Digital Etch-a-Sketch

ENGS 31 Final Project Report

By: Alison Hagen and Himadri Narasimhamurthy

Abstract

For our final project, we created a digital Etch-A-Sketch. The device receives its input from two rotary encoder modules which determine whether the user is drawing and in what direction (left, right, up, or down), which is outputted to a VGA screen. The knob to the left of the FPGA board controls the right (clockwise) or left (counterclockwise) movement and the one to the right controls the up (clockwise) or down (counterclockwise) movement. Similar to the traditional Etch-A-Sketch, we have an option to clear the screen with the input of a button and we have 8 color options for the drawing occurring on the screen, determined by switches.

Table of Contents

1. Introduction …… 4
2. Design Solution ……... 4
    2.1. Design Specifications ……. 4
    2.2. Operating Instructions …… 4
    2.3. Theory of Operation ……... 5
    2.4. Construction and Debugging …….. 8
3. Justification and Evaluation …….. 9
4. Conclusion …… 10
5. Acknowledgements …….. 10
6. References …… 11
7. Appendices …… 12

1. Introduction

The goal of our project is to build an Etch-a-sketch game where a user can turn knobs to draw an image. The input from one rotary encoder knob directs the horizontal direction of the drawing path and another directs the vertical direction. The user can also choose from a variety of colors that will change based on the selected switch input to make their drawing. In order to clear the drawing and start again on a blank background, the user can press the reset button.

2. Design Solution

2.1 Design Specifications

Our project takes input from the user in through turns of the two rotary encoder knobs to determine pixel drawing position, eight switches for colors, and one button to reset the game. The output is drawn on a VGA screen. Therefore, an FPGA with two attachment PMOS Encoders, 8 switches and one button are required for input and a VGA is required for output. See Appendix A for an image of our working device and Appendices B and C for block diagrams of the final design.

2.2 Operating Instructions

First, connect the FPGA (with two rotary encoders attached to the PMOD locations JA and JB) to a VGA with the connector and plug in the VGA. The knob on the left determines horizontal position and the knob on the right determines vertical position. Each click or step turned on the knob will result in an increment (when turned to the right) or a decrement (when turned to the left) of the pixel coordinate. To change the color at any point, the user can flip up one switch at a time. There are eight switches and each one reflects a different color. If multiple switches are up at the same time of drawing, the colors have precedence from the leftmost to the rightmost, so

they will draw in the color of the leftmost switch which is up. Every line drawn on the screen will appear connected unless the user switches to black (leftmost switch) and moves to a new location; black can also be used like an eraser. The user may turn the knobs as much as they want before pressing the reset button which will clear the drawing and return to a black screen with the "pen" location at the same spot as it was before the clear.

2.3 Theory of Operation

Our circuit works as follows (in depth explanations of the modules are located in this section): When the user turns the rotary encoders, the signals are fed through debouncers and then into a controller which outputs signals to our circuit signaling if the pixel coordinates' values should increase or decrease. Then the circuit changes the color of the new pixel drawing position to whichever one is selected through the switches through saving the pixel's location as the address in the block RAM and the color code as the information. The single switch that is selected of the 8 switches corresponds to a specific color for the outputted pixels.

Debouncer

The first step in our design is debouncing the rotary encoder signals since, from the beginning of the project, we had heard about the noisiness of the encoders. Therefore, we fed each signal, the A and the B from each rotary encoder, through its own respective debouncer. The debouncer, for which code can be found in Appendix N, worked by running the input through a series of flip flops while a counter ran. This allowed for us to get smooth signals which we could then feed into our controller.

Rotary Encoder Controller

This became the most important step in deciphering our rotary encoder signals. As mentioned in Appendix F, the PMODEnc modules are low true signals so we had to

modify our original controller. We also realized through the testbenching process that we had to include many more states to account for all possible combinations of A and B signals that we could receive from the debouncer. The state diagram for the controller can be found in Appendix D and we can see the controller was also testbenched and the results are clear in Appendix I. Therefore, we start in the idle state and wait for an input signal to go low. Then, we move to our first hold state where we wait for the other input signal to go low. If the first goes high before that happens, we must return to idle and this same principle holds for all other hold states. If the second also goes low, we move to our next hold state where we wait for the signal that went down first to return to high. Then we move onto the increment or decrement state which outputs an inc or dec enable signal when the second signal returns to high. Then it goes back to idle.

