Mark Lee ECE 357 November 26, 2019

Lab 4: VGA Interfacing and Control

This lab is an extra credit/honors lab for ECE 351/357 that explores the VGA interfacing capabilities on the BASYS3 board. The objective is to generate a checkerboard pattern on a computer monitor using VHDL and the BASYS3 board as a VGA controller, with a red dot that can be moved around using the buttons on the FPGA.

TA Date: 11-26-19

TA Signature: Eun Kim T and R buttons don't work 95% ✓

Introduction

Video graphics are an essential part of any computer, as users must be able to visualize information from the computer hardware to make any use of it. Whether playing video games, typing words in a document, or browsing the web, any human interaction with a computer takes place through graphics displayed on a monitor. How this works is complex and involves a great deal of physics, but in summary, finely-directed electron rays move across the phosphor-coated display screen, making the phosphor glow when hit with electrons. The brightness of each pixel (the unit of area on a display) can be controlled, and the combination of many pixels forms the images we see on computer monitors. The electron beams move horizontally left to right and vertically up and down at a very fast speed, the timing of which must be controlled by carefully-synchronized signals. Video Graphics Array (VGA) cables carry RGBHV (red, green, blue, horizontal sync, vertical sync) signals that tell the monitor what color to display on the screen and the correct time to do it. The VGA standard connection has been used for many years for effective graphics display, and although it is being replaced currently by the newer HDMI standard, it is still prominent and finds many uses in computer engineering.

As computer engineering students, having a basic understanding of video graphic connections and operations is an invaluable skill to have for future projects and employment. This lab provides an opportunity to apply knowledge of VGA fundamentals to a practical project. The objective of the lab is to utilize a VHDL VGA interface module to control the VGA port on the BASYS3 board to produce a 20x15 green-blue checkerboard on a connected display monitor. Moreover, a red highlighted square can be moved on the checkerboard using the BASYS3 board buttons, and a block RAM (BRAM) component must be used to hold the checkerboard color content. Finally, a block diagram of the VGA module must be provided to illustrate the operation of the VGA interface module.

Procedure

For Task 1, I drew a block diagram of the VGA interface module. This handwritten diagram connects all of the components and necessary functions that my VHDL code for Task 2

should perform to successfully display the checkerboard on the monitor, such as the BRAM memory storage and the VGA output from the FPGA to the display screen.

In Task 2 I actually implemented the desired VGA display, using the block diagram and online documentation as guidance for my design. The first thing I did to implement Task 2 - before writing any VHDL code - was read this documentation (located in the Digilent reference manual) detailing the specifics of VGA timing and interfacing with the BASYS3 board. From this I learned that 14 pins are required to produce a VGA signal. 12 of those are three 4-bit color signals representing red, green, and blue colors, while the other 2 signals are HSYNC and VSYNC signals used to drive the timing of each pixel display. I also learned that to drive a 640x480 pixel display I needed a clock rate of 25MHz to achieve a refresh rate of 60Hz for each pixel, fast enough for the human eye to not notice any flickering in the lighting. This knowledge helped me move on to actually writing code for my VGA interface module design.

First, I created a top-level entity in VHDL code specifying the inputs and outputs for the VGA interface. The inputs are four buttons on the BASYS3 board to move the highlighted red square up, down, left, and right. Additionally, a reset switch on SW0 is included to reset the red square to the center of the checkerboard. The output is the display on the computer monitor, consisting of the alternating green/blue blocks and the highlighted red block.

Next, I instantiated several VHDL components for my code. I instantiated a Vivado BRAM module containing 15 20-bit values, each bit representing a color square. I chose a 0 to represent green, a 1 to represent blue, and a dash (-) to represent the red block. This BRAM is used to store the checkerboard color contents, each colored block representing a 32x32 pixel subset of the 640x480 pixel screen. Following this, I included four components of the debounced pulse, one for each button, to ensure a stable signal is received by the code from the mechanical button when it is pressed. Finally, I included the VGA controller component and the clock divider component, both provided on Canvas by our gracious instructor. The VGA controller does most of the heavy lifting in terms of creating HSYNC/VSYNC signals and keeping track of which pixel is currently being targeted for color output. The clock divider simply reduces the 100MHz clock of the BASYS3 board to the required 25MHz frequency to drive the VGA controller.

