ENGS 31 Final Project: Etch-A-Sketch

Taggart Bonham and Will Chisholm

June 5, 2017

Abstract

Our initial goal for this project was to create a fully functional Etch-a-Sketch by wiring knobs to an FPGA and connecting to a VGA. We succeeded in this, successfully creating an Etch-a-Sketch with all of the functions that we intended, allowing the user to draw with two knobs change the color they draw in and clear the screen when desired.

Table of Contents

1. Introduction

In this project, we wanted to create a digital Etch-a-Sketch that would display on a VGA. We had to wire knobs into our FPGA and code the hardware for the FPGA so that it would correctly handle the knob inputs as well as inputs from the board itself, and then output the resulting drawing to the VGA. See Appendix 7A for a system layout.

2. Design Solution

2.1 Specifications:

Our circuit has two knob inputs and four switch inputs and outputs to a VGA display. We have one knob that controls the horizontal motion and one knob that controls the vertical motion of a current pixel location that we track and draw on the screen. Three of the switches correspond to a color: red, green, and blue. The color that the Etch-a-Sketch draws in and clears to is the combination of whichever color switches are in the on position. Our fourth switch clears the screen to whatever color is currently active from the three color switches when it is in the on position, writing every address in the visible portion of the block RAM to the new color. See Appendix 1A for an overview of our final front panel, Appendix 1Da and 1Db for parts overview and map, and Appendix 1E for a full parts list.

2.2 Operating Instructions:

When the Etch-a-Sketch first turns on, there is a dot on the screen in whichever color is currently active.

To draw horizontally, turn the left knob to the right to draw to the right and to the left to draw to the left. To draw vertically, turn the right knob left to draw down and right to draw up.

To clear the screen to the currently selected color, flip the switch furthest to the left to the on position. Once the screen has been cleared to the selected color, flip the switch back to the off position, select a new color, and continue drawing.

The three switches furthest to the right control the color in which the Etch-a-Sketch draws and clears to. From left to right, these switches control the amount of red, green, and blue in the current color. To add any of these colors to the current color, flip the corresponding switch to the on position. To remove any of these colors from the current color, flip the corresponding switch to the off position.

2.3 Theory of Operation:

Our circuit takes in 2 signals each from 2 knobs, one reset switch signal, three color switch signals—which control red, green, and blue for drawing and clearing—and a clock signal. It then outputs to a VGA display. For more detail on our top level functional blocks, see Appendix 1Ba, and for more detail on our top level FPGA board, see Appendix 1Bb. For our top level VHDL code, see Appendix 2Ba1 and for the corresponding test bench see Appendix 2Ba2. For Basys constraints, see Appendix 2Ba3.

Our clock signal is immediately passed to a clock divider and all other blocks in our design take this new, divided clock.

Our knobs output two signals, A and B, which are active low, and when they are turned in one direction A falls and then B falls, and when turned in the other B falls and then A falls. For a schematic diagram, see Appendix 1Ca.

Our controller takes in the 5 signals mentioned above other than the color signals. To determine whether the knobs are moving and which direction they are moving in, each of the 4 knob inputs are first passed through a debouncer (VHDL code in Appendix 2Bd), and then the 2 signals for the x knob and the 2 signals for the y knob are handled separately by 2 knob-decoders, which output a signal indicating if that knob is currently moving clockwise or if it is moving counterclockwise. Both signals are low if the knob is not moving. For VHDL code of the knob decoders, see Appendix 2Bc1, for the internal wiring of this block see Appendix 1Bc2, and for a state diagram see Appendix 2Ab1. The controller also outputs a clear signal based on whether the screen should be cleared or not, determined by the switch signal rst. For the internals of this block see Appendix 1Bc1, and for VHDL code see Appendix 2Bb.

The clockwise and counterclockwise signals for each knob are passed along with a clock signal into our Current Pixel block, which stores the current pixel location and outputs a new 17-bit current pixel location based on whether it was told to move right, left, up, or down by the knobs, or told not to move at all. See Appendix 2Ab2 for a state diagram, Appendix 1Bc3 for internals of this block, and 2Bf1 for VHDL code.

The three color switches go directly into our color decoder along with a clock signal. The color decoder outputs an 8-bit color signal based on which of the switches are on: 3 of the bits for red, 3 for green, and 2 for blue. See Appendix 1bc5 for internals of this block and Appendix 2Bg for VHDL code.

The VGA_Sync block takes in a clock signal and outputs 10-bit x, 9-bit y, H_sync, V_sync, and Video_on signals. The x and y signals are immediately passed to another block that concatenates the two into one 17-bit signal, dropping the least significant bit from each. V_Sync and H_Sync are outputted directly to the VGA. See Appendix 2Be1 for VHDL code.

We also have a multiplexer which takes in the output from Current Pixel as one input and the concatenated x and y from VGA_Sync as a second input. It takes the clear signal from the controller as a select bit; when clear is high it passes through the concatenated xy signal from VGA_Sync, and when clear is low it passes through the current drawing location from Current Pixel. See Appendix 2Bh for VHDL code.

Our final component is our block RAM. BRAM takes 4 inputs, Write_Address, Write_Data, Write_Enable, and Read_Address, and outputs an 8-bit color. The Write_Address is the output of the multiplexer described in the previous paragraph. If clear is low, it just continues to write to the current location with the current color, and if clear is high it writes to the current x and y of VGA_sync, which will clear the screen to the current color. Write_Data takes in the output of Color Decoder, causing the pixel that is currently being written to take whichever color has been selected by the switches. Write_Enable is wired to Video_On, causing only locations reached by the VGA_sync to be written into block RAM, meaning BRAM won't store data for pixels outside of what is visible on the VGA. The final input, Read_Address, is wired to the concatenated x and y from VGA_Sync. The output of BRAM is split into 3 bits of red, 3 bits of green, and 2 bits of blue and passed to the VGA. See Appendix 1Bc4 for a block diagram and 2Ba1 for VHDL code. The Memory map of the BRAM can be found in Appendix 3.

2.4 Construction and Debugging:

Before we began the actual construction of our Etch-a-Sketch, we first created a high level state machine, which can be seen in its revised form in Appendix 2Aa.

We began constructing our Etch-a-Sketch by creating our VGA_Sync. We test benched this component thoroughly, then moved on to our project specific components. See Appendix 2Be2 for test bench.

The next component we built was our Current Pixel. We used a test bench on this, artificially creating right, left, up, and down signals and making sure that our curPixel signal reacted properly. See Appendix 2Bf2 for test bench, Appendix 4A for Waveform. After test benching the Current Pixel component, we wired our directional inputs to buttons on our FPGA.

Once we were confident in Current Pixel and VGA_Sync, we generated our block RAM and interfaced between the BRAM and the two other blocks, not yet worrying about clearing the screen. At this point, we wired our FPGA up to the VGA and were pleased to see that already we had a working Etch-a-Sketch, albeit with buttons instead of knobs and no way to clear the screen other than restarting the program.

The next component we built was our Color Decoder. It was a relatively simple component, just concatenating our three color inputs, so once we finished the component and interfaced it with the BRAM we tested our VGA again and could control the color in which we drew with the three switch inputs to Color Decoder.

After this, we turned our attention to the Controller and the knobs. After creating an FSM based on the knob inputs, we made our first attempt at writing the VHDL for the Controller for one of the knobs, which, though we did not realize it at the time, did not fully represent the behavior of the knobs. After test benching this Controller we believed it was working correctly, but at this point our test bench did not fully capture the behavior of the knobs.

Because we believed that we had a working Controller, we then wired up our knobs. The data sheet for these knobs did not include pins (Appendix 5A), so we struggled for a while to

figure out exactly what the correct configuration would be. Eventually, through oscilloscope testing and with the help of a datasheet for a similar component (Appendix 5B), we were able to wire our knobs correctly, getting the result on the oscilloscope shown in Appendix 4E.

At this point, our construction stagnated as we hit a number of roadblocks. We tested our new design on the VGA and were disappointed to see an empty black screen. The first technique we used to debug was to create a new top file that showed us whether knob inputs were being received, then test bench this top file for correctness, and then wire it to the knobs (top file Appendix 7B1, test bench Appendix 7B2). This showed us that our inputs were not behaving correctly. After carefully checking the wires for our knobs and more oscilloscope testing, we realized that the issues we were having could potentially be caused by oddities in the FPGA ports. We tried different ports for our knob input wires, and the LEDs that we had wired to our knob inputs began behaving correctly, telling us that the knobs were working correctly.

Now our pixel was showing up on the screen again, and it would move when a knob was turned, but not in the direction it should have moved. Each time a knob was turned the pixel would move erratically, jumping a somewhat random amount in either the x or y direction depending on which knob was turned. At this point we realized that our Controller did not fully describe the behavior of the knobs. We completely remade our controller, producing the knob decoder module present in our final design. After test benching our knob decoder (Test bench in Appendix 2Bc2, Waveform in Appendix 4D), we test benched our program with the new Controller containing knob decoders and it worked as expected (Waveform 4C). After reprogramming the FPGA, the knobs were able to move the pixel around the screen almost perfectly.

The one issue that we still had with our pixel motion was that if the knob were turned extremely fast in one direction, the pixel would move quickly in that direction and then back to near its original position. This meant that the user's desired line was drawn, but the pixel would end up back close to where it was before the line was drawn. To fix this, we added debouncer modules for each knob input before they entered our knob decoders, and this quickly fixed the problem (Waveform Appendix 4B). At this point, our design worked completely other than clearing the screen.

Our final task was to get the clear functionality to work properly. For testing purposes, first we wired our concatenated x-y location from VGA_Sync to the Write_Address on BRAM, which theoretically should simply have caused the whole screen to be continually cleared to whichever color was selected from the switches. This did not work. The screen remained black, other than when the clear switch was flipped into the on position, which would cause a flash of horizontal lines of the selected color on the screen before the VGA returned to black. Eventually, we realized that we needed to wire Video_On from the VGA_Sync to Write_Enable on the BRAM, which fixed our problems.

To finish up the project we created a multiplexer with the CLR signal from the control as the selector, and the output of Current Pixel and the concatenated x and y of VGA_Sync as the two inputs, with the output going to Write_Address in the BRAM.

We tested our program again and found we had a fully functional Etch-a-Sketch.

3. Justification and Evaluation:

Our final product is a fully functional Etch-a-Sketch displaying on a VGA. As such, there are very few improvements that could be made to the design, other than potentially adding other

functionality. Our resource utilization and critical timing path can be found in Appendix 2C and Appendix 2D respectively, and both of these indicate that our Etch-a-Sketch is efficient and behaves properly. Appendix 2D also contains analysis of the critical timing path. Our warnings that still occur have each been addressed and don't pose any issues to our Etch-a-Sketch. See Appendix 2E for detailed residual warning analysis.

One component that we would have liked to add if we had had more time is a starting COE file. After completing our intended design, we looked into adding a welcome screen that would display when the Etch-a-Sketch first turned on. Unfortunately, neither of us knew MatLab well enough to generate a COE file there, so we tried to write a Java program to convert an image into a COE file, and after creating a few mangled welcome screens we gave up on this addition to the project.

4. Conclusions:

In short, we achieved all the goals we set out for ourselves at the beginning of the process. In our proposal we stated that we planned to have two knobs to control the x and y motion of drawing, and a button input to clear the screen. We followed the proposed knob operation exactly, though we decided that a switch, which would keep the screen cleared while in the on position, would be more practical for clearing. We also added three switches to control the color, both for the color in which Etch-a-Sketch draws, as well as the color that the screen is cleared to.

For any other groups trying to make a digital Etch-a-Sketch, our largest recommendation would be to fully test each module of the program before implementing it. We did this for the most part and it paid large dividends, allowing us to often have the hardware behave correctly

the first time we generated a bitstream for our FPGA. Particularly with the Etch-a-Sketch, almost all components are relatively straightforward to test bench, so there is no excuse for not making sure that a module works exactly as expected before converting it to hardware. The one time that we did not test bench a module effectively was with our controller, and this caused us to search for a long time without being able to find exactly what the error was. We had used a test bench, but the test bench did not accurately represent the behavior of the hardware inputs, and because of the ineffectiveness of our test bench we struggled to find the error. The most important piece of our success in this project was testing often and efficiently.

5. Acknowledgements:

We would like to thank Dave Picard, Magnus Bigelow, and Professor Geoffrey Luke for their help and advice on this project.

We did most of the work for this project while together, but there were a few pieces which each of us had a larger role in. Taggart did more work on the VGA component; Will did more work on the controller; we collaborated closely for everything else.

For our report, Will did more of the writing, and Taggart worked more on documenting the system to create our appendix, but we both had a large hand in each portion.

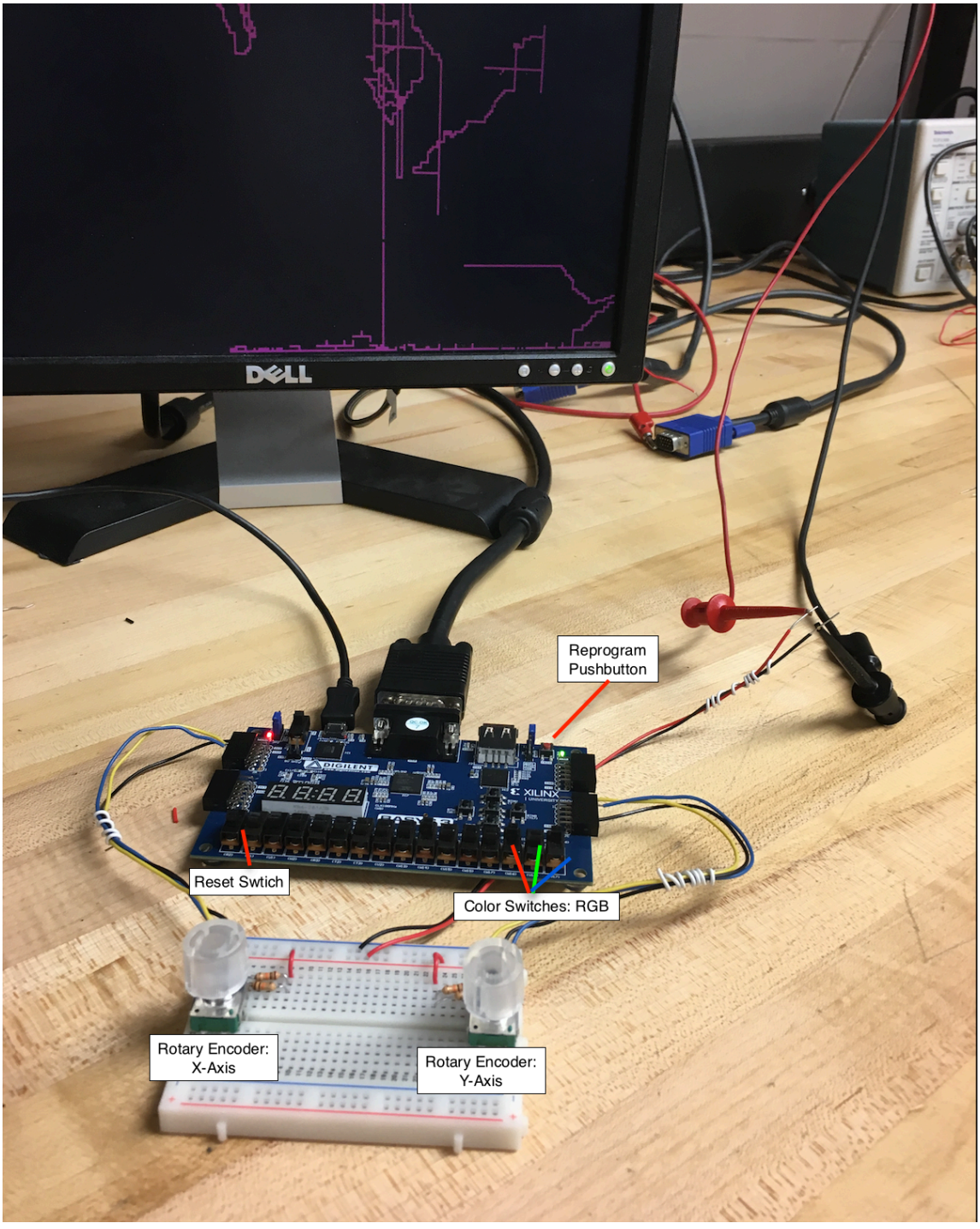
6: References

Revotics, “Understanding Quadrature Encoding,” Applications for Rotary Sensors, January 26, 2009. See Appendix 5B.

7: Appendices

1: Systems level diagrams:

