```
`timescale 1ns / 1ns

module drawSquare
    (
        CLOCK_50,                       //  On Board 50 MHz
        // Your inputs and outputs here
        KEY,                            // On Board Keys
        SW,
        // The ports below are for the VGA output.  Do not change.
        VGA_CLK,                        //  VGA Clock
        VGA_HS,                         //  VGA H_SYNC
        VGA_VS,                         //  VGA V_SYNC
        VGA_BLANK_N,                        //  VGA BLANK
        VGA_SYNC_N,                     //  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;
    // Declare your inputs and outputs here
    input [9:0] SW;
    // Do not change the following outputs
    output          VGA_CLK;                //  VGA Clock
    output          VGA_HS;                 //  VGA H_SYNC
    output          VGA_VS;                 //  VGA V_SYNC
    output          VGA_BLANK_N;                //  VGA BLANK
    output          VGA_SYNC_N;             //  VGA SYNC
    output  [7:0]   VGA_R;                  //  VGA Red[7:0] Changed from 10 to 8-bit DAC
    output  [7:0]   VGA_G;                  //  VGA Green[7:0]
    output  [7:0]   VGA_B;                  //  VGA Blue[7:0]

    wire resetn;
    assign resetn = KEY[0]; // Active low, so don't invert

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

    wire [2:0] colour;
    wire [7:0] x;
    wire [6:0] y;
    wire writeEn;

    // 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(colour),
            .x(x),
            .y(y),
            .plot(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_N),
            .VGA_SYNC(VGA_SYNC_N),
            .VGA_CLK(VGA_CLK));
        defparam VGA.RESOLUTION = "160x120";
        defparam VGA.MONOCHROME = "FALSE";
        defparam VGA.BITS_PER_COLOUR_CHANNEL = 1;
        defparam VGA.BACKGROUND_IMAGE = "black.mif";

    // Put your code here. Your code should produce signals x,y,colour and writeEn
    // for the VGA controller, in addition to any other functionality your design may
    require.

    // Input wires
    wire[6:0] DATA_IN;
    wire[2:0] C_DATA;
    wire go,clear,plot;

    // Control wires
    wire ld_x,ld_y,ld_colour;
    wire ld_xpos,ld_ypos;
    wire set_black,draw_pixel;
    wire[7:0] dx,dy;

    // Assign inputs
    assign plot     = ~KEY[1];
    assign go       = ~KEY[2];
    assign clear    = ~KEY[3];
    assign DATA_IN = SW[6:0];
    assign C_DATA   = SW[9:7];

    // Control module
    control c0(
        .clock(CLOCK_50),.resetn(resetn),
        .go(go),.plot(plot),.clear(clear),
        .ld_x(ld_x),.ld_y(ld_y),.ld_colour(ld_colour),
        .ld_xpos(ld_xpos),.ld_ypos(ld_ypos),
        .set_black(set_black),.draw_pixel(draw_pixel),
        .dx_out(dx),.dy_out(dy)
    );

    // Controls plotting on VGA
    assign writeEn = draw_pixel;

    // Datapath module
    datapath d0(
        .clock(CLOCK_50),.resetn(resetn),
        .DATA_IN(DATA_IN),.COLOUR_DATA(C_DATA),
        .ld_x(ld_x),.ld_y(ld_y),.ld_colour(ld_colour),
        .ld_xpos(ld_xpos),.ld_ypos(ld_ypos),
        .set_black(set_black),.dx(dx),.dy(dy),
        .xpos(x),.ypos(y),.colour(colour)
    );
endmodule

// Tracks state and datapath control signals depending on state
module control(
    input clock,resetn,
    input go,plot,clear,
    output reg ld_x,ld_y,ld_colour,
    output reg ld_xpos,ld_ypos,
    output reg set_black,draw_pixel,
    output reg[7:0] dx,dy
    );

    // Parameters for counters
    localparam  X_SHAPE     = 8'd4,
                Y_SHAPE = 8'd4,
                X_SCREEN = 8'd160,
                Y_SCREEN = 8'd120;

    // Keeps track of state
    reg[3:0] current_state,next_state;
    // dx and dy are counters that count up to x and y size
    // Used to draw shapes and clear screen
    reg[7:0] x_size,y_size;
    reg reset_dx,reset_dy,inc_dx,inc_dy;
    reg set_size,size_select;

    // Assigning state variables
```

```
localparam  S_LOAD_X                = 4'd0,
                S_LOAD_X_WAIT           = 4'd1,
                S_LOAD_Y                = 4'd2,
                S_LOAD_Y_WAIT           = 4'd3,
                S_LOAD_COLOUR           = 4'd4,
                S_LOAD_COLOUR_WAIT  = 4'd5,
                S_CLEAR_SCREEN          = 4'd6,
                S_CYCLE_0               = 4'd7,
                S_CYCLE_1               = 4'd8,
                S_CYCLE_2               = 4'd9,
                S_CYCLE_3               = 4'd10,
                S_CYCLE_4               = 4'd11,
                S_CYCLE_5               = 4'd12;

// Counter registers for dx and dy
always @(posedge clock)
begin
    if(!resetn) // Active low reset
    begin
        dx <= 8'b0;
        dy <= 8'b0;
    end
    else
    begin
        // dx counter
        if(reset_dx)
            dx <= 8'b0;
        else if(inc_dx)
            dx <= dx+1;

        // dy counter
        if(reset_dy)
            dy <= 8'b0;
        else if(inc_dy)
            dy <= dy+1;
    end
end

// Registers for x and y size
always @(posedge clock)
