Engineering 378 Digital Systems Design

Exp. 6: VGA Controller and Pong Game

Submitted by:

Norald Alejo
Farnam Adelkhani

November 17, 2016

Demonstration	_/5
Report: Presentation	_/2
Problem: analysis	_/2
Design: work	_/5
Results	_/5
Conclusion/Discussion	_/1
Total	_/20

Problem Analysis

The objective of this laboratory assignment is to eventually generate verilog code to implement a game of pong but first a sample code will be implemented with the intention of learning to draw color patterns on a computer monitor by means of setting the red/green/blue signals via the VGA port. The major challenge in this assignment was implementing the verilog code to get the pong ball to bounce off of the racket.

Hardware Design

```
97   //=======================================================
98   //  REG/WIRE declarations
99   //=======================================================
100  reg aresetPll = 0; // asynchrous reset for pll
101  wire    pixelClock;
102  wire    [10:0] XPixelPosition;
103  wire    [10:0] YPixelPosition;
104  reg [7:0] redValue;
105  reg [7:0] greenValue;
106  reg [7:0] blueValue;
107
108  // slow clock counter variables
109  reg     [20:0] slowClockCounter = 0;
110  wire    slowClock;
111
112  // variables for the dot
113  reg [10:0] XDotPosition = 500;
114  reg [10:0] YDotPosition = 500;
115
116  // variables for the Paddles
117  reg [10:0] PaddleOneXPosition = 180;
118  reg [10:0] PaddleOneYPosition = 512;
119
120  reg [10:0] PaddleTwoXPosition = 1100;
121  reg [10:0] PaddleTwoYPosition = 512;
122
123  // variables for the ping pong
124  reg [10:0] PingPongXPosition = 500;
125  reg [10:0] PingPongYPosition = 500;
126  reg [1:0]  dir;
```

```
127  //=======================================================
128  //  Structural coding
129  //=======================================================
130
131  // output assignments
132  assign VGA_BLANK_N = 1'b1;
133  assign VGA_SYNC_N = 1'b1;
134  assign VGA_CLK = pixelClock;
135
136  // display the X or Y position of the dot on LEDS (Binary format)
137  // MSB is LEDR[10], LSB is LEDR[0]
138  assign LEDR[10:0] = SW[1] ? YDotPosition : XDotPosition;
139
140
141
142  assign slowClock = slowClockCounter[20]; // take MSB from counter to use as a
     slow clock
143
144  always@ (posedge CLOCK_50) // generates a slow clock by selecting the MSB from
     a large counter
145  begin
146      slowClockCounter <= slowClockCounter + 1;
147  end
148
149  always@(posedge slowClock) // process moves the X position of the dot
150  begin
151      if (KEY[0] == 1'b0)
152          XDotPosition <= XDotPosition + 1;
153      else if (KEY[1] == 1'b0)
154          XDotPosition <= XDotPosition - 1;
155  end
156
157  always@(posedge slowClock) // process moves the Y position of the dot
158  begin
159      if (KEY[2] == 1'b0)
160          YDotPosition <= YDotPosition + 1;
161      else if (KEY[3] == 1'b0)
162          YDotPosition <= YDotPosition - 1;
163  end
164
```

```
164
165   //Paddle Update
166   always@(posedge slowClock) // process moves paddle 1
167   begin
168       if (KEY[0] == 1'b0)
169       begin
170           if ((PaddleTwoYPosition + 200 / 2) < 924)
171               PaddleTwoYPosition <= PaddleTwoYPosition + 10;
172       end
173       else if (KEY[1] == 1'b0)
174       begin
175           if ((PaddleTwoYPosition - 200 / 2) > 100)
176               PaddleTwoYPosition <= PaddleTwoYPosition - 10;
177       end
178   end
179
180   always@(posedge slowClock) // process moves paddle 1
181   begin
182       if (KEY[2] == 1'b0)
183       begin
184           if ((PaddleOneYPosition + 200 / 2) < 924)
185               PaddleOneYPosition <= PaddleOneYPosition + 10;
186       end
187       else if (KEY[3] == 1'b0)
188       begin
189           if ((PaddleOneYPosition - 200 / 2) > 100)
190               PaddleOneYPosition <= PaddleOneYPosition - 10;
191       end
192   end
193
194   always@(posedge slowClock) // process moves paddle 1
195   begin
196
197   end
198
199   // PLL Module (Phase Locked Loop) used to convert a 50Mhz clock signal to a 108
      MHz clock signal for the pixel clock
200   VGAFrequency VGAFreq (aresetPll, CLOCK_50, pixelClock);
201
202   // VGA Controller Module used to generate the vertial and horizontal synch
      signals for the monitor and the X and Y Pixel position of the monitor display
203   VGAController VGAControl (pixelClock, redValue, greenValue, blueValue, VGA_R,
      VGA_G, VGA_B, VGA_VS, VGA_HS, XPixelPosition, YPixelPosition);
204
205
```

```
262
263         if (YPixelPosition == YDotPosition && XPixelPosition == XDotPosition)
264         begin
265             redValue <= 8'b11111111;
266             blueValue <= 8'b11111111;
267             greenValue <= 8'b11111111;
268         end
269         //Boundaries
270         else if (XPixelPosition < 120 || XPixelPosition > 1160)
271         begin
272             redValue   <= 8'b00000000;
273             blueValue  <= 8'b00000000;
274             greenValue <= 8'b11111111;
275         end
276         else if (YPixelPosition < 100 || YPixelPosition > 924)
277         begin
278             redValue   <= 8'b11111111;
279             blueValue  <= 8'b11111111;
280             greenValue <= 8'b00000000;
281         end
282
283         //Paddle 1
284         else if (XPixelPosition > (PaddleOneXPosition - 15 / 2) &&
            XPixelPosition < (PaddleOneXPosition + 15 / 2) && (YPixelPosition > (
            PaddleOneYPosition - 200 / 2) && YPixelPosition < (PaddleOneYPosition +
            200 / 2)))
285         begin
286             redValue   <= 8'b00000000;
287             blueValue  <= 8'b11111111;
288             greenValue <= 8'b11111111;
289         end
290
291         //Paddle 2
292         else if (XPixelPosition > (PaddleTwoXPosition - 15 / 2) &&
            XPixelPosition < (PaddleTwoXPosition + 15 / 2) && (YPixelPosition > (
            PaddleTwoYPosition - 200 / 2) && YPixelPosition < (PaddleTwoYPosition +
            200 / 2)))
293         begin
294             redValue   <= 8'b00000000;
295             blueValue  <= 8'b11111111;
296             greenValue <= 8'b11111111;
297         end
298
```

```
298
299         //Ping Pong
300         else if ((XPixelPosition > (PingPongXPosition - 15 / 2) &&
            XPixelPosition < (PingPongXPosition + 15 / 2) && (YPixelPosition > (
            PingPongYPosition - 15 / 2) && YPixelPosition < (PingPongYPosition + 15
            / 2))))
301         begin
302             redValue   <= 8'b11111111;
303             blueValue  <= 8'b00000000;
304             greenValue <= 8'b00000000;
305         end
306
307         //Black Background
308         else
309         begin
310             redValue   <= 8'b00000000;
311             blueValue  <= 8'b00000000;
312             greenValue <= 8'b00000000;
313         end
314     end
315 end
316
317 endmodule
318
```