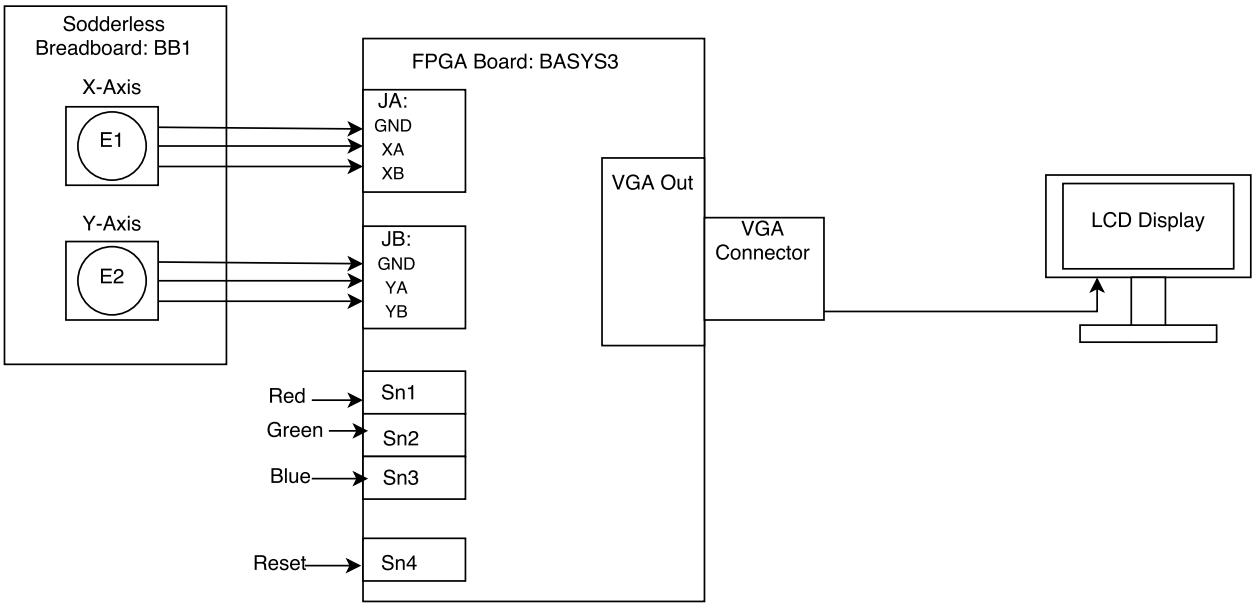
A: Front panel



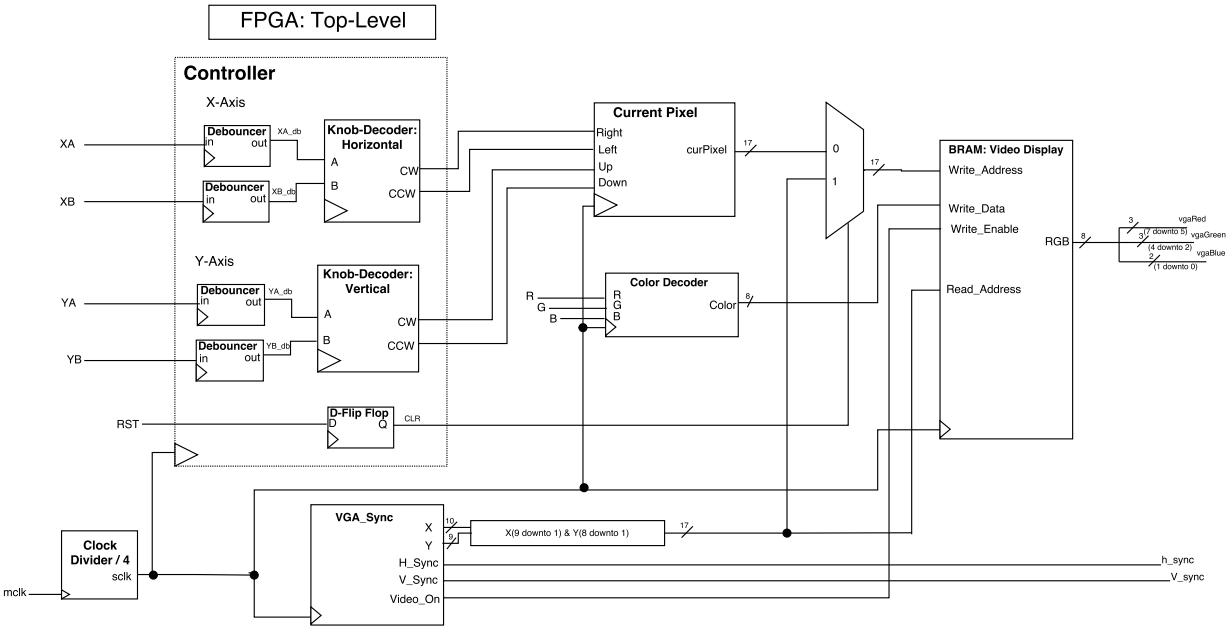
B: Block Diagrams

a: Top Level Functional Blocks

Overview

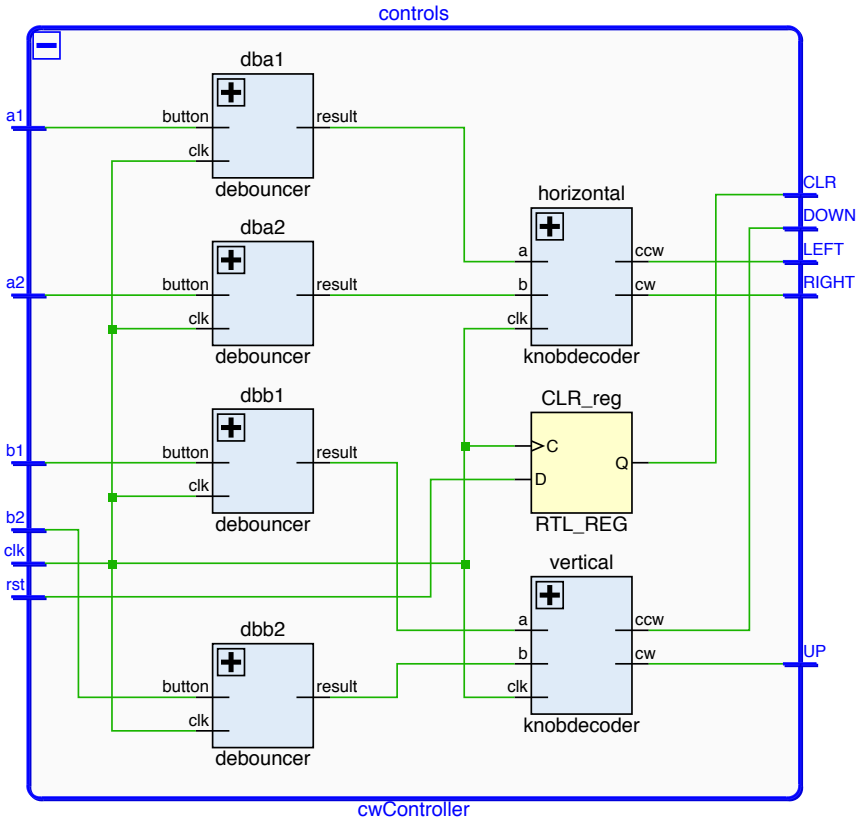


b: Top Level FPGA Board

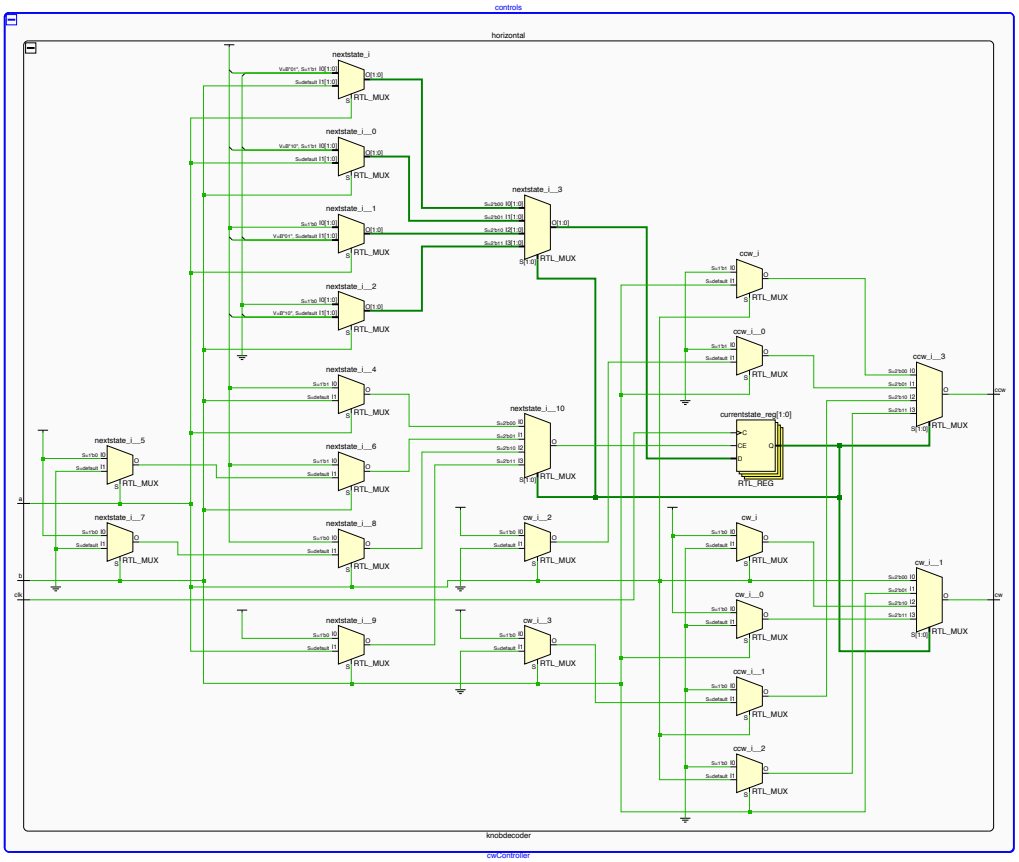


c: Internals of Blocks

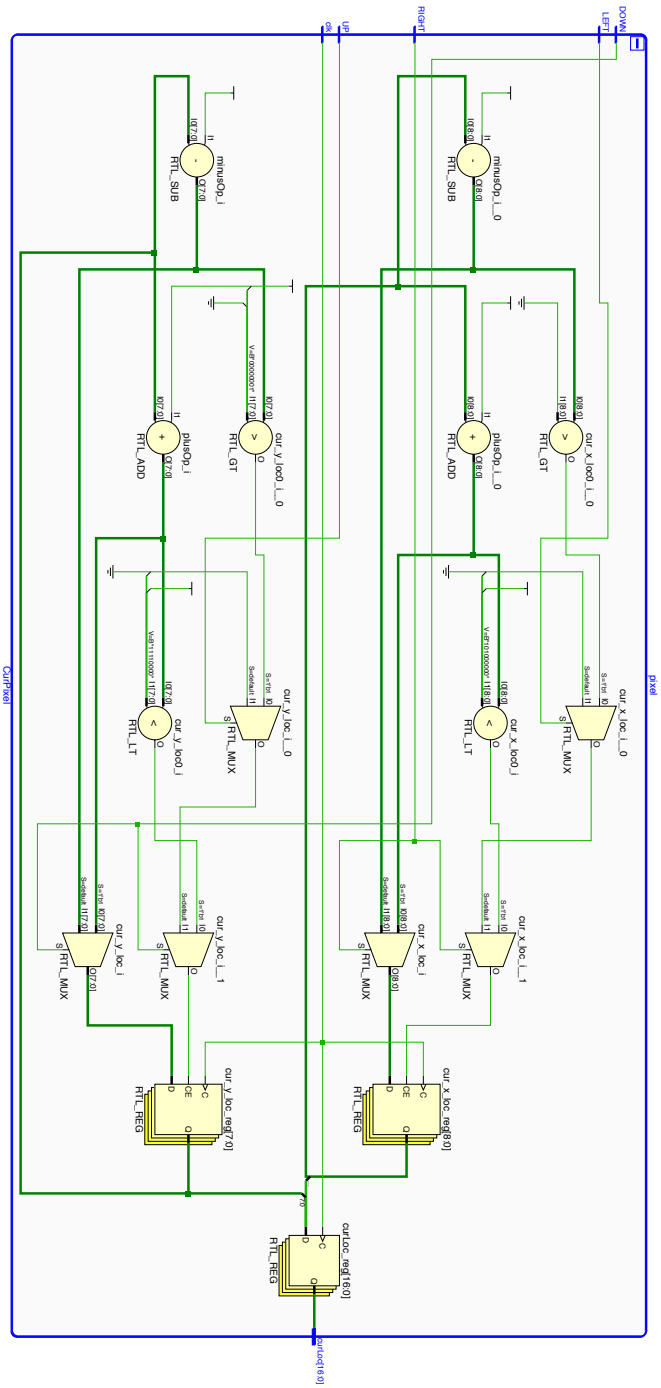
1. Controller



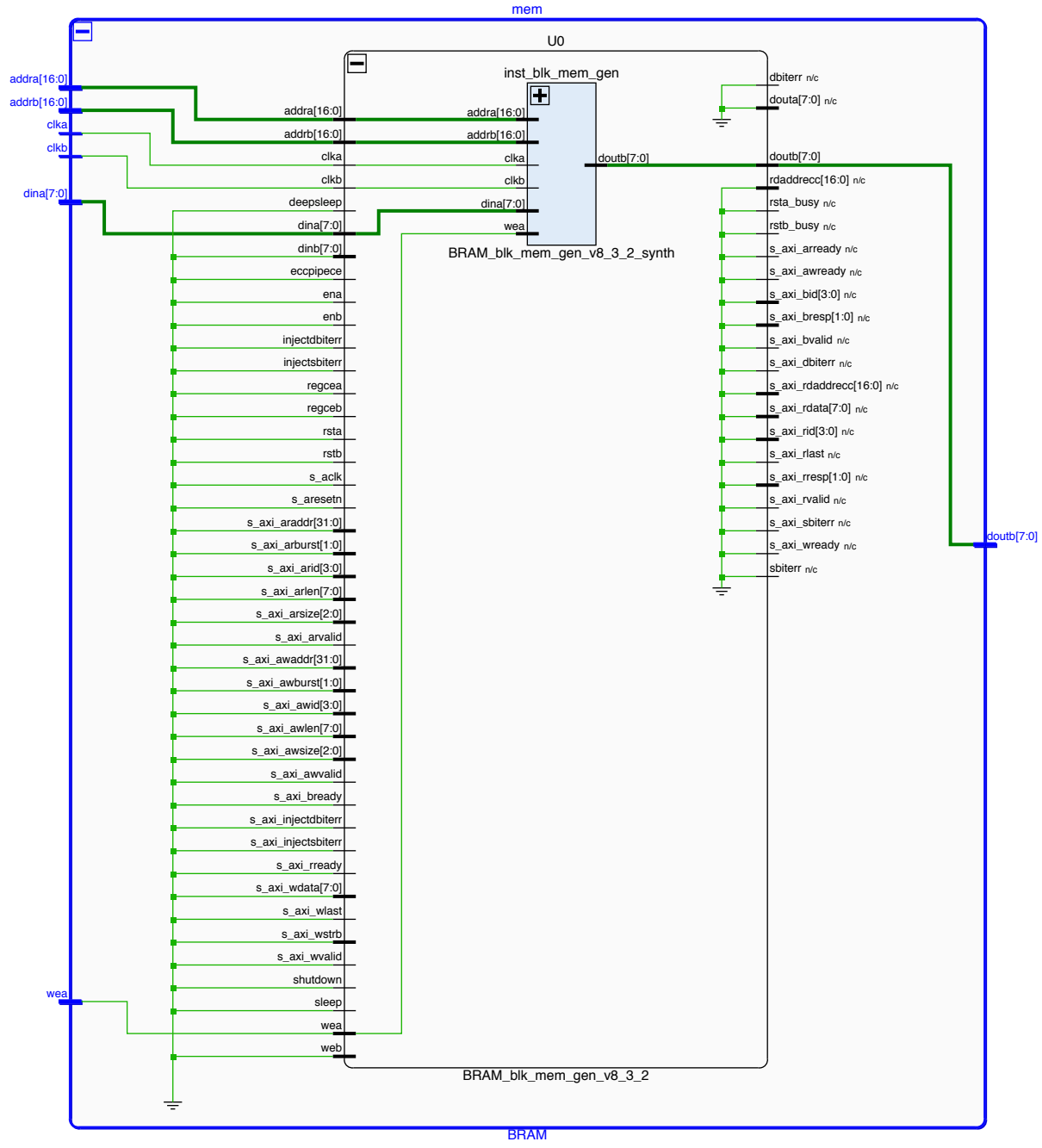
2. Knob Decoder



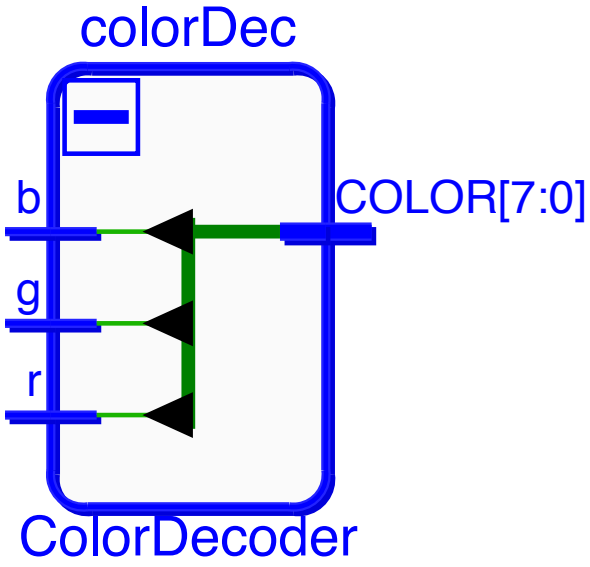
3. CurrentPixel



4. Block RAM Memory

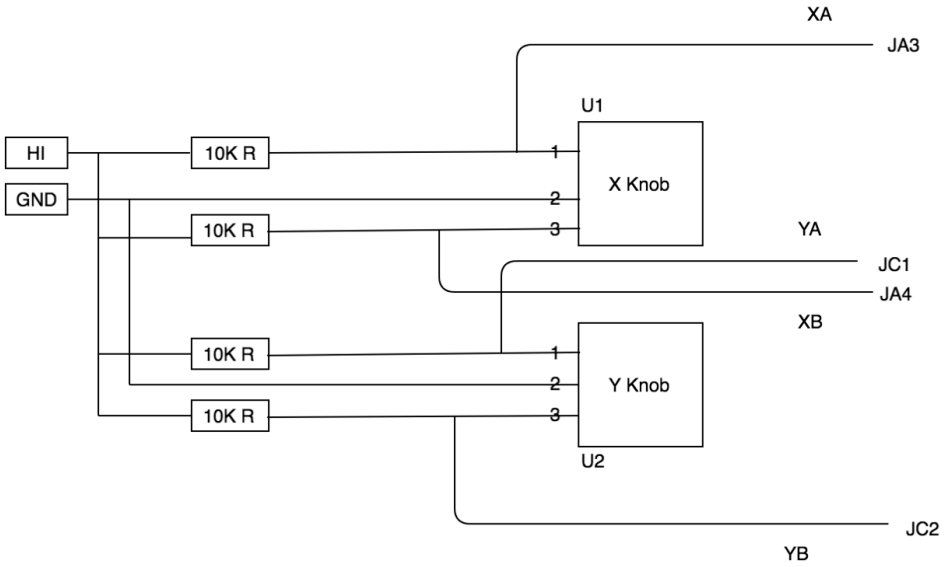


5. Color Decoder



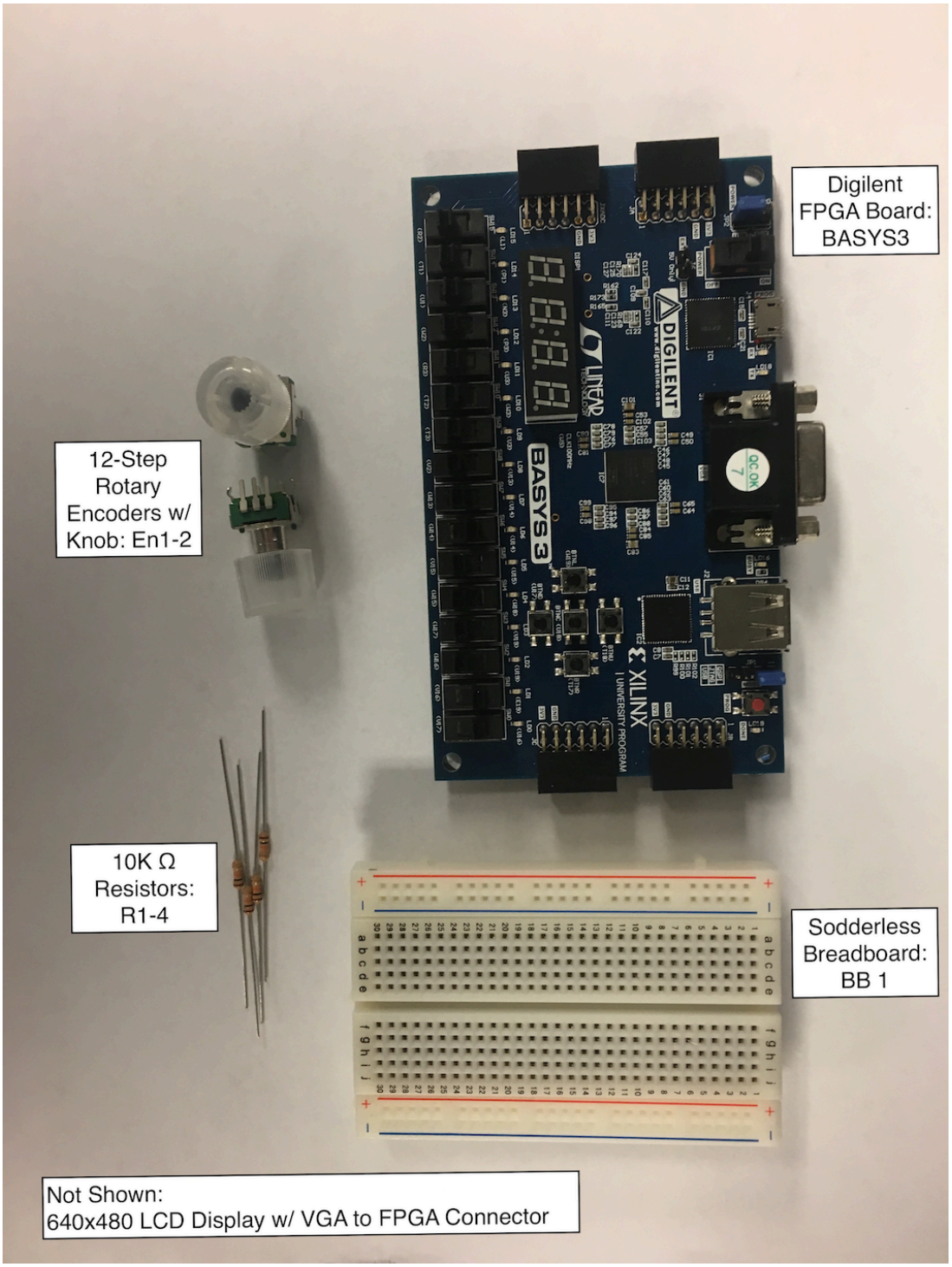
C: Schematic Diagrams

a: Rotary Encoder Inputs

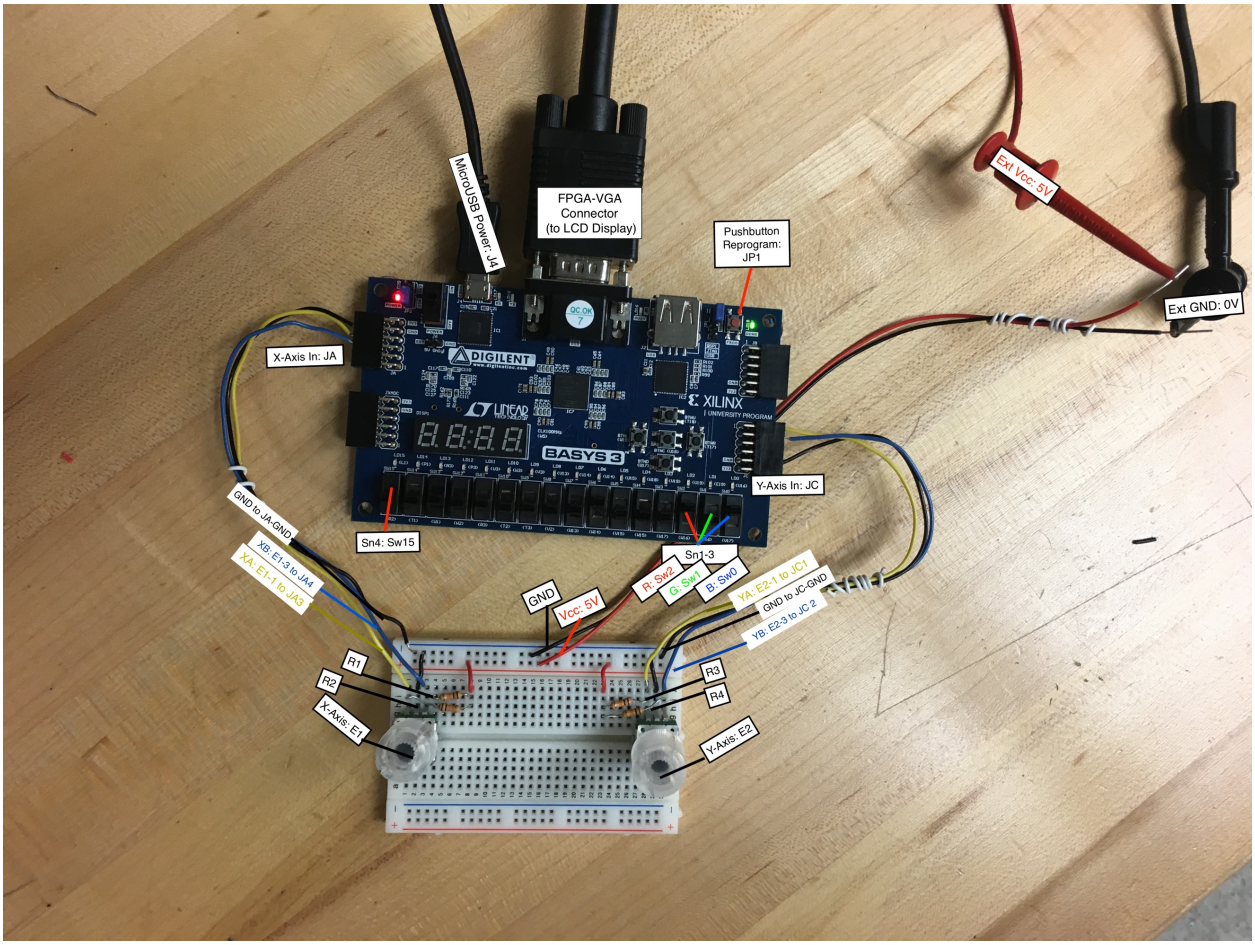


D: Package Map

a: Parts Overview



b: Map



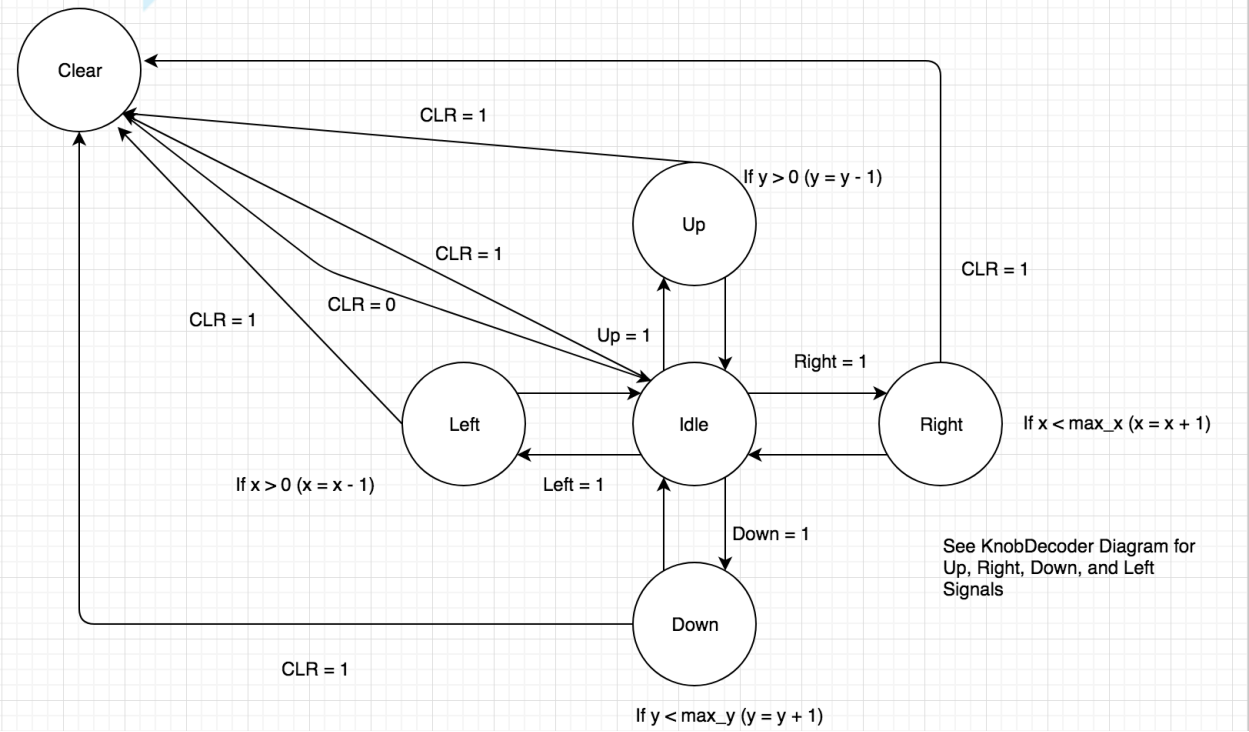
E: Parts list

Reference	Quantity	Part number	Description
Basys3	1	Basys3	Digilent Basys3 board
Display	1		640x480 LCD Display w/ VGA to FPGA Connector
R1-4	4		10K Ω, 1⁄4-watt resistor
En1-2	2	COM-09117	12-step Rotary encoder w/ knob
BB1	1	BB-32621	Sodderless Breadboard

2: Programmed Logic:

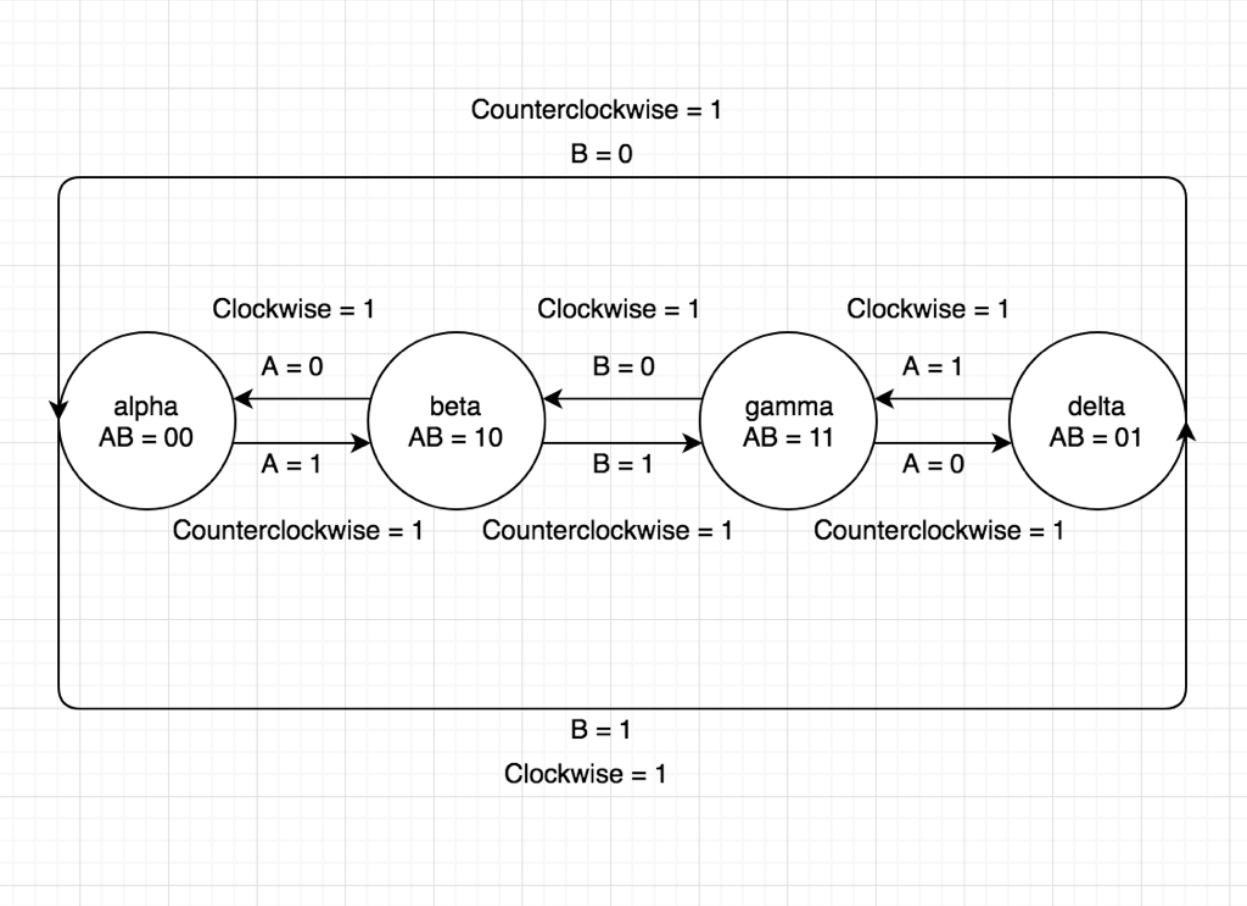
A: State diagrams

a: HLSM

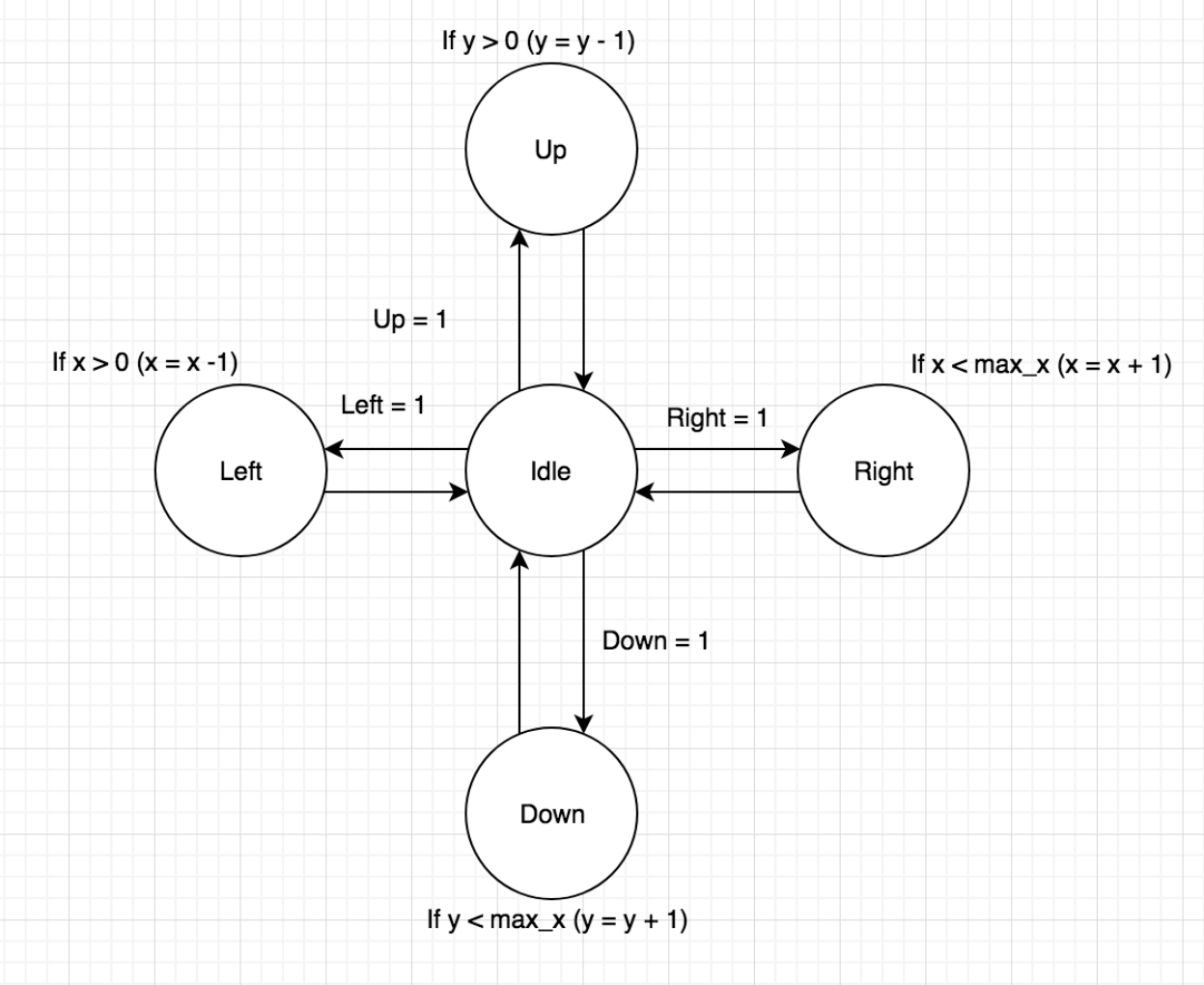


b: State Diagrams

1. Knob Decoder



2. Current Pixel



B: VHDL code for each module

a: Etch-a-Sketch

1: Top Module


```
----------------------------------------------------------------------------------
-- Engineer: Taggart Bonham and Will Chisholm
--
-- Create Date: 05/23/2017 09:05:13 PM
-- Design Name:
-- Module Name: VGA_TEST_TOP - Behavioral
-- Project Name: Etch-a-Sketch final project
-- Target Devices: Digilent Basys3 Board
-- Tool Versions: Vivado 2016.1
-- Description: Top file of final project.
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity EtchaSketch_TOP is
    Port ( mclk : in STD_LOGIC;
           r,g,b : in STD_LOGIC;
           XA, XB, YA, YB : in STD_LOGIC;
           rst : in STD_LOGIC;
           H_sync : out STD_LOGIC;
           V_sync : out STD_LOGIC;
           vgaBlue, vgaGreen, vgaRed : out STD_LOGIC_VECTOR(3 downto 0));
end EtchaSketch_TOP;

architecture Behavioral of EtchaSketch_TOP is

--timing driver for VGA connection
Component VGA_SYNC is
Port (  clk     :   in  STD_LOGIC;
        V_sync  :   out STD_LOGIC;
        H_sync  :   out STD_LOGIC;
        video_on:   out STD_LOGIC;
        pixel_x :   out STD_LOGIC_vector(9 downto 0);
        pixel_y :   out STD_LOGIC_VECTOR(8 downto 0) );
end component;

--logic regarding current location on screen
Component CurPixel is
Port (    clk : in STD_LOGIC;
          UP : in STD_LOGIC;
          DOWN : in STD_LOGIC;
          LEFT : in STD_LOGIC;
          RIGHT : in STD_LOGIC;
          curLoc : out STD_LOGIC_VECTOR(16 downto 0));
end component;

Component ColorDecoder is
    Port ( r : in STD_LOGIC;
           g : in STD_LOGIC;
           b : in STD_LOGIC;
           COLOR : out STD_LOGIC_VECTOR(7 downto 0));
end component;


--Component blk_mem_gen_1 is
--  PORT (
--  clka : IN STD_LOGIC;
--  wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
--  addra : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
```

```
--  dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
--  clkb : IN STD_LOGIC;
--  enb : IN STD_LOGIC;
--  addrb : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
--  doutb : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
--);
--  end component;

 --BRAM storage for VGA
--stores current color at screen locations
 Component BRAM IS
    PORT (
      clka : IN STD_LOGIC;
      wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
      addra : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
      dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      clkb : IN STD_LOGIC;
      addrb : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
      doutb : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
  END component;
--controller for design
--inputs from rotary encoders turned into control signals
--responsible for moving the current location and clearing
Component cwController is
PORT (     clk : in STD_LOGIC;
           a1 : in STD_LOGIC;
           a2 : in STD_LOGIC;
           b1 : in STD_LOGIC;
           b2 : in STD_LOGIC;
           rst : in STD_LOGIC;
           UP : out STD_LOGIC;
           DOWN : out STD_LOGIC;
           LEFT : out STD_LOGIC;
           RIGHT : out STD_LOGIC;
           CLR : out STD_LOGIC);
  end component;

 Component Mux2x1 is
   Port ( A : in STD_LOGIC_VECTOR (16 downto 0);
          B : in STD_LOGIC_VECTOR (16 downto 0);
          sel : in STD_LOGIC;
          y: out STD_LOGIC_VECTOR (16 downto 0));
end component;

-- SIGNAL DECLARATIONS


-- Signals for the serial clock divider, which divides the 100 MHz clock down to 1 MHz
constant SCLK_DIVIDER_VALUE: integer := 2;--100 / 2;
signal sclkdiv: unsigned(1 downto 0) := (others => '0');  -- clock divider counter
signal sclk_unbuf: std_logic := '0';    -- unbuffered serial clock
signal sclk: std_logic := '0';          -- internal serial clock


--wiring
--outputs from VGA_SYNC
signal vid_on : STD_LOGIC;
signal x : STD_LOGIC_VECTOR(9 downto 0);
signal y: STD_LOGIC_VECTOR(8 downto 0);

--inputs to CurrentPixel
signal PIXELUP : STD_LOGIC;
signal PIXELDOWN : STD_LOGIC;
```

```
--  dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
--  clkb : IN STD_LOGIC;
--  enb : IN STD_LOGIC;
--  addrb : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
--  doutb : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
--);
--  end component;

 --BRAM storage for VGA
--stores current color at screen locations
 Component BRAM IS
    PORT (
      clka : IN STD_LOGIC;
      wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
      addra : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
      dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      clkb : IN STD_LOGIC;
      addrb : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
      doutb : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
  END component;
--controller for design
--inputs from rotary encoders turned into control signals
--responsible for moving the current location and clearing
Component cwController is
PORT (     clk : in STD_LOGIC;
           a1 : in STD_LOGIC;
           a2 : in STD_LOGIC;
           b1 : in STD_LOGIC;
           b2 : in STD_LOGIC;
           rst : in STD_LOGIC;
           UP : out STD_LOGIC;
           DOWN : out STD_LOGIC;
           LEFT : out STD_LOGIC;
           RIGHT : out STD_LOGIC;
           CLR : out STD_LOGIC);
  end component;

 Component Mux2x1 is
   Port ( A : in STD_LOGIC_VECTOR (16 downto 0);
          B : in STD_LOGIC_VECTOR (16 downto 0);
          sel : in STD_LOGIC;
          y: out STD_LOGIC_VECTOR (16 downto 0));
end component;

-- SIGNAL DECLARATIONS


-- Signals for the serial clock divider, which divides the 100 MHz clock down to 1 MHz
constant SCLK_DIVIDER_VALUE: integer := 2;--100 / 2;
signal sclkdiv: unsigned(1 downto 0) := (others => '0');  -- clock divider counter
signal sclk_unbuf: std_logic := '0';    -- unbuffered serial clock
signal sclk: std_logic := '0';          -- internal serial clock


--wiring
--outputs from VGA_SYNC
signal vid_on : STD_LOGIC;
