ECE241 Final Project

Brick Breaker

Name: Anas Ahmed

Student Number: 1000441328

Date: 29-11-2014

Introduction:

I chose to create Brick Breaker for my ECE241 project based on two reasons: complexity and interest. Before I expand on the reasons, I will give you some background on the project. Brick Breaker is a clone of an arcade game named Breakout, which was quite popular in the past. The game design is simple, it involves using a paddle which is controlled by the user to hit a moving ball into blocks on the screen. Once the ball hits a block, the block disappears. This continues on until all blocks are gone and the level is complete. On a high level, this game is a really simple game to create, but when you create it on a hardware level, that is when things become complex. When you do not have access to high level programming concepts such as loops, functions, and objects; and have to resort to flip flops, finite state machines, and logical expressions, the project becomes far difficult then what you would initially have expected. I wanted to create a project that challenged me and at the same time catered to my interests. The reason why I think Brick Breaker was an interesting idea is because it is a game, and creating a game is almost as fun as playing it. By creating a project that is a game, I was kept interested and motivated and even enjoyed working on it.

My plan for creating this project started at getting things to first draw on the screen using the VGA module. Then I would proceed to use inputs from the keys on the FPGA to control the paddle. Next I would get a ball moving and colliding with the paddle and walls. Lastly the most complex part, I would get blocks on the screen with collision detection with the wall. My goal was to complete the basic game first and then when it was done I would add more extra features if I had the time.

The Design:

The Verilog code has three major modules. The first one was called BrickBreaker, which was the top level module that takes in every input needed and runs the appropriate sub modules and also wires the sub modules together. The two sub modules are FSM_Main and FSM_Draw. The first one has the main “loop” of the program and deals with the response of the ball, paddle, and bricks. The next one was FSM_Draw. This module draws rectangles to the screen representing the elements (ball,paddle,brick). When everything is put together, the working project is complete.

Detail explanation of the main modules:

BrickBreaker:

-takes in inputs from keys, and outputs to VGA port

-takes in the 50 MHz clock in the DE2 board

-instantiates FSM_Main and FSM_Draw modules

-wires the FSM_Main module positions and other info to the FSM_Draw module

-initializes the VGA adapter

-wires the color, draw position, and enable of FSM_Display module to the VGA adapter

FSM_Draw:

-takes inputs of position of ball and paddle and what bricks are activated

-has a FSM of the draw process which remains in the idle state unless activated

-once activated, it draws a blank background, then draws paddle and ball, then draws bricks

-the plot signal is only ON if a brick is activated, and turns ON when needed for ball and paddle

-the FSM iterates over the x and y pixels and also over x and y relative block positions

-it is only activated during the DISPLAY phase of the FSM_Main else it is in idle

FSM_Main:

-stores and outputs the positions of paddle and ball

-stores the direction ball is moving

-stores and outputs the bricks which are activated

-outputs the draw signal to FSM_Draw, and inputs the draw complete signal

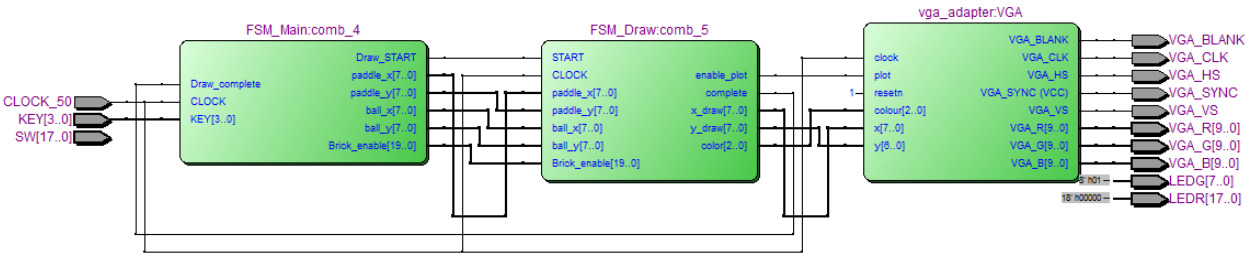
-has a FSM of the main game process that loops 60 times a second

-the update state changes the position of ball and paddle based on the input of keys

-the display state activates FSM_Draw module

-the wait state loops until enough clock cycle have passed such that the game loop frequency is 60 Hz

Block Diagram:



Collision Detection:

The collision detection was coded as “if statements” during the update phase which detects when the ball hits a paddle or wall. To get collision with wall working, more than 20 sets of “if statements” was needed to process each of the bricks. Each of these conditions checks where the ball collides relative to the brick (top, bottom, left, right) and makes the appropriate changes (i.e. reflects

the balls direction and disables the present brick). To make this task easier, I scripted a C++ program that creates the necessary Verilog code to do the collision detection of the bricks. This is possible because the bricks are placed equal distance apart uniformly. That way I did not need to manually code a tremendous amount of “if statements”.

