Embedded Final Project

Shannon L. Sabino

December 20, 2017 (Original)
February 14, 2018 (Edits)

Contents

Introduction

For the final project of the semester, students had free range over the domain and topic for what they wanted to create, as long as it was sufficiently complicated. This past summer I had fun learning to program in Lua in a heavily constrained, low-resolution game development environment. Low-res game dev showed me the ropes of the language and forced me to better grapple with how to implement algorithms and data structures, all while providing a simple way that you can visually check for errors or flaws in the code. This was what I wanted to replicate with VHDL for this project. Throughout the semester, I felt as if I had been lacking the opportunity to fail and fully flesh out why I had failed. To me, this is where meaningful learning occurs, and where you really begin to master a piece of software or language. And many of the finer points of VHDL were indeed laid bare to me, and as for the ones that weren't, I am at least more metacognitively aware of my shortcomings.

Overall, the goal of this project was to create a drawing tool which works similarly to an etch-a-sketch, in which you can manipulate the pixels on screen from external buttons. I made use of my previously created VGA controller and created a logic for a frame buffer which would load up pixels. This project would not create a profitable product — video games have not been built plainly in hardware for a long time [1]. However, the type of logic I needed to implement is scalable to real embedded systems, even if they aren't graphical in nature. I effectively ended up creating a state machine in not so many words. It responds to inputs appropriately and in a timely manner and performs exactly the tasks I want it to. As a result, my product might not be markettable, but I am. Most of the new and exciting logic for this project is in the `new_pix_buff.vhd` file.

For this project I used exclusively programmed the logic fabric on the Zynq chip for our Zybo boards, and made use of the board's buttons and switches.

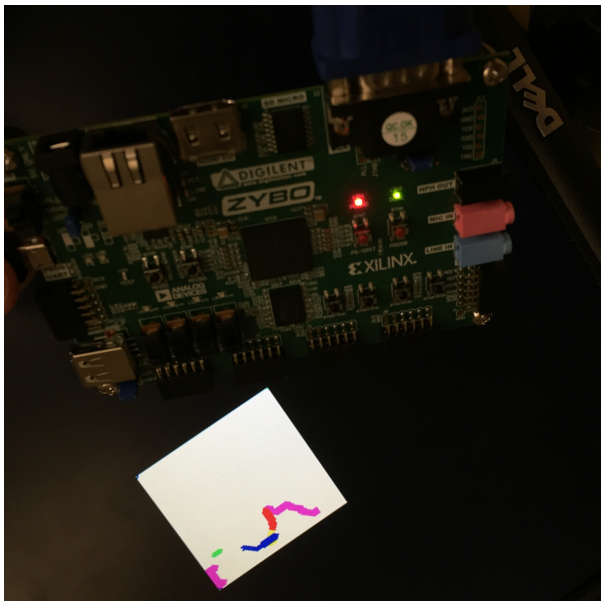


Figure 1: This very dark photo shows the Zybo board above to the monitor that it is interacting with. The white square is my 100x100 pixel result with some colors etched into it.

VGA Revisited

The first part of this project involved anxiously tearing every piece of my VGA project apart until I could discover how it was wrong. And it was somewhat wrong, and it was probably through pure coincidence that it didn't show up in the original test. I hashed out a workaround for this because I didn't want to end up sinking too much time into an "elegant" fix which wouldn't have gotten me any closer to a finished product.

I decided to have my VGA write to a tiny space only to avoid overwhelming me with testing. I originally went with a 33x33 frame so that I could easily test edge conditions and traverse the entire image. Since I was originally writing to only one pixel at a time, this was an appropriate amount of space. I was pressured by my TA to open up the visible area to be larger, and eventually scaled to 100x100 pixels.

All of these edits were actually done in my `pixel_pusher` module, and my `vga_ctrl` was completely untouched.

Frame Buffer

My `pixel_pusher` was originally written to read from a block ROM. Since I wanted to be able to edit my pixels (obviously), I removed the ROM block and created a frame buffer module, `new_pix_buff`. This feeds the correct, requested pixel to the pusher on each clock, while also processing and editing the other pixels behind the scenes.

Since I did not explicitly create a block RAM, I needed to imply one. I created an array of either a 33x33=1089 or 100x100=10,000 8 bit `std_logic_vectors`. This synthesized correctly and without fuss *if* I wrote to it correctly. After much trial and error, I discovered that if I only wrote to one address per clock cycle, then it would synthesize effortlessly as block RAM. But if I tried to write to multiple locations per clock cycle, it would explode and become un-implementable because it needed so many LUTs — I expect it was trying to make a huge multiplexer of some sort.

Another interesting quirk I discovered and was not completely able to resolve was in the vertical wrapping. The pixels should only have been able to be edited within the defined range of the array of RAM. Therefore, when I go off screen, it should "overflow" to the opposite side of the screen. This worked well for the horizontal borders, but on the vertical it appeared to behave as if there was an unseen part of the RAM that it was writing to before wrapping around. Our TA suggested this may be due to it wanting to synthesize RAM sizes with exponents of 2, which seems like a strong possibility, and something that I would want to explore in future edits.

Motion

I handled motion by creating a cursor signal, to hold an address in the block RAM which represents "where" we are. Because of the way the VGA writes to screen, moving left or right is the equivalent of decrementing or incriminating this address by one, and, similarly, moving up or down is the same as decrementing or incriminating the address by the length of a line (33 or 100, for my two resolutions).

I initially started out with a one button press = one pixel system of movement. This was easy for debugging as I knew exactly how it should behave and move for each button press. Eventually, nudged on by making the resolution larger, I decided to implement a "hold" motion. I daisy chained two clock dividers to make this slow, and added an enable to the condition for writing new pixels - now it write for either a new button press or a held button press.

Thickness

I initially started with drawing one pixel only, but as I anticipated scaling up to a larger resolution, I knew I would need to make the thickness that is drawn larger too — in other words, I would need to draw several pixels surrounding the cursor for each button press as well. As mentioned above, I can only write to my frame buffer RAM once per clock cycle, so I effectively ended up creating a state machine to write one surrounding pixel per clock cycle. When in "bigger" mode, it initially writes to the pixels where the cursor is, and then it writes to the four adjacent pixels one at a time, making a line with a thickness of three pixels.

I originally intended to have "one-pixel" mode and "bigger mode" as the two options for the left-most switch. But I eventually began to feel that one pixel would be annoyingly tiny in my higher resolution screen, and decided to create an even bigger mode, called "biggest". This writes 5 pixel wide lines. You can toggle "bigger" and "biggest" mode on the leftmost switch.

