Daniel Self

CompE 470L

Arkajit Dutta

December 10th, 2025

Final Report

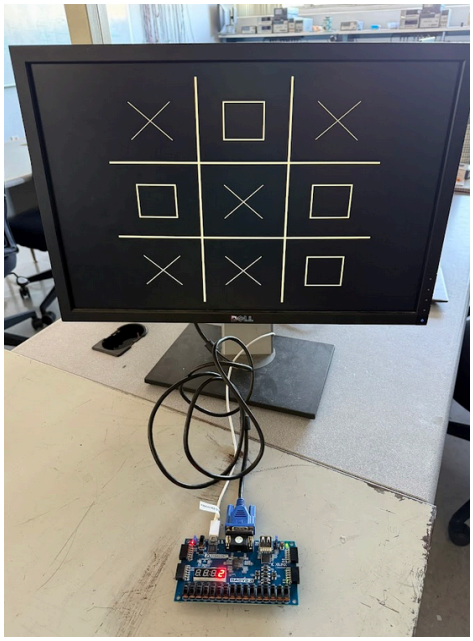
Introduction

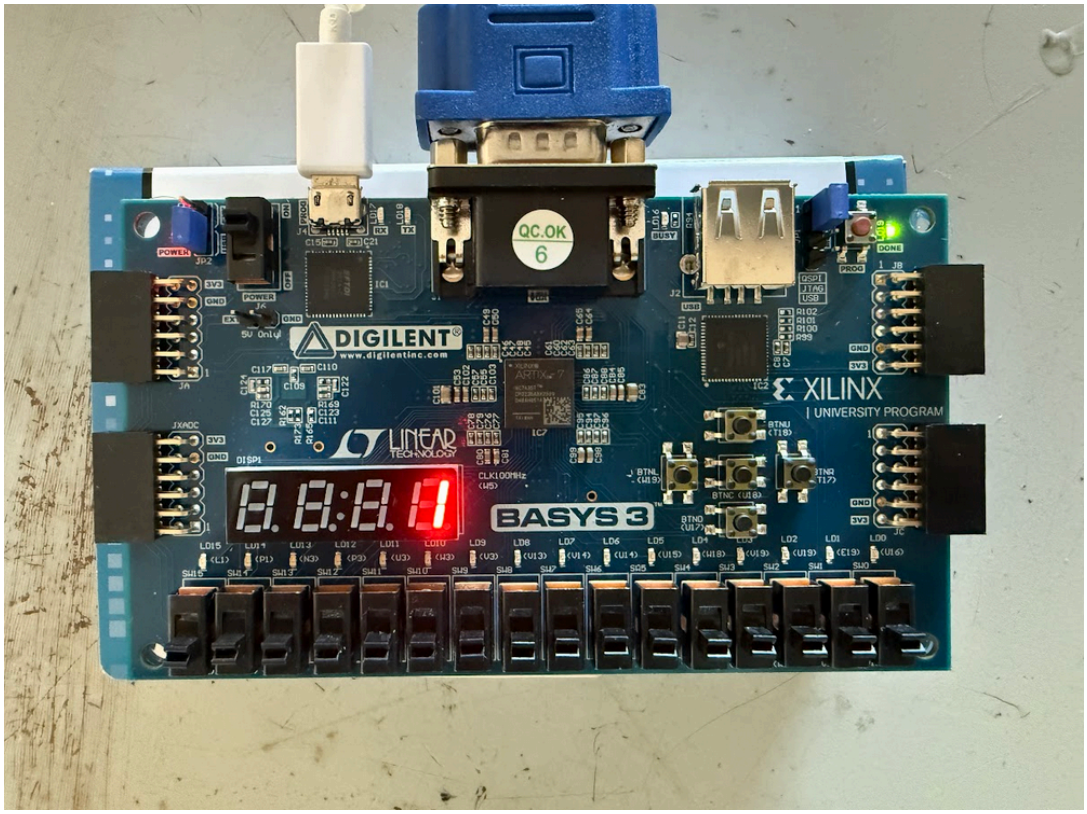
The objective of this final project was to design and implement a fully functional Tic-Tac-Toe game on the BASYS 3 FPGA using Verilog HDL. Building on the initial proposal, the system allows two players to alternate placing X's and O's on a 3×3 grid displayed in real time on a VGA monitor. User input is handled through the first nine switches on the board, each corresponding to a specific cell, while the center push button provides a hardware reset. The game logic tracks turns, validates moves, updates the board, and determines when a winning condition has been reached. When a player wins, the game ends and no more inputs can be placed. To reset the game board, all switches need to be in the off position and then the center push button is pressed to refresh the VGA output.