Ball Movement:

The ball is designed to move in two degrees of freedom where it could go left or right and up or down. This is stored in a reg variable that holds a 1 or 0 depending on direction. In the update state of the main FSM the balls position is updated.

Paddle Movement:

If a key is pressed the paddle should not update instantly every game loop or else it will move too fast on screen. For that reason, I implemented another counter that increments every game loop. When that counter reaches odd values only then would the paddle updates its position. This effectively moves the paddle at a realistic pace.

Report on Success

Overall, this project was a huge success and worked better than I expected. The collision detection worked almost perfectly and pacing of the game was natural. By that I mean it was challenging to play but at the same time possible to win it (by eliminating all the blocks). I did not have enough time to add any extra features besides the bare minimum, but the way I implemented the design, it can easily be upgraded to provide more levels (different number of bricks) by changing the spacing between bricks and adding more rows and/or columns. Below are some images of the working project, and in the appendix is a short video of a demonstration of the project in action.

The only bug my game had was if the ball happen to hit exactly at the corner of a brick or paddle, then undefined behaviour would occur. This is because there is only conditions for if the ball hits on sides of objects, not on the corners. This bug was difficult to spot because it takes time and effort to

force the ball to hit exactly at the corners. This can be fixed by adding more conditional statements to each of the brick and the paddle.

What would you do differently:

If I were to redo this project, I would first get the draw FSM working as soon as possible. I realized that once you have something to draw on the screen through the VGA adapter, it becomes easier to do the rest of the project. That way I would be able to get instant feedback of how the program is working which makes it easier to debug. I would also try to implement multiple levels with different brick layouts to make the game more interesting. The way my game is implemented now is that it is not flexible in implementing game layouts other than a grid pattern. Aside from that, there aren't much things I need to differently because changing anything then what it is now would be just an add-on feature, the base game will still remains the same.

Appendix:

Dropbox link to video:

