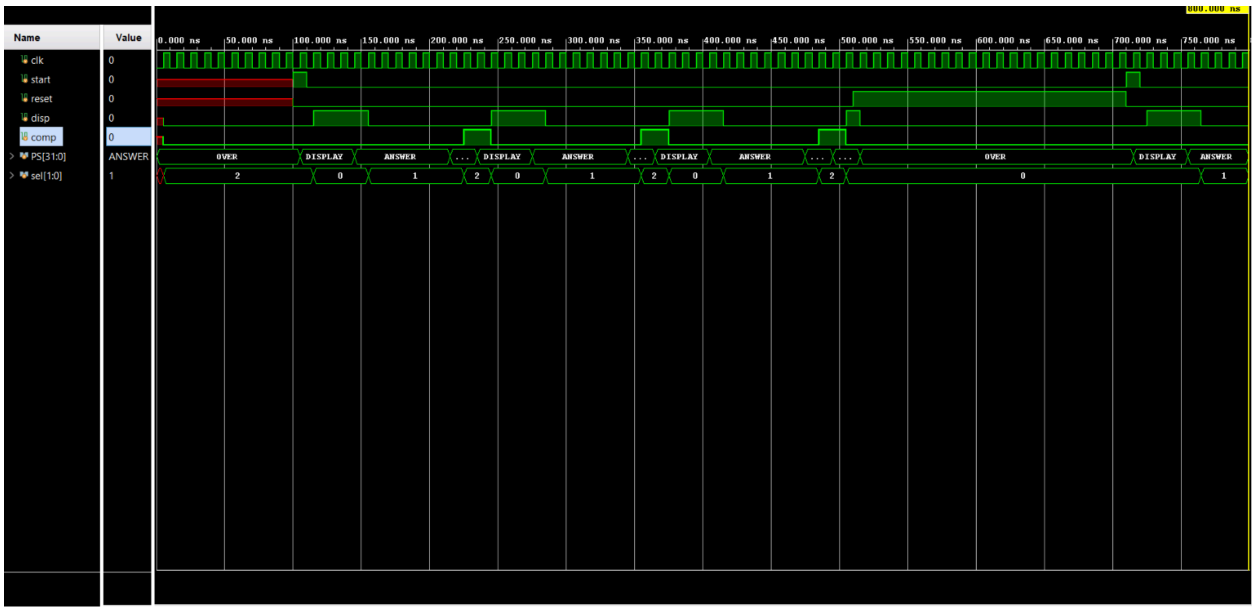
State Diagram



Simulation of FSM Module
Simulation File Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Create Date: 04/11/2024 01:49:48 PM
// Design Name:
// Module Name: TestBench
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module FSMsim();
	logic clk;//10 ns cycle clock
	logic start;
	logic reset;
	logic disp;//should it display the pattern
	logic comp;//should it compare the answer with the pattern
	logic[1:0] sel;//manages display elements
	Game_FSM_2 fail (.clk(clk), .start(start), .reset(reset), .disp(disp), .comp(comp), .sel(sel));

	initial begin
	clk = 0;
	end

	always begin
	#5 clk = ~clk;
	end

	always begin  //8 is biggest value it can display
	#100 start = 1'b1; reset = 1'b0;
	#10 start = 1'b0;
	#400 reset = 1'b1;
	#100;
	end
endmodule
```

Simulation Timing Diagram



New and Modified Code

Top Level

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Create Date: 06/04/2024 12:17:34 PM
// Design Name:
// Module Name: toplevel
// Project Name: cpe 133 final
// Target Devices:
// Tool Versions:
// Description: top level module of memory game
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//left button is reset, right is start
module toplevel(
        input [7:0] sw,
        input start,
        input reset,
        input clk,
        output [7:0] SEGMENTS,
        output [3:0] DISP_EN,
        output [7:0] led
        );
        logic[7:0] t1;
        logic[1:0] t2;
        logic t3;
        logic t4;
        logic t5;
        logic t6;
        logic[13:0] t7;
        logic[3:0] t8;
        logic[7:0] t9;
        logic t10;
        N_Bit_Comp COMP (.B(sw), .A(t1), .EQ(t3));

        //start is in here
        Game_FSM_2 FSM (.clk(clk), .start(start), .reset(t6), .disp(t5), .comp(t4), .sel(t2));

        LFSR LFSR (.en(t5), .reset(t6), .clk(clk), .num(t1));

        DISP_LEDS_MUX MUX1 (.switches(sw), .rand_num(t1), .SEL(t2), .LEDS_OUT(led));

        DISP_SEG_MUX MUX2 (.SEL(t2), .segments(t9),. disp_en(t8), .SEG_OUT(SEGMENTS), .AN_OUT(DISP_EN));

        univ_sseg SSEG (.cnt1({t7}), .ssegs(t9), .disp_en(t8), .clk(clk), .valid(1'b1));

        Acc ACC (.CLR(t6), .LD(t10), .Q(t7), .clk(clk));
        //reset is down here
        RST_MANAGER RST (.is_eq(t3), .comp_state(t4), .reset(reset), .en_acc(t10), .rst_modules(t6));
        endmodule
```

