Lab Code [15 points]
Filename: ChipInterface.sv

```
`default_nettype none

module ChipInterface(
  input  logic [7:0]  KEY,
  input  logic [17:0] SW,
  input  logic        CLOCK_50,
  output logic [6:0]  HEX9, HEX8, HEX7, HEX6, HEX5, HEX3, HEX2, HEX1, HEX0,
  output logic [7:0]  LEDG);

  logic [3:0] hMove, cMove;
  logic [15:0] HHEX, CHEX;
  logic win, clock, reset, reset_N, enter, newGame;
  logic [6:0] segment0, segment1, segment2, segment3, segment4, segment5,
              segment6, segment7, segment8;

  logic newGame_int, newGame_sync, enter_int, enter_sync;

  assign clock = CLOCK_50;
  assign reset = SW[17];
  assign reset_N = ~reset;
  assign hMove = SW[3:0];
  assign enter = ~KEY[3];
  assign newGame = ~KEY[0];

  assign HEX5 = segment0;
  assign HEX3 = segment1;
  assign HEX2 = segment2;
  assign HEX1 = segment3;
  assign HEX0 = segment4;
  assign HEX9 = segment5;
  assign HEX8 = segment6;
  assign HEX7 = segment7;
  assign HEX6 = segment8;
  assign LEDG[7:0] = {8{win}};

  BCDtoSevenSegment BSS0 (.bcd(cMove), .segment(segment0)),
                    BSS1 (.bcd(CHEX[15:12]), .segment(segment1)),
                    BSS2 (.bcd(CHEX[11:8]), .segment(segment2)),
                    BSS3 (.bcd(CHEX[7:4]), .segment(segment3)),
                    BSS4 (.bcd(CHEX[3:0]), .segment(segment4)),
                    BSS5 (.bcd(HHEX[15:12]), .segment(segment5)),
                    BSS6 (.bcd(HHEX[11:8]), .segment(segment6)),
                    BSS7 (.bcd(HHEX[7:4]), .segment(segment7)),
                    BSS8 (.bcd(HHEX[3:0]), .segment(segment8));
  abstractFSM fsm (.hMove, .valid(enter_sync), .newGame(newGame_sync), .cMove,
                   .HHEX, .CHEX, .win, .clock, .reset_N);

  always_ff @(posedge clock)
    newGame_int <= newGame;

  always_ff @(posedge clock)
    newGame_sync <= newGame_int;

  always_ff @(posedge clock)
    enter_int <= enter;

  always_ff @(posedge clock)
    enter_sync <= enter_int;

endmodule: ChipInterface

module BCDtoSevenSegment
    (input logic [3:0] bcd,
    output logic [6:0] segment);

    always_comb
        case (bcd)
            4'b0001: segment = 7'b111_1001;
            4'b0010: segment = 7'b010_0100;
            4'b0011: segment = 7'b011_0000;
            4'b0100: segment = 7'b001_1001;
            4'b0101: segment = 7'b001_0010;
            4'b0110: segment = 7'b000_0010;
            4'b0111: segment = 7'b111_1000;
            4'b1000: segment = 7'b000_0000;
            4'b1001: segment = 7'b001_1000;
            default: segment = 7'b111_1111; // Display is off by default
        endcase
endmodule: BCDtoSevenSegment
```

Lab Code [15 points]
Filename: lab4.sv

```
`default_nettype none

module abstractFSM(
  input  logic [3:0] hMove,
  input  logic valid,
  input  logic newGame,
  output logic [3:0] cMove,
  output logic [15:0] HHEX,
  output logic [15:0] CHEX,
  output logic       win,
  input  logic       clock, reset_N
);

  enum logic [4:0] {new_game, new_game_int,
                    cStart, cStart_int,
                    first, first_int,
                    second,
                    win12, win12_int,
                    win13, win13_int,
                    win14, win14_int,
                    win17, win17_int,
                    win18, win18_int,

                    win22, win22_int,
                    win24, win24_int,
                    win27, win27_int,
                    win28, win28_int}
    currState, nextState;

  // Next State Generator
  always_comb begin
    case (currState)
      new_game: nextState= (valid && newGame) ? new_game_int : new_game;
      new_game_int: nextState= (~valid && ~newGame) ? cStart : new_game_int;

      cStart: nextState = (valid && hMove == 4'd6) ? cStart_int: cStart;
      cStart_int: nextState = (~valid) ? first : cStart_int;

      first: begin
        if(valid) begin