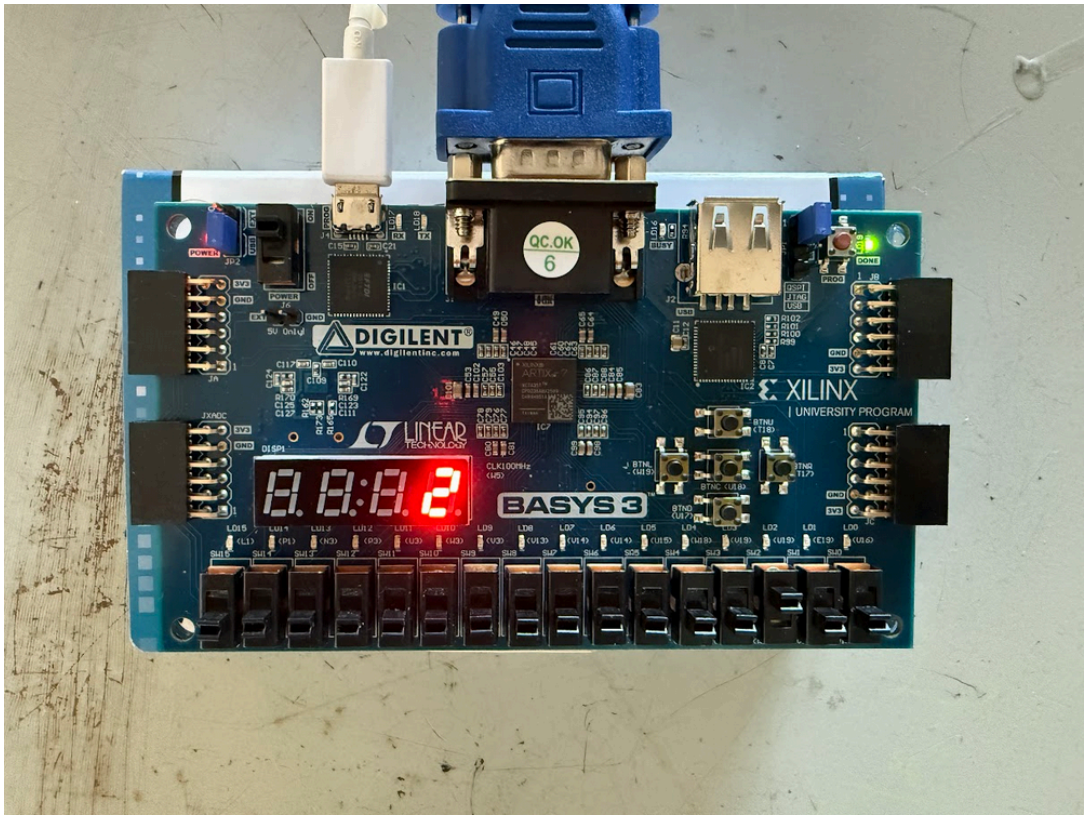
This project integrates several design components. These include VGA timing generation, pixel-level rendering, a finite-state machine for win detection, per-cell state modules, and a seven-segment display used to show the current player's turn. The design was simulated, synthesized, and implemented on the BASYS 3 board using Xilinx Vivado, and functionality was verified through hardware testing. Overall, the project demonstrates the application of Verilog-based digital design techniques to create an interactive, real-time game system.

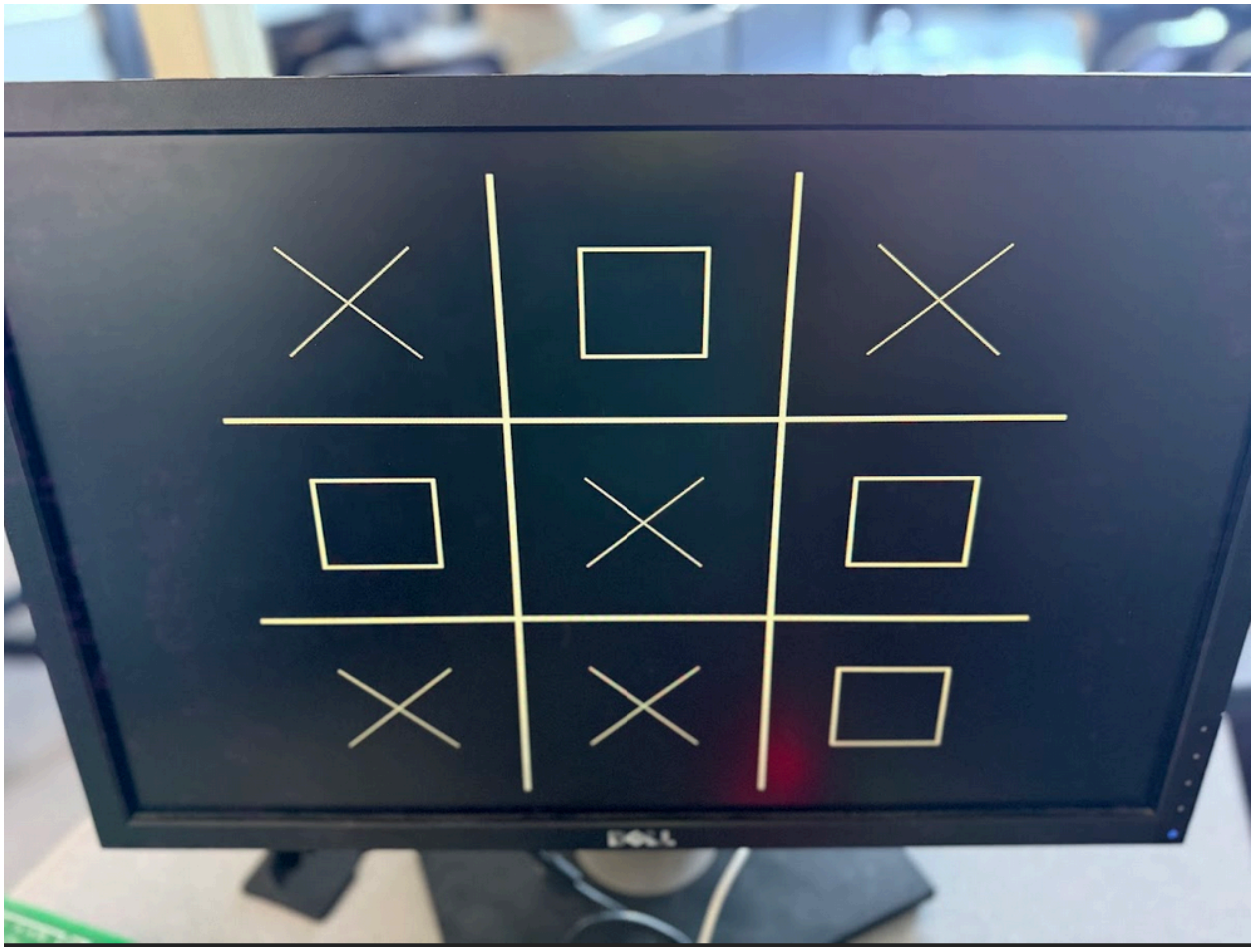
IMAGES/VIDEO:

IMG_6640.MOV




QC.OK
6
DIGILENT
www.digilentinc.com
LINEAR TECHNOLOGY
XILINX
UNIVERSITY PROGRAM
BASYS 3
DISP1
CLK100MHz
(W5)


QC.OK
6
DIGILENT
www.digilentinc.com
LINEAR TECHNOLOGY
XILINX
UNIVERSITY PROGRAM
BASYS 3
DISP1
CLK100MHz
(W5)



Conclusion

This project successfully demonstrated the design and implementation of a complete Tic-Tac-Toe game system on the BASYS 3 FPGA using Verilog HDL. The system accurately processed player inputs, displayed the evolving game board on a VGA monitor, and correctly detected horizontal, vertical, and diagonal win conditions. When a winning pattern occurred, the remaining empty cells could no longer be filled. The use of a dedicated state machine for win detection, individual cell modules for move storage, and a VGA drawing pipeline allowed for clear modularity and reliable hardware behavior. The seven-segment display further enhanced usability by showing the current player's turn throughout gameplay.

This project reinforced several key concepts from the course, including finite-state machine design, synchronous logic, clock-driven video generation, and hardware-based input handling. Completing the implementation on hardware provided valuable experience in debugging timing-based systems and working with real-time graphical interfaces. Overall, the Tic-Tac-Toe project served as a comprehensive demonstration of FPGA design principles learned throughout the semester.

VERILOG:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Daniel Self
//
// Create Date:
// Design Name:
// Module Name: gameLogic
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module gameLogic(
    // Video / clock
    input  wire       clk,
    input  wire       Reset,

    // Control (first 9 switches)
    input  wire [8:0] In,

    // VGA out
    output wire       hsync,
    output wire       vsync,
    output wire [11:0] rgb,

    // Status LED (winner)
    output wire       winState,

    // Seven-seg display
    output wire [6:0] seg,
    output wire [3:0] an,
    output wire       dp
```

```
);

    reg [8:0]  sqrSel;
    wire [17:0] Cells;
    reg [8:0]  prevIn;
    reg [8:0]  myIn;
    reg        Turn;      // 0 = X, 1 = O

    wire [8:0] Color;
    wire       winState_int;

    assign winState = winState_int;

    // Input decoding (same logic, just pure Verilog always @(*))
    always @(*) begin
        if (!winState_int) begin
            if      (In[6] && !Cells[16]) sqrSel = 9'b100000000;
            else if (In[7] && !Cells[14]) sqrSel = 9'b010000000;
            else if (In[8] && !Cells[12]) sqrSel = 9'b001000000;

            else if (In[3] && !Cells[10]) sqrSel = 9'b000100000;
            else if (In[4] && !Cells[8])  sqrSel = 9'b000010000;
            else if (In[5] && !Cells[6])  sqrSel = 9'b000001000;

            else if (In[0] && !Cells[4])  sqrSel = 9'b000000100;
            else if (In[1] && !Cells[2])  sqrSel = 9'b000000010;
            else if (In[2] && !Cells[0])  sqrSel = 9'b000000001;

            else                          sqrSel = 9'b000000000;
        end else begin
            sqrSel = 9'b000000000;
        end
    end

    // Turn toggling based on input edge
    always @(negedge clk or posedge Reset) begin
        if (Reset) begin
            Turn   <= 1'b0;       // X starts
            prevIn <= 9'b0;
            myIn   <= 9'b0;
        end else begin
            if (prevIn != myIn && myIn != 9'b0)
                Turn <= ~Turn;
```

```
        prevIn <= myIn;
        myIn   <= In;
    end
end

// Win checking
gameState state_inst (
    .clk      (clk),
    .Reset    (Reset),
    .Cells    (Cells),
    .winState (winState_int),
    .Color    (Color)
);

// Video generator (grid + X/O + highlighting)
videoElements VGA_inst (
    .clk   (clk),
    .reset (Reset),
    .Cells (Cells),
    .Color (Color),
    .Turn  (Turn),
    .hsync (hsync),
    .vsync (vsync),
    .rgb   (rgb)
);

