Lab7Part1

// Background image display

module background

    (

        CLOCK\_50,                        //    On Board 50 MHz

        KEY,                            //    Push Button[0:0]

        VGA\_CLK,                           //    VGA Clock

        VGA\_HS,                            //    VGA H\_SYNC

        VGA\_VS,                            //    VGA V\_SYNC

        VGA\_BLANK,                        //    VGA BLANK

        VGA\_SYNC,                        //    VGA SYNC

        VGA\_R,                           //    VGA Red[9:0]

        VGA\_G,                            //    VGA Green[9:0]

        VGA\_B                             //    VGA Blue[9:0]

    );

    input    CLOCK\_50;                //    50 MHz

    input    [0:0] KEY;                //    Button[0:0]

    output    VGA\_CLK;                   //    VGA Clock

    output    VGA\_HS;                    //    VGA H\_SYNC

    output    VGA\_VS;                    //    VGA V\_SYNC

    output    VGA\_BLANK;                //    VGA BLANK

    output    VGA\_SYNC;                //    VGA SYNC

    output    [9:0] VGA\_R;               //    VGA Red[9:0]

    output    [9:0] VGA\_G;            //    VGA Green[9:0]

    output    [9:0] VGA\_B;               //    VGA Blue[9:0]

    wire resetn, gnd;

    assign resetn = KEY[0];

    assign gnd = 1'b0;

    // Create the color, x, y and writeEn wires that are inputs to the controller.

    wire [2:0] color;

    wire [7:0] x;

    wire [6:0] y;

    assign x = 8'd0;

    assign y = 7'd0;

    assign color = 3'd0;

    // Create an Instance of a VGA controller - "There can be only one!"

    // Define the number of colours as well as the initial background

    // image file (.MIF) for the controller.

    vga\_adapter VGA(

            .resetn(resetn),

            .clock(CLOCK\_50),

            .colour(color),

            .x(x),

            .y(y),

            .plot(gnd),

            /\* Signals for the DAC to drive the monitor. \*/

            .VGA\_R(VGA\_R),

            .VGA\_G(VGA\_G),

            .VGA\_B(VGA\_B),

            .VGA\_HS(VGA\_HS),

            .VGA\_VS(VGA\_VS),

            .VGA\_BLANK(VGA\_BLANK),

            .VGA\_SYNC(VGA\_SYNC),

            .VGA\_CLK(VGA\_CLK));

        defparam VGA.RESOLUTION = "160x120";

        defparam VGA.MONOCHROME = "FALSE";

        defparam VGA.BITS\_PER\_COLOUR\_CHANNEL = 1;

        defparam VGA.BACKGROUND\_IMAGE = "display.mif";

endmodule

Lab7Part1b

// Background image display

module background

    (

        CLOCK\_50,                        //    On Board 50 MHz

        SW,                            //    Push Button[0:0]

        VGA\_CLK,                           //    VGA Clock

        VGA\_HS,                            //    VGA H\_SYNC

        VGA\_VS,                            //    VGA V\_SYNC

        VGA\_BLANK,                        //    VGA BLANK

        VGA\_SYNC,                        //    VGA SYNC

        VGA\_R,                           //    VGA Red[9:0]

        VGA\_G,                            //    VGA Green[9:0]

        VGA\_B,

        KEY

        //    VGA Blue[9:0]

    );

    input    CLOCK\_50;                //    50 MHz

    input    [3:0] KEY;                //    Button[0:0]

    input    [17:0] SW;                //    Button[0:0]

    output    VGA\_CLK;                   //    VGA Clock

    output    VGA\_HS;                    //    VGA H\_SYNC

    output    VGA\_VS;                    //    VGA V\_SYNC

    output    VGA\_BLANK;                //    VGA BLANK

    output    VGA\_SYNC;                //    VGA SYNC

    output    [9:0] VGA\_R;               //    VGA Red[9:0]

    output    [9:0] VGA\_G;            //    VGA Green[9:0]

    output    [9:0] VGA\_B;               //    VGA Blue[9:0]

    wire resetn, plot;

    wire [2:0] color;

    wire [7:0] x;

    wire [6:0] y;

    // Further assignments go here...