With the underlying components included, I was actually halfway finished with my code. I wrote additional logic to fill the BRAM with the required values, and output the proper color for each pixel at a frequency of 25MHz. I accomplished the latter part by dividing each pixel index by 32 to find the current index in the 20x15 checkerboard grid. Once I found that, I knew which color to send to the display screen through the VGA cable (blue/green depending on position, or red if the pixel was inside the area designated to the movable square). I also made sure to not output any color signals when the current pixel was outside the visible 640x480 range. Lastly, I wrote code that changes the position of the highlighted red square when the user presses one of the four buttons or activates the reset switch.

At this point my code was finished! Well, almost. I ran simulations and had to do some debugging until the correct checkerboard showed up on the screen and the red block moved around properly. Then, I was done.

Results

The block diagram for Task 1 is shown in Appendix A. It shows the relationships between inputs, outputs, and the various VHDL "blocks" that perform different functions for the final VGA display generation. I pretty much explained it already in the Procedure, so I won't repeat everything again. The drawing is fairly self-explanatory.

The simulation code for Task 2 is shown below in Figure 1. Prominently portrayed towards the bottom of the figure are the output signals red/green/blue. The alternation between green and blue can easily be observed (representing the alternating green/blue blocks on the display), as well as the red output whenever a pixel is within the red square region of the checkerboard. You can see that 20 color signals occur before signal "blank" is asserted, corresponding to the 20 columns of the 20x15 checkerboard pattern. The blank signal signifies the area where the current pixel is not within the visible range and no color is sent to the screen. This signal is located at the bottom of the simulation.

One important aspect to emphasize is the clock timing. Every 4 clock cycles of the 100MHz system clock, the 25MHz clock goes through one cycle. Every 32 of these cycles is required to change a color of the checkerboard, so 128 clock cycles are needed to switch colors

on every column. This is why the clock signal in the simulation below appears pure green - it has to change a lot!

Based on the simulation waveform below and the successful demonstration of my VGA design in the lab, the lab objective and its requirements have been satisfied by my design. The complete top-level code for the VGA interface module I created is shown in Appendix B. This code is another main result of my implementation, containing the component instantiation and VGA output logic described in the Procedure section.