// Board cells (9 instances)
Cell cell1 (
    .clk  (clk), .Sel(sqrSel[0]), .Turn(Turn), .Reset(Reset),
    .State(Cells[1:0])
);

Cell cell2 (
    .clk  (clk), .Sel(sqrSel[1]), .Turn(Turn), .Reset(Reset),
    .State(Cells[3:2])
);

Cell cell3 (
    .clk  (clk), .Sel(sqrSel[2]), .Turn(Turn), .Reset(Reset),
    .State(Cells[5:4])
);

Cell cell4 (
    .clk  (clk), .Sel(sqrSel[3]), .Turn(Turn), .Reset(Reset),
```

```
        .State(Cells[7:6])
    );

    Cell cell5 (
        .clk  (clk), .Sel(sqrSel[4]), .Turn(Turn), .Reset(Reset),
        .State(Cells[9:8])
    );

    Cell cell6 (
        .clk  (clk), .Sel(sqrSel[5]), .Turn(Turn), .Reset(Reset),
        .State(Cells[11:10])
    );

    Cell cell7 (
        .clk  (clk), .Sel(sqrSel[6]), .Turn(Turn), .Reset(Reset),
        .State(Cells[13:12])
    );

    Cell cell8 (
        .clk  (clk), .Sel(sqrSel[7]), .Turn(Turn), .Reset(Reset),
        .State(Cells[15:14])
    );

    Cell cell9 (
        .clk  (clk), .Sel(sqrSel[8]), .Turn(Turn), .Reset(Reset),
        .State(Cells[17:16])
    );

    // Seven segment turn indicator (X/O)
    sevenseg_turn turnDisplay (
        .clk  (clk),
        .turn (Turn),
        .seg  (seg),
        .an   (an),
        .dp   (dp)
    );

endmodule

module gameState(
    input  wire       Reset,
    input  wire       clk,
    input  wire [17:0] Cells,    // 9 cells * 2 bits
```

```
    output reg        winState,  // 1 when someone has won
    output reg [8:0]  Color      // which cells to highlight
);

    // PLAY / WIN FSM
    localparam PLAY = 1'b0;
    localparam WIN  = 1'b1;

    reg PS, NS;

    // state register
    always @(posedge clk) begin
        if (Reset)
            PS <= PLAY;
        else
            PS <= NS;
    end

    // next-state and outputs
    always @(*) begin
        winState = 1'b0;
        Color    = 9'b000000000;
        NS       = PS;

        case (PS)
            PLAY: begin
                // Horizontal wins
                if (Cells[1:0] == Cells[3:2] &&
                    Cells[3:2] == Cells[5:4] &&
                    Cells[1:0] != 2'b00) begin
                    Color = 9'b000000111;
                    NS    = WIN;
                end
                else if (Cells[7:6] == Cells[9:8] &&
                         Cells[9:8] == Cells[11:10] &&
                         Cells[7:6] != 2'b00) begin
                    Color = 9'b000111000;
                    NS    = WIN;
                end
                else if (Cells[13:12] == Cells[15:14] &&
                         Cells[15:14] == Cells[17:16] &&
                         Cells[13:12] != 2'b00) begin
                    Color = 9'b111000000;
                    NS    = WIN;
```

```
    end

    // Vertical wins
    else if (Cells[1:0] == Cells[7:6] &&
            Cells[7:6] == Cells[13:12] &&
            Cells[1:0] != 2'b00) begin
        Color = 9'b001001001;
        NS    = WIN;
    end
    else if (Cells[3:2] == Cells[9:8] &&
            Cells[9:8] == Cells[15:14] &&
            Cells[3:2] != 2'b00) begin
        Color = 9'b010010010;
        NS    = WIN;
    end
    else if (Cells[5:4] == Cells[11:10] &&
            Cells[11:10] == Cells[17:16] &&
            Cells[5:4] != 2'b00) begin
        Color = 9'b100100100;
        NS    = WIN;
    end

    // Diagonals
    else if (Cells[1:0] == Cells[9:8] &&
            Cells[9:8] == Cells[17:16] &&
            Cells[1:0] != 2'b00) begin
        Color = 9'b100010001;
        NS    = WIN;
    end
    else if (Cells[5:4] == Cells[9:8] &&
            Cells[9:8] == Cells[13:12] &&
            Cells[5:4] != 2'b00) begin
        Color = 9'b001010100;
        NS    = WIN;
    end
    else begin
        Color = 9'b000000000;
        NS    = PLAY;
    end
end

WIN: begin
    winState = 1'b1;
    // keep Color from previous cycle (stays red)
```

```
            NS       = WIN;
         end

         default: begin
            winState = 1'b0;
            Color    = 9'b000000000;
            NS       = PLAY;
         end
      endcase
   end

endmodule

module videoElements(
   input  wire        clk,
   input  wire        reset,
   input  wire [17:0] Cells,
   input  wire [8:0]  Color,
   input  wire        Turn,    // not used but kept for compatibility

   output wire        hsync,
   output wire        vsync,
   output wire [11:0] rgb
);