    // Define the number of colours as well as the initial background

    // image file (.MIF) for the controller.

    vga\_adapter VGA(

            .resetn(KEY[0]),

            .clock(CLOCK\_50),

            .colour(SW[17:15]),

            .x(SW[7:0]),

            .y(SW[14:8]),

            .plot(~KEY[1]),

            /\* Signals for the DAC to drive the monitor. \*/

            .VGA\_R(VGA\_R),

            .VGA\_G(VGA\_G),

            .VGA\_B(VGA\_B),

            .VGA\_HS(VGA\_HS),

            .VGA\_VS(VGA\_VS),

            .VGA\_BLANK(VGA\_BLANK),

            .VGA\_SYNC(VGA\_SYNC),

            .VGA\_CLK(VGA\_CLK));

        defparam VGA.RESOLUTION = "160x120";

        defparam VGA.MONOCHROME = "FALSE";

        defparam VGA.BITS\_PER\_COLOUR\_CHANNEL = 1;

        defparam VGA.BACKGROUND\_IMAGE = "display.mif";

endmodule

lab7Part2

// Etch-and-sketch

module sketch

    (

        CLOCK\_50,//    On Board 50 MHz

        KEY,    //    Push Button[3:0]

        SW,   //    DPDT Switch[17:0]

        VGA\_CLK,       //    VGA Clock

        VGA\_HS,    //    VGA H\_SYNC

        VGA\_VS,    //    VGA V\_SYNC

        VGA\_BLANK,        //    VGA BLANK

        VGA\_SYNC,        //    VGA SYNC

        VGA\_R,           //    VGA Red[9:0]

        VGA\_G,    //    VGA Green[9:0]

        VGA\_B       //    VGA Blue[9:0]

    );

    input            CLOCK\_50;                //    50 MHz

    input    [3:0]    KEY;                    //    Button[3:0]

    input    [17:0]    SW;                        //    Switches[17:0]

    output            VGA\_CLK;                   //    VGA Clock

    output            VGA\_HS;            //    VGA H\_SYNC

    output            VGA\_VS;                    //    VGA V\_SYNC

    output            VGA\_BLANK;                //    VGA BLANK

    output            VGA\_SYNC;                //    VGA SYNC

    output    [9:0]    VGA\_R;                   //    VGA Red[9:0]

    output    [9:0]    VGA\_G;                    //    VGA Green[9:0]

    output    [9:0]    VGA\_B;                   //    VGA Blue[9:0]

    wire resetn;

    assign resetn = KEY[0];

    // Create the color, x, y and writeEn wires that are inputs to the controller.

    reg [2:0] color;

    reg [7:0] x;

    reg [6:0] y;

    wire writeEn;

    wire [2:0]inputColor;

    wire [7:0]inputX;

    wire [6:0]inputY;

    assign inputX = SW[7:0];

    assign inputY = SW[14:8];

    assign inputColor = SW[17:15];

    assign writeEn = ~KEY[1];

    // Create an Instance of a VGA controller - there can be only one!

    // Define the number of colours as well as the initial background

    // image file (.MIF) for the controller.

    vga\_adapter VGA(

            .resetn(resetn),

            .clock(CLOCK\_50),

            .colour(color),

            .x(x),

            .y(y),

            .plot(~resetn||~writeEn),

            /\* Signals for the DAC to drive the monitor. \*/

            .VGA\_R(VGA\_R),

            .VGA\_G(VGA\_G),

            .VGA\_B(VGA\_B),

            .VGA\_HS(VGA\_HS),

            .VGA\_VS(VGA\_VS),

            .VGA\_BLANK(VGA\_BLANK),

            .VGA\_SYNC(VGA\_SYNC),

            .VGA\_CLK(VGA\_CLK));

        defparam VGA.RESOLUTION = "160x120";

        defparam VGA.MONOCHROME = "FALSE";

        defparam VGA.BITS\_PER\_COLOUR\_CHANNEL = 1;

        defparam VGA.BACKGROUND\_IMAGE = "display.mif";

    reg [2:0]currentState,nextState;

    parameter A=3'b000, B=3'b001, C=3'b010;

    parameter D=3'b011, E=3'b100;

    reg [2:0]countX,countY;

    reg [7:0]resetCountX;

    reg [6:0]resetCountY;

    always@(\*) // describe state transitions

        case(currentState)