Pixel Diagram

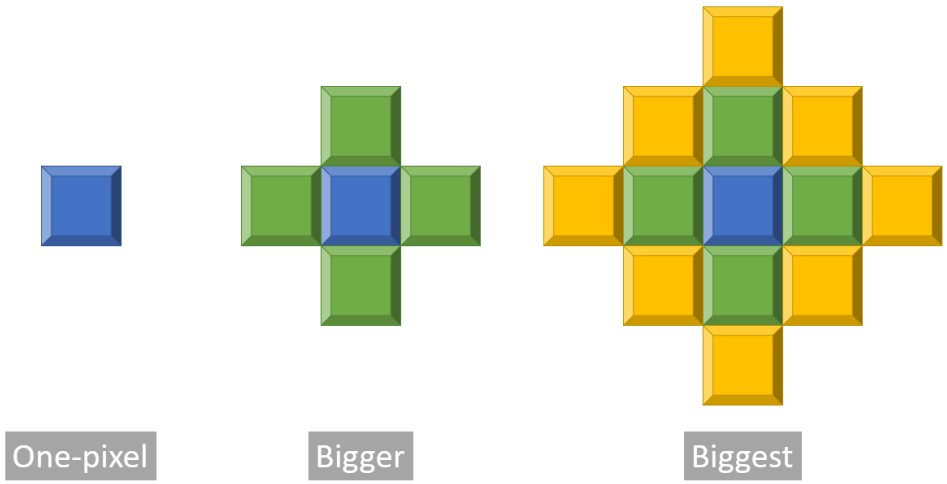


Figure 2: Size comparison of different widths.

Color

Changing the color being drawn was a simple affair — the three remaining switches represent red, green and blue, as the VGA controller takes a traditional RGB value. I have it cut down to only 8 bits between all three colors with padded zeros, so the colors are very bright when isolated.

Switch Diagram

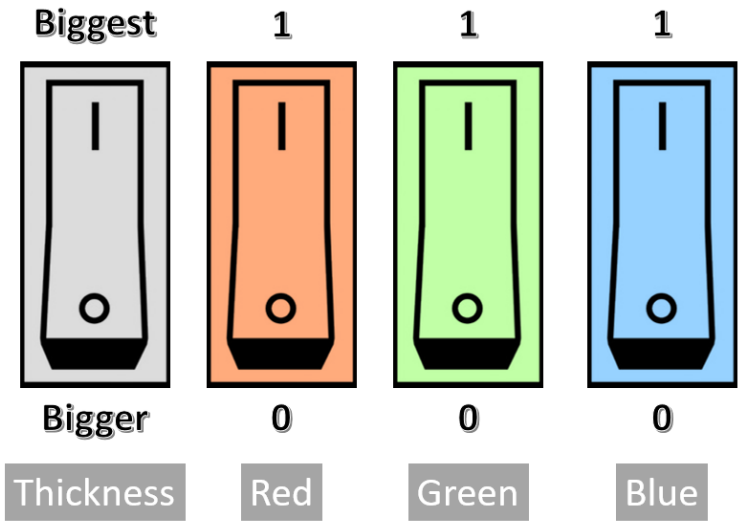


Figure 3: Layout of the switches.

The real difficulty on the color front came from making the cursor blink. I wanted to make the cursor blink in place because I felt it would make the project feel very clean and complete. Without it, it is quite difficult to tell where you are going to draw when you press a button, which makes it look like a rough work in progress — a beta version. To achieve the "blink", I invert the inputs to invert the color and then return to normal. I have it flip on an internal counter. When I press a new button, I don't want the pixels to remain inverted, so before I move the cursor I always return the pixels to their non-inverted color. I additionally have it color the entire width of the cursor. This added quite a bit of logic to the frame buffer, as this piece of the project turned out to be the most challenging. It pulled together my ability to work with virtually every other piece of the project without breaking anything along the way.

Design Process

System Architecture

I knew from the get-go that this would be an application-specific processor, as I did not want to get bogged down in writing assembly for a general-purpose processor given that this is a hardware class. I also knew that, in removing the block ROM, I would need to create a block RAM memory frame buffer to hold the image. This is no instruction memory, only the data memory i.e. the frame buffer.

Implementation

If I was to implement this differently, I might have created more modules for my frame buffer to talk to, instead of grouping it all into one, self-contained "state machine". I additionally think it's disingenuous to call this a state machine — my many nested if-statements look, in retrospect, like I was using my familiarity with high-level programming languages as a crutch. This machine surely would had enough states to rationalize setting up one-hot encoding, and a case statement would have likely executed more efficiently and in a more understandable way to outsiders.

Starting Point

My starting point was my pre-existing VGA controller.

Discussion

I like my project enough to want to use my personal Git for the first time. I wish it was easier to take pictures of the finished result, but writing a module to take a screenshot from my VGA in this time span may have taken years off my life..

The biggest things I learned were how to imply block RAM, and that everything eventually becomes a state machine. Albeit, reading my own code back to me, the nested if-statements still make more sense to me, and I can better visualize how they would turn into combinational logic in the RTL.

If I wanted to sell this product it would need a lot more functionality. As is, this hardware is too much to etch by hand at home with some tracing paper and ferric acid, but to buy FPGA or ASICs would be way too much money at any volume given how much little you could sell this product for. It would make sense as a part of a low-res bundle of little games, so that you can cram more onto a chip and sell the product for more profit.

References

[1] Wikipedia: First generation of video game consoles,
`https://en.wikipedia.org/wiki/First_generation_of_video_game_consoles`