   // VGA timing constants from original design
   localparam hRes = 640;
   localparam vRes = 480;

   localparam hBorder = 100;
   localparam vBorder = 20;

   localparam hLinePos1 = vBorder + 147;
   localparam hLinePos2 = (vRes - 20) - 147;

   localparam vLinePos1 = hBorder + 147;
   localparam vLinePos2 = (hRes - 100) - 147;

   localparam sqBorder  = 40;
   localparam [4:0] plsBorder = 30;   // kept but no longer used for shapes

   localparam lineWeight = 2;

   reg [1:0] pDisp;
```

```
wire [9:0] hPos, vPos;
wire       p_tick;
wire       video_on;

// VGA timing generator
vga_sync vga_sync_unit (
    .clk(clk),
    .reset(reset),
    .hsync(hsync),
    .vsync(vsync),
    .video_on(video_on),
    .p_tick(p_tick),
    .x(hPos),
    .y(vPos)
);

// helper variables for shape drawing
integer cellX1, cellX2, cellY1, cellY2;
integer cx, cy;
integer dx, dy;

// MAIN PIXEL LOGIC
always @(posedge p_tick or posedge reset) begin
    if (reset) begin
        pDisp <= 2'b00;
    end
    else if (!video_on) begin
        pDisp <= 2'b00;
    end
    else begin

        // ============================
        // GRID LINES
        // ============================

        // Horizontal lines
        if (hPos > hBorder && hPos < (hRes - hBorder) &&
           ((vPos > hLinePos1 - lineWeight && vPos < hLinePos1 + lineWeight) ||
            (vPos > hLinePos2 - lineWeight && vPos < hLinePos2 + lineWeight))) begin
            pDisp <= 2'b01;
        end

        // Vertical lines
```

```
else if (vPos > vBorder && vPos < (vRes - vBorder) &&
        ((hPos > vLinePos1 - lineWeight && hPos < vLinePos1 + lineWeight) ||
         (hPos > vLinePos2 - lineWeight && hPos < vLinePos2 + lineWeight))) begin
    pDisp <= 2'b01;
end


// ======================================================
// SHAPE DRAWING
//   X = diagonal X across full cell
//   O = hollow rectangle across full cell
// ======================================================


// ----------------------
//  CELL 1  (Cells[1:0])
// ----------------------
else begin
    // Default no pixel
    pDisp <= 2'b00;

    // CELL 1 bounds
    if (hPos > hBorder + sqBorder && hPos < vLinePos1 - sqBorder &&
        vPos > vBorder + sqBorder && vPos < hLinePos1 - sqBorder) begin

        cellX1 = hBorder + sqBorder;
        cellX2 = vLinePos1 - sqBorder;
        cellY1 = vBorder + sqBorder;
        cellY2 = hLinePos1 - sqBorder;

        cx = (cellX1 + cellX2) >>> 1;
        cy = (cellY1 + cellY2) >>> 1;

        dx = hPos - cx;
        dy = vPos - cy;

        case (Cells[1:0])
            // --- X: full diagonal across cell ---
            2'b01: begin
                if (dx == dy || dx == -dy)
                    pDisp <= {Color[0], 1'b1};
            end

            // --- O: hollow rectangle (same style as before) ---
```

```
        2'b11: begin
           if ( (hPos - cellX1 < 3) || (cellX2 - hPos < 3) ||
                (vPos - cellY1 < 3) || (cellY2 - vPos < 3) )
               pDisp <= {Color[0],1'b1};
        end

        default: ;
     endcase
  end

  // CELL 2  (Cells[3:2])
  if (hPos > vLinePos1 + sqBorder && hPos < vLinePos2 - sqBorder &&
     vPos > vBorder + sqBorder && vPos < hLinePos1 - sqBorder) begin

     cellX1 = vLinePos1 + sqBorder;
     cellX2 = vLinePos2 - sqBorder;
     cellY1 = vBorder + sqBorder;
     cellY2 = hLinePos1 - sqBorder;

     cx = (cellX1 + cellX2) >>> 1;
     cy = (cellY1 + cellY2) >>> 1;

     dx = hPos - cx;
     dy = vPos - cy;

     case (Cells[3:2])
        2'b01: begin
           if (dx == dy || dx == -dy)
              pDisp <= {Color[1],1'b1};
        end
        2'b11: begin
           if ( (hPos - cellX1 < 3) || (cellX2 - hPos < 3) ||
                (vPos - cellY1 < 3) || (cellY2 - vPos < 3) )
               pDisp <= {Color[1],1'b1};
        end
        default: ;
     endcase
  end

  // CELL 3  (Cells[5:4])
  if (hPos > vLinePos2 + sqBorder && hPos < (hRes - hBorder) - sqBorder &&
     vPos > vBorder + sqBorder && vPos < hLinePos1 - sqBorder) begin

     cellX1 = vLinePos2 + sqBorder;
```

```
    cellX2 = (hRes - hBorder) - sqBorder;
    cellY1 = vBorder + sqBorder;
    cellY2 = hLinePos1 - sqBorder;

    cx = (cellX1 + cellX2) >>> 1;
    cy = (cellY1 + cellY2) >>> 1;

    dx = hPos - cx;
    dy = vPos - cy;

    case (Cells[5:4])
       2'b01: begin
          if (dx == dy || dx == -dy)
             pDisp <= {Color[2],1'b1};
       end
       2'b11: begin
          if ( (hPos - cellX1 < 3) || (cellX2 - hPos < 3) ||
               (vPos - cellY1 < 3) || (cellY2 - vPos < 3) )
             pDisp <= {Color[2],1'b1};
       end
       default: ;
    endcase
 end