            case(hMove)
                4'd2: nextState= win12_int;
                4'd3: nextState= win13_int;
                4'd4: nextState= win14_int;
                4'd7: nextState= win17_int;
                4'd8: nextState= win18_int;
                4'd9: nextState= first_int;
                default: nextState= first;
            endcase
        end
        else nextState= first;
      end
      first_int: nextState= (~valid) ? second : first_int;

      second: begin
        if(valid) begin
            case(hMove)
                4'd2: nextState= win22_int;
                4'd4: nextState= win24_int;
                4'd7: nextState= win27_int;
                4'd8: nextState= win28_int;
                default: nextState= second;
            endcase
        end
        else nextState= second;
      end

      win12_int: nextState= (~valid) ? win12 : win12_int;
      win12: nextState= newGame ? new_game : win12;

      win13_int: nextState= (~valid) ? win13 : win13_int;
      win13: nextState= newGame ? new_game : win13;

      win14_int: nextState= (~valid) ? win14 : win14_int;
      win14: nextState= newGame ? new_game : win14;

      win17_int: nextState= (~valid) ? win17 : win17_int;
      win17: nextState= newGame ? new_game : win17;

      win18_int: nextState= (~valid) ? win18 : win18_int;
      win18: nextState= newGame ? new_game : win18;



      win22_int: nextState= (~valid) ? win22 : win22_int;
      win22: nextState= newGame ? new_game : win22;

      win24_int: nextState= (~valid) ? win24 : win24_int;
      win24: nextState= newGame ? new_game : win24;

      win27_int: nextState= (~valid) ? win27 : win27_int;
      win27: nextState= newGame ? new_game : win27;

      win28_int: nextState= (~valid) ? win28 : win28_int;
      win28: nextState= newGame ? new_game : win28;



      default: nextState = cStart; //never exec'd
    endcase
  end

  // Output Generator
  always_comb begin
    win = 1'b0;
    HHEX = 16'h00_00;
    unique case (currState)
      new_game, new_game_int: begin
        cMove= 4'd15; //invalid
        HHEX= 16'd0;
        CHEX= 16'd0;
      end
      cStart, cStart_int: begin
          cMove= 4'd5;
          CHEX= 16'h50_00;
      end

      first, first_int: begin
          cMove = 4'd1;
          HHEX= 16'h60_00;
          CHEX= 16'h15_00;
      end

      win12, win12_int: begin
        cMove = 4'd9;
        HHEX= 16'h26_00;
        CHEX= 16'h15_90;
        win = 1'b1;
      end

      win13, win13_int: begin
        cMove = 4'd9;
        HHEX= 16'h36_00;
        CHEX= 16'h15_90;
        win = 1'b1;
      end

      win14, win14_int: begin
        cMove = 4'd9;
        HHEX= 16'h46_00;
        CHEX= 16'h15_90;
        win = 1'b1;
      end

      win17, win17_int: begin
        cMove = 4'd9;
        HHEX= 16'h67_00;
        CHEX= 16'h15_90;
        win = 1'b1;
      end

      win18, win18_int: begin
        cMove = 4'd9;
        HHEX= 16'h68_00;
        CHEX= 16'h15_90;
        win = 1'b1;
      end




      second: begin
          cMove = 4'd3;
          HHEX= 16'h69_00;
          CHEX= 16'h13_50;
      end

      win22, win22_int: begin
          cMove = 4'd2;
          HHEX= 16'h26_90;
          CHEX= 16'h12_35;
          win = 1'b1;
      end

      win24, win24_int: begin
          cMove = 4'd2;
          HHEX= 16'h46_90;
          CHEX= 16'h12_35;
          win = 1'b1;
      end

      win27, win27_int: begin
          cMove = 4'd2;
          HHEX= 16'h67_90;
          CHEX= 16'h12_35;
          win = 1'b1;
      end

      win28, win28_int: begin
          cMove = 4'd2;
          HHEX= 16'h68_90;
          CHEX= 16'h12_35;
          win = 1'b1;
      end

    endcase
  end

  always_ff @(posedge clock)
    if (~reset_N)
      currState <= cStart;
    else
      currState <= nextState;
endmodule: abstractFSM
```