Block Diagram

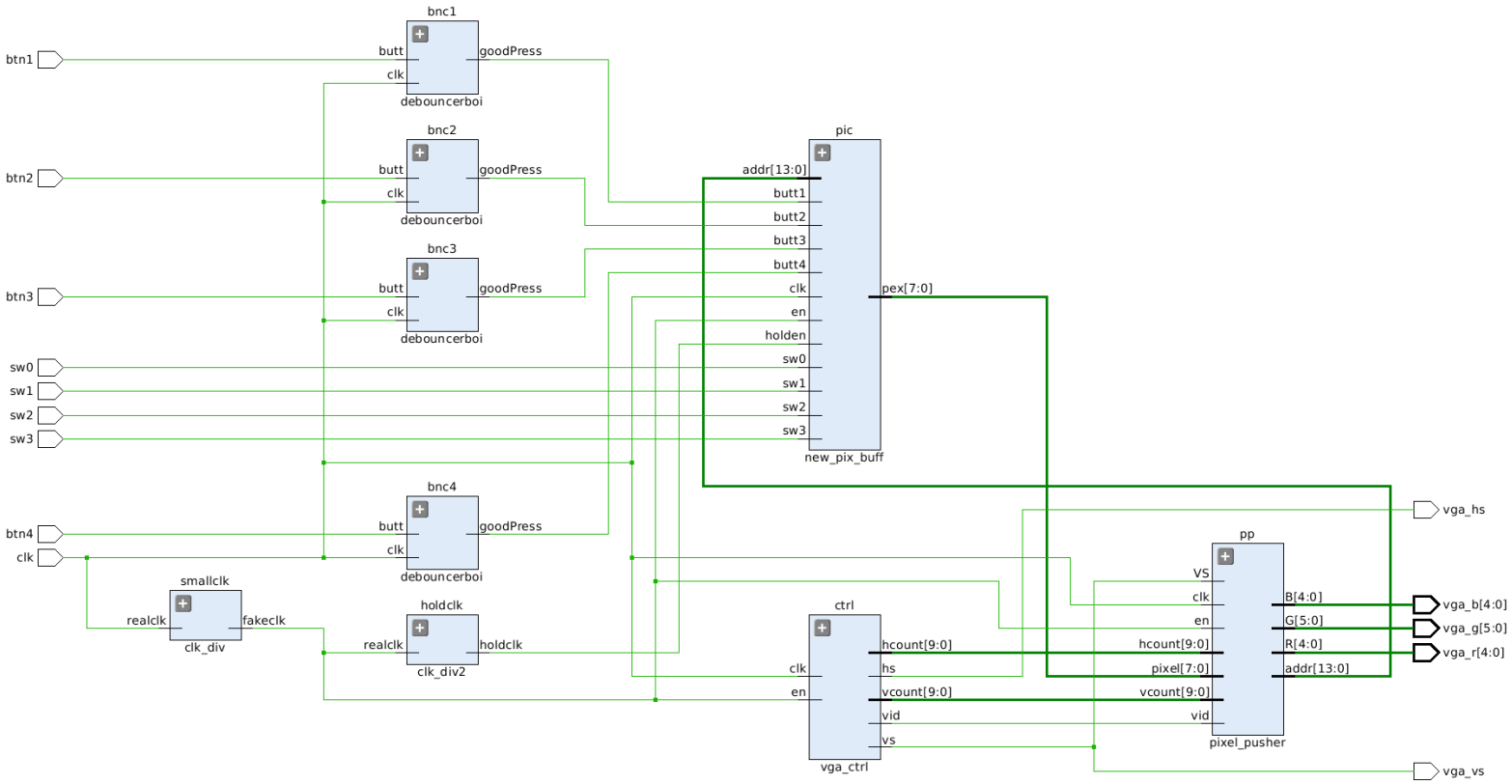


Figure 4: Entire project Block diagram.

Synthesis

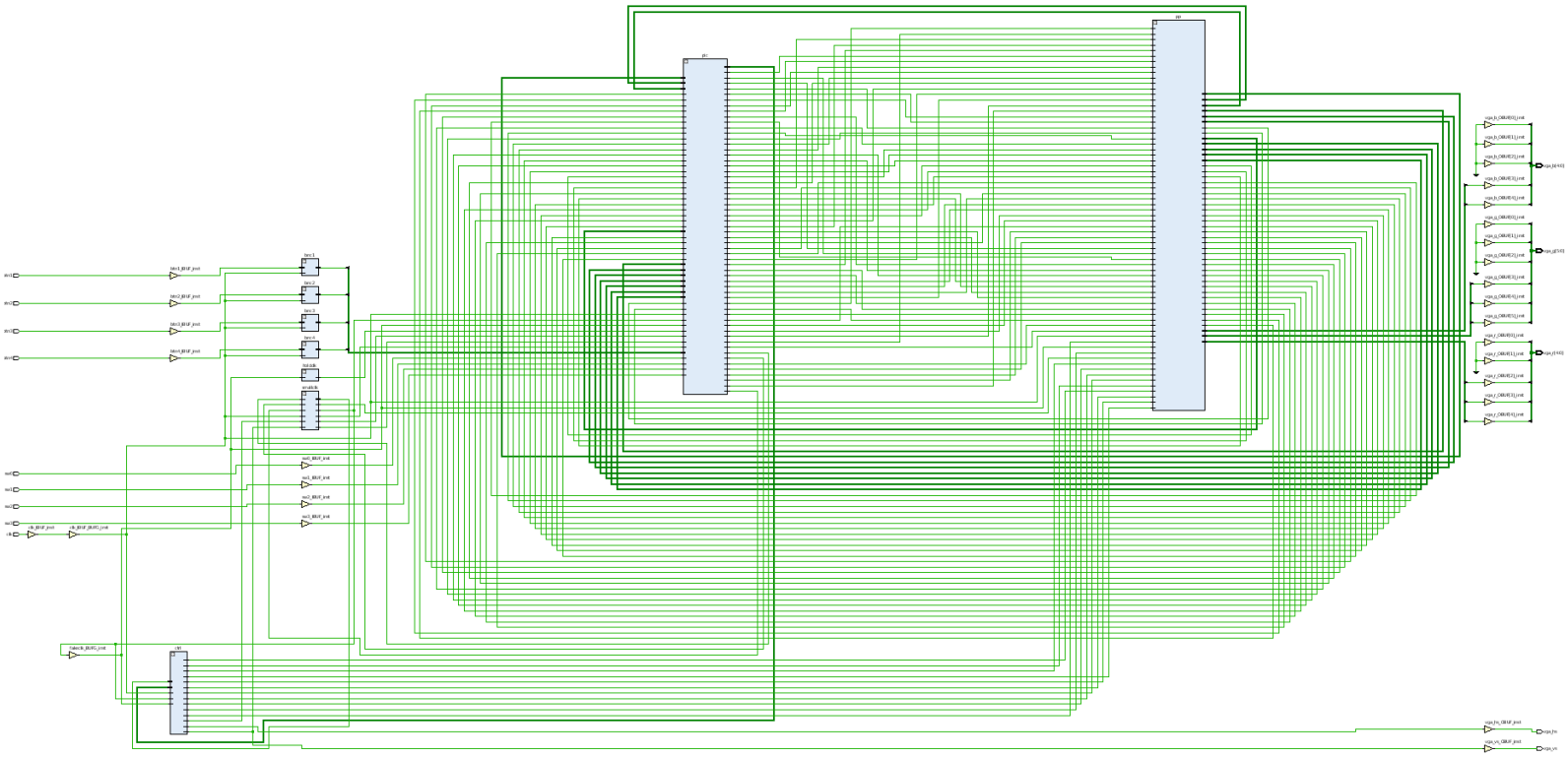
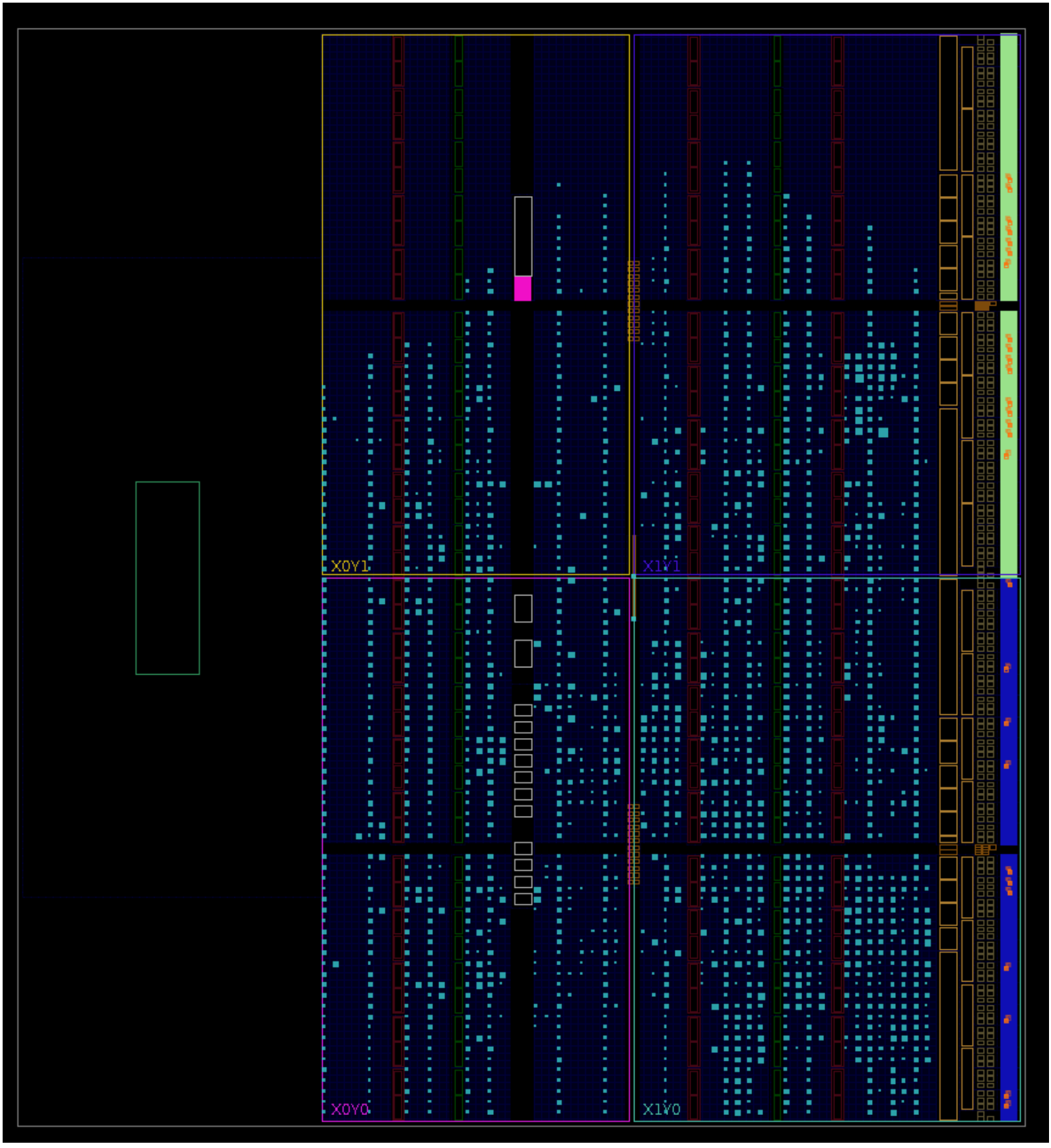