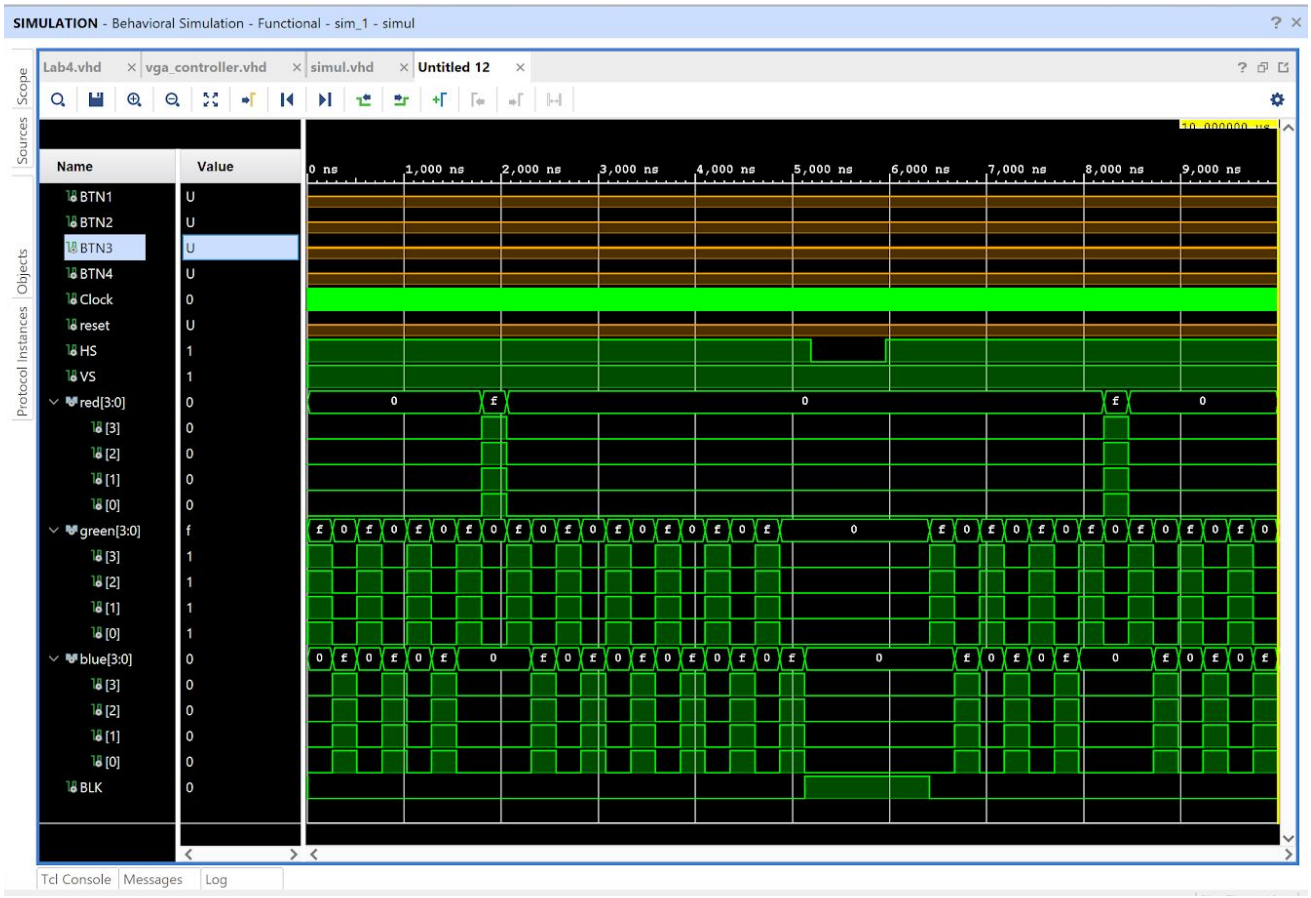


Figure 1. Simulation Data for Task 2.

Conclusion

In summary, the main results of my VGA interface implementation include the Task 1 block diagram in Appendix A, my VHDL code in Appendix B, and the successful operation of the checkerboard with movable red square demonstrated during lab. I am really proud of my