Colors

The colors module is a simple one, the code for which is found in Appendix P. It takes in the input of 8 switches, each of which corresponds to a 4 bit code which is inputted into memory and then a 12 bit color code which is read out later and dealt with in the VGA part of the constraints file.

Board Top

In order to modularize our code and test just the parts of the code which dealt with the FPGA, we created a top module for the board, found in Appendix O. This initialized four debouncers, one for each signal, and two rotary encoder controllers, one per encoder. It also initialized color switches by initializing the color module and translated the reset button press into a signal.

PixelCalc

We create a module called PixelCalc in order to do the arithmetic for moving across the screen at every click of the encoders. This works as a series of if statements into an increment or decrement statement. We want to increment or decrement depending on if the enable signal has been passed in and if we aren't at a boundary. This code, found in Appendix Q, was also tested with a testbench while we ran into bugs with our boundaries and the results can be found in Appendix H.

Block RAM

We decided to do our memory through a True dual port block RAM that was generated through Vivado. The write enable signal for the A side, or the writing side, was the video on from the VGA controller and we wrote in the mem_color code from the Colors module. Then, for the B side, or the read side, we had a write enable as well, which took in the clear signal as the input and wrote all the addresses to black. The read output for this was the color code at the VGA output's address locations which were deciphered in the top level. The memory map can be found in Appendix G. This made it easy for us to store our addresses after we received a pixel x and y location from PixelCalc as a simple concatenated vector, though it did end up taking up more memory than if we had used arithmetic to simplify the addresses used for memory locations.

VGAController

The VGA Controller module, seen in Appendix R, was very similar to that of Professor Hansen's which was available to us in our class notes. We did not have to modify this for our project much other than adding signals to actually increment pixel x and y locations,

since we were confident in the testbenched waveforms, in Appendix J, and their ability to match those from the class notes.

Top

Finally, as with most projects, we had a Top file, for which the code is in Appendix S, where we wired up all the modules with their FPGA inputs as inputs to the top and the VGA outputs, i.e. Hsync, Vsync, and a 12-bit color vector, as outputs. The only non-modular code in this file would be our color decoder which takes in the memory color code after reading it and translates that into the 12-bit color that is needed for the VGA constraints file.

2.4 Construction and debugging

Initially, we talked through our design and decided to write very modular code. First, we designed and built the state machine for our rotary encoder controllers. Though this code later on changed once we wrote a test bench to simulate the input, this was crucial in understanding how the user's input would be handled in the controller. Next, we focused on the VGA controller and our code to calculate the pixel coordinates based on the VGA and user inputs. After writing a top controller for the board and and overall top controller which combined all of our smaller functions, we decided to focus on BRAM.

We tried to write the BRAM code ourselves instead of using Vivado's tools and this is where we ran into many problems. When trying to run our code, the VGA would simply not turn on and we had many errors about lookup tables. We concluded that the BRAM was not functioning properly and switched to use the Vivado tools for a True Dual Port Ram. After simulating our rotary encoder controller with a test bench, we edited it to add more wait states. Finally, our screen was able to turn on but the game was far from finished.

The initial image we drew on our VGA once working appeared twice, one on top of the other with a unique border limitation that did not fill the entire screen. Our switches changed the colors of the pixels drawn, but not in the order we expected. The reset button was also peculiar and cleared the drawing but changed the background color to the previous pixel color. It also had a random different tone for a vertical rectangle block on the screen that marked the borders for our drawings. To debug this, we looked through our VGA controller and realized that we were mapping into memory with a 10-bit X address and a 9-bit y address which meant that when they became 17-bit addresses, they shifted the full concatenated memory address, causing the images to be multiplied. We solved the problem by dividing both the x and y addresses by two (shifting the vectors) when they were mapped as addresses into the memory.