https://www.dropbox.com/s/bzb8liz9ok6dzw3/VID_20141124_011808.mp4?dl=0

Code:

```
/*
Anas Ahmed
November 2014
ECE241 Digital Systems Final Project
Brick Breaker
*/

module FSM_Main(paddle_x,paddle_y,ball_x,ball_y, Brick_enable, Draw_START,Draw_complete, KEY, CLOCK);
	input CLOCK;
	input [3:0] KEY; //controls for paddle
	output reg [19:0] Brick_enable; //holds which blocks are enabled as 1
	output reg [7:0] paddle_x,paddle_y,ball_x,ball_y;
	reg x_dir,y_dir; //stores 1 or 0 (direction of ball)
	//used to activate Draw_FSM
	output Draw_START;
	input Draw_complete;
	reg [25:0] clockCount;//stores present clock cycle
	reg [5:0] clockCount60; //increment 60 times a second
	reg [3:0] state; //state variables
	reg [3:0] Nstate;
	parameter [3:0]	INITIAL=4'd0,
							UPDATE= 4'd1,
							DISPLAY=4'd2,
							WAIT = 4'd3,
							DONE = 4'd4;

	assign Draw_START=(state==DISPLAY);//only starts the draw module during DISPLAY state

	always@(*)//new state assignment
	begin
		case(state)
			INITIAL: Nstate<=UPDATE;
```

```
                UPDATE:   if (ball_y>paddle_y+6 || Brick_enable==0) Nstate<=INITIAL;// resets game if ball falls out or all
blocks disabled
                                        else Nstate<=DISPLAY;

                DISPLAY: if (Draw_complete) Nstate<=DISPLAY;
                                         else Nstate<=WAIT;

                WAIT:     if (clockCount < 26'd833_333) Nstate<=WAIT;          //waits till appropriate time has passed
(1/60th of a second)
                                        else Nstate<=DONE;

                DONE:   Nstate<=UPDATE;
                default: Nstate<=INITIAL;
        endcase
    end

    always@(posedge CLOCK)// Operations during states
    begin
        case(state)
            INITIAL:
                begin
                Brick_enable<=20'b1111_1111_1111_1111_1111;//enables all blocks
                //assigns starting positions and directions
                paddle_x<=75;
                paddle_y<=110;
                ball_x<=122;
                ball_y<=47;
                x_dir=0;
                y_dir=1;
                clockCount60<=0;
                end

            UPDATE:
                begin
                //code generated with a c++ program that checks condition for when ball collides with each block
                if (Brick_enable[0])
                if (ball_x>=35 && ball_x <=40 && (ball_y==19 || ball_y==26))
                begin Brick_enable[0]<=0;y_dir=~y_dir; end
                else if(ball_y>=20 && ball_y<=25 && (ball_x==34 || ball_x==41))
                begin Brick_enable[0]<=0; x_dir=~x_dir; end
                if (Brick_enable[1])
                if (ball_x>=45 && ball_x <=50 && (ball_y==19 || ball_y==26))
                begin Brick_enable[1]<=0;y_dir=~y_dir; end
                else if(ball_y>=20 && ball_y<=25 && (ball_x==44 || ball_x==51))
                begin Brick_enable[1]<=0; x_dir=~x_dir; end
                if (Brick_enable[2])
                if (ball_x>=55 && ball_x <=60 && (ball_y==19 || ball_y==26))
                begin Brick_enable[2]<=0;y_dir=~y_dir; end
                else if(ball_y>=20 && ball_y<=25 && (ball_x==54 || ball_x==61))
                begin Brick_enable[2]<=0; x_dir=~x_dir; end
                if (Brick_enable[3])
                if (ball_x>=65 && ball_x <=70 && (ball_y==19 || ball_y==26))
                begin Brick_enable[3]<=0;y_dir=~y_dir; end
                else if(ball_y>=20 && ball_y<=25 && (ball_x==64 || ball_x==71))
                begin Brick_enable[3]<=0; x_dir=~x_dir; end
                if (Brick_enable[4])
                if (ball_x>=75 && ball_x <=80 && (ball_y==19 || ball_y==26))
                begin Brick_enable[4]<=0;y_dir=~y_dir; end
                else if(ball_y>=20 && ball_y<=25 && (ball_x==74 || ball_x==81))
                begin Brick_enable[4]<=0; x_dir=~x_dir; end
```

```
if (Brick_enable[5])
if (ball_x>=85 && ball_x <=90 && (ball_y==19 || ball_y==26))
begin Brick_enable[5]<=0;y_dir=~y_dir; end
else if(ball_y>=20 && ball_y<=25 && (ball_x==84 || ball_x==91))
begin Brick_enable[5]<=0; x_dir=~x_dir; end
if (Brick_enable[6])
if (ball_x>=95 && ball_x <=100 && (ball_y==19 || ball_y==26))
begin Brick_enable[6]<=0;y_dir=~y_dir; end
else if(ball_y>=20 && ball_y<=25 && (ball_x==94 || ball_x==101))
begin Brick_enable[6]<=0; x_dir=~x_dir; end
if (Brick_enable[7])
if (ball_x>=105 && ball_x <=110 && (ball_y==19 || ball_y==26))
begin Brick_enable[7]<=0;y_dir=~y_dir; end
else if(ball_y>=20 && ball_y<=25 && (ball_x==104 || ball_x==111))
begin Brick_enable[7]<=0; x_dir=~x_dir; end
if (Brick_enable[8])
if (ball_x>=115 && ball_x <=120 && (ball_y==19 || ball_y==26))
begin Brick_enable[8]<=0;y_dir=~y_dir; end
else if(ball_y>=20 && ball_y<=25 && (ball_x==114 || ball_x==121))
begin Brick_enable[8]<=0; x_dir=~x_dir; end
if (Brick_enable[9])
if (ball_x>=125 && ball_x <=130 && (ball_y==19 || ball_y==26))
begin Brick_enable[9]<=0;y_dir=~y_dir; end
else if(ball_y>=20 && ball_y<=25 && (ball_x==124 || ball_x==131))
begin Brick_enable[9]<=0; x_dir=~x_dir; end

if (Brick_enable[10])
if (ball_x>=35 && ball_x <=40 && (ball_y==29 || ball_y==36))
begin Brick_enable[10]<=0;y_dir=~y_dir; end
else if(ball_y>=30 && ball_y<=35 && (ball_x==34 || ball_x==41))
begin Brick_enable[10]<=0; x_dir=~x_dir; end