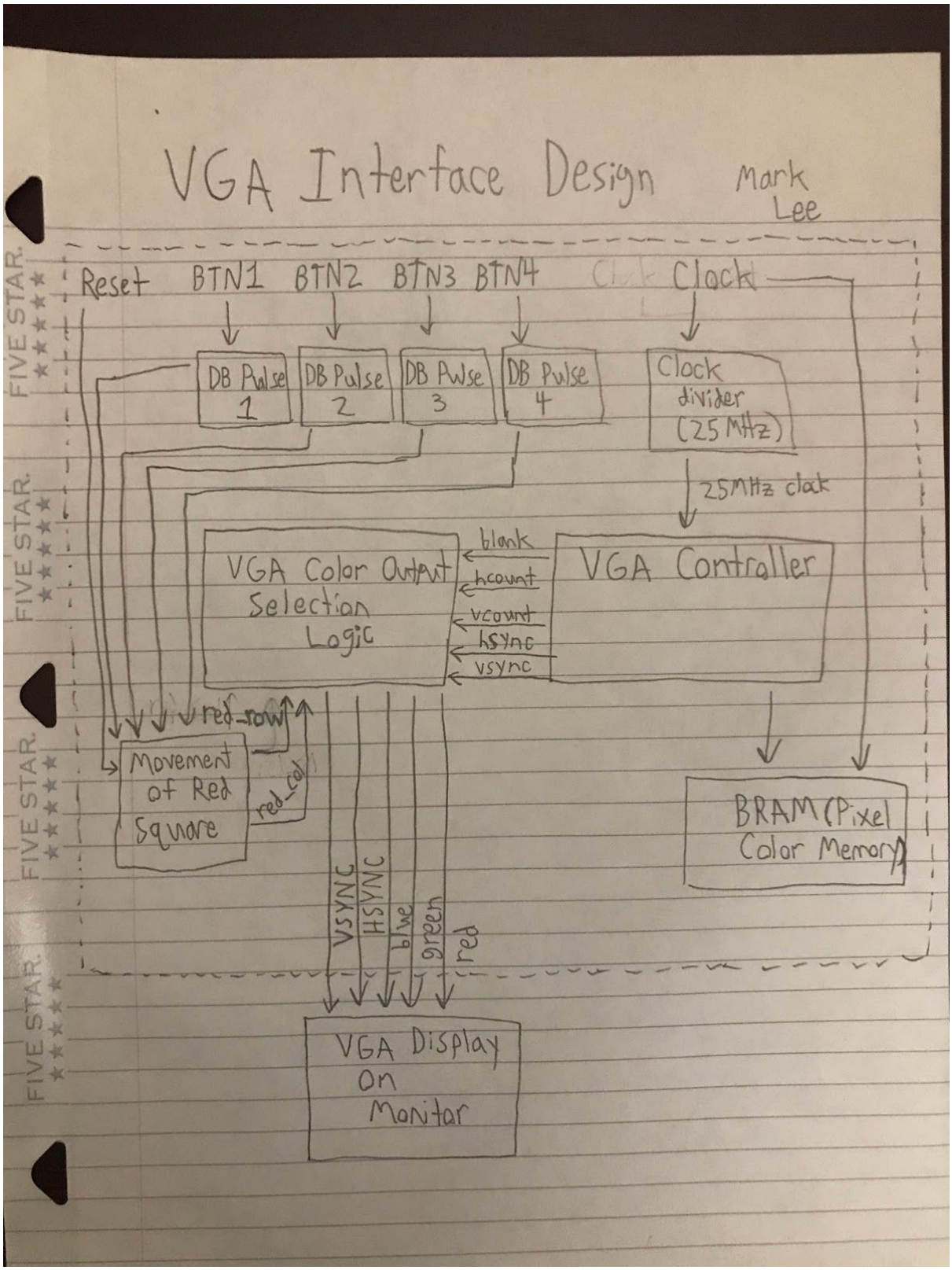
design, as it works almost perfectly and is honestly one of the coolest projects I've done. I haven't worked much with graphics before, so seeing the computer monitor light up with the green and blue checkerboard was an ecstatic moment for me.

One issue I had while doing this lab was getting a computer monitor to receive a VGA signal from the BASYS3 board. I must have spent two days trying to figure out what was wrong. Eventually it started working all of a sudden, but I am still unsure what the issue was. Another problem I could never fix was an extremely perplexing bug where only the bottom and left buttons worked, while the right and top buttons didn't move the red square at all. I spent a long time trying to figure this out, yet couldn't prevail, even with the TA's help.

One possible improvement on my design would be to fix this issue with the buttons, as I'm sure I could figure out what's wrong if I put more time into it. However, it is an insignificant enough error that it's not worth taking a late turn-in penalty. Another cool thing I could do to improve my design would be to implement a simple game like snake, using this lab as a starting point. I don't think it would take much more effort at all to implement this - I would just add a yellow block that the red one could collect to increase its length, and write some logic to keep score and kill the snake if it hits itself.

Ultimately, I'm happy I was one of the few students to complete the honors/extra credit lab, because it has been a priceless learning experience. I now know so much more about VGAs and VHDL in general. It's a shame this is the last for ECE 351/357 because each one was getting more fun than the last. I will definitely use some of these labs - especially this one - as future inspiration for other projects.