Figure 5: Synthesis schematic.

Implementation



Power

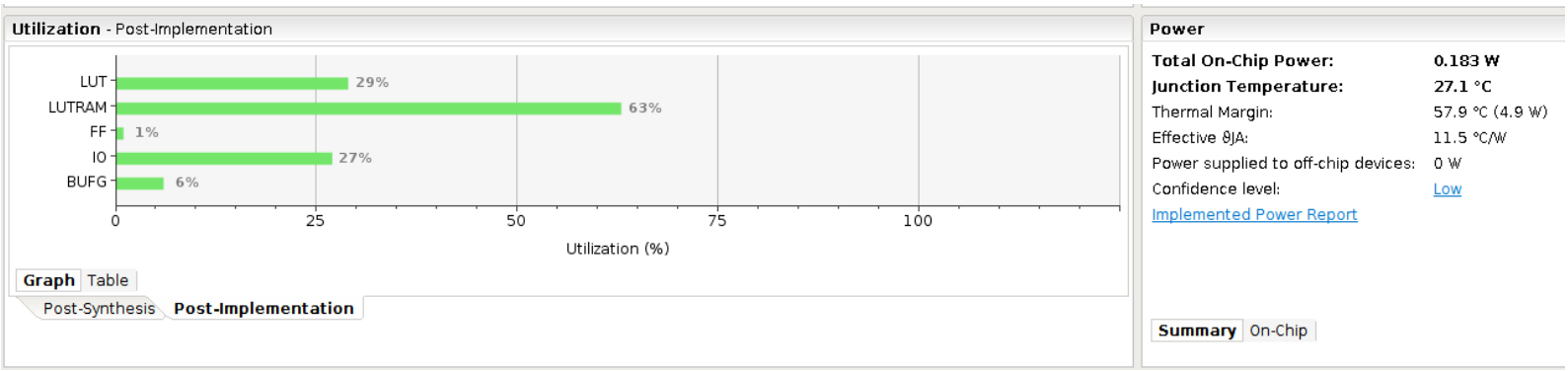


Figure 7: Power and Utilization post-implementation.

Appendix

I will only include the parts of this code that are changed from my original VGA project to avoid redundancy.

image_top.vhd

```
entity image_top is
    Port ( clk : in STD_LOGIC;
           vga_r : out STD_LOGIC_VECTOR (4 downto 0);
           vga_g : out STD_LOGIC_VECTOR (5 downto 0);
           vga_b : out STD_LOGIC_VECTOR (4 downto 0);
           vga_hs : out STD_LOGIC;
           vga_vs : out STD_LOGIC;
           btn1 : in std_logic;
           btn2 : in std_logic;
           btn3 : in std_logic;
           btn4 : in std_logic;
           sw0,sw1,sw2,sw3 : in std_logic);
end image_top;

architecture Behavioral of image_top is

component new_pix_buff
    Port (addr : in STD_LOGIC_VECTOR (13 downto 0) := (others => '0');
          butt1          : in STD_LOGIC; --all signals from the button
          butt2          : in STD_LOGIC; --all signals from the button
          butt3          : in STD_LOGIC; --all signals from the button
          butt4          : in STD_LOGIC; --all signals from the button
          pex  : out std_logic_vector(7 downto 0);
          clk  : in std_logic;
          en   : in std_logic;
          holden : in std_logic;
          sw0,sw1,sw2, sw3 : in std_logic
           );
end component;
component pixel_pusher
    Port ( clk : in STD_LOGIC;
           en : in STD_LOGIC;
           VS : in STD_LOGIC;
           pixel : in STD_LOGIC_VECTOR (7 downto 0);
           hcount : in STD_LOGIC_VECTOR (9 downto 0);
           vcount : in STD_LOGIC_VECTOR (9 downto 0);
           vid : in STD_LOGIC;
           R : out STD_LOGIC_VECTOR (4 downto 0) := (others => '0');
           B : out STD_LOGIC_VECTOR (4 downto 0) := (others => '0');
           G : out STD_LOGIC_VECTOR (5 downto 0) := (others => '0');
           addr : out STD_LOGIC_VECTOR (13 downto 0));
end component;
component clk_div
    Port ( realclk : in STD_LOGIC;
           fakeclk : out STD_LOGIC
           );
end component;
component debouncerboi
Port (butt          : in STD_LOGIC; --all signals from the button
      clk           : in STD_LOGIC; --125MHz clock
      goodPress     : out STD_LOGIC); --high iff this is a real press (test
    for led)
```

```
52
53 end component;
54 component vga_ctrl
55     Port ( clk : in STD_LOGIC := '0';
56            en : in STD_LOGIC := '0';
57            hcount : out STD_LOGIC_VECTOR (9 downto 0) := (others => '0');
58            vcount : out STD_LOGIC_VECTOR (9 downto 0) := (others => '0');
59            vid : out STD_LOGIC := '0';
60            hs : out STD_LOGIC := '0';
61            vs : out STD_LOGIC := '0');
62 end component;
63
64 component clk_div2
65     Port ( realclk : in STD_LOGIC;
66            holdclk : out STD_LOGIC
67            );
68 end component;
69
70 -- signals
71 signal en, vid, hs, vs, press1,press2, press3, press4, holden : std_logic :=
       '0';
72 signal hcount, vcount :  STD_LOGIC_VECTOR (9 downto 0) := (others => '0');
73 signal pixel : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
74 signal addr : STD_LOGIC_VECTOR (13 downto 0) := (others => '0');
75 begin
76
77     smallclk : clk_div port map(
78                     realclk => clk,
79                     fakeclk => en);
80   holdclk :   clk_div2 port map(
81                     realclk => en,
82                     holdclk => holden);
83    bnc1 : debouncerboi port map(
84                     clk => clk,
85                     butt => btn1,
86                     goodPress => press1
87                     );
88    bnc2 : debouncerboi port map(
89                     clk => clk,
90                     butt => btn2,
91                     goodPress => press2
92                     );
93    bnc3 : debouncerboi port map(
94                     clk => clk,
95                     butt => btn3,
96                     goodPress => press3
97                     );
98    bnc4 : debouncerboi port map(
99                     clk => clk,
100                    butt => btn4,
101                    goodPress => press4
102                    );
103       ctrl : vga_ctrl port map(
104                    clk => clk,
105                    en => en,
106                    hcount => hcount, -- pp
107                    vcount => vcount,
108                    vid => vid, -- pp
109                    hs => hs,
110                    vs => vs); -- pp
```

```
111      pp : pixel_pusher port map(
112                      clk => clk,
113                      en => en,
114                      VS => vs, -- ctrl
115                      pixel => pixel, -- pic ?
116                      hcount => hcount, -- ctrl
117                      vcount => vcount,
118                      vid => vid, --ctrl
119                      R => vga_r, -- top?
120                      B => vga_b, -- top
121                      G => vga_g, --top
122                      addr => addr); -- pic ?
123 
124     pic : new_pix_buff port map(
125                      clk => clk,
126                      addr => addr, -- pp
127                      pex => pixel,
128                      butt1 => press1,
129                      butt2 => press2,
130                      butt3 => press3,
131                      butt4 => press4,
132                      en => en,
133                      sw0 => sw0,
134                      sw1 => sw1,
135                      sw2 => sw2,
136                      sw3 => sw3,
137                      holden => holden); -- pp
138 
139                      vga_hs <= hs;
140                      vga_vs <= vs;
141 
142 end Behavioral;
```