signal x : STD_LOGIC_VECTOR(9 downto 0);
signal y: STD_LOGIC_VECTOR(8 downto 0);

--inputs to CurrentPixel
signal PIXELUP : STD_LOGIC;
signal PIXELDOWN : STD_LOGIC;
```

```
        DOWN => PIXELDOWN,
        LEFT => PIXELLEFT,
        RIGHT => PIXELRIGHT,
        curLoc => currentPixel);

--ties x and y together from VGA_sync
        screenLoc <= x(9 downto 1) & y(8 downto 1);

--driven by clear, inputs either the current location on screen
--or the VGA screen sync signal to enable the BRAM to be written over
mux: Mux2x1 port map(
    A => currentPixel,
    B => screenLoc,
    sel => CLR,
    y => wr_add_in
);


--BRAM stores the RGB(8) colors of the current location
--read-write enabled during vid_on to ensure only legitimate positions on screen stored
--to protect against porches being stored
memEn(0) <= vid_on; --need to feed into bram as a std_logic_vector
--mem: blk_mem_gen_1 port map(
--    clka => sclk,
--    wea => memEn,
--    addra => wr_add_in,
--    dina => decodedColor,
--    enb => '1',
--    clkb => sclk,
--    addrb => screenLoc,
--    doutb => color);

mem: bram port map(
    clka => sclk,
    wea => memEn,
    addra => wr_add_in,
    dina => decodedColor,
    clkb => sclk,
    addrb => screenLoc,
    doutb => color);

--controller converts rotary signals into Left right up down signals
--additionally drives the clearing of the screen
controls: cwController port map(
    clk => sclk,
    a1 => XA,
    a2 => XB,
    b1 => YA,
    b2 => YB,
    rst => rst,
    UP => PIXELUP,
    DOWN => PIXELDOWN,
    LEFT => PIXELLEFT,
    RIGHT => PIXELRIGHT,
    CLR => CLR);


--splits color into red green and blue for vga
vgaRed <= color(7 downto 5) & '0';
vgaGreen <= color(4 downto 2) & '0';
vgaBlue <= color(1 downto 0) & "00";

end Behavioral;
```

2: Top Testbench

O:\ENGS31\finalProject\finalProject.srcs\sim_1\new\Top_tb.vhd Monday, June 05, 2017 1:35 PM

```
----------------------------------------------------------------------------------
-- Engineer: Taggart Bonham and Will Chisholm
--
-- Create Date: 05/27/2017 01:50:16 PM
-- Design Name: Top_tb
-- Module Name: ControllerTestBench - Behavioral
-- Project Name: Etchasketch
-- Target Devices: BASYS3
-- Description: top test bench
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Top_tb is
end Top_tb;

architecture Behavioral of Top_tb is

component Etchasketch_TOP  is
   Port ( mclk : in STD_LOGIC;
             r,g,b : in STD_LOGIC;
             XA, YA, XB, YB : in STD_LOGIC;
             rst : STD_LOGIC;
             H_sync : out STD_LOGIC;
              V_sync : out STD_LOGIC;
              vgaGreen, vgaRed, vgaBlue: out STD_LOGIC_VECTOR(3 downto 0) );
end component;

signal clk, rst : std_logic:= '0';
signal ax : std_logic:= '0';
signal ay : std_logic:= '0';
signal bx : std_logic:= '0';
signal by : std_logic:= '0';
signal r, g, b : std_logic:= '1';
signal H_sync : std_logic;
signal V_sync : std_logic;
signal vgaGreen, vgaRed, vgaBlue : std_logic_vector(3 downto 0);


begin
uut :   VGA_CONTROLLER_TOP
Port map ( mclk => clk,
           xa => ax,
           ya => ay,
           xb => bx,
           yb => by,
            rst => rst,
           r => r,
           g => g,
           b => b,
           H_sync => H_sync,
           V_sync => V_sync,
```

```
            vgaBlue => vgaBlue,
            vgaGreen => vgaGreen,
            vgaRed => vgaRed);


clk_proc : process
BEGIN

  CLK <= '0';
  wait for 5 ns;

  CLK <= '1';
  wait for 5 ns;

END PROCESS clk_proc;

stim_proc : process


begin

wait for 80 ns;
ax <= '1';
wait for 80 ns;

wait for 80 ns;

ax <= '0';
wait for 80 ns;
rst <= '1';
wait for 80 ns;
rst<= '0';
bx <= '0';
ax <= '0';
wait for 80 ns;
ax <= '1';
ay <= '1';
wait for 80 ns;

end process;
end Behavioral;
```

3: BASYS Master Constraints

C:\Users\f002qbx\Desktop\FILES TO COPY\Basys3_Master.vhd Thursday, June 01, 2017 10:18 PM

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal
names in the project

##Taggart Bonham and Will Chisholm
##Etch-a-Sketch Constraints file

#Clock signal
set_property PACKAGE_PIN W5 [get_ports mclk]
    set_property IOSTANDARD LVCMOS33 [get_ports mclk]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports mclk]

## Switches
set_property PACKAGE_PIN V17 [get_ports b]
    set_property IOSTANDARD LVCMOS33 [get_ports b]
set_property PACKAGE_PIN V16 [get_ports g]
    set_property IOSTANDARD LVCMOS33 [get_ports g]
set_property PACKAGE_PIN W16 [get_ports r]
    set_property IOSTANDARD LVCMOS33 [get_ports r]

set_property PACKAGE_PIN R2 [get_ports rst]
    set_property IOSTANDARD LVCMOS33 [get_ports rst]



##Pmod Header JA
#    ##Sch name = JA3
    set_property PACKAGE_PIN J2 [get_ports XA]
        set_property IOSTANDARD LVCMOS33 [get_ports XA]
#    ##Sch name = JA4
    set_property PACKAGE_PIN G2 [get_ports XB]
        set_property IOSTANDARD LVCMOS33 [get_ports XB]


##Pmod Header JC
##Sch name = JC1
set_property PACKAGE_PIN K17 [get_ports YA]
    set_property IOSTANDARD LVCMOS33 [get_ports YA]
#Sch name = JC2
set_property PACKAGE_PIN M18 [get_ports YB]
    set_property IOSTANDARD LVCMOS33 [get_ports YB]



##VGA Connector
set_property PACKAGE_PIN G19 [get_ports {vgaRed[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[0]}]
set_property PACKAGE_PIN H19 [get_ports {vgaRed[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[1]}]
set_property PACKAGE_PIN J19 [get_ports {vgaRed[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[2]}]
set_property PACKAGE_PIN N19 [get_ports {vgaRed[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[3]}]
set_property PACKAGE_PIN N18 [get_ports {vgaBlue[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[0]}]
set_property PACKAGE_PIN L18 [get_ports {vgaBlue[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[1]}]
set_property PACKAGE_PIN K18 [get_ports {vgaBlue[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[2]}]
set_property PACKAGE_PIN J18 [get_ports {vgaBlue[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[3]}]
set_property PACKAGE_PIN J17 [get_ports {vgaGreen[0]}]
```

```
    set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[0]}]
set_property PACKAGE_PIN H17 [get_ports {vgaGreen[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[1]}]
set_property PACKAGE_PIN G17 [get_ports {vgaGreen[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[2]}]
set_property PACKAGE_PIN D17 [get_ports {vgaGreen[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[3]}]
set_property PACKAGE_PIN P19 [get_ports H_sync]
    set_property IOSTANDARD LVCMOS33 [get_ports H_sync]
set_property PACKAGE_PIN R19 [get_ports V_sync]
    set_property IOSTANDARD LVCMOS33 [get_ports V_sync]
```

b: Controller — cwController VHDL Module


```
----------------------------------------------------------------------------------
-- Engineer: Taggart Bonham and Will Chisholm
--
-- Create Date: 05/23/2017 09:05:13 PM
-- Design Name:
-- Module Name: CWController- Architecture
-- Project Name: Etch-a-Sketch final project
-- Target Devices: Digilent Basys3 Board
-- Tool Versions: Vivado 2016.1
-- Description: Architecture of the Controller. Bundles outputs of debounced
--rotary encoders inputted into a knob-decoder. This is the controller for the
--system.