Reset Manager

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
```

```
// Create Date: 05/30/2024 02:23:14 PM
// Design Name:
// Module Name: RST_MANAGER
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module RST_MANAGER(
        input is_eq,
        input comp_state,
        input reset,
        output en_acc,
        output rst_modules
        );

        //assigns the enable signal for the accumulator
        assign en_acc = is_eq&comp_state;
        //assigns the signal that resets all the modules
        assign rst_modules = (comp_state&~is_eq)|reset;
endmodule
```

Modified Benson Accumulator

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Create Date: 6/6/2024 03:08:43 PM
// Description: 8-bit accumulator. Adds new value to the
// current value when LD is 1.
//////////////////////////////////////////////////////////////////////////////////
```

```
module Acc(
        input clk,              // Clock signal
        input LD,     // Load signal
        input CLR,    // Clear signal
        output logic [13:0] Q = 0  // 14-bit output register
                //Removed D variable as 10 is added every enable instead of a variable input
        );

        logic LD_prev, CLR_prev;

        always_ff @ (posedge clk) begin
        LD_prev <= LD;
        CLR_prev <= CLR;
        end

        always_ff @ (posedge clk) begin
        if (CLR & ~CLR_prev)  // Rising edge of CLR
        Q <= 0;
        else if (LD & ~LD_prev)  // Rising edge of LD as to only add once every enable instead of every clk cycle
        Q <= Q + 14'd10;
        end
endmodule
```

Finite State Machine

```
 `timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Create Date: 05/30/2024 07:23:22 PM
// Design Name:
// Module Name: Game_FSM_2
// Project Name:
```

```
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module Game_FSM_2(
    input clk,           // 10 ns cycle clock
    input start,         // Start signal to begin the game
    input reset,         // Reset signal to reset the FSM
    output logic disp,    // Output to control display of the pattern
    output logic comp,    // Output to control comparison of the answer with the pattern
    output logic [1:0] sel// Output to manage display elements
  );

  // Define the FSM states
  typedef enum {OVER, DISPLAY, ANSWER, COMPARE} STATES;
  STATES PS = OVER; // Initialize the present state to OVER

  // Local parameters for the states of the select variable
  localparam logic [1:0] over_leds = 2'b10;
  localparam logic [1:0] pattern_leds = 2'b00; // Connects to the random number generator
  localparam logic [1:0] show_leds = 2'b01; // Shows current switch state as LEDs

  // Counter and clock cycle thresholds
  localparam DISPLAY_CYCLES = 300_000_000; // Number of cycles to display the pattern
  localparam ANSWER_CYCLES = 600_000_000; // Number of cycles to allow the user to answer
  localparam COMPARE_CYCLES = 10_000; // Number of cycles to compare the answer with the pattern

  // Parameters for testing (uncomment for testing)
//   localparam int DISPLAY_CYCLES = 3;
//   localparam int ANSWER_CYCLES = 6; // example value, adjust as needed
//   localparam int COMPARE_CYCLES = 1; // example value, adjust as needed

  logic [31:0] counter = 32'b0; // Counter variable

  // Sequential logic block triggered on the positive edge of the clock
  always_ff @(posedge clk)
    begin
      if (reset == 1'b1) begin
        // Reset the FSM and outputs
        counter <= 32'b0;
        disp <= 1'b0;
        comp <= 1'b0;
        PS <= OVER;
      end else if (start == 1'b1) begin
        // Start the FSM and initialize for DISPLAY state
        counter <= 32'b0;
        disp <= 1'b0;
        comp <= 1'b0;
        PS <= DISPLAY;
      end else begin
        // FSM behavior based on the current state
        case(PS)
            OVER:
            begin
                // State when game is over or reset
                disp <= 1'b0;
                comp <= 1'b0;
                sel <= over_leds;
                counter <= 32'b0;
            end

            DISPLAY:
            begin
                // State to display the pattern
                disp <= 1'b1;
                comp <= 1'b0;
                sel <= pattern_leds;
                if (counter >= DISPLAY_CYCLES) begin
                    PS <= ANSWER;
                    counter <= 32'b0;
                end else begin
                    PS <= DISPLAY;
                    counter <= counter + 1;
                end
            end

            ANSWER:
            begin
                // State to allow user to answer
                disp <= 1'b0;
                comp <= 1'b0;
                sel <= show_leds;
                if (counter >= ANSWER_CYCLES) begin
                    PS <= COMPARE;
                    counter <= 32'b0;
                end else begin
                    PS <= ANSWER;
                    counter <= counter + 1;
                end
            end
```

```
                COMPARE:
                begin
                    // State to compare the user's answer with the pattern
                    disp <= 1'b0;
                    comp <= 1'b1;
                    sel <= over_leds;
                    if (counter >= COMPARE_CYCLES) begin
                        PS <= DISPLAY;
                        counter <= 32'b0;
                    end else begin
                        PS <= COMPARE;
                        counter <= counter + 1;
                    end
                end

                default:
                begin
                    // Default state
                    PS <= OVER;
                    counter <= 32'b0;
                    disp <= 1'b0;
                    comp <= 1'b0;
                    sel <= 2'b10;
                end
            endcase
        end
    end
endmodule
```