if (Brick_enable[11])
if (ball_x>=45 && ball_x <=50 && (ball_y==29 || ball_y==36))
begin Brick_enable[11]<=0;y_dir=~y_dir; end
else if(ball_y>=30 && ball_y<=35 && (ball_x==44 || ball_x==51))
begin Brick_enable[11]<=0; x_dir=~x_dir; end
if (Brick_enable[12])
if (ball_x>=55 && ball_x <=60 && (ball_y==29 || ball_y==36))
begin Brick_enable[12]<=0;y_dir=~y_dir; end
else if(ball_y>=30 && ball_y<=35 && (ball_x==54 || ball_x==61))
begin Brick_enable[12]<=0; x_dir=~x_dir; end
if (Brick_enable[13])
if (ball_x>=65 && ball_x <=70 && (ball_y==29 || ball_y==36))
begin Brick_enable[13]<=0;y_dir=~y_dir; end
else if(ball_y>=30 && ball_y<=35 && (ball_x==64 || ball_x==71))
begin Brick_enable[13]<=0; x_dir=~x_dir; end
if (Brick_enable[14])
if (ball_x>=75 && ball_x <=80 && (ball_y==29 || ball_y==36))
begin Brick_enable[14]<=0;y_dir=~y_dir; end
else if(ball_y>=30 && ball_y<=35 && (ball_x==74 || ball_x==81))
begin Brick_enable[14]<=0; x_dir=~x_dir; end
if (Brick_enable[15])
if (ball_x>=85 && ball_x <=90 && (ball_y==29 || ball_y==36))
begin Brick_enable[15]<=0;y_dir=~y_dir; end
else if(ball_y>=30 && ball_y<=35 && (ball_x==84 || ball_x==91))
begin Brick_enable[15]<=0; x_dir=~x_dir; end
if (Brick_enable[16])
if (ball_x>=95 && ball_x <=100 && (ball_y==29 || ball_y==36))
begin Brick_enable[16]<=0;y_dir=~y_dir; end
else if(ball_y>=30 && ball_y<=35 && (ball_x==94 || ball_x==101))
begin Brick_enable[16]<=0; x_dir=~x_dir; end
```

```
                                        if (Brick_enable[17])
                                        if (ball_x>=105 && ball_x <=110 && (ball_y==29 || ball_y==36))
                                        begin Brick_enable[17]<=0;y_dir=~y_dir; end
                                        else if(ball_y>=30 && ball_y<=35 && (ball_x==104 || ball_x==111))
                                        begin Brick_enable[17]<=0; x_dir=~x_dir; end
                                        if (Brick_enable[18])
                                        if (ball_x>=115 && ball_x <=120 && (ball_y==29 || ball_y==36))
                                        begin Brick_enable[18]<=0;y_dir=~y_dir; end
                                        else if(ball_y>=30 && ball_y<=35 && (ball_x==114 || ball_x==121))
                                        begin Brick_enable[18]<=0; x_dir=~x_dir; end
                                        if (Brick_enable[19])
                                        if (ball_x>=125 && ball_x <=130 && (ball_y==29 || ball_y==36))
                                        begin Brick_enable[19]<=0;y_dir=~y_dir; end
                                        else if(ball_y>=30 && ball_y<=35 && (ball_x==124 || ball_x==131))
                                        begin Brick_enable[19]<=0; x_dir=~x_dir; end

                                        //checks when ball hits paddle
                                        if (ball_x >= paddle_x && ball_x <=(paddle_x+15) && ball_y >= (paddle_y-1) && ball_y <=
(paddle_y+1))
                                                y_dir=~y_dir;
                                        //checks when ball hits walls
                                        if (ball_x<=4 ||  ball_x>=158)
                                                x_dir=~x_dir;
                                        if (ball_y>=118 || ball_y<=0)
                                                y_dir=~y_dir;
                                        //updates ball position
                                        ball_x<= x_dir? ball_x+1: ball_x-1;
                                        ball_y<= y_dir? ball_y+1: ball_y-1;

                                        //only when clockCount60 is odd will it allow condition to occur
                                        //this prevents the paddle from moving too fast
                                        if (            clockCount60==0 || clockCount60==2 || clockCount60==4 || clockCount60==6 ||
clockCount60==8 ||
                                                                clockCount60==10 || clockCount60==12 || clockCount60==14 ||
clockCount60==16 || clockCount60==18 ||
                                                                clockCount60==20 || clockCount60==22 || clockCount60==24 ||
clockCount60==26 || clockCount60==28 ||
                                                                clockCount60==30 || clockCount60==32 || clockCount60==34 ||
clockCount60==36 || clockCount60==38 ||
                                                                clockCount60==40 || clockCount60==42 || clockCount60==44 ||
clockCount60==46 || clockCount60==48 ||
                                                                clockCount60==50 || clockCount60==52 || clockCount60==54 ||
clockCount60==56 || clockCount60==58)
                                        begin
                                                //updates position of paddle if key is pressed
                                                if (~KEY[3] && paddle_x>=6)
                                                        paddle_x<= paddle_x-3;
                                                else if(~KEY[2]&& paddle_x<=143)
                                                        paddle_x<= paddle_x+3;
                                        end
                                        //keep clockCount between 0 and 59