Appendix A: Task 1 Block Diagram



Appendix B: Task 2 Code

Project Summary | Schematic | Lab4.vhd | vga_controller.vhd | simul.vhd

C:/Users/Alexander/Desktop/School_Assignments/ECE_357_Labs/ECE357_Lab4/ECE357_Lab4.srcs/sources_1/new/Lab4.vhd

```
1  -- Mark Lee    ECE 357   Lab 4
2  --
3  -- This program utilizes the VGA capabilties of the BASYS3 board to
4  -- display a blue-green checkerboard pattern on a connected display.
5  -- One square is highlighted red, and the user can move it around
6  -- the checkerboard using buttons. A BRAM is instantiated to store
7  -- the initial color contents.
8
9  library IEEE;
10 use IEEE.STD_LOGIC_1164.ALL;
11 use ieee.numeric_std.ALL;
12 use ieee.std_logic_unsigned.all;
13
14 entity Lab4 is
15     Port (BTN1, BTN2, BTN3, BTN4: in std_logic;
16           Clock: in std_logic;
17           reset: in std_logic;
18           HS, VS: out std_logic;
19           red, green, blue: out std_logic_vector(3 downto 0)
20     );
21 end Lab4;
```

```
22
23 architecture Behavioral of Lab4 is
24
25     signal DB1, DB2, DB3, DB4: std_logic;
26     signal address: std_logic_vector(3 downto 0);
27     signal temp_col, temp_row: integer;
28     signal ena_sig: std_logic := '1';
29     signal wea_sig: std_logic_vector(0 downto 0) := "1";
30     signal BRAM_clock: std_logic := '0';
31     signal pixel_clock: std_logic;
32     signal row_in, row_out: std_logic_vector(19 downto 0);
33     signal BRAM_counter: integer := 0;
34     signal row, column: std_logic_vector(10 downto 0);
35     signal blank: std_logic;
36     signal i: integer := 0;
37     signal hs_sig, vs_sig: std_logic;
38     signal red_row: integer := 7;
39     signal red_col: integer := 9;
40     signal ignore: std_logic := '0';
41
42     component DB_CLK
43         port(CLK, BTNC: in std_logic;
44              DB: out std_logic);
45     end component;
46
47     component clockdivider
48         port(clk_in: in std_logic;
49              clk_out: out std_logic);
50     end component;
51
```

```
52      --BRAM
53      component blk_mem_gen_0 is
54          port (
55              clka : in STD_LOGIC;
56              ena : in STD_LOGIC;
57              wea : in STD_LOGIC_VECTOR ( 0 downto 0 );
58              addra : in STD_LOGIC_VECTOR ( 3 downto 0 );
59              dina : in STD_LOGIC_VECTOR ( 19 downto 0 );
60              douta : out STD_LOGIC_VECTOR ( 19 downto 0 )
61          );
62      end component;
63
64      component vga_controller_640_60 is
65          port(
66              rst, pixel_clk : in std_logic;
67              HS, VS, blank : out std_logic;
68              hcount : out std_logic_vector(10 downto 0);
69              vcount : out std_logic_vector(10 downto 0)
70          );
71      end component;
72
73  begin
74
75      --Instantiate components
76      CLK_Component0 : DB_CLK port map(CLK => Clock, BTNC => BTN1, DB => DB1);
77      CLK_Component1 : DB_CLK port map(CLK => Clock, BTNC => BTN2, DB => DB2);
78      CLK_Component2 : DB_CLK port map(CLK => Clock, BTNC => BTN3, DB => DB3);
79      CLK_Component3 : DB_CLK port map(CLK => Clock, BTNC => BTN4, DB => DB4);
80      BRAM : blk_mem_gen_0 port map (clka => BRAM_clock, ena => ena_sig, wea => wea_sig,
81             addra => address, dina => row_in, douta => row_out);
82      DivideCLK: clockdivider port map(clk_in => Clock, clk_out => pixel_clock);
83      VGA_Control: vga_controller_640_60 port map(rst => reset, pixel_clk => pixel_clock,
84                  HS => hs_sig, VS => vs_sig, hcount => column, vcount => row, blank => blank);
85
```

```
85
86      --Initialize BRAM to contain checkerboard display color contents
87      --A 1 represents a green pixel, a 0 represents a blue pixel, and the dash "-"
88      --represents a red pixel
89      process(Clock)
90      begin
91          if(rising_edge(Clock)) then
92
93              --Counter variable to divide clock for each BRAM write