Verilog Modeling:

The pong module has two instances of the paddle module. One of the modules has the two buttons to the left as inputs and the other one has the buttons at the right as inputs. The output from these two modules is then saved in the two wires (y-axis paddle left) and (y-axis paddle right) which indicated the y-coordinate of the left and right paddle respectively.

Results/Verification:

The results for this lab are verified by playing the implemented pong game. The instructor is keeping an eye out for how the pong ball is bouncing off the paddles, which was the most difficult portion of this lab assignment. We are required to set up the bars across the top and sides of the verilog module to represent a particular color that is specified. When the pong ball enters the field on the panel where it should bounce back off it is critical for the response to work even if few pixels make contact.

Conclusion and Discussion:

The final product will have green bars flowing from top to bottom to the left and right of the screen. The bars are set to the specified thickness. The bottom bars are magenta with a black background. The verilog code was modified until the paddles are moving at a desirable and reasonable speed. It was a challenge to generate a circular ball, it was found that a square ball was more straight-forward in regard to its implementation. The dimensions of the game features are left up to the user. The user can determine the dimensions of the game features

such as paddle and ball sizes. The objective is to have the ball bounce off of the paddles into the intended direction.

Work Breakdown

- Pre-lab assignments were completed individually.
- The laboratory assignment was completed as a team effort.

Prelabs

Farnam Adelkhani
Pre-lab 6
915815724