hold_clock.vhd

```
1  entity clk_div2 is
2      Port ( realclk : in STD_LOGIC;
3             holdclk : out STD_LOGIC
4             );
5  end clk_div2;
6 
7  architecture Behavioral of clk_div2 is
8  signal countguy : std_logic_vector (19 downto 0) := (others =>'0');
9  --signal countguy : std_logic_vector (1 downto 0) := (others =>'0');
10 signal middleman : std_logic := '0';
11     begin
12     cntproc: process(realclk) begin
13     if rising_edge(realclk) then
14               if countguy="11111111111111111111" then
15               --if unsigned(countguy)=2 then
16                   middleman<='1';
17                   countguy <= (others =>'0');
18               else
19                   middleman<='0';
20                   countguy <= std_logic_vector(unsigned(countguy)+1);
21               end if;
22     end if;
23     end process cntproc;
24     holdclk<=middleman;
25 end Behavioral;
```

pixel_pusher.vhd

```
entity pixel_pusher is
    Port ( clk : in STD_LOGIC;
           en : in STD_LOGIC;
           VS : in STD_LOGIC;
           pixel : in STD_LOGIC_VECTOR (7 downto 0);
           hcount : in STD_LOGIC_VECTOR (9 downto 0);
           vcount : in STD_LOGIC_VECTOR (9 downto 0);
           vid : in STD_LOGIC;
           R : out STD_LOGIC_VECTOR (4 downto 0) := (others => '0');
           B : out STD_LOGIC_VECTOR (4 downto 0) := (others => '0');
           G : out STD_LOGIC_VECTOR (5 downto 0) := (others => '0');
           addr : out STD_LOGIC_VECTOR (13 downto 0) := (others => '0'));
end pixel_pusher;

architecture Behavioral of pixel_pusher is
signal countin : std_logic_vector(13 downto 0) := (others => '0'); --
    internal counter
signal flag : std_logic := '0';
begin
    process(clk) begin
        if rising_edge(clk) AND en='1' then
            if VS='0' then
            --reset
                R <= (others => '0');
                B <= (others => '0');
                G <= (others => '0');
                countin <= (others => '0');
            elsif vid='1' AND unsigned(hcount)<480 then
--                if unsigned(hcount)=280 AND unsigned(vcount)=279 then --
    trying to fix issue w first pixel print twice vcount=279, hcount=280
                if unsigned(hcount)=200 AND unsigned(vcount)=199 then
                    countin <= std_logic_vector(unsigned(countin)+1);
                end if;

--                if unsigned(hcount)>279 AND unsigned(vcount)<313 AND
    unsigned(vcount)>279 AND unsigned(hcount)<313 then -- 33x33 square = 1089
    pixels
                if unsigned(hcount)>199 AND unsigned(vcount)<300 AND unsigned
    (vcount)>199 AND unsigned(hcount)<300 then -- 100x100 square = 10000
    pixels
--                    if countin = "10001000000" then --count to 1088 and see
     what happends
                    if countin = "10011100001111" then --count to 9999 and
    see what happends
                    countin <= (others => '0');
                    else
                    countin <= std_logic_vector(unsigned(countin)+1);
                    end if;
                end if;
--                if unsigned(hcount)>281 AND unsigned(vcount)<313 AND
    unsigned(vcount)>279 AND unsigned(hcount)<315 then -- 33x33 square = 1089
    pixels
                if unsigned(hcount)>201 AND unsigned(vcount)<300 AND unsigned
    (vcount)>199 AND unsigned(hcount)<302 then -- 100x100 square = 10000
    pixels
                    R <= pixel(7 downto 5) & "00";
                    G <= pixel(4 downto 2) & "000";
                    B <= pixel(1 downto 0) & "000";
```

```
                else
                    R <= "00000";
                    G <= "000000";
                    B <= "00000";
                end if;
        else
            R <= (others => '0');
            B <= (others => '0');
            G <= (others => '0');
        end if;
            --signal assignment
            addr <= countin;
        end if;

    end process;
end Behavioral;
```