Our final major bug was one where the right border of the screen was showing up blank even though the pixel location was still incrementing. Initially we assumed this was an error in the VGA controller due to the small size of the border. After some unsuccessful tries shifting the H_video_on signal and Hsync signal by small factors, we realized this was not causing the problem. We then decided to increase the amount of addresses which we could possibly store in our Block RAM. Since we were simply concatenating X and Y for the BRAM address, we were running out of memory early and therefore needed to increase memory depth for the full screen to display. This was able to resolve our issue and leave us with a functional Etch-a-Sketch.

3. Justification and Evaluation

Overall, our project was designed well with very modular code and a resulting working Etch-a-sketch. The colors chosen are bright and the switches are an intuitive way to change it. The way that we decided to have larger scaled pixels for drawing paid off as the images come out clear. Our decision to use Block RAM through Vivado's IP tools also ended up working well and this

was better than the alternative with manually written Block RAM. If we were to do the project again we would consider including the ability to change the background color in addition to the pixel color. In addition to that, we also have some code blocks in our top level which assign final colors which could have been included in our Colors module.

4. Conclusions

Our project successfully reached our goal of allowing the user to draw an image like the classic Etch-a-sketch but in a digital version. As a bonus, our Etch-a-sketch allows you to change colors, even to black so that you can erase or move across the screen without a mark, and reset the screen with a button to start all over again. While we changed our design along the way, most notably our initial rotary encoder controller, VGA controller logic and Block RAM, we think the result is much improved from our initial design. We would recommend future groups to test each section of their code as each part is built, to avoid dealing with lots of issues at once after building the whole game. In general, starting as early as possible is useful since it is hard to predict how much time it will take to fix bugs that come up.

5. Acknowledgements

We would like to thank Professor Hansen for all of his advice and patience during our time designing and coding our project as well as with his guidance when it came to debugging our code. We would also like to thank Dave for all of his patience throughout the project as well as throughout labs this entire term. We would also like to thank all the learning fellows and lab TAs that spent countless hours in the digital lab towards the end of the term helping us reason through a lot of the logic from our device. A huge thanks to Ruthie for being our assigned learning fellow who was always available to offer us guidance on bugs we ran into later on in the construction process.

In general, we worked together whenever possible in terms of writing code so that we would both understand the project and all of its functional blocks. Most of our design process was done together from the block diagrams to the rotary encoder state diagram. Both members of the team were present throughout most of the code writing process and through the debugging process. Himadri produced most of the code for the Pixelcalc and Colors modules and Ali worked on writing the testbenches for testing the modules. The top modules and the rotary encoder controller were written by both. Most of the work that was put into the project, in terms of hours, was done together. Throughout the testing processes, we consulted each other for advice.