Lab Code [15 points]
Filename: testb.sv

```

module abstractFSM_test();
  logic [3:0] cMove, hMove;
  logic       win, clock, reset;

  myStructuralFSM DUT (.*);

  initial begin
    $monitor($time,, "state=%d, hMove=%d, cMove=%d, win=%b",
            {DUT.q2, DUT.q1, DUT.q0}, hMove, cMove, win);
    clock = 0;
    forever #5 clock = ~clock;
  end

  initial begin
    // Initialize Values
    hMove <= 4'b0000; reset <= 1;

    @(posedge clock);

    // Release reset
    reset <= 0;
    // Start test path for cStart
    hMove <= 4'd0;
    @(posedge clock);
    hMove <= 4'd1;
    @(posedge clock);
    hMove <= 4'd2;
    @(posedge clock);
    hMove <= 4'd3;
    @(posedge clock);
    hMove <= 4'd4;
    @(posedge clock);
    hMove <= 4'd5;
    @(posedge clock);
    hMove <= 4'd7;
    @(posedge clock);
    hMove <= 4'd8;
    @(posedge clock);
    hMove <= 4'd9;
    @(posedge clock);
    hMove <= 4'd10;
    @(posedge clock);
    hMove <= 4'd11;
    @(posedge clock);
    hMove <= 4'd12;
    @(posedge clock);
    hMove <= 4'd13;
    @(posedge clock);
    hMove <= 4'd14;
    @(posedge clock);
    hMove <= 4'd15;
    @(posedge clock);

    hMove <= 4'd6;
    @(posedge clock); // To first

    // Start testing First
    hMove <= 4'd0;
    @(posedge clock);
    hMove <= 4'd1;
    @(posedge clock);
    hMove <= 4'd5;
    @(posedge clock);
    hMove <= 4'd6;
    @(posedge clock);
    hMove <= 4'd10;
    @(posedge clock);
    hMove <= 4'd11;
    @(posedge clock);
    hMove <= 4'd12;
    @(posedge clock);
    hMove <= 4'd13;
    @(posedge clock);
    hMove <= 4'd14;
    @(posedge clock);
    hMove <= 4'd15;
    @(posedge clock);

    // To win1
    hMove <= 4'd2;
    @(posedge clock);
    reset <= 1;
    @(posedge clock);
    reset <= 0;

    hMove <= 4'd3;
    @(posedge clock);
    reset <= 1;
    @(posedge clock);
    reset <= 0;

    hMove <= 4'd4;
    @(posedge clock);
    reset <= 1;
    @(posedge clock);
    reset <= 0;

    hMove <= 4'd7;
    @(posedge clock);
    reset <= 1;
    @(posedge clock);
    reset <= 0;

    hMove <= 4'd8;
    @(posedge clock);

    // Start testing win1
    for (logic [3:0] i = 4'd0; i < 4'b1111; i++) begin
      hMove <= i;
      @(posedge clock);
    end
    hMove <= 4'b1111;
    @(posedge clock);
    reset <= 1;
    @(posedge clock);
    reset <= 0;

    hMove <= 4'd6;
    @(posedge clock);
    reset <= 1;
    @(posedge clock);
    reset <= 0;

    // Start testing second
    hMove <= 4'd6;
    @(posedge clock);
    hMove <= 4'd9;
    @(posedge clock);
    reset <= 1;
    @(posedge clock); // test reset
    reset <=0;
    hMove <= 4'd6;
    @(posedge clock);
    hMove <= 4'd9;
    @(posedge clock);
    hMove <= 4'd0;
    @(posedge clock); // test invalid cases
    hMove <= 4'd1;
    @(posedge clock);
    hMove <= 4'd3;
    @(posedge clock);
    hMove <= 4'd5;
    @(posedge clock);
    hMove <= 4'd9;
    @(posedge clock);
    hMove <= 4'd10;
    @(posedge clock);
    hMove <= 4'd11;
    @(posedge clock);
    hMove <= 4'd12;
    @(posedge clock);
    hMove <= 4'd13;
    @(posedge clock);
    hMove <= 4'd14;
    @(posedge clock);
    hMove <= 4'd15;
    @(posedge clock);

    hMove <= 4'd2;
    @(posedge clock); // to win3

    // Test win3
    for (logic [3:0] i = 4'd0; i < 4'b1111; i++) begin
      hMove <= i;
      @(posedge clock);
    end
    hMove <= 4'b1111;
    @(posedge clock);
    reset <= 1;
    @(posedge clock);
    reset <= 0;
    hMove <= 4'd6;
    @(posedge clock);
    hMove <= 4'd9;
    @(posedge clock);
    hMove <= 4'd4;
    @(posedge clock);

    reset <= 1;
    @(posedge clock);
    reset <= 0;
    hMove <= 4'd6;
    @(posedge clock);
    hMove <= 4'd9;
    @(posedge clock);
    hMove <= 4'd7;
    @(posedge clock);

    reset <= 1;
    @(posedge clock);
    reset <= 0;
    hMove <= 4'd6;
    @(posedge clock);
    hMove <= 4'd9;