----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity cwController is
    Port ( clk : in STD_LOGIC;
           a1 : in STD_LOGIC;
           a2 : in STD_LOGIC;
           b1 : in STD_LOGIC;
           b2 : in STD_LOGIC;
           rst : in STD_LOGIC;
           UP : out STD_LOGIC;
           DOWN : out STD_LOGIC;
           LEFT : out STD_LOGIC;
           RIGHT : out STD_LOGIC;
           CLR : out STD_LOGIC);
end cwController;

architecture Behavioral of cwController is


Component debouncer is
PORT (   clk, button    : in STD_LOGIC;
         result         : out std_logic );
  end component;


Component knobdecoder is
PORT (    clk : in STD_LOGIC;
          a   : in STD_LOGIC;
          b   : in STD_LOGIC;
          cw  : out STD_LOGIC;
          ccw : out STD_LOGIC);
  end component;

signal a1_db, a2_db, b1_db, b2_db : std_logic; --wires to go from debouncers' output to
knobdecoders' input

begin
```

```
clearScreen: process (clk) --synchronizes clear signal
begin
if rising_edge(clk) then
    CLR <= '0';
    if rst = '1' then
        CLR <= '1';
    end if;
end if;
end process;

--debouncers debounce the a and b signals from each of the knobs
dba1: debouncer port map (
    clk => clk,
    button => a1,
    result => a1_db);

dba2: debouncer port map (
    clk => clk,
    button => a2,
    result => a2_db);

dbb1: debouncer port map (
    clk => clk,
    button => b1,
    result => b1_db);

dbb2: debouncer port map (
    clk => clk,
    button => b2,
    result => b2_db);

--knob decoders receive the debounced signals and output their clockwise and counterclockwise
for each knob
--to the controller's 4 directions
horizontal: knobdecoder port map(
    clk => clk,
    a => a1_db,
    b => a2_db,
    cw => RIGHT,
    ccw => LEFT);

vertical: knobdecoder port map(
    clk => clk,
    a => b1_db,
    b => b2_db,
    cw => UP,
    ccw => DOWN);

end Behavioral;
```

c: KnobDecoder – Entity in Controller

1. VHDL Code


```
----------------------------------------------------------------------------------
-- Engineer: Taggart Bonham and Will Chisholm
--
-- Create Date: 05/23/2017 09:05:13 PM
-- Design Name:
-- Module Name: KnobDecoder- Behavioral
-- Project Name: Etch-a-Sketch final project
-- Target Devices: Digilent Basys3 Board
-- Tool Versions: Vivado 2016.1
-- Description: Translates debounced A,B signals from a rotary encoder
--and puts out CW/CCW control signals based on the rotation. This state machine
--is the underlying logic behind our controller.

----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity knobdecoder is
    Port ( a : in STD_LOGIC;  --knob signal a
           b : in STD_LOGIC;  --knob signal b
           cw : out STD_LOGIC;  --1 if knob is turning clockwise, 0 else
           ccw : out STD_LOGIC;  --1 if knob is turning counter-clockwise, 0 else
           clk : in STD_LOGIC);
end knobdecoder;

architecture Behavioral of knobdecoder is
type statetype is (alpha , beta, gamma, delta); --four states correspond to 4 possible
combinations of AB
--alpha = "00", beta = "10", gamma = "11", delta = "01"
signal currentstate, nextstate : statetype;

begin
process(clk, currentstate, a , b)
begin
    cw <= '0'; --default both directions to '0'
    ccw <= '0';
    nextstate <= currentstate; --default nextstate back to curstate

    if (rising_edge(clk)) then
      currentstate <= nextstate; --uepdate currentstate
    end if;
    case currentstate is
        when alpha => --"00"
            if (a = '1') then
                cw <= '1';
                nextstate <= beta;
            elsif (b = '1') then
                ccw <= '1';
                nextstate <= delta;
            end if;
        when beta => --"10"
            if (b = '1') then
                cw <= '1';
                nextstate <= gamma;
```

```
            elsif (a = '0') then
                ccw <= '1';
                nextstate <= alpha;
            end if;
        when gamma => --"11"
            if (a = '0') then
                cw <= '1';
                nextstate <= delta;
            elsif (b = '0') then
                ccw <= '1';
                nextstate <= beta;
            end if;
        when delta => --"01"
            if (b = '0') then
                cw <= '1';
                nextstate <= alpha;
            elsif (a = '1') then
                ccw <= '1';
                nextstate <= gamma;
            end if;
        end case;
end process;

end Behavioral;
```

2. KnobDecoder Testbench


```
----------------------------------------------------------------------------------
-- Engineer: Taggart Bonham and Will Chisholm
-- Create Date: 05/23/2017 09:05:13 PM
-- Module Name: knobdecoder_tb- Behavioral
-- Project Name: Etch-a-Sketch final project
-- Target Devices: Digilent Basys3 Board
-- Tool Versions: Vivado 2016.1
-- Description: testbench of knobdecoder
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity knobdecoder_tb is
--  Port ( );
end knobdecoder_tb;



architecture Behavioral of knobdecoder_tb is

component knobdecoder is
    Port ( a : in STD_LOGIC;  --knob signal a
           b : in STD_LOGIC;  --knob signal b
           cw : out STD_LOGIC;  --1 if knob is turning clockwise, 0 else
           ccw : out STD_LOGIC;  --1 if knob is turning counter-clockwise, 0 else
           clk : in STD_LOGIC);
end component;


signal A, B, cW, CCW : std_logic := '0';
signal mclk: std_logic := '0';



begin

uut : knobdecoder
port map (clk => mclk,
            a => a,
            b => b,
            cw => cw,
            ccw => ccw);


clk_proc : process
            BEGIN

              mclk <= '0';
              wait for 5 ns;

              mclk <= '1';
              wait for 5 ns;
```

```
            END PROCESS clk_proc;


stim_proc : process
 begin


 --simulation of clockwise knob turning
 wait for 80 ns;
 a <= '1';
 wait for 10 ns;
 b <= '1';
wait for 10 ns;
 a <= '0';
wait for 10 ns;
 b <= '0';

 --ccw turning
 wait for 40 ns;
 b <= '1';
wait for 10 ns;
a <= '1';
 wait for 10 ns;
 b <= '0';
wait for 10 ns;
 a <= '0';
wait for 40 ns;
 end process;

end Behavioral;
```

d: Debouncer – Entity in Controller

new 1 *Thursday, June 01, 2017 10:14 PM*

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;


--WRITTEN BY DAVE
ENTITY debouncer IS

  PORT(
    clk     : IN  STD_LOGIC;  -- constants adjusted for 25 MHZ clk
    button  : IN  STD_LOGIC;  -- input signal to be debounced
    result  : OUT STD_LOGIC); -- debounced signal out
END debouncer;

ARCHITECTURE logic OF debouncer IS
  SIGNAL flipflops   : STD_LOGIC_VECTOR(1 DOWNTO 0); --input flip flops
  SIGNAL counter_set : STD_LOGIC;                    --sync reset to zero
  SIGNAL counter_out : UNSIGNED(10 DOWNTO 0) := (OTHERS => '0');
    --counter size (20 bits gives 10.5ms with 100MHz clock)
BEGIN

  counter_set <= flipflops(0) xor flipflops(1);   --determine when to start/reset counter

  PROCESS(clk)
  BEGIN
    IF(clk'EVENT and clk = '1') THEN
      flipflops(0) <= button;
      flipflops(1) <= flipflops(0);
      If(counter_set = '1') THEN      -- reset counter because input is still changing
        counter_out <= (OTHERS => '0');
--      ELSIF(counter_out(5) = '0') THEN         -- use for simulation
      ELSIF(counter_out(10) = '0') THEN      -- stable input time is not yet met
        counter_out <= counter_out + 1;
      ELSE                                    --stable input time is met
        result <= flipflops(1);
      END IF;
    END IF;
  END PROCESS;
END logic;
```

e: VGA Sync

1: VHDL Module


```
----------------------------------------------------------------------------------
-- Engineer: Taggart Bonham and Will Chisholm
--
-- Create Date: 05/23/2017 09:05:13 PM
-- Design Name:
-- Module Name: VGA_SYNC - Behavioral
-- Project Name: Etch-a-Sketch final project
-- Target Devices: Digilent Basys3 Board
-- Tool Versions: Vivado 2016.1
-- Description:

----------------------------------------------------------------------------------

-- Code your design here
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

ENTITY VGA_SYNC IS
PORT (  clk     :   in  STD_LOGIC; --100 MHz clock
        V_sync  :   out STD_LOGIC;
        H_sync  :   out STD_LOGIC;
        video_on:   out STD_LOGIC;
        pixel_x :   out STD_LOGIC_vector(9 downto 0);
        pixel_y :   out STD_LOGIC_VECTOR(8 downto 0) );
end VGA_SYNC;


architecture behavior of VGA_SYNC is


--video on signals
signal H_video_on : STD_LOGIC := '1';
signal V_video_on : STD_LOGIC := '1';

--internal signals to determine when last Hsync went
signal h_complete : STD_LOGIC := '1'; --used to take place of rising_edge(H_sync)

--VGA Constants

constant left_border : integer := 48;
constant h_display : integer := 640;
constant right_border : integer := 16;
constant h_retrace : integer := 96;
constant HSCAN : integer := left_border + h_display + right_border + h_retrace - 1; --number of
PCLKs in an H_sync period

--counts
signal H_CNT : integer:= 0;
signal V_CNT : integer:= 0;

constant top_border : integer := 29;
constant v_display : integer := 480;
constant bottom_border : integer := 10;
constant v_retrace : integer := 2;
constant VSCAN : integer := top_border + v_display + bottom_border + v_retrace - 1; --number of
H_syncs in an V_sync period

BEGIN

hCounter: process(clk)
begin
if rising_edge(clk) then
    if (H_CNT < HSCAN) then
```

```
        H_CNT <= H_CNT + 1;
    elsif(H_CNT = HSCAN) then
        H_CNT <= 0;
    end if;
end if;
end process;

vCounter: process(clk)
begin
if rising_edge(clk) then
    if (H_complete = '1') then -- works as rising clk on h
         if (V_CNT < VSCAN) then
            V_CNT <= V_CNT + 1;
        elsif(V_CNT = VSCAN) then
            V_CNT <= 0;
    end if;
end if;
end if;
end process;

H_Logic: process (H_CNT)
begin
    H_sync <= '1';
    H_video_on <= '1';
    H_complete <= '0';

    pixel_x <=STD_LOGIC_VECTOR(to_unsigned(H_CNT, 10));

       if (H_CNT <  h_display) then
         H_video_on <= '1';
         H_sync <= '1';
           H_complete <= '0';

         elsif (H_CNT < h_display + right_border) then
            H_video_on <= '0';
             H_sync <= '1';
           H_complete <= '0';

         elsif ( H_CNT <  h_display + right_border + h_retrace) then
            H_video_on <= '0';
             H_sync <= '0';
             H_complete <= '0';

          elsif (H_CNT < HSCAN) then
                    H_video_on <= '0';
                     H_sync <= '1';
                H_complete <= '0';

          elsif (H_CNT = HSCAN) then
            H_video_on <= '0';
             H_complete <= '1';
              H_sync <= '1';

          end if;
          end process;



--V_sync generating process
Vert_logic : process(V_CNT)
begin


   V_sync <= '1';
```

```
   V_video_on <= '1';

   pixel_y <=STD_LOGIC_VECTOR(to_unsigned(V_CNT, 9));

      if (V_CNT <  v_display) then
        V_video_on <= '1';
        V_sync <= '1';

        elsif (V_CNT < v_display + bottom_border) then
           V_video_on <= '0';
            V_sync <= '1';

        elsif ( V_CNT <  v_display + bottom_border + v_retrace) then
           V_video_on <= '0';
            V_sync <= '0';

         elsif (V_CNT <= VSCAN) then
                   V_video_on <= '0';
                    V_sync <= '1';

         end if;

end process;

video_on <= H_video_on AND V_video_on;

end behavior;
```

2: VGA SYNC Testbench


```
----------------------------------------------------------------------------------
-- Engineer: Taggart Bonham and Will Chisholm
-- Create Date: 05/23/2017 09:05:13 PM
-- Module Name: vga_sync_tb- Behavioral
-- Project Name: Etch-a-Sketch final project
-- Target Devices: Digilent Basys3 Board
-- Tool Versions: Vivado 2016.1
-- Description: testbench of vga_SYNC
----------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

entity VGA_tb is
end VGA_tb;

architecture testbench of VGA_tb is

component VGA_SYNC IS
PORT (  clk     :   in  STD_LOGIC; --100 MHz clock
        V_sync  :   out STD_LOGIC;
        H_sync  :   out STD_LOGIC;
        video_on:   out STD_LOGIC;
        pixel_x :   out STD_LOGIC_VECTOR(9 downto 0);
        pixel_y :   out STD_LOGIC_VECTOR(8 downto 0));
end component;



signal  clk     :   STD_LOGIC; --100 MHz clock
signal  V_sync  :   STD_LOGIC;
signal  H_sync  :   STD_LOGIC;
signal  video_on:   STD_LOGIC;
signal  pixel_x :   STD_LOGIC_VECTOR(9 downto 0);
signal  pixel_y :   STD_LOGIC_VECTOR(8 downto 0);


begin

uut : VGA_SYNC PORT MAP(
        clk  => CLK,
        V_sync => V_sync,
        H_sync => H_sync,
        Video_on => video_on,
        pixel_x => pixel_x,
        pixel_y => pixel_y);


clk_proc : process
BEGIN

  CLK <= '0';
  wait for 5 ns;

  CLK <= '1';
  wait for 5 ns;

END PROCESS clk_proc;

stim_proc : process
begin



    wait;
```

```
end process stim_proc;
end testbench;
```

f: Current Pixel

1: VHDL Module


```
----------------------------------------------------------------------------------
-- Engineer: Taggart Bonham and Will Chisholm
--
-- Create Date: 05/23/2017 09:05:13 PM
-- Design Name:
-- Module Name: CurPixel- Behavioral
-- Project Name: Etch-a-Sketch final project
-- Target Devices: Digilent Basys3 Board
-- Tool Versions: Vivado 2016.1
-- Description: Logic for the current pixel on the screen. Reacts to control
--signals based on rotary encoder input.

----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity CurPixel is
    Port ( clk: in STD_LOGIC;
           UP : in STD_LOGIC; --tells which direction to move the current location of the pixel
           DOWN : in STD_LOGIC;
           LEFT : in STD_LOGIC;
           RIGHT : in STD_LOGIC;
           curLoc : out STD_LOGIC_VECTOR(16 downto 0)); --new current location of pixel
end CurPixel;

architecture Behavioral of CurPixel is

--x,y locations
-- since 640 x 480, in order to make lines bigger we divided both values by 2 for storage
signal cur_x_loc: unsigned(8 downto 0):= "100000000"; --initializes to a spot far away from the
edges so it's easy to see where the pixel starts
signal cur_y_loc : unsigned(7 downto 0):= "10000000";

constant MAX_X : integer := 320; --screen x from 0 to 640/2 = 320
constant MAX_Y: integer := 240; --y from 0 to 480/2 = 240


begin

xReg: process(clk) --updates x location
begin

if rising_edge(clk) then
    if (RIGHT = '1') then
        if (cur_x_loc + 1 < MAX_X) then
            cur_x_loc <= cur_x_loc + 1;
        end if;

    elsif (LEFT = '1') then
        if (cur_x_loc - 1 > 0) then
            cur_x_loc <= cur_x_loc - 1;
        end if;
    end if;
```

```
end if;

end process;

yReg: process(clk) --updates y location
begin

if rising_edge(clk) then
    if (DOWN = '1') then
        if (cur_y_loc + 1 < MAX_Y) then
            cur_y_loc <= cur_y_loc + 1;
        end if;

    elsif (UP = '1') then
        if (cur_y_loc - 1 > 1) then
            cur_y_loc <= cur_y_loc - 1;
        end if;
    end if;
end if;

end process;

--combines the x and y into one std_logic_vector to output
output: process(clk, cur_x_loc, cur_y_loc)
begin
    if (rising_edge(clk)) then
        curLoc <= STD_LOGIC_VECTOR(cur_x_loc) & STD_LOGIC_VECTOR(cur_y_loc);
    end if;

end process;

end Behavioral;
```