                                        clockCount60<=clockCount60>60? 0 : clockCount60+1;
                                        end
                                DISPLAY:
                                        begin
                                        end
                                WAIT:
                                        begin
                                        end
                                DONE:
                                        begin
```

```
                                        end
                                default:
                                        begin
                                        //similar to initial state
                                        paddle_x<=75;
                                        paddle_y<=110;
                                        ball_x<=79;
                                        ball_y<=108;
                                        x_dir=0;
                                        y_dir=1;
                                        clockCount60<=0;
                                        end
                        endcase
                end

                always@(posedge CLOCK)//update state
                begin
                        if (state==INITIAL||state==DONE)
                                clockCount<=0;
                        else
                                clockCount<=clockCount+1;
                        state<=Nstate;
                end
endmodule

module FSM_Draw(START,paddle_x,paddle_y,ball_x,ball_y,x_draw,y_draw, Brick_enable, color,enable_plot,complete,CLOCK);
                input START; //signal to begin FSM_Draw
                input [19:0] Brick_enable; //stores which bricks are enable
                reg [3:0] brick_counterX; //holds current brick relative x position on the brick grid
                reg [3:0] brick_counterY; //holds current brick relative y position on the brick grid

                input [7:0] paddle_x,paddle_y,ball_x,ball_y; //positions
                output [7:0] x_draw,y_draw; //position of the pixel to draw
                output [2:0] color;

                input CLOCK;
                output enable_plot;
                output complete; //signal turns on when FSM_Draw ends in DONE state

                reg [7:0] x,y;
                reg [3:0] state;
                reg [3:0] Nstate;

                parameter [3:0] IDLE=4'd0,
                                                        ITERX_clear = 4'd1,
                                                        ITERY_clear=4'd2,

                                                        PADDLE = 4'd3,
                                                        ITERX_paddle=4'd4,
                                                        ITERY_paddle=4'd5,

                                                        BALL = 4'd6,
                                                        ITERX_ball=4'd7,
                                                        ITERY_ball=4'd8,

                                                        BRICK = 4'd10,
                                                        BRICK_incX = 4'd11,
                                                        BRICK_incY = 4'd14,
                                                        ITERX_brick = 4'd12,
                                                        ITERY_brick = 4'd13,
```

```
                                        DONE=4'd9;

        //based on state, the pixel position is different
        assign x_draw= (state==ITERX_clear) ? x : (state==ITERX_paddle) ? paddle_x+x : (state==ITERX_ball) ? ball_x+x :
35+10*brick_counterX+x;
        assign y_draw= (state==ITERX_clear) ? y : (state==ITERX_paddle) ? paddle_y+y : (state==ITERX_ball) ? ball_y+y :
20+10*brick_counterY+y;
        assign color= (state==ITERX_clear) ? 3'b000: 3'b111; //color is black for background and white for paddle and ball
        //only enable plot if drawing ball or paddle, or drawing bricks that are enabled
        assign enable_plot=(state==ITERX_clear || state==ITERX_paddle || state==ITERX_ball || (state==ITERX_brick &&
Brick_enable[brick_counterX+10*brick_counterY]));
        assign complete=(state==DONE);

        always@(*)//new state assignment
        begin
                case(state)
                        IDLE: if(START) Nstate<=ITERX_clear;
                                else Nstate<=IDLE;

                        ITERX_clear: if(x<160)          Nstate<=ITERX_clear; //when the background if drawn
                                                        else Nstate<=ITERY_clear;

                        ITERY_clear: if(y<120)  Nstate<=ITERX_clear;
                                                        else Nstate<=PADDLE;

                        PADDLE: Nstate<=ITERX_paddle;

                        ITERX_paddle: if(x<15) Nstate<=ITERX_paddle; //when the paddle is drawn
                                                        else    Nstate<=ITERY_paddle;

                        ITERY_paddle: if(y<1) Nstate<=ITERX_paddle;
                                                        else Nstate<=BALL;

                        BALL: Nstate<=ITERX_ball;

                        ITERX_ball: if(x<1) Nstate<=ITERX_ball; //when the ball is drawn