image_top.vhd

```
entity new_pix_buff is
    Port ( addr : in STD_LOGIC_VECTOR (13 downto 0) := (others => '0');
           butt1         : in STD_LOGIC; --all signals from the button
           butt2         : in STD_LOGIC; --all signals from the button
           butt3         : in STD_LOGIC; --all signals from the button
           butt4         : in STD_LOGIC; --all signals from the button
           pex   : out std_logic_vector(7 downto 0);
           clk   : in std_logic;
           en    : in std_logic;
           holden : in std_logic;
           sw0,sw1,sw2, sw3 : in std_logic);
end new_pix_buff;

architecture Behavioral of new_pix_buff is
--type image is array (0 to 1088) of std_logic_vector(7 downto 0); --made for
     33x33 square
type image is array (0 to 9999) of std_logic_vector(7 downto 0); --made for
    100x100 square
signal thisboi : image := (others => "11111111"); --1024 bois of 8 bits
signal incount : std_logic_vector(3 downto 0) := (others => '1');
signal step : std_logic_vector(3 downto 0) := (others => '0');
--signal cursor : integer range 0 to 824 ; --1088
--signal cursorint : integer range 0 to 824; --1088
signal cursor : integer range 0 to 8500; --1088
signal cursorint : integer range 0 to 8500; --1088
signal buttons : std_logic_vector(3 downto 0) := (others => '0');
signal color : std_logic_vector(7 downto 0) := (others => '0');
signal blinkcnt : std_logic_vector(11 downto 0) := (others => '0');
signal blinkflg : std_logic := '0';
signal bigboi : std_logic_vector(1 downto 0);
signal biggerboi : std_logic_vector(3 downto 0);
signal bigflag,cursorrst,biggerflag : std_logic := '0';
signal colors : std_logic_vector(7 downto 0) := (others=>'0');
signal counter : std_logic_vector(21 downto 0) := (others =>'0');
begin
--max address is 100*100=10000 = 10011100010000 = 14 bits
process(clk, buttons, en) begin
if rising_edge(clk) AND en='1' then
pex <= thisboi(to_integer(unsigned(addr)));

    if bigflag = '1' then  --NEW REGULAR SIZE SURROUNDING PIXELS
```

```
40              case bigboi is --fill in sourrounding pixels one at a time
41                  when "01"
42                      => thisboi(cursorint+100) <= colors;--down
43                         bigboi <= "10";
44
45                  when "10"
46                      =>thisboi(cursorint-100) <= colors; --up
47                      bigboi <= "11";
48
49                  when "11"
50                      => thisboi(cursorint+1) <= colors;--right
51                      bigflag <= '0';
52
53                  when "00"
54                      => thisboi(cursorint-1) <= colors;  --left
55                       bigboi <= "01";
56                  when others
57                      => bigflag <= '0';
58                  end case;
59 --
60      elsif biggerflag = '1' then --THICC SURROUNDING PIXELS
61              case biggerboi is --fill in outtermost pixels
62                  when "0000"
63                      => thisboi(cursorint+200) <= colors;--down
64                     biggerboi <= "0101";
65
66                  when "0101"
67                      =>thisboi(cursorint-200) <= colors; --up
68                     biggerboi <= "0110";
69
70                  when "0110"
71                      => thisboi(cursorint+2) <= colors;--right
72                      biggerboi <= "0111";
73
74                  when "0111"
75                      => thisboi(cursorint-2) <= colors;  --left
76                      biggerboi <= "1000";
77
78                  when "1000"
79                      => thisboi(cursorint-99) <= colors;  --left
80                      biggerboi <= "1001";
81
82                  when "1001"
83                      => thisboi(cursorint-101) <= colors;--right
84                      biggerboi <= "1010";
85
86                  when "1010"
87                      => thisboi(cursorint+99) <= colors;  --left
88                      biggerboi <= "1100";
89
90                  when "1100"
91                      => thisboi(cursorint+101) <= colors;  --left
92                      biggerflag <= '0';
93
94                  when others
95                      => biggerflag <= '0';
96                  end case;
97 --
98      else
99          -- Restore pixels to normal if mid-blink
```

```
100         if buttons /= (butt1 & butt2 & butt3 & butt4) AND cursorrst='1' then
     --fix cursor blinkning inversion
101                 thisboi(cursor) <= sw2&sw2&sw2&sw1&sw1&sw1&sw0&sw0; --regular
     color again
102                 colors <= sw2&sw2&sw2&sw1&sw1&sw1&sw0&sw0;
103                 cursorrst <= '0';
104                 bigflag <= '1';--fill in next circle
105                 bigboi <= (others => '0');
106                 if sw3 ='1'then  --GO TO THICKEST BOI,  fill in outtermost
    circle
107                     biggerflag <= '1';
108                     biggerboi <= (others => '0');
109                 end if;
110
111       -- Plot new pixels -> do for new button presses and held button presses
     (meat n potatoes)
112         elsif (buttons /= (butt1 & butt2 & butt3 & butt4) AND cursorrst='0')
    OR ((buttons = (butt1 & butt2 & butt3 & butt4)) AND buttons/="0000" AND
    holden='1') then  --NEW BUTTON PRESS!!! RREADY TO MOVE!!!! This is
    actually the main chunk
113             colors <= sw2&sw2&sw2&sw1&sw1&sw1&sw0&sw0; --get current color
114             buttons <= butt1 & butt2 & butt3 & butt4; -- make button bus
115                 case buttons is --move cursor according to button presses, on
     press onl
116                     when "0000"
117                         =>
118                     when "1000"
119                         => cursorint <= cursor+100; --down
120
121                     when "0100"
122                         => cursorint <= cursor-100;  --up
123
124                     when "0010"
125                         => cursorint <= cursor+1;  --right
126
127                     when "0001"
128                         => cursorint <= cursor-1;   --left
129
130                     when others
131                         => cursorint <= cursorint;
132                end case;
133
134               cursor<=cursorint; --need buffer to get the cursor right
135               thisboi(cursor) <= colors;     --put color on that pixel
136               bigflag <= '1'; --go to thickER pexs
137               bigboi <= (others => '0'); --reset state machine
138
139               if sw3 ='1'then  --GO TO THICKEST BOI
140                     biggerflag<= '1';
141                     biggerboi <= (others => '0'); --reset state machine
142               end if;
143
144          --If idling, blink cursor:
145           elsif buttons ="0000" then
146                if counter="11111111111111111110" then -- BLINK CURSOR TO
    INVERT
147                     thisboi(cursorint) <= NOT(sw2&sw2&sw2&sw1&sw1&sw1&sw0&sw0);
      --invert colors
148                     colors<=NOT(sw2&sw2&sw2&sw1&sw1&sw1&sw0&sw0); --store
    inversion for filling in other pixels
```

```
149                     cursorrst <= '1'; -- set inverted flag
150                     bigflag <= '1';-- get the inner ring of pixels
151                     bigboi <= (others => '0'); --reset state machine
152                     if sw3 ='1'then  --GO TO THICKEST BOI
153                         biggerflag <= '1';    --get the outter ring of pixels
154                         biggerboi <= (others => '0'); --reset state machine
155                     end if;
156                 elsif counter="01111111111111111111" then  --blink to
    regular color
157                     thisboi(cursorint) <= sw2&sw2&sw2&sw1&sw1&sw1&sw0&sw0;
158                     colors<=sw2&sw2&sw2&sw1&sw1&sw1&sw0&sw0;
159                     cursorrst <= '0'; -- not inverted flag
160                     bigflag <= '1';
161                     bigboi <= (others => '0');
162                     if sw3 ='1'then  --GO TO THICKEST BOI
163                         biggerflag <= '1';
164                         biggerboi <= (others => '0');
165                     end if;
166                 end if;
167                 counter <= std_logic_vector(unsigned(counter)+1); -- blinking
    counter
168
169         end if; -- end getting new pixels/maintaing those pixels
170     end if; -- end filling in pixels/getting new pixels
171 end if; -- end if clock and clock enable
172 end process;
173
174 end Behavioral;
```