2: Current Pixel Testbench


```
----------------------------------------------------------------------------------
-- Engineer: Taggart Bonham and Will Chisholm
-- Create Date: 05/23/2017 09:05:13 PM
-- Module Name: curpixel_tb- Behavioral
-- Project Name: Etch-a-Sketch final project
-- Target Devices: Digilent Basys3 Board
-- Tool Versions: Vivado 2016.1
-- Description: testbench of curpixel
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity curPixel_tb is
--  Port ( );
end curPixel_tb;



architecture Behavioral of curPixel_tb is

component curpixel is
Port ( clk: in STD_LOGIC;
           UP : in STD_LOGIC; --tells which direction to move the current location of the pixel
           DOWN : in STD_LOGIC;
           LEFT : in STD_LOGIC;
           RIGHT : in STD_LOGIC;
           curLoc : out STD_LOGIC_VECTOR(16 downto 0)); --new current location of pixel
end component;


signal UP : std_logic := '0';
signal DOWN: std_logic := '0';
signal LEFT : std_logic := '0';
signal RIGHT : std_logic := '0';
signal mclk: std_logic := '0';
 signal curLoc : STD_LOGIC_VECTOR(16 downto 0); --new current location of pixel



begin

uut :  curpixel
port map (clk => mclk,
            UP => up,
            DOWN => DOWN,
            LEFT => LEFT,
            RIGHT => RIGHT,
         curLoc => curloc);


clk_proc : process
            BEGIN
```

```
            mclk <= '0';
            wait for 5 ns;

            mclk <= '1';
            wait for 5 ns;

        END PROCESS clk_proc;


stim_proc : process

        begin

        wait for 80 ns;
        UP <= '1';
        wait for 10 ns;
        UP <= '0';
        wait for 80 ns;

        left <= '1';
        Up <= '1';
        wait for 10 ns;
        left <= '0';
        wait for 80 ns;

        end process;

end Behavioral;
```

g: Color Decoder


```
----------------------------------------------------------------------------------
-- Engineer: Taggart Bonham and Will Chisholm
--
-- Create Date: 05/23/2017 09:05:13 PM
-- Design Name:
-- Module Name: COLORDecoder- Behavioral
-- Project Name: Etch-a-Sketch final project
-- Target Devices: Digilent Basys3 Board
-- Tool Versions: Vivado 2016.1
-- Description: Decodes switch RGB input into an 8-bit color vector

----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ColorDecoder is
    Port ( r : in STD_LOGIC; --takes in three switch values and ouputs a combination of those
    colors
           g : in STD_LOGIC;
           b : in STD_LOGIC;
           COLOR : out STD_LOGIC_VECTOR(7 downto 0));
end ColorDecoder;

architecture Behavioral of ColorDecoder is
signal Red, Green : STD_LOGIC_VECTOR(2 downto 0):= (others =>'1'); --3 bits for red and green
signal Blue : STD_LOGIC_VECTOR(1 downto 0):= (others =>'1'); --2 bits for blue
begin

--turns colors on and off based on inputs
Red <= "111" when r = '1' else
       "000" when r = '0';

Green <= "111" when g = '1' else
         "000" when g = '0';

 Blue <= "11" when b = '1' else
                "00" when b = '0';

--wires output from three intermediate wires
COLOR <= Red & Green & Blue;
end Behavioral;
```

h: Mux2x1


```
----------------------------------------------------------------------------------
-- Engineer: Taggart Bonham and Will Chisholm
--
-- Create Date: 05/23/2017 09:05:13 PM
-- Design Name:
-- Module Name: Mux2x1- Behavioral
-- Project Name: Etch-a-Sketch final project
-- Target Devices: Digilent Basys3 Board
-- Tool Versions: Vivado 2016.1
-- Description: A basic mux, used to explicitly show how our clear procedure works

----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Mux2x1 is --simples 2x1 Mux
  Port ( A : in STD_LOGIC_VECTOR (16 downto 0);
         B : in STD_LOGIC_VECTOR (16 downto 0);
         sel : in STD_LOGIC;
         y: out STD_LOGIC_VECTOR (16 downto 0));
end Mux2x1;

architecture Behavioral of Mux2x1 is


begin

with sel select y<= --Mux logic
     A when '0',
    B when '1';


end Behavioral;
```

i: BRAM – Memory


```
-- (c) Copyright 1995-2017 Xilinx, Inc. All rights reserved.
--
-- This file contains confidential and proprietary information
-- of Xilinx, Inc. and is protected under U.S. and
-- international copyright and other intellectual property
-- laws.
--
-- DISCLAIMER
-- This disclaimer is not a license and does not grant any
-- rights to the materials distributed herewith. Except as
-- otherwise provided in a valid license issued to you by
-- Xilinx, and to the maximum extent permitted by applicable
-- law: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND
-- WITH ALL FAULTS, AND XILINX HEREBY DISCLAIMS ALL WARRANTIES
-- AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING
-- BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-
-- INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and
-- (2) Xilinx shall not be liable (whether in contract or tort,
-- including negligence, or under any other theory of
-- liability) for any loss or damage of any kind or nature
-- related to, arising under or in connection with these
-- materials, including for any direct, or any indirect,
-- special, incidental, or consequential loss or damage
-- (including loss of data, profits, goodwill, or any type of
-- loss or damage suffered as a result of any action brought
-- by a third party) even if such damage or loss was
-- reasonably foreseeable or Xilinx had been advised of the
-- possibility of the same.
--
-- CRITICAL APPLICATIONS
-- Xilinx products are not designed or intended to be fail-
-- safe, or for use in any application requiring fail-safe
-- performance, such as life-support or safety devices or
-- systems, Class III medical devices, nuclear facilities,
-- applications related to the deployment of airbags, or any
-- other applications that could lead to death, personal
-- injury, or severe property or environmental damage
-- (individually and collectively, "Critical
-- Applications"). Customer assumes the sole risk and
-- liability of any use of Xilinx products in Critical
-- Applications, subject only to applicable laws and
-- regulations governing limitations on product liability.
--
-- THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS
-- PART OF THIS FILE AT ALL TIMES.
--
-- DO NOT MODIFY THIS FILE.

-- IP VLNV: xilinx.com:ip:blk_mem_gen:8.3
-- IP Revision: 2

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY BRAM IS
  PORT (
    clka : IN STD_LOGIC;
    wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addra : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
    dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    clkb : IN STD_LOGIC;
    addrb : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
    doutb : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
```

```
  );
END BRAM;

ARCHITECTURE BRAM_arch OF BRAM IS
  ATTRIBUTE DowngradeIPIdentifiedWarnings : STRING;
  ATTRIBUTE DowngradeIPIdentifiedWarnings OF BRAM_arch: ARCHITECTURE IS "yes";
  COMPONENT blk_mem_gen_v8_3_2 IS
    GENERIC (
      C_FAMILY : STRING;
      C_XDEVICEFAMILY : STRING;
      C_ELABORATION_DIR : STRING;
      C_INTERFACE_TYPE : INTEGER;
      C_AXI_TYPE : INTEGER;
      C_AXI_SLAVE_TYPE : INTEGER;
      C_USE_BRAM_BLOCK : INTEGER;
      C_ENABLE_32BIT_ADDRESS : INTEGER;
      C_CTRL_ECC_ALGO : STRING;
      C_HAS_AXI_ID : INTEGER;
      C_AXI_ID_WIDTH : INTEGER;
      C_MEM_TYPE : INTEGER;
      C_BYTE_SIZE : INTEGER;
      C_ALGORITHM : INTEGER;
      C_PRIM_TYPE : INTEGER;
      C_LOAD_INIT_FILE : INTEGER;
      C_INIT_FILE_NAME : STRING;
      C_INIT_FILE : STRING;
      C_USE_DEFAULT_DATA : INTEGER;
      C_DEFAULT_DATA : STRING;
      C_HAS_RSTA : INTEGER;
      C_RST_PRIORITY_A : STRING;
      C_RSTRAM_A : INTEGER;
      C_INITA_VAL : STRING;
      C_HAS_ENA : INTEGER;
      C_HAS_REGCEA : INTEGER;
      C_USE_BYTE_WEA : INTEGER;
      C_WEA_WIDTH : INTEGER;
      C_WRITE_MODE_A : STRING;
      C_WRITE_WIDTH_A : INTEGER;
      C_READ_WIDTH_A : INTEGER;
      C_WRITE_DEPTH_A : INTEGER;
      C_READ_DEPTH_A : INTEGER;
      C_ADDRA_WIDTH : INTEGER;
      C_HAS_RSTB : INTEGER;
      C_RST_PRIORITY_B : STRING;
      C_RSTRAM_B : INTEGER;
      C_INITB_VAL : STRING;
      C_HAS_ENB : INTEGER;
      C_HAS_REGCEB : INTEGER;
      C_USE_BYTE_WEB : INTEGER;
      C_WEB_WIDTH : INTEGER;
      C_WRITE_MODE_B : STRING;
      C_WRITE_WIDTH_B : INTEGER;
      C_READ_WIDTH_B : INTEGER;
      C_WRITE_DEPTH_B : INTEGER;
      C_READ_DEPTH_B : INTEGER;
      C_ADDRB_WIDTH : INTEGER;
      C_HAS_MEM_OUTPUT_REGS_A : INTEGER;
      C_HAS_MEM_OUTPUT_REGS_B : INTEGER;
      C_HAS_MUX_OUTPUT_REGS_A : INTEGER;
      C_HAS_MUX_OUTPUT_REGS_B : INTEGER;
      C_MUX_PIPELINE_STAGES : INTEGER;
      C_HAS_SOFTECC_INPUT_REGS_A : INTEGER;
      C_HAS_SOFTECC_OUTPUT_REGS_B : INTEGER;
      C_USE_SOFTECC : INTEGER;
```

```
    C_USE_ECC : INTEGER;
    C_EN_ECC_PIPE : INTEGER;
    C_HAS_INJECTERR : INTEGER;
    C_SIM_COLLISION_CHECK : STRING;
    C_COMMON_CLK : INTEGER;
    C_DISABLE_WARN_BHV_COLL : INTEGER;
    C_EN_SLEEP_PIN : INTEGER;
    C_USE_URAM : INTEGER;
    C_EN_RDADDRA_CHG : INTEGER;
    C_EN_RDADDRB_CHG : INTEGER;
    C_EN_DEEPSLEEP_PIN : INTEGER;
    C_EN_SHUTDOWN_PIN : INTEGER;
    C_EN_SAFETY_CKT : INTEGER;
    C_DISABLE_WARN_BHV_RANGE : INTEGER;
    C_COUNT_36K_BRAM : STRING;
    C_COUNT_18K_BRAM : STRING;
    C_EST_POWER_SUMMARY : STRING
  );
  PORT (
    clka : IN STD_LOGIC;
    rsta : IN STD_LOGIC;
    ena : IN STD_LOGIC;
    regcea : IN STD_LOGIC;
    wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addra : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
    dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
    clkb : IN STD_LOGIC;
    rstb : IN STD_LOGIC;
    enb : IN STD_LOGIC;
    regceb : IN STD_LOGIC;
    web : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addrb : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
    dinb : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    doutb : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
    injectsbiterr : IN STD_LOGIC;
    injectdbiterr : IN STD_LOGIC;
    eccpipece : IN STD_LOGIC;
    sbiterr : OUT STD_LOGIC;
    dbiterr : OUT STD_LOGIC;
    rdaddrecc : OUT STD_LOGIC_VECTOR(16 DOWNTO 0);
    sleep : IN STD_LOGIC;
    deepsleep : IN STD_LOGIC;
    shutdown : IN STD_LOGIC;
    rsta_busy : OUT STD_LOGIC;
    rstb_busy : OUT STD_LOGIC;
    s_aclk : IN STD_LOGIC;
    s_aresetn : IN STD_LOGIC;
    s_axi_awid : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    s_axi_awaddr : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    s_axi_awlen : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    s_axi_awsize : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    s_axi_awburst : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
    s_axi_awvalid : IN STD_LOGIC;
    s_axi_awready : OUT STD_LOGIC;
    s_axi_wdata : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    s_axi_wstrb : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    s_axi_wlast : IN STD_LOGIC;
    s_axi_wvalid : IN STD_LOGIC;
    s_axi_wready : OUT STD_LOGIC;
    s_axi_bid : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    s_axi_bresp : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
    s_axi_bvalid : OUT STD_LOGIC;
    s_axi_bready : IN STD_LOGIC;
```

```
      s_axi_arid : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      s_axi_araddr : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
      s_axi_arlen : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      s_axi_arsize : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
      s_axi_arburst : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
      s_axi_arvalid : IN STD_LOGIC;
      s_axi_arready : OUT STD_LOGIC;
      s_axi_rid : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
      s_axi_rdata : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
      s_axi_rresp : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
      s_axi_rlast : OUT STD_LOGIC;
      s_axi_rvalid : OUT STD_LOGIC;
      s_axi_rready : IN STD_LOGIC;
      s_axi_injectsbiterr : IN STD_LOGIC;
      s_axi_injectdbiterr : IN STD_LOGIC;
      s_axi_sbiterr : OUT STD_LOGIC;
      s_axi_dbiterr : OUT STD_LOGIC;
      s_axi_rdaddrecc : OUT STD_LOGIC_VECTOR(16 DOWNTO 0)
    );
  END COMPONENT blk_mem_gen_v8_3_2;
  ATTRIBUTE X_CORE_INFO : STRING;
  ATTRIBUTE X_CORE_INFO OF BRAM_arch: ARCHITECTURE IS "blk_mem_gen_v8_3_2,Vivado 2016.1";
  ATTRIBUTE CHECK_LICENSE_TYPE : STRING;
  ATTRIBUTE CHECK_LICENSE_TYPE OF BRAM_arch : ARCHITECTURE IS "BRAM,blk_mem_gen_v8_3_2,{}";
  ATTRIBUTE CORE_GENERATION_INFO : STRING;
  ATTRIBUTE CORE_GENERATION_INFO OF BRAM_arch: ARCHITECTURE IS
  "BRAM,blk_mem_gen_v8_3_2,{x_ipProduct=Vivado
  2016.1,x_ipVendor=xilinx.com,x_ipLibrary=ip,x_ipName=blk_mem_gen,x_ipVersion=8.3,x_ipCoreRevisi
  on=2,x_ipLanguage=VHDL,x_ipSimLanguage=VHDL,C_FAMILY=artix7,C_XDEVICEFAMILY=artix7,C_ELABORATIO
  N_DIR=./,C_INTERFACE_TYPE=0,C_AXI_TYPE=1,C_AXI_SLAVE_TYPE=0,C_USE_BRAM_BLOCK=0,C_ENABLE_32BIT_A
  DDRESS=0,C_CTRL_ECC_ALGO=NONE,C_HAS_AXI_ID=0,C_AXI_ID_WIDTH=4,C_MEM_TYPE=1,C_BYTE_SIZE=9,C_ALGO
  RITHM=1,C_PRIM_TYPE=1,C_LOAD_INIT_FILE=0,C_INIT_FILE_NAME=no_coe_file_loaded" &
",C_INIT_FILE=BRAM.mem,C_USE_DEFAULT_DATA=1,C_DEFAULT_DATA=0,C_HAS_RSTA=0,C_RST_PRIORITY_A=CE,C_R
STRAM_A=0,C_INITA_VAL=0,C_HAS_ENA=0,C_HAS_REGCEA=0,C_USE_BYTE_WEA=0,C_WEA_WIDTH=1,C_WRITE_MODE_A=
NO_CHANGE,C_WRITE_WIDTH_A=8,C_READ_WIDTH_A=8,C_WRITE_DEPTH_A=131072,C_READ_DEPTH_A=131072,C_ADDRA
_WIDTH=17,C_HAS_RSTB=0,C_RST_PRIORITY_B=CE,C_RSTRAM_B=0,C_INITB_VAL=0,C_HAS_ENB=0,C_HAS_REGCEB=0,
C_USE_BYTE_WEB=0,C_WEB_WIDTH=1,C_WRITE_MODE_B=READ_FIRST,C_WRITE_WIDTH_B=8,C_READ_WIDTH_B=8,C_WRI
TE_DEPTH_B=13107" &
"2,C_READ_DEPTH_B=131072,C_ADDRB_WIDTH=17,C_HAS_MEM_OUTPUT_REGS_A=0,C_HAS_MEM_OUTPUT_REGS_B=1,C_H
AS_MUX_OUTPUT_REGS_A=0,C_HAS_MUX_OUTPUT_REGS_B=0,C_MUX_PIPELINE_STAGES=0,C_HAS_SOFTECC_INPUT_REGS
_A=0,C_HAS_SOFTECC_OUTPUT_REGS_B=0,C_USE_SOFTECC=0,C_USE_ECC=0,C_EN_ECC_PIPE=0,C_HAS_INJECTERR=0,
C_SIM_COLLISION_CHECK=ALL,C_COMMON_CLK=1,C_DISABLE_WARN_BHV_COLL=0,C_EN_SLEEP_PIN=0,C_USE_URAM=0,
C_EN_RDADDRA_CHG=0,C_EN_RDADDRB_CHG=0,C_EN_DEEPSLEEP_PIN=0,C_EN_SHUTDOWN_PIN=0,C_EN_SAFETY_CKT=0,
C_DISABLE_WARN_B" &
"HV_RANGE=0,C_COUNT_36K_BRAM=32,C_COUNT_18K_BRAM=0,C_EST_POWER_SUMMARY=Estimated Power for
IP           34.891 mW}";
  ATTRIBUTE X_INTERFACE_INFO : STRING;
  ATTRIBUTE X_INTERFACE_INFO OF clka: SIGNAL IS "xilinx.com:interface:bram:1.0 BRAM_PORTA CLK";
  ATTRIBUTE X_INTERFACE_INFO OF wea: SIGNAL IS "xilinx.com:interface:bram:1.0 BRAM_PORTA WE";
  ATTRIBUTE X_INTERFACE_INFO OF addra: SIGNAL IS "xilinx.com:interface:bram:1.0 BRAM_PORTA ADDR";
  ATTRIBUTE X_INTERFACE_INFO OF dina: SIGNAL IS "xilinx.com:interface:bram:1.0 BRAM_PORTA DIN";
  ATTRIBUTE X_INTERFACE_INFO OF clkb: SIGNAL IS "xilinx.com:interface:bram:1.0 BRAM_PORTB CLK";
  ATTRIBUTE X_INTERFACE_INFO OF addrb: SIGNAL IS "xilinx.com:interface:bram:1.0 BRAM_PORTB ADDR";
  ATTRIBUTE X_INTERFACE_INFO OF doutb: SIGNAL IS "xilinx.com:interface:bram:1.0 BRAM_PORTB DOUT";
BEGIN
  U0 : blk_mem_gen_v8_3_2
    GENERIC MAP (
      C_FAMILY => "artix7",
      C_XDEVICEFAMILY => "artix7",
      C_ELABORATION_DIR => "./",
      C_INTERFACE_TYPE => 0,
      C_AXI_TYPE => 1,
      C_AXI_SLAVE_TYPE => 0,
      C_USE_BRAM_BLOCK => 0,
```

```
        C_ENABLE_32BIT_ADDRESS => 0,
        C_CTRL_ECC_ALGO => "NONE",
        C_HAS_AXI_ID => 0,
        C_AXI_ID_WIDTH => 4,
        C_MEM_TYPE => 1,
        C_BYTE_SIZE => 9,
        C_ALGORITHM => 1,
        C_PRIM_TYPE => 1,
        C_LOAD_INIT_FILE => 0,
        C_INIT_FILE_NAME => "no_coe_file_loaded",
        C_INIT_FILE => "BRAM.mem",
        C_USE_DEFAULT_DATA => 1,
        C_DEFAULT_DATA => "0",
        C_HAS_RSTA => 0,
        C_RST_PRIORITY_A => "CE",
        C_RSTRAM_A => 0,
        C_INITA_VAL => "0",
        C_HAS_ENA => 0,
        C_HAS_REGCEA => 0,
        C_USE_BYTE_WEA => 0,
        C_WEA_WIDTH => 1,
        C_WRITE_MODE_A => "NO_CHANGE",
        C_WRITE_WIDTH_A => 8,
        C_READ_WIDTH_A => 8,
        C_WRITE_DEPTH_A => 131072,
        C_READ_DEPTH_A => 131072,
        C_ADDRA_WIDTH => 17,
        C_HAS_RSTB => 0,
        C_RST_PRIORITY_B => "CE",
        C_RSTRAM_B => 0,
        C_INITB_VAL => "0",
        C_HAS_ENB => 0,
        C_HAS_REGCEB => 0,
        C_USE_BYTE_WEB => 0,
        C_WEB_WIDTH => 1,
        C_WRITE_MODE_B => "READ_FIRST",
        C_WRITE_WIDTH_B => 8,
        C_READ_WIDTH_B => 8,
        C_WRITE_DEPTH_B => 131072,
        C_READ_DEPTH_B => 131072,
        C_ADDRB_WIDTH => 17,
        C_HAS_MEM_OUTPUT_REGS_A => 0,
        C_HAS_MEM_OUTPUT_REGS_B => 1,
        C_HAS_MUX_OUTPUT_REGS_A => 0,
        C_HAS_MUX_OUTPUT_REGS_B => 0,
        C_MUX_PIPELINE_STAGES => 0,
        C_HAS_SOFTECC_INPUT_REGS_A => 0,
        C_HAS_SOFTECC_OUTPUT_REGS_B => 0,
        C_USE_SOFTECC => 0,
        C_USE_ECC => 0,
        C_EN_ECC_PIPE => 0,
        C_HAS_INJECTERR => 0,
        C_SIM_COLLISION_CHECK => "ALL",
        C_COMMON_CLK => 1,
        C_DISABLE_WARN_BHV_COLL => 0,
        C_EN_SLEEP_PIN => 0,
        C_USE_URAM => 0,
        C_EN_RDADDRA_CHG => 0,
        C_EN_RDADDRB_CHG => 0,
        C_EN_DEEPSLEEP_PIN => 0,
        C_EN_SHUTDOWN_PIN => 0,
        C_EN_SAFETY_CKT => 0,
        C_DISABLE_WARN_BHV_RANGE => 0,
        C_COUNT_36K_BRAM => "32",
```

```
      C_COUNT_18K_BRAM => "0",
      C_EST_POWER_SUMMARY => "Estimated Power for IP     :     34.891 mW"
    )
    PORT MAP (
      clka => clka,
      rsta => '0',
      ena => '0',
      regcea => '0',
      wea => wea,
      addra => addra,
      dina => dina,
      clkb => clkb,
      rstb => '0',
      enb => '0',
      regceb => '0',
      web => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 1)),
      addrb => addrb,
      dinb => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 8)),
      doutb => doutb,
      injectsbiterr => '0',
      injectdbiterr => '0',
      eccpipece => '0',
      sleep => '0',
      deepsleep => '0',
      shutdown => '0',
      s_aclk => '0',
      s_aresetn => '0',
      s_axi_awid => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 4)),
      s_axi_awaddr => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 32)),
      s_axi_awlen => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 8)),
      s_axi_awsize => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 3)),
      s_axi_awburst => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 2)),
      s_axi_awvalid => '0',
      s_axi_wdata => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 8)),
      s_axi_wstrb => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 1)),
      s_axi_wlast => '0',
      s_axi_wvalid => '0',
      s_axi_bready => '0',
      s_axi_arid => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 4)),
      s_axi_araddr => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 32)),
      s_axi_arlen => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 8)),
      s_axi_arsize => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 3)),
      s_axi_arburst => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 2)),
      s_axi_arvalid => '0',
      s_axi_rready => '0',
      s_axi_injectsbiterr => '0',
      s_axi_injectdbiterr => '0'
    );
END BRAM_arch;
```

C: Resource Utilization for your FPGA

Utilization Report

Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs	341	0	20800	1.64
LUT as Logic	341	0	20800	1.64
LUT as Memory	0	0	9600	0.00
Slice Registers	166	0	41600	0.40
Register as Flip Flop	166	0	41600	0.40
Register as Latch	0	0	41600	0.00

Slice Logic Distribution

Site Type	Used	Fixed	Available	Util%
Slice	121	0	8150	1.48
SLICEL	56	0		
SLICEM	65	0		
LUT as Logic	341	0	20800	1.64
using O6 output only	269			
using O5 and O6	72			
LUT Flip Flop Pairs	113	0	20800	0.54
Unique Control Sets	10			

Memory

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	32	0	50	64.00
RAMB36/FIFO*	32	0	50	64.00
RAMB36E1 only	32			
RAMB18	0	0	100	0.00