    @(posedge clock); // return to second
    hMove <= 4'd8;
    @(posedge clock);

    // Test win2
    for (logic [3:0] i = 4'd0; i < 4'b1111; i++) begin
      hMove <= i;
      @(posedge clock);
    end
    hMove <= 4'b1111;
    @(posedge clock);
    reset <= 1;
    @(posedge clock);
    reset <= 0;
    @(posedge clock);
    #1 $finish;
  end
```

```
endmodule: abstractFSM_test
```

Lab Code [15 points]
Filename: tictac.sv

```
module dFlipFlop
    (output logic q,
     input logic d, clock, reset);

    always @(posedge clock)
    if (reset == 1)
        q <= 0;
    else
        q <= d;
endmodule: dFlipFlop

module myStructuralFSM
    (input logic [3:0] hMove,
     output logic [3:0] cMove,
     output logic win,
     input logic clock, reset);

logic q0, q1, q2;
logic d0, d1, d2;

dFlipFlop ff2(.d(d2), .q(q2), .clock, .reset),
          ff1(.d(d1), .q(q1), .clock, .reset),
          ff0(.d(d0), .q(q0), .clock, .reset);

logic tmp, tmp1, tmp2, tmp3;
assign tmp= (~q2 & ~q1 & ~q0) &
            ((hMove[3]                                  ) |
            (~hMove[3] & ~hMove[2]                      ) |
            (~hMove[3] & hMove[2] & ~hMove[1]           ) |
            (~hMove[3] & hMove[2] & hMove[1] & hMove[0] ));

assign tmp1= (~q2 & ~q1 & q0) &
            ((~hMove[3] & ~hMove[2] & ~hMove[1]          ) |
            (~hMove[3] & hMove[2] & ~hMove[1] & hMove[0]) |
            (~hMove[3] & hMove[2] & hMove[1] & ~hMove[0]) |
            (hMove[3] & ~hMove[2] & hMove[1]            ) |
            (hMove[3] & hMove[2]                        ));

assign tmp2= (~q2 & ~q1 & q0) &
           ((~hMove[3] & ~hMove[2] & hMove[1]           ) |
           (~hMove[3] & hMove[2] & ~hMove[1] & ~hMove[0] ) |
           (~hMove[3] & hMove[2] & hMove[1] & hMove[0] ) |
           (hMove[3] & ~hMove[2] & ~hMove[1] & ~hMove[0]));

assign tmp3=(~q2 & q1 & ~q0) &
            ((~hMove[3] & ~hMove[2] & ~hMove[1]           ) |
            (~hMove[3] & ~hMove[2] & hMove[1] & hMove[0] ) |
            (~hMove[3] & hMove[2] & ~hMove[1] & hMove[0] ) |
            (~hMove[3] & hMove[2] & hMove[1] & ~hMove[0] ) |
            (hMove[3] & ~hMove[2] & ~hMove[1] & hMove[0] ) |
            (hMove[3] & ~hMove[2] & hMove[1]             ) |
            (hMove[3] & hMove[2]                         ));

/* Next State Logic */
assign d2= (~hMove[3] & ~hMove[2] & hMove[1] & ~hMove[0] & ~q2 & q1 & ~q0) |
           (~hMove[3] & hMove[2] & ~hMove[1] & ~hMove[0] & ~q2 & q1 & ~q0) |
           (~hMove[3] & hMove[2] & hMove[1] & hMove[0]   & ~q2 & q1 & ~q0) |
           (hMove[3] & ~hMove[2] & ~hMove[1] & ~hMove[0] & ~q2 & q1 & ~q0) |
           (q2 & ~q1 & ~q0) |
           (q2 & ~q1 & q0);

assign d1= (hMove[3] & ~hMove[2] & ~hMove[1] & hMove[0]  & ~q2 & ~q1 & q0) |
           tmp2 |
           tmp3 |
           (~q2 & q1 & q0);

assign d0= (~hMove[3] & hMove[2] & hMove[1] & ~hMove[0]  & ~q2 & ~q1 & ~q0) |
           tmp1 |
           tmp2 |
           (~hMove[3] & ~hMove[2] & hMove[1] & ~hMove[0] & ~q2 & q1 & ~q0) |
           (~q2 & q1 & q0) |
           (q2 & ~q1 & q0);


/* Output Logic */
assign cMove[3]= ~q2 & q1 & q0;

assign cMove[2]= (~hMove[3] & hMove[2] & hMove[1] & ~hMove[0] & ~q2 & ~q1 & ~q0)
                 | tmp |
                 (q2 & ~q1 & q0);

assign cMove[1]= tmp3 |
                 (~hMove[3] & ~hMove[2] & hMove[1] & ~hMove[0] & ~q2 & q1 & ~q0)
                 | (~hMove[3] & hMove[2] & ~hMove[1] & ~hMove[0] & ~q2 & q1 &
                 ~q0) | (~hMove[3] & hMove[2] & hMove[1] & hMove[0]   & ~q2 & q1
                 & ~q0) | (hMove[3] & ~hMove[2] & ~hMove[1] & ~hMove[0] & ~q2 &
                 q1 & ~q0) | (q2 & ~q1 & ~q0) | (q2 & ~q1 & q0);

assign cMove[0]= ~(q2 & ~q1 & ~q0);

assign win= (~q2 & q1 & q0) |
            (q2 & ~q1 & ~q0) |
            (q2 & ~q1 & q0);

endmodule : myStructuralFSM
```