ZYBO_Master.xdc

```
1  ## This file is a general .xdc for the ZYBO Rev B board
2  ## To use it in a project:
3  ## - uncomment the lines corresponding to used pins
4  ## - rename the used signals according to the project
5
6
7  ##Clock signal
8  set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { clk
       }]; #IO_L11P_T1_SRCC_35 Sch=sysclk
9  create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports {
       clk }];
10
11
12 ##Switches
13 set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { sw0
       }]; #IO_L19N_T3_VREF_35 Sch=SW0
14 set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { sw1
       }];  #IO_L24P_T3_34 Sch=SW1
15 set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports { sw2
       }]; #IO_L4N_T0_34 Sch=SW2
16 set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { sw3
       }]; #IO_L9P_T1_DQS_34 Sch=SW3
17
18
19 ##Buttons
20 set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports {
       btn1 }]; #IO_L20N_T3_34 Sch=BTN0
21 set_property -dict { PACKAGE_PIN P16   IOSTANDARD LVCMOS33 } [get_ports {
       btn2 }]; #IO_L24N_T3_34 Sch=BTN1
```

```
22 set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {
       btn3 }]; #IO_L18P_T2_34 Sch=BTN2
23 set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports {
       btn4 }]; #IO_L7P_T1_34 Sch=BTN3
24
25
26 ##LEDs
27 #set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports {
       led[0] }]; #IO_L23P_T3_35 Sch=LED0
28 #set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports {
       led[1] }]; #IO_L23N_T3_35 Sch=LED1
29 #set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports {
       led[2] }]; #IO_0_35=Sch=LED2
30 #set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports {
       led[3] }]; #IO_L3N_T0_DQS_AD1N_35 Sch=LED3
31
32
33 ##I2S Audio Codec
34 #set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS33 } [get_ports
       ac_bclk]; #IO_L12N_T1_MRCC_35 Sch=AC_BCLK
35 #set_property -dict { PACKAGE_PIN T19   IOSTANDARD LVCMOS33 } [get_ports
       ac_mclk]; #IO_25_34 Sch=AC_MCLK
36 #set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports
       ac_muten]; #IO_L23N_T3_34 Sch=AC_MUTEN
37 #set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports
       ac_pbdat]; #IO_L8P_T1_AD10P_35 Sch=AC_PBDAT
38 #set_property -dict { PACKAGE_PIN L17   IOSTANDARD LVCMOS33 } [get_ports
       ac_pblrc]; #IO_L11N_T1_SRCC_35 Sch=AC_PBLRC
39 #set_property -dict { PACKAGE_PIN K17   IOSTANDARD LVCMOS33 } [get_ports
       ac_recdat]; #IO_L12P_T1_MRCC_35 Sch=AC_RECDAT
40 #set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports
       ac_reclrc]; #IO_L8N_T1_AD10N_35 Sch=AC_RECLRC
41
42
43 ##Audio Codec/external EEPROM IIC bus
44 #set_property -dict { PACKAGE_PIN N18   IOSTANDARD LVCMOS33 } [get_ports
       ac_scl]; #IO_L13P_T2_MRCC_34 Sch=AC_SCL
45 #set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports
       ac_sda]; #IO_L23P_T3_34 Sch=AC_SDA
46
47
48 ##Additional Ethernet signals
49 #set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports
       eth_int_b]; #IO_L6P_T0_35 Sch=ETH_INT_B
50 #set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33 } [get_ports
       eth_rst_b]; #IO_L3P_T0_DQS_AD1P_35 Sch=ETH_RST_B
51
52
53 ##HDMI Signals
54 #set_property -dict { PACKAGE_PIN H17   IOSTANDARD TMDS_33 } [get_ports
       hdmi_clk_n]; #IO_L13N_T2_MRCC_35 Sch=HDMI_CLK_N
55 #set_property -dict { PACKAGE_PIN H16   IOSTANDARD TMDS_33 } [get_ports
       hdmi_clk_p]; #IO_L13P_T2_MRCC_35 Sch=HDMI_CLK_P
56 #set_property -dict { PACKAGE_PIN D20   IOSTANDARD TMDS_33 } [get_ports {
       hdmi_d_n[0] }]; #IO_L4N_T0_35 Sch=HDMI_D0_N
57 #set_property -dict { PACKAGE_PIN D19   IOSTANDARD TMDS_33 } [get_ports {
       hdmi_d_p[0] }]; #IO_L4P_T0_35 Sch=HDMI_D0_P
58 #set_property -dict { PACKAGE_PIN B20   IOSTANDARD TMDS_33 } [get_ports {
       hdmi_d_n[1] }]; #IO_L1N_T0_AD0N_35 Sch=HDMI_D1_N
59 #set_property -dict { PACKAGE_PIN C20   IOSTANDARD TMDS_33 } [get_ports {
```

```
    hdmi_d_p[1] }]; #IO_L1P_T0_AD0P_35 Sch=HDMI_D1_P
#set_property -dict { PACKAGE_PIN A20   IOSTANDARD TMDS_33 } [get_ports {
    hdmi_d_n[2] }]; #IO_L2N_T0_AD8N_35 Sch=HDMI_D2_N
#set_property -dict { PACKAGE_PIN B19   IOSTANDARD TMDS_33 } [get_ports {
    hdmi_d_p[2] }]; #IO_L2P_T0_AD8P_35 Sch=HDMI_D2_P
#set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports
    hdmi_cec]; #IO_L5N_T0_AD9N_35 Sch=HDMI_CEC
#set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS33 } [get_ports
    hdmi_hpd]; #IO_L5P_T0_AD9P_35 Sch=HDMI_HPD
#set_property -dict { PACKAGE_PIN F17   IOSTANDARD LVCMOS33 } [get_ports
    hdmi_out_en]; #IO_L6N_T0_VREF_35 Sch=HDMI_OUT_EN
#set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports
    hdmi_scl]; #IO_L16P_T2_35 Sch=HDMI_SCL
#set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 } [get_ports
    hdmi_sda]; #IO_L16N_T2_35 Sch=HDMI_SDA


##Pmod Header JA (XADC)
#set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports {
    ja_p[0] }]; #IO_L21P_T3_DQS_AD14P_35 Sch=JA1_R_p
#set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS33 } [get_ports {
    ja_p[1] }]; #IO_L22P_T3_AD7P_35 Sch=JA2_R_P
#set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports {
    ja_p[2] }]; #IO_L24P_T3_AD15P_35 Sch=JA3_R_P
#set_property -dict { PACKAGE_PIN K14   IOSTANDARD LVCMOS33 } [get_ports {
    ja_p[3] }]; #IO_L20P_T3_AD6P_35 Sch=JA4_R_P
#set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports {
    ja_n[0] }]; #IO_L21N_T3_DQS_AD14N_35 Sch=JA1_R_N
#set_property -dict { PACKAGE_PIN L15   IOSTANDARD LVCMOS33 } [get_ports {
    ja_n[1] }]; #IO_L22N_T3_AD7N_35 Sch=JA2_R_N
#set_property -dict { PACKAGE_PIN J16   IOSTANDARD LVCMOS33 } [get_ports {
    ja_n[2] }]; #IO_L24N_T3_AD15N_35 Sch=JA3_R_N
#set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports {
    ja_n[3] }]; #IO_L20N_T3_AD6N_35 Sch=JA4_R_N


##Pmod Header JB