IO and GT Specific

```
------------------

+----------------------------+------+-------+-----------+-------+
|          Site Type         | Used | Fixed | Available | Util% |
+----------------------------+------+-------+-----------+-------+
| Bonded IOB                 |   23 |    23 |       106 | 21.70 |
|   IOB Master Pads          |   11 |       |           |       |
|   IOB Slave Pads           |   11 |       |           |       |
+----------------------------+------+-------+-----------+-------+


Clocking
--------

+------------+------+-------+-----------+-------+
|  Site Type | Used | Fixed | Available | Util% |
+------------+------+-------+-----------+-------+
| BUFGCTRL   |    2 |     0 |        32 |  6.25 |
| BUFIO      |    0 |     0 |        20 |  0.00 |
| MMCME2_ADV |    0 |     0 |         5 |  0.00 |
| PLLE2_ADV  |    0 |     0 |         5 |  0.00 |
| BUFMRCE    |    0 |     0 |        10 |  0.00 |
| BUFHCE     |    0 |     0 |        72 |  0.00 |
| BUFR       |    0 |     0 |        20 |  0.00 |
+------------+------+-------+-----------+-------+


Primitives
----------

+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+
| LUT2     |  235 |                 LUT |
| FDRE     |  164 |        Flop & Latch |
| LUT3     |   63 |                 LUT |
| CARRY4   |   55 |          CarryLogic |
| LUT6     |   33 |                 LUT |
| RAMB36E1 |   32 |        Block Memory |
| LUT4     |   30 |                 LUT |
| LUT5     |   26 |                 LUT |
| LUT1     |   26 |                 LUT |
| OBUF     |   14 |                  IO |
| IBUF     |    9 |                  IO |
| FDCE     |    2 |        Flop & Latch |
| BUFG     |    2 |               Clock |
+----------+------+---------------------+
```

D: Critical Timing Path

```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             8.744ns  (required time - arrival time)
  Source:                 sclkdiv_reg[0]/C
                            (rising edge-triggered cell FDRE clocked by sys_clk_pin  {rise@0.000ns fall@5.000ns
period=10.000ns})
  Destination:            sclkdiv_reg[0]/D
                            (rising edge-triggered cell FDRE clocked by sys_clk_pin  {rise@0.000ns fall@5.000ns
period=10.000ns})
  Path Group:             sys_clk_pin
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            10.000ns  (sys_clk_pin rise@10.000ns - sys_clk_pin rise@0.000ns)
  Data Path Delay:        1.252ns  (logic 0.580ns (46.336%)  route 0.672ns (53.664%))
  Logic Levels:           1  (LUT1=1)
  Clock Path Skew:        0.000ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    4.786ns = ( 14.786 - 10.000 )
    Source Clock Delay      (SCD):    5.086ns
    Clock Pessimism Removal (CPR):    0.300ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

Our critical path comes from the Clock divider that turns the Board's 100 MHz clock into a 25 MHz clock in order to interface with the VGA's output. The route starts at the sclkdiv_reg (slow clock divider register) and travels through one level of logic, a look up table, before reaching it's destination, sclkdiv_reg, the same register. The logic portion of the delay is .580 ns, about 45% of the path, and the route takes .670 ns, or 55%. The maximum estimated time of slack, or MET, is 8.744 ns meaning that the design could be clocked faster. However, based on specifications in the datasheets for VGA Sync we saw in class, and the limits of human speed when interfacing with a debounced knob, there is no incentive to run it faster.

E: Analysis of residual warnings

Residual Warnings: 110.

See Table Below for More Details.

Errors 1-8: *Synth 8-3917: design EtchaSketch_TOP has port vga[COLOR][#] driven by constant 0.*

These errors are due to the fact of how we wired the VGA into the FPGA board. As we only output an 8-bit RGB and tied some of the LSBs to 0 as there are ports for 12-bit color, this caused this message. Our wiring here is acceptable and actually preferred as ignoring these ports would be poor form. Furthermore, if these ports are ignored by commenting them out of the Constraints file, then the full RGB colors are never fully realized on the screen, resulting in off-looking coloring.

Error 9-109: *Project 1-486: Could not resolve non-primitive black box cell 'BRAM' instantiated as'mem'[O:/ENGS31/final/final.srcs/sources_1/imports/sources_1/imports/new/VGA_Controller_top.vhd:225].* And, *Synth 8-3331: design blk_mem_gen_mux__parameterized0 has unconnected port [name][#].*

After discussing with Prof. Luke, these errors resulted from the fact that BRAMs have many available features and since we didn't use them all, they were left unconnected. As we do not

need these features and the BRAM runs fine, we determined with Prof. Luke that these errors would not affect performance of our system.

Errors 110: *DRC 23-20:* See below for description.

Unresolved: We followed the instructions in the file and added the following lines to the Constraints file. This changed into a Critical Warning, but the design still worked. Therefore we deleted back the following lines from the Constraints file.