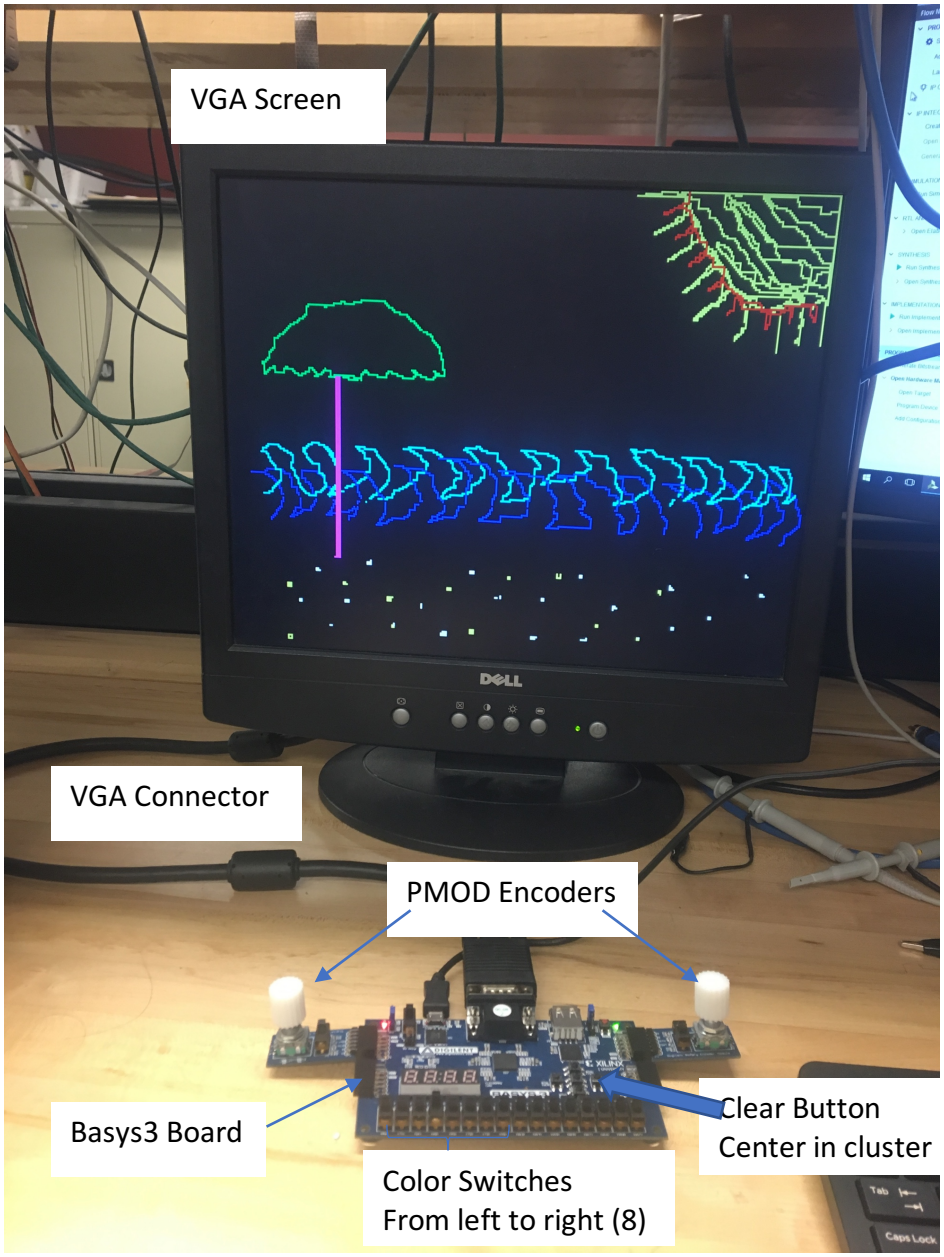
6. References

Our code also has comments where references were included. Most of our code was only referencing previous labs and lectures from Professor Hansen. We consulted Professor Hansen's VGA controller code as well when we ran into some bugs with our own. For our Debouncer module, we sought outside references from Yourigh on Github who had provided a rotary encoder debouncer which we modified for our project.