            A: if(!resetn) nextState = D;

                else if(!writeEn) nextState = B;

                else nextState = A;

            B: if(countX<5) nextState = B;

                else nextState = C;

            C: if(countY<5) nextState = B;

                else nextState = A;

            D: if(resetCountX<159) nextState = D;

                else nextState = E;

            E: if(resetCountY<119) nextState = D;

                else nextState = A;

        endcase

    always@(posedge CLOCK\_50)

        begin

            currentState <= nextState;

        end

    always @(posedge CLOCK\_50)

        case(currentState)

            A: if(!resetn)

                    begin

                        x <= 8'b00000000;

                        y <= 7'b0000000;

                        resetCountX <= 8'b00000000;

                        resetCountY <= 7'b0000000;

                    end

                else if(!writeEn)

                    begin

                        x <= inputX;

                        y <= inputY;

                        color <= inputColor;

                        countX <= 3'b000;

                        countY <= 3'b000;

                    end

            B: if(countX<5)

                    begin

                        countX <= countX + 1;

                        x <= x + 1;

                    end

            C: if(countY<5)

                    begin

                        if(countY<2)

                            color <= inputColor;

                        else

                            color <= ~inputColor;

                        countY <= countY + 1;

                        y <= y + 1;

                        x <= inputX;

                        countX <= 3'b000;

                    end

                else

                    begin

                        countX <= 3'b000;

                        countY <= 3'b000;

                        x <= inputX;

                        y <= inputY;

                        color <= inputColor;

                    end

            D: if(resetCountX<159)

                    begin

                        color <= 3'b000;

                        resetCountX <= resetCountX + 1;

                        x <= x + 1;

                    end

            E: if(resetCountY<119)

                    begin

                        resetCountY <= resetCountY + 1;

                        resetCountX <= 8'b00000000;

                        y <= y + 1;

                        x <= 8'b00000000;

                    end

                else

                    begin

                        resetCountX <= 8'b00000000;

                        resetCountY <= 7'b0000000;

                        x <= inputX;

                        y <= inputY;

                    end

        endcase

endmodule

module sketch

    (

        CLOCK\_50,                        //    On Board 50 MHz

        KEY,                            //    Push Button[3:0]

        SW,                                //    DPDT Switch[17:0]

        VGA\_CLK,                           //    VGA Clock

        VGA\_HS,                            //    VGA H\_SYNC

        VGA\_VS,                            //    VGA V\_SYNC

        VGA\_BLANK,                        //    VGA BLANK

        VGA\_SYNC,                        //    VGA SYNC

        VGA\_R,                           //    VGA Red[9:0]

        VGA\_G,                            //    VGA Green[9:0]

        VGA\_B                           //    VGA Blue[9:0]

    );

    input            CLOCK\_50;                //    50 MHz

    input    [3:0]    KEY;                    //    Button[3:0]

    input    [17:0]    SW;                        //    Switches[17:0]

    output            VGA\_CLK;                   //    VGA Clock

    output            VGA\_HS;                    //    VGA H\_SYNC

    output            VGA\_VS;                    //    VGA V\_SYNC

    output            VGA\_BLANK;                //    VGA BLANK

    output            VGA\_SYNC;                //    VGA SYNC

    output    [9:0]    VGA\_R;                   //    VGA Red[9:0]

    output    [9:0]    VGA\_G;                    //    VGA Green[9:0]

    output    [9:0]    VGA\_B;                   //    VGA Blue[9:0]

    wire resetn;

    assign resetn = KEY[0];