```
set_property CFGBVS GND [current_design]
set_property CONFIG_VOLTAGE VCCO [current_design]
```

#	Name	Details
1	Synth 8-3917	design EtchaSketch_TOP has port vgaBlue[1] driven by constant 0
2	Synth 8-3917	design EtchaSketch_TOP has port vgaBlue[0] driven by constant 0
3	Synth 8-3917	design EtchaSketch_TOP has port vgaGreen[0] driven by constant 0
4	Synth 8-3917	design EtchaSketch_TOP has port vgaRed[0] driven by constant 0
5	Synth 8-3917	design EtchaSketch_TOP has port vgaBlue[1] driven by constant 0
6	Synth 8-3917	design EtchaSketch_TOP has port vgaBlue[0] driven by constant 0
7	Synth 8-3917	design EtchaSketch_TOP has port vgaGreen[0] driven by constant 0
8	Synth 8-3917	design EtchaSketch_TOP has port vgaRed[0] driven by constant 0
9	Project 1-486	Could not resolve non-primitive black box cell 'BRAM' instantiated as 'mem' [O:/ENGS31/final/final.srcs/sources_1/imports/sources_1/imports/new/VGA_Controller_top.vhd:225]
10	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port MUX_RST[0]
11	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port MEM_LAT_RST
12	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port MUX_REGCE[0]
13	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port WE
14	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[16]
15	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[15]

16	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[14]
17	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[13]
18	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[12]
19	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[11]
20	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[10]
21	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[9]
22	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[8]
23	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[7]
24	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[6]
25	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[5]
26	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[4]
27	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[3]
28	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[2]
29	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[1]

30	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port ADDR_IN[0]
31	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[63]
32	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[62]
33	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[61]
34	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[60]
35	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[59]
36	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[58]
37	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[57]
38	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[56]
39	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[55]
40	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[54]
41	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[53]
42	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[52]
43	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[51]

44	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[50]
45	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[49]
46	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[48]
47	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[47]
48	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[46]
49	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[45]
50	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[44]
51	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[43]
52	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[42]
53	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[41]
54	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[40]
55	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[39]
56	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[38]
57	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[37]

58	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[36]
59	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[35]
60	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[34]
61	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[33]
62	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[32]
63	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[31]
64	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[30]
65	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[29]
66	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[28]
67	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[27]
68	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[26]
69	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[25]
70	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[24]
71	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[23]

72	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[22]
73	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[21]
74	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[20]
75	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[19]
76	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[18]
77	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[17]
78	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[16]
79	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[15]
80	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[14]
81	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[13]
82	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[12]
83	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[11]
84	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[10]
85	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[9]

86	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[8]
87	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[7]
88	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[6]
89	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[5]
90	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[4]
91	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[3]
92	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[2]
93	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[1]
94	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port SBITERRIN[0]
95	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port DBITERRIN[63]
96	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port DBITERRIN[62]
97	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port DBITERRIN[61]
98	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port DBITERRIN[60]
99	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port DBITERRIN[59]

100	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port DBITERRIN[58]
101	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port DBITERRIN[57]
102	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port DBITERRIN[56]
103	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port DBITERRIN[55]
104	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port DBITERRIN[54]
105	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port DBITERRIN[53]
106	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port DBITERRIN[52]
107	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port DBITERRIN[51]
108	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port DBITERRIN[50]
109	Synth 8-3331	design blk_mem_gen_mux__parameterized0 has unconnected port DBITERRIN[49]

110	DRC 23-20	Rule violation (CFGBVS-1) Missing CFGBVS and CONFIG_VOLTAGE Design Properties - Neither the CFGBVS nor CONFIG_VOLTAGE voltage property is set in the current_design. Configuration bank voltage select (CFGBVS) must be set to VCCO or GND, and CONFIG_VOLTAGE must be set to the correct configuration voltage, in order to determine the I/O voltage support for the pins in bank 0. It is suggested to specify these either using the 'Edit Device Properties' function in the GUI or directly in the XDC file using the following syntax: set_property CFGBVS value1 [current_design] #where value1 is either VCCO or GND set_property CONFIG_VOLTAGE value2 [current_design] #where value2 is the voltage provided to configuration bank 0 Refer to the device configuration user guide for more information.

3: Memory Map

Memory Map

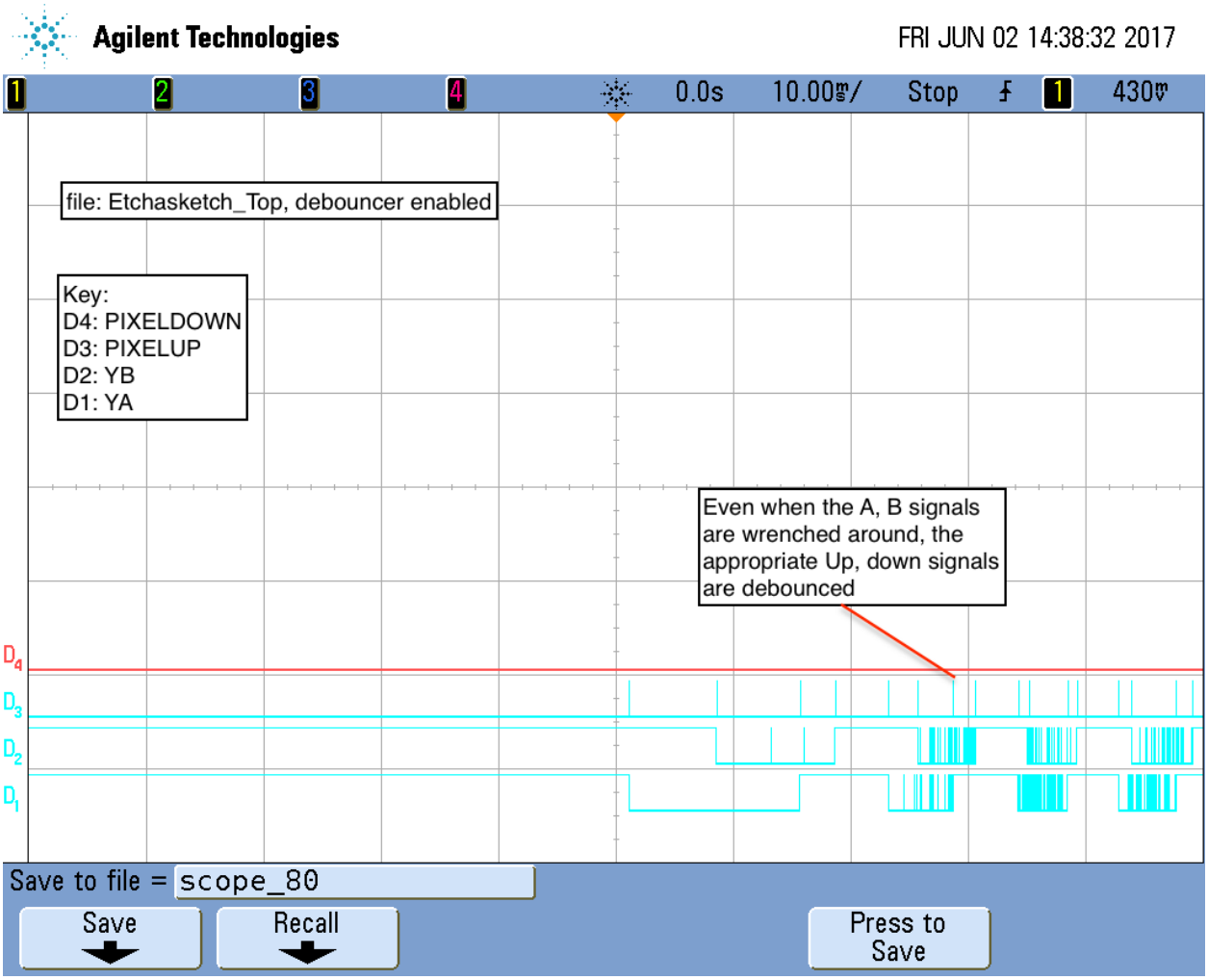
Video Ram:

RAM Address	RAM Data
00000000000000000	0 -> 2^8
00000000000000001	00->ff
⋮	⋮
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ |----------X----------| |----------Y---------|	
⋮	
2^{17} - 1	

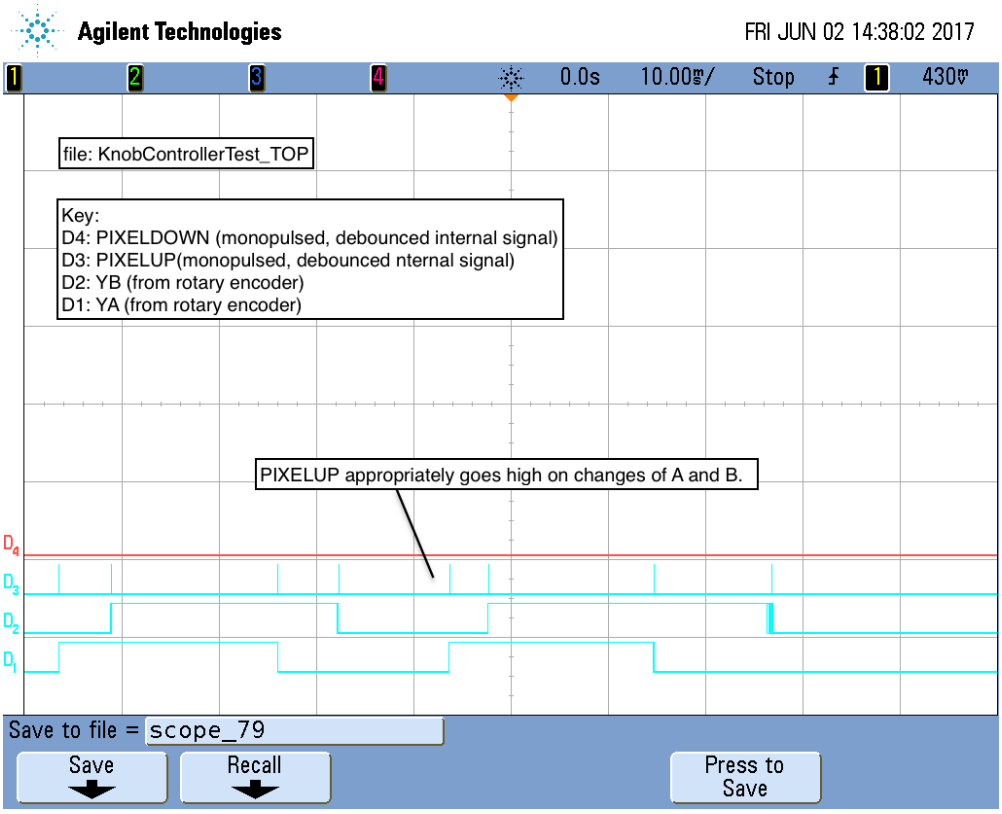
4: Waveform graphs

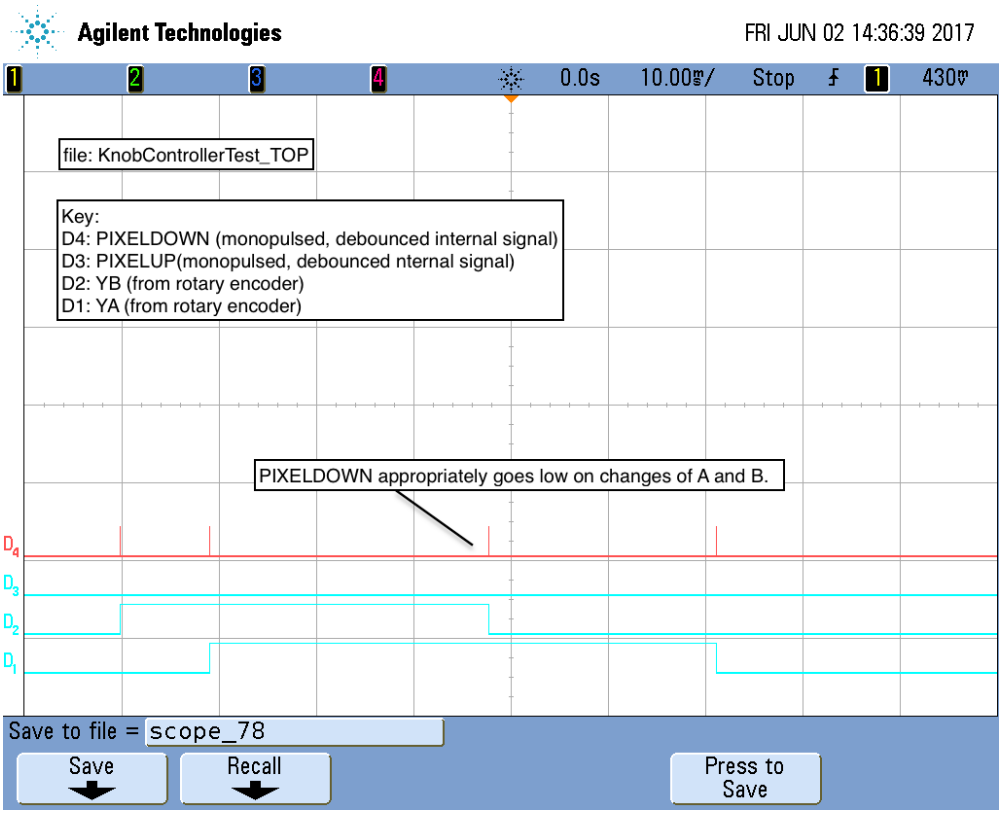
A: CurPixel

B: Debouncer

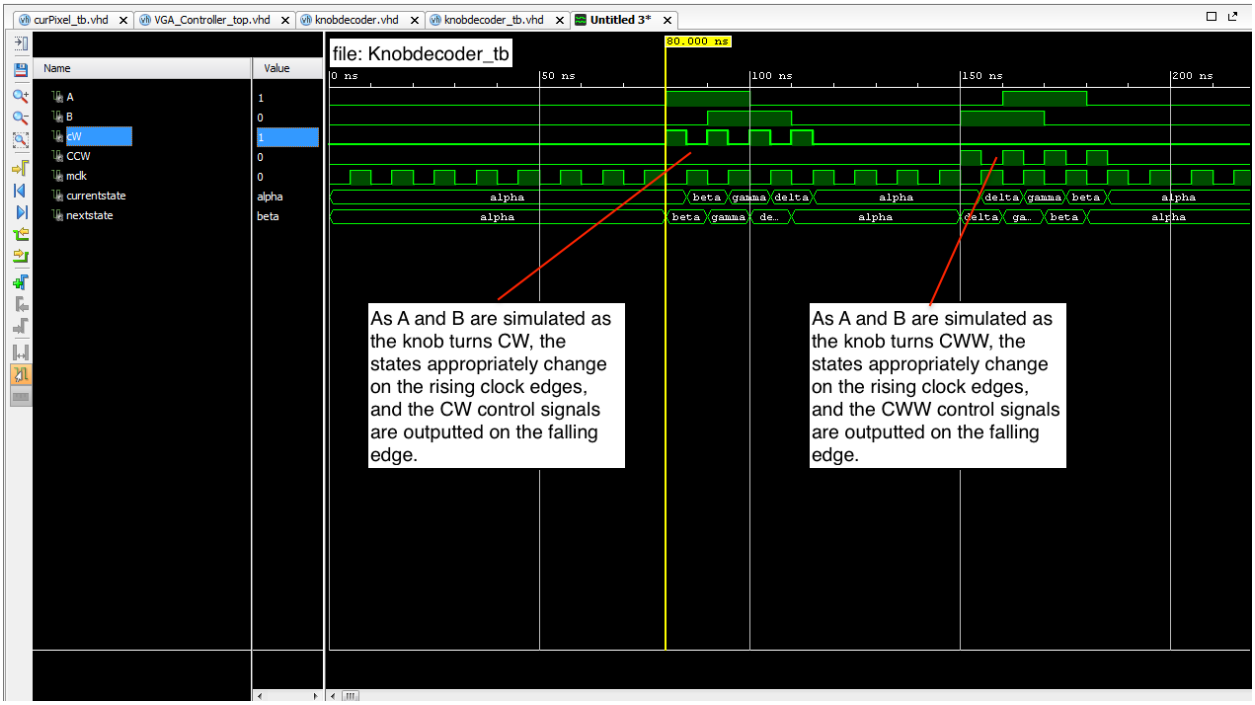

Agilent Technologies
FRI JUN 02 14:38:32 2017
1
2
3
4
0.0s
10.00ms/
Stop
1
430mV
file: Etchasketch_Top, debouncer enabled
Key:
D4: PIXELDOWN
D3: PIXELUP
D2: YB
D1: YA
Even when the A, B signals are wrenched around, the appropriate Up, down signals are debounced
D4
D3
D2
D1
Save to file = scope_80
Save
Recall
Press to Save

C: KnobController



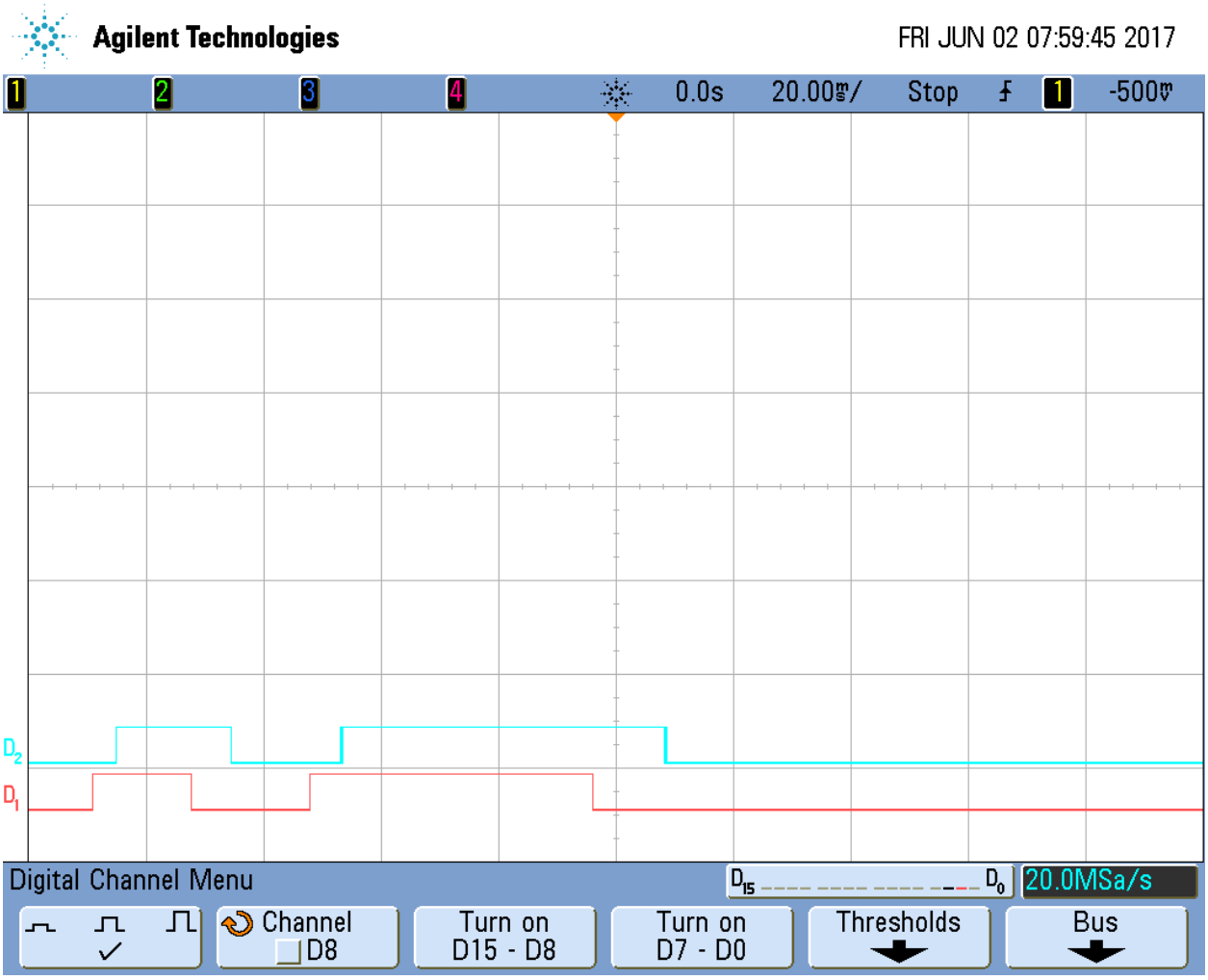


D: KnobDecoder



E: Knob Wiring

The red line represents the A signal from one of the knobs, which falls first when turning clockwise, and the blue line represents the B signal from one of the knobs, which falls second when turning clockwise.



5: Datasheets

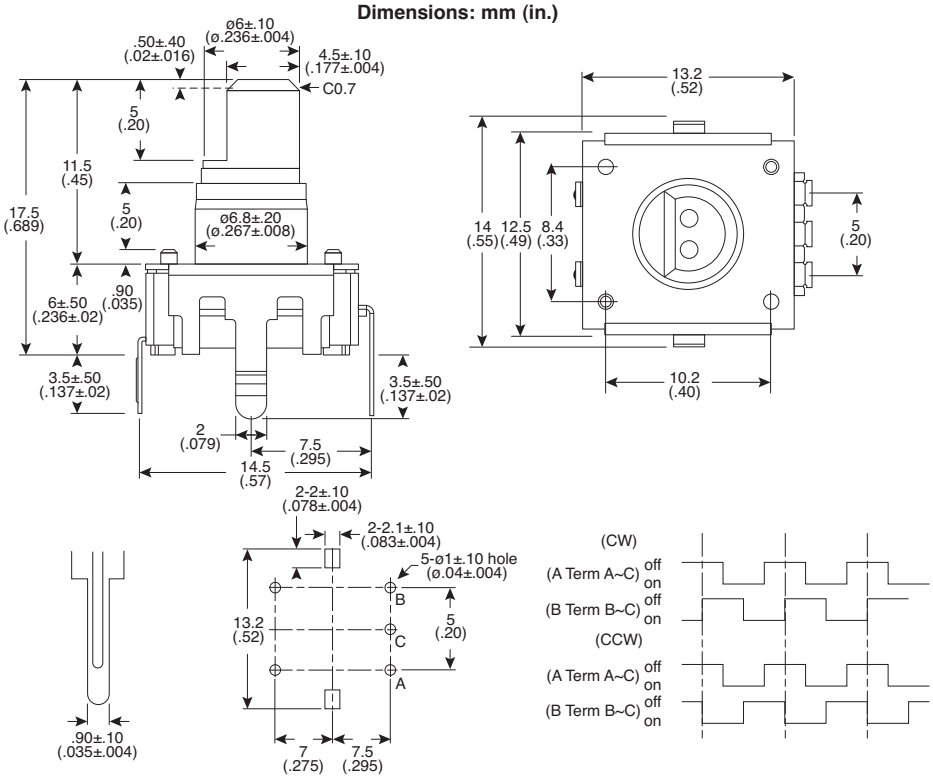
A: Rotary Encoder

Date: 07-16-08 **Ref.#:** TW-700198

ALPHA

Rotary Encoder
318-ENC130175F-12PS

Dimensions: mm (in.)



Mechanical Specifications:
- Operating Temp: -10°C to 70°C
- Storage Temp: -40°C to 85°C
- Rotational Torque: 50gf.cm max.
- No. and Pos. of detents: 12 detents (Step angle 30°±3°)
- Terminal Strength: A static load for 300gf.cm shall be applied to the tip of the terminals for 10 sec. in any direction
- Shaft push-pull strength: 5.1kgf
- Rotational life: 30,000 cycles

Note:
- RoHS Compliant

Electrical Specifications:
- Rating: 1mA/10VDC
- Insulation Resistance: 50VDC 10MΩ Min.
- Dielectric Strength: 50VAC for 1 min.
- Resolution: 12 pulses/360° for each phase

Soldering Specifications:
- Soldering: To be performed in 5 seconds within 260±5°C
- Manual Soldering: To be performed in 3 seconds within 350±5°C
- Preheating: The entire flow duration should not exceed 2 min., and soldering surface temperature (undersurface of PCB) shall be settled within 100°C

Push-on Switch Specifications:
- Type: Single Pole Single Throw (Push on)
- Rating: 10mA/5VDC
- Switch Travel (mm): 0.5±0.4
- Operating Force: 200~460gf
- Operating Life: 20,000 times

Available from Mouser Electronics www.mouser.com (800) 346-6873

B: Quad Encoder Additional Information used as Data Sheet for Rotary Encoder

R

Articles > Understanding Quadrature Encoding

UNDERSTANDING QUADRATURE ENCODING

January 26th, 2009 by Aaron Moore (Original posted to Prototalk.net)

Applications for rotary sensors range from fax machines to computer mice, motor controllers, and even robot localization. Rotary sensors are fundamental to the robotics and motion control systems found today. In this article we'll explore a common and quite popular rotary encoder method called quadrature encoding and also take a look at the Nubotics WheelWatcher, a quadrature encoder for popular dc and servo motors.

A quadrature encoder is a type of incremental encoder which has the ability to indicate both position and direction of movement (as well as velocity). This type of encoder is found in numerous consumer and commercial applications. The versatility of quadrature encoding is one of its strengths as they are capable of high resolution, high precision measurement and can work in a wide range of speeds from a few RPMs to some exceeding 5000RPMs. These encoders typically use optical or magnetic (hall-effect) sensors making them extremely durable and easy to utilize. A variety of ways exist to generate quadrature encoded signals, but first we'll examine the fundamentals of the sensor and how quadrature works.

Fundamentals of the Sensor

Optical sensor are typically used to read data from an encoded track. A light sensing optical sensor passes over an encoded track as shown in Figure 1 and light and dark segments of the encoded track reflect varying degrees of light back to the sensor. The corresponding data is a high or low signal depending on weather the sensor is seeing light or dark. This generates the resulting pulse train data output (again, refer to Figure 1).

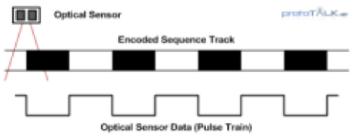


Figure 1 – Track and Sensor Data (Pulse Train)

Figure 2 – Accelerating Pulse Train Sensor Data

This one bit system has two states, high or low. And by counting these transitions and looking at the time between transitions we can determine the velocity of our system. By looking at the transition between high and low we have enough information to determine velocity. As the track moves faster past the optical sensor, the pulse train will increase in frequency, Figure 2. Likewise, as we slow down, the pulses will get wider (less frequent); no change means the system has come to a full stop. From this stationary state, as the system starts to move again,

the pulses return, getting wider as the speed increases. Unfortunately, from this binary one bit system, we can determine velocity, but not position because the direction of rotation is unknown. This lack of direction creates ambiguity (in systems that can change direction), and cannot be used to determine position. An example of an ambiguous pulse train is shown in Figure 3 and this is the limitation of a single sensor/track system; speed but no position.

Figure 3 – Directional Ambiguity of Pulse Train Data

Dealing with Ambiguity

The ambiguity in quadrature encoding is resolved though the additional of another data channel (Figure 4). By offsetting the phase of these two, otherwise identical, tracks by 90 degrees, it establishes two bits of data per cycle (Figure 5). These two channels are referred to as Channel A and Channel B in most quadrature systems.

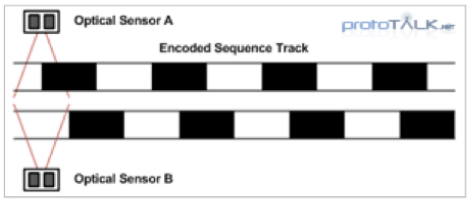


Figure 4 – Dual Sensor Track

With Channels A and B, the track defines 4 quadrants or state in which the system resides at any given moment. The change in state is limited to one of two states (the previous or the next quadrant). This form of encoding created by this phase offset is called Grey code encoding where by only one bit can change at any given moment (no point does the data on both channels change at the same time). By comparing the current state against the next state, direction can be inferred (see truth table in Figure 6).

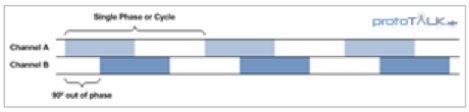


Figure 5 – Quadrature Encoding Phase Offset

Current State			Next State			
CH A	CH B		CH A	CH B		Direction
HIGH	HIGH		HIGH	LOW		Reverse
			LOW	HIGH		Forward
HIGH	LOW		HIGH	HIGH		Forward
			LOW	LOW		Reverse
LOW	HIGH		LOW	LOW		Forward
			HIGH	HIGH		Reverse
LOW	LOW		HIGH	LOW		Forward
			LOW	HIGH		Reverse

Figure 6 – Quadrature Encoding Direction Truth Table

The 90 deg phase offset essentially causes one of the channels to “follow” the other. So by looking at which channel is leading, the direction can be determined. Figure 7 shows this where in forward motion, Channel B is following Channel A during the first half. When A goes high, B goes high after the phase offset delay; likewise when A goes

low, B goes low soon-after. In the second half of the graph, after the change in direction, Channel A follows Channel B indicating that the system is now operating in the reverse direction.

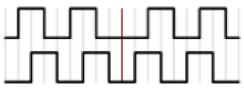


Figure 7 – Direction Change Example

Resolving Ambiguity

Let's revisit the previously ambiguous case (Figure 3) and see how Channel B (highlighted in blue) resolves ambiguity. In Figure 8, it is now clear that no change in direction occurs because Channel B always follows Channel A. We say follow because Channel B reflects what Channel A is doing, but with a delay. Channel A in this case leads the changes indicating that no change in direction occurred.

Figure 8 – No Change in Direction

Figure 9 is the second case where the direction reverses as indicated where Channel B begins to lead. By examine which channel is leading, we can determine the direction.

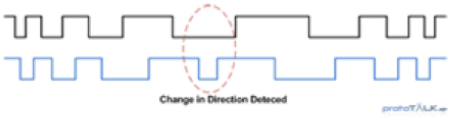


Figure 9 – Direction Change

Wheel Encoders

Up until now we've only looked at linear encoding tracks, and yes this technique does work on linear systems. However, we are also covering rotary encoders so let's take a look at a few wheels encoders. We'll look at some different wheel examples and go over the track resolution.

A wheel encoder is simply an encoding track wrapped on the plane of the wheel. Figure 10 is an example of a very basic encoder wheel with only 2 bits of resolution per revolution. The resolution of the encoder wheel is determined by the number of cycles or complete phases. In this case we have two cycles from black to white on Channel A and B, as such, the resolution of this wheel is 2 bits.

Figure 10 – 2bit Encoder Wheel

This wheel also includes a third bit closest to the center called an index bit. This is sometimes used as a point of reference in rotary encoder systems that support the use of an index bit (typically a 3rd sensor is required to read this bit).

The encoder resolution can be increased by adding more phases to the wheel. In Figure 11 we have a 64 bit encoder wheel. The wheel resolution is 64 bits because of the number of phases. However, since the encoder system uses an 90 degree off-phase signal to provide 4 unique states per phase, the offset provides 4x the resolution meaning it can detect up to 256 position along the wheel!

Figure 11 – 64bit Encoder Wheel

Sometimes it's easier to see things in action. Check out this video overview of the Nubotics WheelWatcher showing how the waveforms look on an oscilloscope as the speed and direction changes.

Optimizing the Encoder

As the resolution increases, sensors placement becomes critical. The sensor placement limits the variety of encoder designs and resolutions that can be used and as a result, placement of the sensor often limits the resolution and

size. However, because of this correlation, we can look at an optimization that simplifies the encoding by incorporating the 90 degree offset into the sensor placement (Figure 12). As long as the 90 degree offset is maintained, the sensors can be separated by multiples of the cycle width (offset by a phase width in multiples of 360 degrees). This not only removes the need to have a redundant encoder track, but gives greater flexibility in the physical placement of the sensor itself. This idea can be extended to encoder wheels as well flat track encoders, although precautions must be taken in the alignment and orientation (rotation) of the sensors when using a wheel.

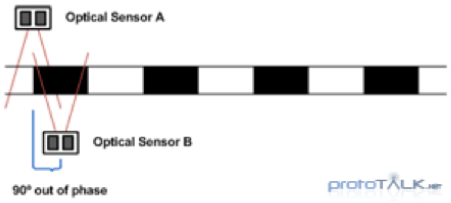


Figure 12 – Offset Sensors

Figure 13 – Single Track Wheel

Nubotics Wheel Watcher

The Nubotics WheelWatcher quadrature encoders are an example of a single track, offset sensor encoder (Figure 14). The reflective code wheel, superimposed over the sensor (Figure 15), shows that the offset is about 2-3 phase widths in addition to the 90 degree offset. Notice that the sensors are rotated to align with the encoder wheel.

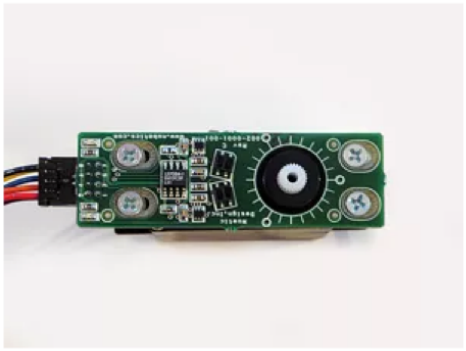


Figure 14 – Nubotics WheelWatcher Encoder

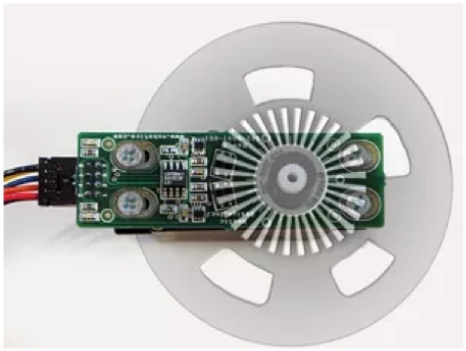


Figure 15 – Encoder with Wheel Overlay

The Nubotics encoder uses a pair of [illegible] reflective photo detectors to generate the A and B channel signals. An LSI LS7084 Quadrature Clock Converter IC is used to decode the A and B channels to produce direction and clock signals. The clock signal can be useful as it removes the need to decode the signal on a separate microcontroller. The 4x output mode of the LS7084 means that a clock is generated at each transition of the 4 states (per phase). The unit itself makes it easy to get rotational data from popular R/C servos and DC gear motors. This purpose built unit is great for small robotics projects where position, direction, and velocity information is needed. The Nubotics WheelWatcher makes it easy to obtain basic odometry information from a platform for localization and other feedback applications.

Summary

Quadrature encoders are an evolution over single output rotary sensors and their added ability to determine direction greatly improves their functionality. This sensor method makes it possible to add accurate positional feedback into a variety of motor control applications and have already been proven effective in numerous consumer applications as a low cost, simple sensor. The variety in encoder designs and sensor placement allow custom solution to be built easily and fit into a variety of platforms. Commercial products such as the Nubotics WheelWatcher make it even easier to integrate with standardized motor drive systems. Their simple yet effective interface provides the kind of data which will help to take robotics to the next level.

C: FPGA-VGA Datasheet

VGA Display Controller

Design Challenge 6

Controller (https://learn.digilentinc.com/list?tag=controller) Comparator (https://learn.digilentinc.com/list?tag=comparators) VGA (https://learn.digilentinc.com/list?tag=vga) Basys2 (https://learn.digilentinc.com/list?tag=basys) Nexys (https://learn.digilentinc.com/list?tag=nexys)

Introduction

In this design, you are going to be asked to do a VGA controller to display something on your monitor. Actually, a VGA controller is quite a simple design which only requires two counters and several comparators.

Before you begin, you should:

- Have the Xilinx® ISE WebPACK™ installed.
- Have your FPGA board set up.
- Be able to implement adders, counters, multiplexers, and comparators behaviorally in Verilog® HDL.

After you're done, you should:

- Understand the timing specification of a VGA interface.
- Be able to design a complex digital system using basic combinational and sequential digital circuit components.

Inventory:

Qty	Description
1	Digilent® Nexys™4 (http://digilentinc.com/Products/Detail.cfm?NavPath=2,400,1184&Prod=NEXYS4), Nexys™3 (http://www.digilentinc.com/Nexys3), Nexys™2 (http://www.digilentinc.com/Nexys2), or Basys™2 (http://www.digilentinc.com/Basys2) FPGA Board
1	Xilinx ISE Design Suite: WebPACK (14.6 Recommended) (http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools.html)
1	Digilent Adept (http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,66,828&Prod=ADEPT2)

Requirements

In this design challenge, you are asked to implement a circuit that generates VGA signals to draw a blue screen on your monitor in a resolution of 640x480. The following materials provide detailed descriptions of how VGA signals operate and some hints on how to design the VGA controller.

Video Graphics Array (VGA)

VGA stands for Video Graphics Array. Initially, it refers specifically to the display hardware first introduced with IBM® PS/2 computer in 1987. With the widespread adoption, it now usually refers to the analog computer display standards (defined by VESA®), the DE-15 Connector (commonly known as VGA connector), or the 640x480 resolution itself.

The analog computer display standards is specified, published, copyrighted and sold by the VESA organization (www.vesa.org). The timing information used in this project is an example of how a VGA monitor might be driven in a 640x480 resolution.

A DE-15 connector, commonly known as a VGA connector, is a three row 15-pin D-subminiature Connector (named after their D-shaped metal shield). The name of each pin is shown in Fig. 1 below. We will only concentrate on the 5 signals out of 15 pins in this project. These signals are Red, Grn, Blue, HS, and VS. Red, Grn, and Blue are three analog signals that specify the color of a point on the screen, while HS and VS provide a positional reference of where the point should be displayed on the screen. By properly driving these five signals according to the VGA timing specification, we can display everything we want on any monitors. To understand how these signals should be driven, we need to take a look at how our monitors actually work.

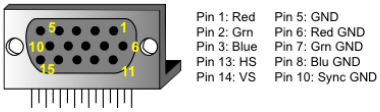


Figure 1. VGA connector.

How do Our Monitors Work?

CRT-based VGA displays use amplitude-modulated moving electron beams (or cathode rays) to display information on a phosphor-coated screen. LCD displays use an array of switches that can impose a voltage across a small amount of liquid crystal, thereby changing light permittivity through the crystal on a pixel-by-pixel basis. Although the following description is limited to CRT displays, LCD displays have evolved to use the same signal timings as CRT displays (so the "signals" discussion below pertains to both CRTs and LCDs). Color CRT displays use three electron beams (one for red, one for blue, and one for green) to energize the phosphor that coats the inner side of the display end of a cathode ray tube (see Fig. 2 below).

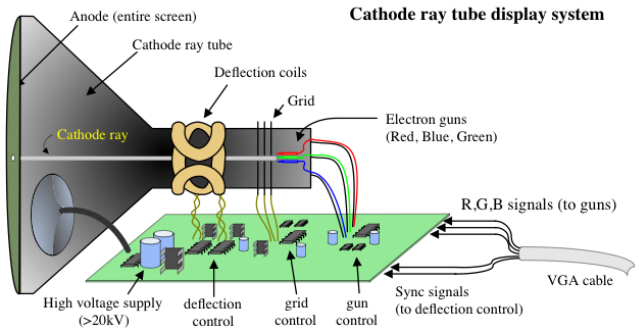


Figure 2. Color CRT display.

Electron beams emanate from "electron guns" which are finely-pointed, heated cathodes placed in close proximity to a positively charged annular plate called a "grid." The electrostatic force imposed by the grid pulls rays of energized electrons from the cathodes, and those rays are fed by the current that flows into the cathodes. These particle rays are initially accelerated towards the grid, but they soon fall under the influence of the much larger electrostatic force that results from the entire phosphor-coated display surface of the CRT being charged to 20kV (or more). The rays are focused to a fine beam as they pass through the center of the grids, and then they accelerate to impact on the phosphor-coated display surface. The phosphor surface glows brightly at the impact point, and it continues to glow for several hundred microseconds after the beam is removed. The larger the current fed into the cathode, the brighter the phosphor will glow.

Between the grid and the display surface, the beam passes through the neck of the CRT where two coils of wire produce orthogonal electromagnetic fields. Because cathode rays are composed of charged particles (electrons), they can be deflected by these magnetic fields. Current waveforms are passed through the coils to produce magnetic fields that interact with the cathode rays and cause them to transverse the display surface in a "raster" pattern, horizontally from left to right and vertically from top to bottom, as shown in Fig. 3. As the cathode ray moves over the surface of the display, the current sent to the electron guns can be increased or decreased to change the brightness of the display at the cathode ray impact point.

Information is only displayed when the beam is moving in the "forward" direction (left to right and top to bottom), and not during the time the beam is reset back to the left or top edge of the display. Much of the potential display time is therefore lost in "blanking" periods when the beam is reset and stabilized to begin a new horizontal or vertical display pass. The size of the beams, the frequency at which the beam can be traced across the display, and the frequency at which the electron beam can be modulated determine the display resolution.

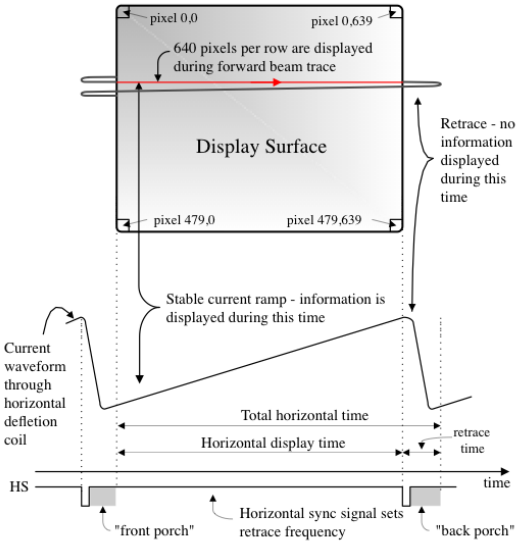


Figure 3. VGA horizontal synchronization.

VGA Timing Specification

Modern VGA displays can accommodate different resolutions, and a VGA controller circuit dictates the resolution by producing timing signals to control the raster patterns. The controller must produce synchronizing pulses at 3.3V (or 5V) to set the frequency at which current flows through the deflection coils, and it must ensure that video data is applied to the electron guns at the correct time. Raster video displays define a number of "rows" that corresponds to the number of horizontal passes the cathode makes over the display area, and a number of "columns" that corresponds to an area on each row that is assigned to one "picture element", or pixel. Typical displays use from 240 to 1200 rows and from 320 to 1600 columns. The overall size of a display and the number of rows and columns determines the size of each pixel.

Video data typically comes from a video refresh memory; with one or more bytes assigned to each pixel location (the Nexys4 uses 12-bits per pixel, Nexys 2, Nexys 3 and Basys2 uses 8-bits). The controller must index into video memory as the beams move across the display, and retrieve and apply video data to the display at precisely the time the electron beam is moving across a given pixel.

A VGA controller circuit must generate the HS and VS timings signals and coordinate the delivery of video data based on the pixel clock. The pixel clock defines the time available to display one pixel of information. The VS signal defines the "refresh" frequency of the display, or the frequency at which all information on the display is redrawn. The minimum refresh frequency is a function of the display's phosphor and electron beam intensity, with practical refresh frequencies falling in the 50Hz to 120Hz range. The number of lines to be displayed at a given refresh frequency defines the horizontal "retrace" frequency.

Example Timing Specification for 640x480@60Hz

Figure 4 and the table below provide the timing specification for 640x480 resolution at 60Hz Frame Rate:

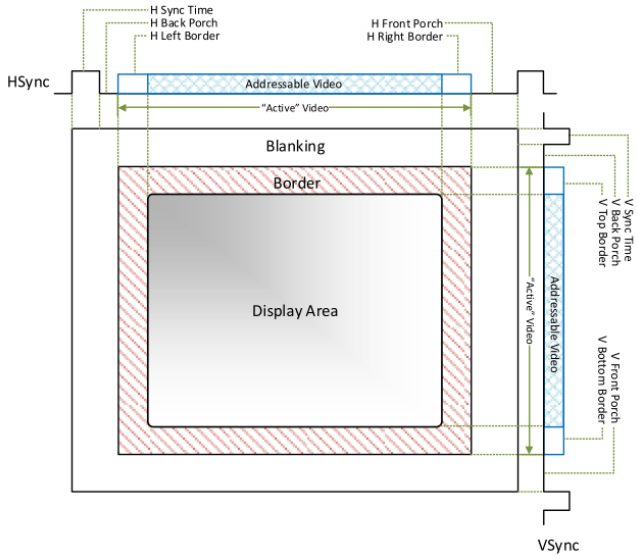


Figure 4. VGA timing specification.

Description	Notation	Time	Width/Freq
Pixel Clock	t_{clk} |	39.7 ns (± 0.5%)	25.175MHz
Hor Sync Time	t_{hs} |	3.813 µs	96 Pixels
Hor Back Porch	t_{hbp} |	1.907 µs	48 Pixels
Hor Front Porch	t_{hfp} |	0.636 µs	16 Pixels
Hor Addr Video Time	t_{haddr} |	25.422 µs	640 Pixels
Hor L/R Border	t_{hbd} |	0 µs	0 Pixels
V Sync Time	t_{vs} |	0.064 ms	2 Lines
V Back Porch	t_{vbp} |	1.048 ms	33 Lines
V Front Porch	t_{vfp} |	0.318 ms	10 Lines
V Addr Video Time	t_{vaddr} |	15.253 ms	480 Lines
V T/B Border	t_{vbd} |	0 ms	0 Lines

Hints

First of all, you need a clock divider to generate the pixel clock, which provides a timing reference to HS and VS signals. The pixel clock frequency is 25.175MHz in the specification. However, with ±0.5% accuracy, 25MHz can be acceptable as well. And obviously, 25MHz is easy to generate on your FPGA boards with the clock divider you implemented in previous projects.

Secondly, you need two counters, a counter (horizontal counter) to count pixels in each line and another counter (vertical counter) to count lines in a frame. The horizontal counter needs to reset itself when it reaches the end of the line (799 in this case), and when it resets itself, it needs to provide a Terminal Count signal to the Enable input of vertical counter so that vertical counter can add 1 when a new line begins. Similarly, vertical counter needs to reset itself when it reaches the end of the frame. So, some changes need to be adapted into the counter you implemented in previous projects into this design.

Based on the counter values, we can compare them to the constant defined in the specification to generate HS and VS signals. Note that you have to drive Red, Grn, and Blue to GND outside the display area. Figure 5 shows the HS and VS generation based on the counter values.

Figure 5. HS and VS generation based on counter values.

The reference block diagram is provided in Fig. 6 below.

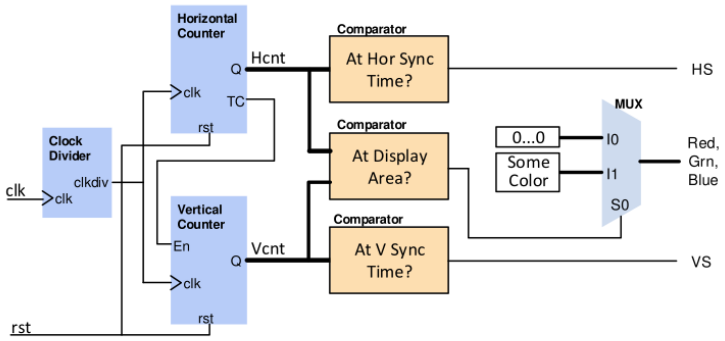


Figure 6. VGA Controller Reference Block Diagram

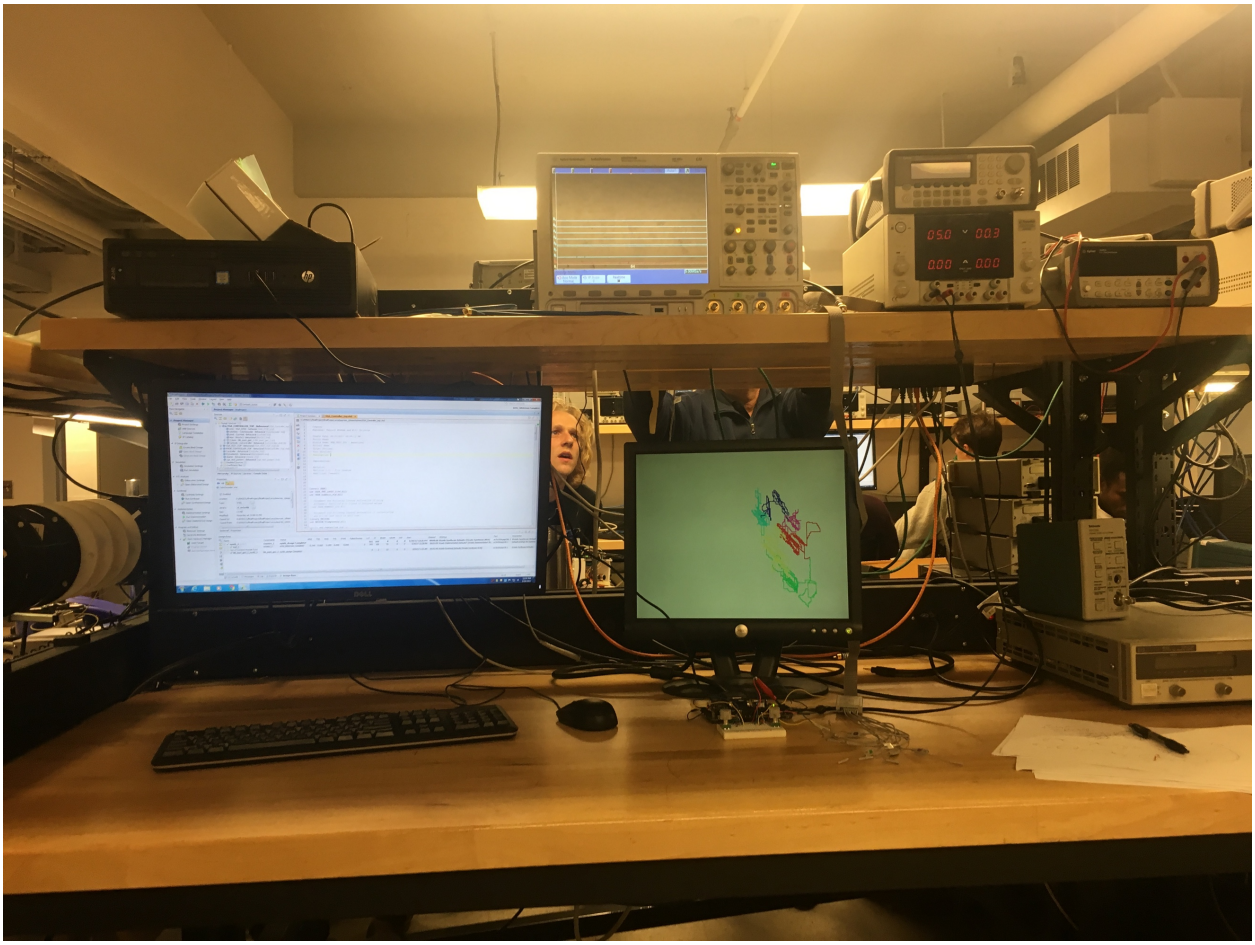
Test Your Knowledge!

Now that you've completed this challenge, try these modifications:

1. Draw a “CrossHair” pattern on the display. A Cross Hair pattern is a 480 pixel high, 1 pixel wide vertical column in the center of the display and a 640 pixel long, 1 pixel high horizontal line across the middle of the display.
2. In your CrossHair pattern, is the vertical line on the screen shaking? Why does that happen? How can you fix it?
3. Can you draw a 5 pixel by 5 pixel square on the screen?
4. Can you make the 5 by 5 square flying on the screen at a speed of 60 pixels per second?
5. Can you make the cross hair pattern rotate on the display slowly?

7: Other documentation

A: System Layout



B: Additional Testing Modules and Files in Final

1: Knob Controller Top: Used to test correct connection of knobs to the controller

a: Top Module