#set_property -dict { PACKAGE_PIN T20   IOSTANDARD LVCMOS33 } [get_ports {
    jb_p[0] }]; #IO_L15P_T2_DQS_34 Sch=JB1_p
#set_property -dict { PACKAGE_PIN U20   IOSTANDARD LVCMOS33 } [get_ports {
    jb_n[0] }]; #IO_L15N_T2_DQS_34 Sch=JB1_N
#set_property -dict { PACKAGE_PIN V20   IOSTANDARD LVCMOS33 } [get_ports {
    jb_p[1] }]; #IO_L16P_T2_34 Sch=JB2_P
#set_property -dict { PACKAGE_PIN W20   IOSTANDARD LVCMOS33 } [get_ports {
    jb_n[1] }]; #IO_L16N_T2_34 Sch=JB2_N
#set_property -dict { PACKAGE_PIN Y18   IOSTANDARD LVCMOS33 } [get_ports {
    jb_p[2] }]; #IO_L17P_T2_34 Sch=JB3_P
#set_property -dict { PACKAGE_PIN Y19   IOSTANDARD LVCMOS33 } [get_ports {
    jb_n[2] }]; #IO_L17N_T2_34 Sch=JB3_N
#set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports {
    jb_p[3] }]; #IO_L22P_T3_34 Sch=JB4_P
#set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports {
    jb_n[3] }]; #IO_L22N_T3_34 Sch=JB4_N


##Pmod Header JC
#set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports {
    jc_p[0] }]; #IO_L10P_T1_34 Sch=JC1_P
#set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports {
    jc_n[0] }]; #IO_L10N_T1_34 Sch=JC1_N
```

```
#set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { jc_p[1] }]; #IO_L1P_T0_34 Sch=JC2_P
#set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { jc_n[1] }]; #IO_L1N_T0_34 Sch=JC2_N
#set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports { jc_p[2] }]; #IO_L8P_T1_34 Sch=JC3_P
#set_property -dict { PACKAGE_PIN Y14   IOSTANDARD LVCMOS33 } [get_ports { jc_n[2] }]; #IO_L8N_T1_34 Sch=JC3_N
#set_property -dict { PACKAGE_PIN T12   IOSTANDARD LVCMOS33 } [get_ports { jc_p[3] }]; #IO_L2P_T0_34 Sch=JC4_P
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { jc_n[3] }]; #IO_L2N_T0_34 Sch=JC4_N


##Pmod Header JD
#set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { jd_p[0] }]; #IO_L5P_T0_34 Sch=JD1_P
#set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { jd_n[0] }]; #IO_L5N_T0_34 Sch=JD1_N
#set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { jd_p[1] }]; #IO_L6P_T0_34 Sch=JD2_P
#set_property -dict { PACKAGE_PIN R14   IOSTANDARD LVCMOS33 } [get_ports { jd_n[1] }]; #IO_L6N_T0_VREF_34 Sch=JD2_N
#set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { jd_p[2] }]; #IO_L11P_T1_SRCC_34 Sch=JD3_P
#set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports { jd_n[2] }]; #IO_L11N_T1_SRCC_34 Sch=JD3_N
#set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { jd_p[3] }]; #IO_L21P_T3_DQS_34 Sch=JD4_P
#set_property -dict { PACKAGE_PIN V18   IOSTANDARD LVCMOS33 } [get_ports { jd_n[3] }]; #IO_L21N_T3_DQS_34 Sch=JD4_N


##Pmod Header JE
#set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { je[0] }]; #IO_L4P_T0_34 Sch=JE1
#set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports { je[1] }]; #IO_L18N_T2_34 Sch=JE2
#set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { je[2] }]; #IO_25_35 Sch=JE3
#set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { je[3] }]; #IO_L19P_T3_35 Sch=JE4
#set_property -dict { PACKAGE_PIN V13   IOSTANDARD LVCMOS33 } [get_ports { je[4] }]; #IO_L3N_T0_DQS_34 Sch=JE7
#set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { je[5] }]; #IO_L9N_T1_DQS_34 Sch=JE8
#set_property -dict { PACKAGE_PIN T17   IOSTANDARD LVCMOS33 } [get_ports { je[6] }]; #IO_L20P_T3_34 Sch=JE9
#set_property -dict { PACKAGE_PIN Y17   IOSTANDARD LVCMOS33 } [get_ports { je[7] }]; #IO_L7N_T1_34 Sch=JE10


##USB-OTG overcurrent detect pin
#set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports otg_oc]; #IO_L3P_T0_DQS_PUDC_B_34 Sch=OTG_OC


##VGA Connector
set_property -dict { PACKAGE_PIN M19   IOSTANDARD LVCMOS33 } [get_ports { vga_r[0] }]; #IO_L7P_T1_AD2P_35 Sch=VGA_R1
```

```
130 set_property -dict { PACKAGE_PIN L20   IOSTANDARD LVCMOS33 } [get_ports {
      vga_r[1] }]; #IO_L9N_T1_DQS_AD3N_35 Sch=VGA_R2
131 set_property -dict { PACKAGE_PIN J20   IOSTANDARD LVCMOS33 } [get_ports {
      vga_r[2] }]; #IO_L17P_T2_AD5P_35 Sch=VGA_R3
132 set_property -dict { PACKAGE_PIN G20   IOSTANDARD LVCMOS33 } [get_ports {
      vga_r[3] }]; #IO_L18N_T2_AD13N_35 Sch=VGA_R4
133 set_property -dict { PACKAGE_PIN F19   IOSTANDARD LVCMOS33 } [get_ports {
      vga_r[4] }]; #IO_L15P_T2_DQS_AD12P_35 Sch=VGA_R5
134 set_property -dict { PACKAGE_PIN H18   IOSTANDARD LVCMOS33 } [get_ports {
      vga_g[0] }]; #IO_L14N_T2_AD4N_SRCC_35 Sch=VGA_G0
135 set_property -dict { PACKAGE_PIN N20   IOSTANDARD LVCMOS33 } [get_ports {
      vga_g[1] }]; #IO_L14P_T2_SRCC_34 Sch=VGA_G1
136 set_property -dict { PACKAGE_PIN L19   IOSTANDARD LVCMOS33 } [get_ports {
      vga_g[2] }]; #IO_L9P_T1_DQS_AD3P_35 Sch=VGA_G2
137 set_property -dict { PACKAGE_PIN J19   IOSTANDARD LVCMOS33 } [get_ports {
      vga_g[3] }]; #IO_L10N_T1_AD11N_35 Sch=VGA_G3
138 set_property -dict { PACKAGE_PIN H20   IOSTANDARD LVCMOS33 } [get_ports {
      vga_g[4] }]; #IO_L17N_T2_AD5N_35 Sch=VGA_G4
139 set_property -dict { PACKAGE_PIN F20   IOSTANDARD LVCMOS33 } [get_ports {
      vga_g[5] }]; #IO_L15N_T2_DQS_AD12N_35 Sch=VGA=G5
140 set_property -dict { PACKAGE_PIN P20   IOSTANDARD LVCMOS33 } [get_ports {
      vga_b[0] }]; #IO_L14N_T2_SRCC_34 Sch=VGA_B1
141 set_property -dict { PACKAGE_PIN M20   IOSTANDARD LVCMOS33 } [get_ports {
      vga_b[1] }]; #IO_L7N_T1_AD2N_35 Sch=VGA_B2
142 set_property -dict { PACKAGE_PIN K19   IOSTANDARD LVCMOS33 } [get_ports {
      vga_b[2] }]; #IO_L10P_T1_AD11P_35 Sch=VGA_B3
143 set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports {
      vga_b[3] }]; #IO_L14P_T2_AD4P_SRCC_35 Sch=VGA_B4
144 set_property -dict { PACKAGE_PIN G19   IOSTANDARD LVCMOS33 } [get_ports {
      vga_b[4] }]; #IO_L18P_T2_AD13P_35 Sch=VGA_B5
145 set_property -dict { PACKAGE_PIN P19   IOSTANDARD LVCMOS33 } [get_ports
      vga_hs]; #IO_L13N_T2_MRCC_34 Sch=VGA_HS
146 set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports
      vga_vs]; #IO_0_34 Sch=VGA_VS
```