    // Create the color, x, y and writeEn wires that are inputs to the controller.

  reg [7:0]movex;

    reg [6:0]movey;

    reg [2:0]newcolor;

    //reg [7:0]resetx;

    //reg [6:0]resety;

    wire [2:0] color;

    wire [7:0] x;

    wire [6:0] y;

    wire writeEn;

  assign writeEn = ~KEY[1];

    assign x = SW[7:0];

    assign y = SW[14:8];

    assign color = SW[17:15];

    reg [2:0]countx;

    reg [2:0]county;

    reg[2:0]y\_Q, Y\_D;

    parameter A = 3'b000, B = 3'b001, C = 3'b010, D = 3'b011, E = 3'b100;

    always@(\*)

    begin

    case(y\_Q)

    A: if(!writeEn) Y\_D = B;

      else if(!resetn) Y\_D = D;

        else Y\_D = A;

    B: if(countx < 3'd5) Y\_D = B;

      else Y\_D = C;

    C: if(county < 3'd5) Y\_D = B;

      else Y\_D = A;

    D: if(movex < 8'd159) Y\_D = D;

      else Y\_D = E;

    E: if(movey < 7'd119) Y\_D = D;

      else Y\_D = A;

    default Y\_D = 3'bxxx;

    endcase

    end

    always@(posedge CLOCK\_50)

    begin

    y\_Q <= Y\_D;

    end

    always@(posedge CLOCK\_50)

    begin

    case(y\_Q)

    A: if(!resetn)

     begin

      movex <= 8'b00000000;

        movey <= 7'b0000000;

     end

     else if(!writeEn)

     begin

     movex <= x;

     movey <= y;

     newcolor <= color;

     countx <= 3'b000;

     county <= 3'b000;

     end

  B: if(countx < 3'd5)

     begin

      countx <= countx + 1;

        movex <= movex + 1;

        end

    C: if(county < 3'd5)

        begin

         if (county < 3'd2)

          begin

          newcolor <= color;

            county <= county + 1;

            end

            else

          begin

            newcolor <= ~color;

            county <= county + 1;

            end

        movex <= x;

        countx <= 3'b000;

        end

        else

        begin

         countx <= 3'b000;

         county <= 3'b000;

         movex <= x;

         movey <= y;

         newcolor <= color;

        end

    D: if(movex < 8'd159)

      begin

        newcolor <= 3'b000;

      movex <= movex + 1;

        end

    E: if(movey < 7'd119)

      begin

        movex <= 8'b00000000;

        movey <= movey + 1;

        end

      else

      begin

      movex <= 8'b00000000;

      movey <= 7'b0000000;

      end

    endcase

    end

    // Create an Instance of a VGA controller - there can be only one!

    // Define the number of colours as well as the initial background

    // image file (.MIF) for the controller.

    vga\_adapter VGA(

            .resetn(resetn),

            .clock(CLOCK\_50),

            .colour(color),

            .x(movex),

            .y(movey),

            .plot(!writeEn||!resetn),

            /\* Signals for the DAC to drive the monitor. \*/

            .VGA\_R(VGA\_R),

            .VGA\_G(VGA\_G),

            .VGA\_B(VGA\_B),

            .VGA\_HS(VGA\_HS),

            .VGA\_VS(VGA\_VS),

            .VGA\_BLANK(VGA\_BLANK),

            .VGA\_SYNC(VGA\_SYNC),

            .VGA\_CLK(VGA\_CLK));

        defparam VGA.RESOLUTION = "160x120";

        defparam VGA.MONOCHROME = "FALSE";

        defparam VGA.BITS\_PER\_COLOUR\_CHANNEL = 1;

        defparam VGA.BACKGROUND\_IMAGE = "display.mif";

    // Put your code here. Your code should produce signals x,y,color and writeEn

    // for the VGA controller, in addition to any other functionality your design may require.

endmodule