```
----------------------------------------------------------------------------------
-- Engineer: Taggart Bonham and Will Chisholm
-- Create Date: 05/23/2017 09:05:13 PM
-- Module Name: knobcontrollertest_top- Behavioral
-- Project Name: Etch-a-Sketch final project
-- Target Devices: Digilent Basys3 Board
-- Tool Versions: Vivado 2016.1
-- Description: test top for knobs and controllers for oscillioscopes
----------------------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity kct_TOP is
    Port ( mclk : in STD_LOGIC;
           r,g,b : in STD_LOGIC;
           XA, XB, YA, YB : in STD_LOGIC;
           rst : in STD_LOGIC;
           u, d : out STD_LOGIC;
end kct_TOP;

architecture Behavioral of kct_TOP is


--controller for design
--inputs from rotary encoders turned into control signals
--responsible for moving the current location and clearing
Component cwController is
PORT (     clk : in STD_LOGIC;
           a1 : in STD_LOGIC;
           a2 : in STD_LOGIC;
           b1 : in STD_LOGIC;
           b2 : in STD_LOGIC;
           rst : in STD_LOGIC;
           UP : out STD_LOGIC;
           DOWN : out STD_LOGIC;
           LEFT : out STD_LOGIC;
           RIGHT : out STD_LOGIC;
           CLR : out STD_LOGIC);
  end component;


-- Signals for the serial clock divider, which divides the 100 MHz clock down to 1 MHz
constant SCLK_DIVIDER_VALUE: integer := 2;--100 / 2;
signal sclkdiv: unsigned(1 downto 0) := (others => '0');  -- clock divider counter
signal sclk_unbuf: std_logic := '0';    -- unbuffered serial clock
signal sclk: std_logic := '0';          -- internal serial clock


--wiring
--outputs from VGA_SYNC
signal vid_on : STD_LOGIC;
signal x : STD_LOGIC_VECTOR(9 downto 0);
signal y: STD_LOGIC_VECTOR(8 downto 0);

--inputs to CurrentPixel
signal PIXELUP : STD_LOGIC;
signal PIXELDOWN : STD_LOGIC;
signal PIXELLEFT : STD_LOGIC;
```

```
signal PIXELRIGHT : STD_LOGIC;



--inputs to 2x1 Mux
signal CLR : std_logic;
signal currentPixel : STD_LOGIC_VECTOR(16 downto 0);


------------------------------------------------
begin
-- Clock buffer for sclk
-- The BUFG component puts the signal onto the FPGA clocking network

u <= PIXELUP;
d <= PIXELDOWN;

Slow_clock_buffer: BUFG
    port map (I => sclk_unbuf,
              O => sclk );

-- Divide the 100 MHz clock down to 25 MHz, then toggling a flip flop gives the final
-- 1 MHz system clock
Serial_clock_divider: process(mclk)
begin
    if rising_edge(mclk) then
        if sclkdiv = 1 then
            sclkdiv <= (others => '0');
            sclk_unbuf <= NOT(sclk_unbuf);
        else
            sclkdiv <= sclkdiv + 1;
        end if;
    end if;
end process Serial_clock_divider;


--controller converts rotary signals into Left right up down signals
--additionally drives the clearing of the screen
controls: cwController port map(
    clk => sclk,
    a1 => XA,
    a2 => XB,
    b1 => YA,
    b2 => YB,
    rst => '0',
    UP => PIXELUP,
    DOWN => PIXELDOWN,
    LEFT => PIXELLEFT,
    RIGHT => PIXELRIGHT,
    CLR => CLR);

end Behavioral;
```

```
----------------------------------------------------------------------------------
-- Engineer: Taggart Bonham and Will Chisholm
-- Create Date: 05/23/2017 09:05:13 PM
-- Module Name: knobcontrollertest_top- Behavioral
-- Project Name: Etch-a-Sketch final project
-- Target Devices: Digilent Basys3 Board
-- Tool Versions: Vivado 2016.1
-- Description: test top for knobs and controllers for oscillioscopes
----------------------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity kct_TOP is
    Port ( mclk : in STD_LOGIC;
           r,g,b : in STD_LOGIC;
           XA, XB, YA, YB : in STD_LOGIC;
           rst : in STD_LOGIC;
           u, d : out STD_LOGIC;
end kct_TOP;

architecture Behavioral of kct_TOP is


--controller for design
--inputs from rotary encoders turned into control signals
--responsible for moving the current location and clearing
Component cwController is
PORT (     clk : in STD_LOGIC;
           a1 : in STD_LOGIC;
           a2 : in STD_LOGIC;
           b1 : in STD_LOGIC;
           b2 : in STD_LOGIC;
           rst : in STD_LOGIC;
           UP : out STD_LOGIC;
           DOWN : out STD_LOGIC;
           LEFT : out STD_LOGIC;
           RIGHT : out STD_LOGIC;
           CLR : out STD_LOGIC);
  end component;


-- Signals for the serial clock divider, which divides the 100 MHz clock down to 1 MHz
constant SCLK_DIVIDER_VALUE: integer := 2;--100 / 2;
signal sclkdiv: unsigned(1 downto 0) := (others => '0');  -- clock divider counter
signal sclk_unbuf: std_logic := '0';    -- unbuffered serial clock
signal sclk: std_logic := '0';          -- internal serial clock


--wiring
--outputs from VGA_SYNC
signal vid_on : STD_LOGIC;
signal x : STD_LOGIC_VECTOR(9 downto 0);
signal y: STD_LOGIC_VECTOR(8 downto 0);

--inputs to CurrentPixel
signal PIXELUP : STD_LOGIC;
signal PIXELDOWN : STD_LOGIC;
signal PIXELLEFT : STD_LOGIC;
```

```
signal PIXELRIGHT : STD_LOGIC;



--inputs to 2x1 Mux
signal CLR : std_logic;
signal currentPixel : STD_LOGIC_VECTOR(16 downto 0);


------------------------------------------------
begin
-- Clock buffer for sclk
-- The BUFG component puts the signal onto the FPGA clocking network

u <= PIXELUP;
d <= PIXELDOWN;

Slow_clock_buffer: BUFG
    port map (I => sclk_unbuf,
              O => sclk );

-- Divide the 100 MHz clock down to 25 MHz, then toggling a flip flop gives the final
-- 1 MHz system clock
Serial_clock_divider: process(mclk)
begin
    if rising_edge(mclk) then
        if sclkdiv = 1 then
            sclkdiv <= (others => '0');
            sclk_unbuf <= NOT(sclk_unbuf);
        else
            sclkdiv <= sclkdiv + 1;
        end if;
    end if;
end process Serial_clock_divider;


--controller converts rotary signals into Left right up down signals
--additionally drives the clearing of the screen
controls: cwController port map(
    clk => sclk,
    a1 => XA,
    a2 => XB,
    b1 => YA,
    b2 => YB,
    rst => '0',
    UP => PIXELUP,
    DOWN => PIXELDOWN,
    LEFT => PIXELLEFT,
    RIGHT => PIXELRIGHT,
    CLR => CLR);

end Behavioral;
```

b: Knob-Controller Test BASYS 3

new 1 *Friday, June 02, 2017 2:50 PM*

```
set_property PACKAGE_PIN W5 [get_ports mclk]
set_property IOSTANDARD LVCMOS33 [get_ports mclk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports mclk]

# Switches
set_property PACKAGE_PIN V17 [get_ports b]
set_property IOSTANDARD LVCMOS33 [get_ports b]
set_property PACKAGE_PIN V16 [get_ports g]
set_property IOSTANDARD LVCMOS33 [get_ports g]
set_property PACKAGE_PIN W16 [get_ports r]
set_property IOSTANDARD LVCMOS33 [get_ports r]
set_property PACKAGE_PIN W16 [get_ports rst]
set_property IOSTANDARD LVCMOS33 [get_ports rst]




#Pmod Header JA
#    #Sch name = JA1
    set_property PACKAGE_PIN J1 [get_ports AX]
        set_property IOSTANDARD LVCMOS33 [get_ports AX]
#    #Sch name = JA2
    set_property PACKAGE_PIN L2 [get_ports BX]
        set_property IOSTANDARD LVCMOS33 [get_ports BX]
    ##Sch name = JA3
   set_property PACKAGE_PIN J2 [get_ports XA]
       set_property IOSTANDARD LVCMOS33 [get_ports XA]
    ##Sch name = JA4
   set_property PACKAGE_PIN G2 [get_ports XB]
       set_property IOSTANDARD LVCMOS33 [get_ports XB]


#Pmod Header JB
#Sch name = JB1
set_property PACKAGE_PIN A14 [get_ports u]
set_property IOSTANDARD LVCMOS33 [get_ports u]
#Sch name = JB2
set_property PACKAGE_PIN A16 [get_ports d]
set_property IOSTANDARD LVCMOS33 [get_ports d]



#Pmod Header JC
#Sch name = JC1
set_property PACKAGE_PIN K17 [get_ports YA]
set_property IOSTANDARD LVCMOS33 [get_ports YA]
#Sch name = JC2
set_property PACKAGE_PIN M18 [get_ports YB]
set_property IOSTANDARD LVCMOS33 [get_ports YB]
```