 // CELL 4  (Cells[7:6])
 if (hPos > hBorder + sqBorder && hPos < vLinePos1 - sqBorder &&
    vPos > hLinePos1 + sqBorder && vPos < hLinePos2 - sqBorder) begin

    cellX1 = hBorder + sqBorder;
    cellX2 = vLinePos1 - sqBorder;
    cellY1 = hLinePos1 + sqBorder;
    cellY2 = hLinePos2 - sqBorder;

    cx = (cellX1 + cellX2) >>> 1;
    cy = (cellY1 + cellY2) >>> 1;

    dx = hPos - cx;
    dy = vPos - cy;

    case (Cells[7:6])
       2'b01: begin
          if (dx == dy || dx == -dy)
             pDisp <= {Color[3],1'b1};
       end
```

```
        2'b11: begin
           if ( (hPos - cellX1 < 3) || (cellX2 - hPos < 3) ||
                (vPos - cellY1 < 3) || (cellY2 - vPos < 3) )
              pDisp <= {Color[3],1'b1};
        end
        default: ;
     endcase
  end

  // CELL 5  (Cells[9:8])
  if (hPos > vLinePos1 + sqBorder && hPos < vLinePos2 - sqBorder &&
     vPos > hLinePos1 + sqBorder && vPos < hLinePos2 - sqBorder) begin

     cellX1 = vLinePos1 + sqBorder;
     cellX2 = vLinePos2 - sqBorder;
     cellY1 = hLinePos1 + sqBorder;
     cellY2 = hLinePos2 - sqBorder;

     cx = (cellX1 + cellX2) >>> 1;
     cy = (cellY1 + cellY2) >>> 1;

     dx = hPos - cx;
     dy = vPos - cy;

     case (Cells[9:8])
        2'b01: begin
           if (dx == dy || dx == -dy)
              pDisp <= {Color[4],1'b1};
        end
        2'b11: begin
           if ( (hPos - cellX1 < 3) || (cellX2 - hPos < 3) ||
                (vPos - cellY1 < 3) || (cellY2 - vPos < 3) )
              pDisp <= {Color[4],1'b1};
        end
        default: ;
     endcase
  end

  // CELL 6  (Cells[11:10])
  if (hPos > vLinePos2 + sqBorder && hPos < (hRes-hBorder) - sqBorder &&
     vPos > hLinePos1 + sqBorder && vPos < hLinePos2 - sqBorder) begin

     cellX1 = vLinePos2 + sqBorder;
     cellX2 = (hRes - hBorder) - sqBorder;
```

```
    cellY1 = hLinePos1 + sqBorder;
    cellY2 = hLinePos2 - sqBorder;

    cx = (cellX1 + cellX2) >>> 1;
    cy = (cellY1 + cellY2) >>> 1;

    dx = hPos - cx;
    dy = vPos - cy;

    case (Cells[11:10])
       2'b01: begin
          if (dx == dy || dx == -dy)
             pDisp <= {Color[5],1'b1};
       end
       2'b11: begin
          if ( (hPos - cellX1 < 3) || (cellX2 - hPos < 3) ||
               (vPos - cellY1 < 3) || (cellY2 - vPos < 3) )
             pDisp <= {Color[5],1'b1};
       end
       default: ;
    endcase
end

// CELL 7  (Cells[13:12])
if (hPos > hBorder + sqBorder && hPos < vLinePos1 - sqBorder &&
    vPos > hLinePos2 + sqBorder && vPos < (vRes - vBorder) - sqBorder) begin

    cellX1 = hBorder + sqBorder;
    cellX2 = vLinePos1 - sqBorder;
    cellY1 = hLinePos2 + sqBorder;
    cellY2 = (vRes - vBorder) - sqBorder;

    cx = (cellX1 + cellX2) >>> 1;
    cy = (cellY1 + cellY2) >>> 1;

    dx = hPos - cx;
    dy = vPos - cy;

    case (Cells[13:12])
       2'b01: begin
          if (dx == dy || dx == -dy)
             pDisp <= {Color[6],1'b1};
       end
       2'b11: begin
```

```
            if ( (hPos - cellX1 < 3) || (cellX2 - hPos < 3) ||
                 (vPos - cellY1 < 3) || (cellY2 - vPos < 3) )
                pDisp <= {Color[6],1'b1};
        end
        default: ;
    endcase
end

// CELL 8  (Cells[15:14])
if (hPos > vLinePos1 + sqBorder && hPos < vLinePos2 - sqBorder &&
    vPos > hLinePos2 + sqBorder && vPos < (vRes - vBorder) - sqBorder) begin

    cellX1 = vLinePos1 + sqBorder;
    cellX2 = vLinePos2 - sqBorder;
    cellY1 = hLinePos2 + sqBorder;
    cellY2 = (vRes - vBorder) - sqBorder;

    cx = (cellX1 + cellX2) >>> 1;
    cy = (cellY1 + cellY2) >>> 1;

    dx = hPos - cx;
    dy = vPos - cy;

    case (Cells[15:14])
        2'b01: begin
            if (dx == dy || dx == -dy)
                pDisp <= {Color[7],1'b1};
        end
        2'b11: begin
            if ( (hPos - cellX1 < 3) || (cellX2 - hPos < 3) ||
                 (vPos - cellY1 < 3) || (cellY2 - vPos < 3) )
                pDisp <= {Color[7],1'b1};
        end
        default: ;
    endcase
end