94              BRAM_counter <= BRAM_counter + 1;
95
96              --Initialize the color contents for each 32x32 block of pixels on the
97              --display. The red dot is initialized in the center, row 7 column 9.
98              if(BRAM_counter mod 500000 = 0 and i < 15) then
99
100                 if(i = 7) then
101                     row_in <= "101010101-1010101010";
102                 elsif(i mod 2 = 0) then
103                     row_in <= "01010101010101010101";
104                 else
105                     row_in <= "10101010101010101010";
106                 end if;
107
108                 address <= std_logic_vector(to_unsigned(i, 4));
109
110                 --Increment address and activate BRAM clock to input the color values
111                 if(BRAM_clock = '0') then
112                     i <= i + 1;
113                 end if;
114
115                 BRAM_clock <= not BRAM_clock;
116             end if;
117         end if;
118     end process;
119
```

```
119
120     --Display each pixel on the screen, outputing the appropriate color based on which
121     --32x32 checkerboard block it's in.
122     process(column)
123     begin
124         HS <= hs_sig;
125         VS <= vs_sig;
126
127         --Only output color in the visible pixel range.
128         if(blank = '0') then
129
130             --Divide pixel index by 32 to determine checkerboard index
131             temp_row <= to_integer(shift_right(unsigned(row), 5));
132             temp_col <= to_integer(shift_right(unsigned(column), 5));
133
134             --Output colors accordingly
135             if(temp_row = red_row and temp_col = red_col) then
136                 red <= "1111";
137                 green <= "0000";
138                 blue <= "0000";
139             elsif((temp_row * (20 + 1) + temp_col) mod 2 = 0) then
140                 red <= "0000";
141                 green <= "1111";
142                 blue <= "0000";
143             else
144                 red <= "0000";
145                 green <= "0000";
146                 blue <= "1111";
147             end if;
```

```
148         else
149             red <= "0000";
150             green <= "0000";
151             blue <= "0000";
152         end if;
153     end process;
154 
155     --Move the red square!
156     process(DB1, DB2, DB3, DB4, reset)
157     begin
158         --Reset the red square to the middle of the checkerboard
159         --when reset is active.
160         if(reset = '1' and ignore = '0') then
161             red_row <= 7;
162             red_col <= 9;
163             ignore <= '1';
164         elsif(reset = '0') then
165             ignore <= '0';
166         end if;
167 
168         --If the top button is pressed, the red square moves up
169         if falling_edge(DB1) then
170             if(red_row = 0) then
171                 red_row <= 14;
172             else
173                 red_row <= red_row - 1;
174             end if;
175         end if;
```

```
176
177         --If the right button is pressed, the red square moves right
178         if falling_edge(DB2) then
179             if(red_col = 19) then
180                 red_col <= 0;
181             else
182                 red_col <= red_col + 1;
183             end if;
184         end if;
185
186         --If the bottom button is pressed, the red square moves down
187         if falling_edge(DB3) then
188             if(red_row = 14) then
189                 red_row <= 0;
190             else
191                 red_row <= red_row + 1;
192             end if;
193         end if;
194
195         --If the left button is pressed, the red square moves left
196         if falling_edge(DB4) then
197             if(red_col = 0) then
198                 red_col <= 19;
199             else
200                 red_col <= red_col - 1;
201             end if;
202         end if;
203     end process;
204 end Behavioral;
205
```