begin
    if(!resetn) // Active low reset to shape size
    begin
        x_size <= X_SHAPE;
        y_size <= Y_SHAPE;
    end
    else if(set_size) // Set the size to selected size
    begin
        x_size <= size_select ? X_SHAPE:X_SCREEN;
        y_size <= size_select ? Y_SHAPE:Y_SCREEN;
    end
end

// State table
always @(*)
begin
    case(current_state)
        S_LOAD_X: // Loop in state until value is input
            next_state = go ? S_LOAD_X_WAIT:S_LOAD_X;
        S_LOAD_X_WAIT: // Loop in state until go signal goes low
            next_state = go ? S_LOAD_X_WAIT:S_LOAD_Y;
        S_LOAD_Y: // Loop in state until value is input
            next_state = go ? S_LOAD_Y_WAIT:S_LOAD_Y;
        S_LOAD_Y_WAIT: // Loop in state until go signal goes low
            next_state = go ? S_LOAD_Y_WAIT:S_LOAD_COLOUR;
        S_LOAD_COLOUR: // Loop in state until value is input
            next_state = go ? S_LOAD_COLOUR_WAIT:S_LOAD_COLOUR;
        S_LOAD_COLOUR_WAIT: // Loop until go goes low and plot goes high
            next_state = (go | !plot) ? S_LOAD_COLOUR_WAIT:S_CYCLE_0;
        S_CLEAR_SCREEN: // Clears (x and y size = screen size), reset dy
            next_state = S_CYCLE_1;
        S_CYCLE_0: // Sets x and y size to shape size, reset dy
            next_state = S_CYCLE_1;
        S_CYCLE_1: // Reset dx
            next_state = S_CYCLE_2;
        S_CYCLE_2: // Set actual x and y positions
            next_state = S_CYCLE_3;
        S_CYCLE_3: // Fill pixel with colour
            next_state = S_CYCLE_4;
        S_CYCLE_4: // Increment dx
            next_state = (dx==x_size-1) ? S_CYCLE_5:S_CYCLE_2;
        S_CYCLE_5: // Increment dy
            next_state = (dy==y_size-1) ? S_LOAD_X:S_CYCLE_1;
        default: next_state = S_LOAD_X;
    endcase
end

// Changing data control signals
always @(*)
begin
    // Initializing signals to 0 to avoid latches
    // Internal controls
    reset_dx = 0; reset_dy = 0; inc_dx = 0; inc_dy = 0;
    set_size = 0; size_select = 0;
    // External controls
    ld_x = 0; ld_y = 0; ld_colour = 0;
    ld_xpos = 0; ld_ypos = 0;
    set_black = 0; draw_pixel = 0;

    case(current_state)
        S_LOAD_X: // Load x
        begin
            ld_x = 1;
        end
        S_LOAD_Y: // Load y
        begin
            ld_y = 1;
        end
        S_LOAD_COLOUR: // Load colour
        begin
            ld_colour = 1;
        end
        S_CLEAR_SCREEN: // Clears (x and y size = screen size), reset dy
        begin
            reset_dy = 1;
            set_size = 1; size_select = 0; // Screen size
            set_black = 1;
        end
        S_CYCLE_0: // Sets x and y size to shape size, reset dy
        begin
            reset_dy = 1;
            set_size = 1; size_select = 1; // Shape size
        end
        S_CYCLE_1: // Reset dx
        begin
            reset_dx = 1;
        end
        S_CYCLE_2: // Set actual x and y positions
        begin
            ld_xpos = 1;
            ld_ypos = 1;
        end
        S_CYCLE_3: // Fill pixel with colour
        begin
            draw_pixel = 1;
        end
        S_CYCLE_4: // Increment dx
        begin
```

```
                inc_dx = 1;
            end
            S_CYCLE_5: // Increment dy
            begin
                inc_dy = 1;
            end
        endcase
    end

    // Register for current state
    always @(posedge clock)
    begin
        if(!resetn) // Reset to value input, active low
            current_state <= S_LOAD_X;
        else if(clear)
            current_state <= S_CLEAR_SCREEN;
        else // Load next state
            current_state <= next_state;
    end
endmodule

// Modifies data and outputs depending on control signals
module datapath(
    input clock,resetn,
    input[6:0] DATA_IN,
    input[2:0] COLOUR_DATA,
    input ld_x,ld_y,ld_colour,
    input ld_xpos,ld_ypos,
    input set_black,
    input[7:0] dx,dy,
    output reg[7:0] xpos,
    output reg[6:0] ypos,
    output reg[2:0] colour
    );

    // Internal registers
    reg[7:0] x,y;

    // Registers x,y,xpos,ypos and colour with input logic
    always @(posedge clock)
    begin
        if(!resetn) // Active low reset
        begin
            x <= 8'b0;
            y <= 8'b0;
            xpos <= 8'b0;
            ypos <= 7'b0;
            colour <= 3'b0;
        end
        else
        begin
            // x register
            if(ld_x)
                x <= {1'b0,DATA_IN};
            else if(set_black)
                x <= 8'b0;
            // y register
            if(ld_y)
                y <= {1'b0,DATA_IN};
            else if(set_black)
                y <= 8'b0;
            // xpos register
            if(ld_xpos)
                xpos <= x+dx;
            // ypos register
            if(ld_ypos)
                ypos <= y[6:0]+dy[6:0];
            // colour register
            if(ld_colour)
```

```
                colour <= COLOUR_DATA;
            else if(set_black)
                colour <= 3'b0;
        end
    end
endmodule

```