// CELL 9  (Cells[17:16])
if (hPos > vLinePos2 + sqBorder && hPos < (hRes-hBorder) - sqBorder &&
    vPos > hLinePos2 + sqBorder && vPos < (vRes - vBorder) - sqBorder) begin

    cellX1 = vLinePos2 + sqBorder;
    cellX2 = (hRes - hBorder) - sqBorder;
    cellY1 = hLinePos2 + sqBorder;
```

```
                cellY2 = (vRes - vBorder) - sqBorder;

                cx = (cellX1 + cellX2) >>> 1;
                cy = (cellY1 + cellY2) >>> 1;

                dx = hPos - cx;
                dy = vPos - cy;

                case (Cells[17:16])
                   2'b01: begin
                      if (dx == dy || dx == -dy)
                         pDisp <= {Color[8],1'b1};
                   end
                   2'b11: begin
                      if ( (hPos - cellX1 < 3) || (cellX2 - hPos < 3) ||
                           (vPos - cellY1 < 3) || (cellY2 - vPos < 3) )
                         pDisp <= {Color[8],1'b1};
                   end
                   default: ;
                endcase
             end
          end
       end
    end

    // FINAL COLOR OUTPUT
    assign rgb = (pDisp[0])
              ? (pDisp[1] ? 12'hF00 : 12'hFFF)
              : 12'h000;

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Daniel Self
//
// Create Date:
// Design Name:
// Module Name: vga_sync
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

// Generates 640x480 @ 60 Hz timing with 25 MHz pixel tick
module vga_sync(
    input wire clk,          // 100 MHz Basys3 clock
    input wire reset,
    output wire hsync,
    output wire vsync,
    output wire video_on,
    output wire p_tick,
    output wire [9:0] x,     // horizontal pixel position
    output wire [9:0] y      // vertical pixel position
);

    // ----------------------------------------------------
    // VGA TIMING CONSTANTS (640x480 @ 60Hz)
    // ----------------------------------------------------
    localparam H_DISPLAY       = 640;
    localparam H_L_BORDER      = 48;
    localparam H_R_BORDER      = 16;
    localparam H_RETRACE       = 96;
    localparam H_MAX           = H_DISPLAY + H_L_BORDER + H_R_BORDER + H_RETRACE
- 1;
    localparam START_H_RETRACE = H_DISPLAY + H_R_BORDER;
    localparam END_H_RETRACE   = H_DISPLAY + H_R_BORDER + H_RETRACE - 1;

    localparam V_DISPLAY       = 480;
    localparam V_T_BORDER      = 10;
    localparam V_B_BORDER      = 33;
    localparam V_RETRACE       = 2;
    localparam V_MAX           = V_DISPLAY + V_T_BORDER + V_B_BORDER + V_RETRACE -
1;
    localparam START_V_RETRACE = V_DISPLAY + V_B_BORDER;
    localparam END_V_RETRACE   = V_DISPLAY + V_B_BORDER + V_RETRACE - 1;

    // ----------------------------------------------------
    // PIXEL CLOCK: divide 100 MHz → 25 MHz (mod-4 counter)
```

```
// ----------------------------------------------------
reg [1:0] pixel_reg;
wire [1:0] pixel_next;
wire pixel_tick;

always @(posedge clk or posedge reset) begin
   if (reset)
      pixel_reg <= 2'b00;
   else
      pixel_reg <= pixel_next;
end

assign pixel_next = pixel_reg + 1'b1;
assign pixel_tick = (pixel_reg == 2'b00);
assign p_tick     = pixel_tick;

// ----------------------------------------------------
// HORIZONTAL & VERTICAL COUNTERS
// ----------------------------------------------------
reg [9:0] h_count_reg, h_count_next;
reg [9:0] v_count_reg, v_count_next;

reg hsync_reg, vsync_reg;
wire hsync_next, vsync_next;

always @(posedge clk or posedge reset) begin
   if (reset) begin
      h_count_reg <= 10'd0;
      v_count_reg <= 10'd0;
      hsync_reg   <= 1'b0;
      vsync_reg   <= 1'b0;
   end
   else begin
      h_count_reg <= h_count_next;
      v_count_reg <= v_count_next;
      hsync_reg   <= hsync_next;
      vsync_reg   <= vsync_next;
   end
end

// Next-state logic
always @(*) begin
   // Horizontal counter advances on pixel tick
   h_count_next = pixel_tick
```

```
            ? (h_count_reg == H_MAX ? 0 : h_count_reg + 1)
            : h_count_reg;

        // Vertical counter advances only when hcounter wraps
        v_count_next = (pixel_tick && h_count_reg == H_MAX)
            ? (v_count_reg == V_MAX ? 0 : v_count_reg + 1)
            : v_count_reg;
    end