                                                        else Nstate<=ITERY_ball;

                        ITERY_ball: if(y<1) Nstate<=ITERX_ball;
                                                        else Nstate<=BRICK;

                        BRICK: Nstate<=ITERX_brick;

                        BRICK_incX:             if (brick_counterX<9) Nstate<=ITERX_brick; //increments the relative brick positions
                                                        else Nstate<=BRICK_incY;

                        BRICK_incY:             if (brick_counterY<1) Nstate<=ITERX_brick;
                                                        else Nstate<=DONE;

                        ITERX_brick:if(x<4) Nstate<=ITERX_brick; //when the paddle is drawn
                                                        else Nstate<=ITERY_brick;

                        ITERY_brick: if(y<4) Nstate<=ITERX_brick;
                                                        else Nstate<=BRICK_incX;

                        DONE: Nstate<=IDLE;

                        default: Nstate<=IDLE;
                endcase
        end
```

```
always@(posedge CLOCK)// Operations during states
begin
        case(state)//increments x or y depending on state
                IDLE:
                        begin
                        x<=0;
                        y<=0;
                        end
                ITERX_clear:
                        begin
                        x<=x+1;
                        y<=y;
                        end
                ITERY_clear:
                        begin
                        x<=0;
                        y<=y+1;
                        end
                PADDLE:
                        begin
                        x<=0;
                        y<=0;
                        end
                ITERX_paddle:
                        begin
                        x<=x+1;
                        y<=y;
                        end
                ITERY_paddle:
                        begin
                        x<=0;
                        y<=y+1;
                        end
                BALL:
                        begin
                        x<=0;
                        y<=0;
                        end
                ITERX_ball:
                        begin
                        x<=x+1;
                        y<=y;
                        end
                ITERY_ball:
                        begin
                        x<=0;
                        y<=y+1;
                        end
                BRICK:
                        begin
                        x<=0;
                        y<=0;
                        brick_counterX<=0;
                        brick_counterY<=0;
                        end
                BRICK_incX:
                        //relative x
                        begin
                        x<=0;
                        y<=0;
                        brick_counterX<=brick_counterX+1;
```

```
                brick_counterY<=brick_counterY;
                end
            BRICK_incY:
                //relative y
                begin
                x<=0;
                y<=0;
                brick_counterX<=0;
                brick_counterY<=brick_counterY+1;
                end
            ITERX_brick:
                begin
                x<=x+1;
                y<=y;
                end
            ITERY_brick:
                begin
                x<=0;
                y<=y+1;
                end
            DONE:
                begin
                x<=0;
                y<=0;
                brick_counterX<=0;
                brick_counterY<=0;
                end
            default:
                begin
                x<=0;
                y<=0;
                brick_counterX<=0;
                brick_counterY<=0;
                end
        endcase
    end

    always@(posedge CLOCK)//update state
    begin
        state<=Nstate;
    end
endmodule


module BrickBreaker
    (
        CLOCK_50,                       //  On Board 50 MHz
        KEY,                            //  Push Button[3:0]
        VGA_CLK,                        //  VGA Clock
        SW,
        LEDG,
        LEDR,
        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 [17:0] SW;
```

```
		input		CLOCK_50;				//		50 MHz
		input		[3:0] KEY;				//		Button[0:0]
		output [17:0] LEDR;
		output [7:0] LEDG;

		//ignore this
		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]
		//ignore this

		wire [2:0] color;
		wire [7:0] x_draw,y_draw; //location of the pixel to draw
		wire enable_plot;//connect from FSM_Draw to plot
		wire Draw_START;
		wire Draw_complete;
		wire [19:0] Brick_enable;
		wire [7:0] paddle_x,paddle_y,ball_x,ball_y;
		//initializes both modules and wires them appropriately
		FSM_Main(paddle_x,paddle_y,ball_x,ball_y, Brick_enable,Draw_START,Draw_complete, KEY, CLOCK_50);
		FSM_Draw(Draw_START,paddle_x,paddle_y,ball_x,ball_y,x_draw,y_draw,
Brick_enable,color,enable_plot,Draw_complete,CLOCK_50);

		vga_adapter VGA(
					.resetn(1'b1), //turn reset off
					.clock(CLOCK_50),
					.colour(color), //always draws black
					.x(x_draw),
					.y(y_draw),
					.plot(enable_plot), //draws when FSM lets it

					/* 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
```