    // ------------------------------------------------
    // SYNC SIGNALS (active low)
    // ------------------------------------------------
    assign hsync_next = (h_count_reg >= START_H_RETRACE &&
                         h_count_reg <= END_H_RETRACE);

    assign vsync_next = (v_count_reg >= START_V_RETRACE &&
                         v_count_reg <= END_V_RETRACE);

    // Output sync signals
    assign hsync = ~hsync_reg;   // active low
    assign vsync = ~vsync_reg;   // active low

    // ------------------------------------------------
    // VIDEO ON/OFF FLAG
    // ------------------------------------------------
    assign video_on =
        (h_count_reg < H_DISPLAY) &&
        (v_count_reg < V_DISPLAY);

    // Current pixel coordinates
    assign x = h_count_reg;
    assign y = v_count_reg;

endmodule

module Cell(
    input  wire       clk,
    input  wire       Sel,
    input  wire       Turn,   // 0 = X, 1 = O
    input  wire       Reset,
    output reg  [1:0] State   // 00 = N, 01 = X, 11 = O
);

    // enum replacement
```

```
localparam N = 2'b00;
localparam X = 2'b01;
localparam O = 2'b11;

reg [1:0] PS, NS;

// state register
always @(posedge clk) begin
   if (Reset)
      PS <= N;
   else
      PS <= NS;
end

// next-state and output logic
always @(*) begin
   case (PS)
      N: begin
         State = 2'b00;
         if (Sel && ~Turn)
            NS = X;
         else if (Sel && Turn)
            NS = O;
         else
            NS = N;
      end

      X: begin
         State = 2'b01;
         NS    = X;
      end

      O: begin
         State = 2'b11;
         NS    = O;
      end

      default: begin
         State = 2'b00;
         NS    = N;
      end
   endcase
end
```

```
endmodule


// Shows X when turn = 0, O when turn = 1 on digit 0.
module sevenseg_turn(
    input  wire clk,
    input  wire turn,        // 0 = X, 1 = O
    output reg  [6:0] seg,   // active-low segments CA..CG
    output reg  [3:0] an,    // active-low anodes
    output reg        dp     // decimal point (active-low)
);

    always @(posedge clk) begin
        // enable only rightmost digit (AN0)
        an <= 4'b1110;   // AN0 = 0, AN1-3 = 1
        dp <= 1'b1;      // decimal point off

        if (turn == 1'b0) begin
            // Digit "1"
            seg <= 7'b1111001;
        end
        else begin
            // Digit "2"
            seg <= 7'b0100100;
        end
    end

endmodule
```

CONSTRAINTS:

```
## Clock
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

## Switches -> In[8:0]
set_property PACKAGE_PIN V17 [get_ports {In[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {In[0]}]
set_property PACKAGE_PIN V16 [get_ports {In[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {In[1]}]
set_property PACKAGE_PIN W16 [get_ports {In[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {In[2]}]
set_property PACKAGE_PIN W17 [get_ports {In[3]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {In[3]}]
set_property PACKAGE_PIN W15 [get_ports {In[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {In[4]}]
set_property PACKAGE_PIN V15 [get_ports {In[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {In[5]}]
set_property PACKAGE_PIN W14 [get_ports {In[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {In[6]}]
set_property PACKAGE_PIN W13 [get_ports {In[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {In[7]}]
set_property PACKAGE_PIN V2  [get_ports {In[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {In[8]}]

## Reset button (center)
set_property PACKAGE_PIN U18 [get_ports Reset]
set_property IOSTANDARD LVCMOS33 [get_ports Reset]

## winState LED (LED0)
set_property PACKAGE_PIN U16 [get_ports winState]
set_property IOSTANDARD LVCMOS33 [get_ports winState]

## VGA
set_property PACKAGE_PIN G19 [get_ports {rgb[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rgb[8]}]
set_property PACKAGE_PIN H19 [get_ports {rgb[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rgb[9]}]
set_property PACKAGE_PIN J19 [get_ports {rgb[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rgb[10]}]
set_property PACKAGE_PIN N19 [get_ports {rgb[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rgb[11]}]
set_property PACKAGE_PIN N18 [get_ports {rgb[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rgb[0]}]
set_property PACKAGE_PIN L18 [get_ports {rgb[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rgb[1]}]
set_property PACKAGE_PIN K18 [get_ports {rgb[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rgb[2]}]
set_property PACKAGE_PIN J18 [get_ports {rgb[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rgb[3]}]
set_property PACKAGE_PIN J17 [get_ports {rgb[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rgb[4]}]
set_property PACKAGE_PIN H17 [get_ports {rgb[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rgb[5]}]
set_property PACKAGE_PIN G17 [get_ports {rgb[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {rgb[6]}]
set_property PACKAGE_PIN D17 [get_ports {rgb[7]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {rgb[7]}]
set_property PACKAGE_PIN P19 [get_ports hsync]
set_property IOSTANDARD LVCMOS33 [get_ports hsync]
set_property PACKAGE_PIN R19 [get_ports vsync]
set_property IOSTANDARD LVCMOS33 [get_ports vsync]

## Seven-segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
set_property PACKAGE_PIN V7 [get_ports dp]
set_property IOSTANDARD LVCMOS33 [get_ports dp]

set_property PACKAGE_PIN U2 [get_ports {an[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]
```