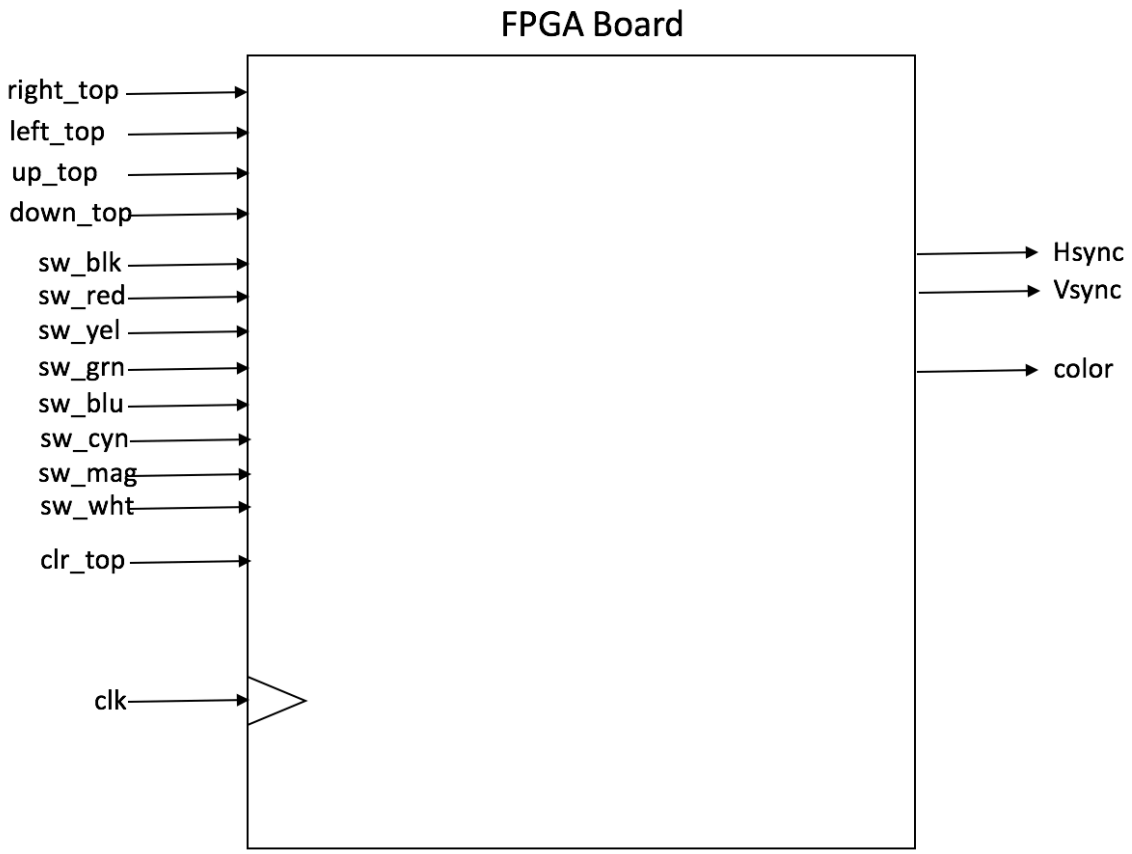
Appendices

ENGS 31: Digital Electronics
Digital Etch-a-Sketch
Alison Hagen and Himadri Narasimhamurthy

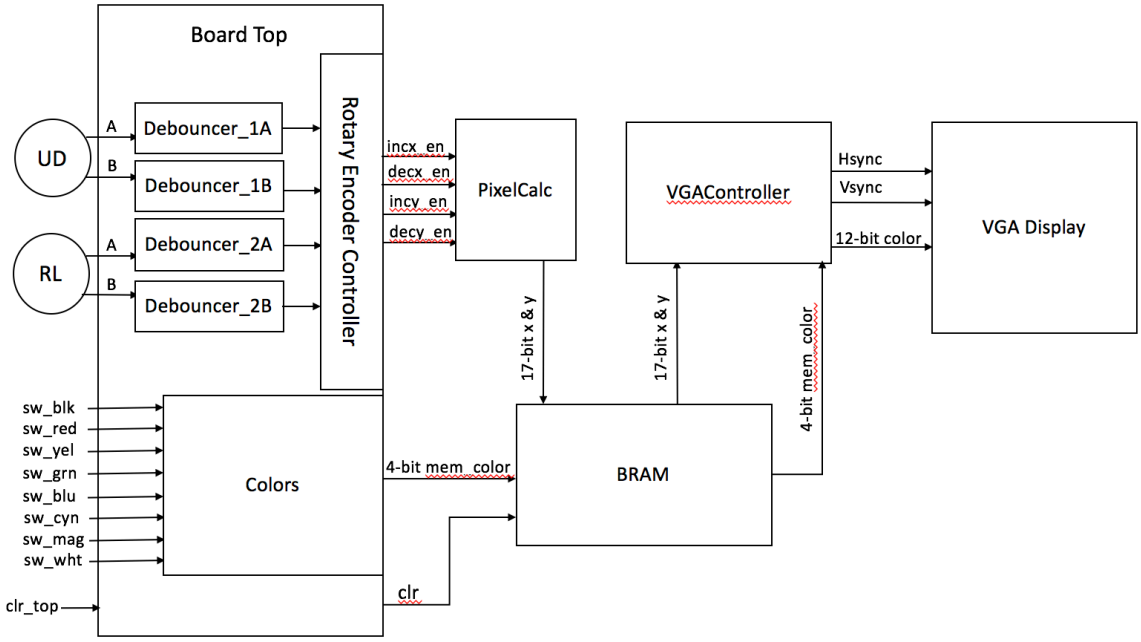
Appendix A: Annotated Photo



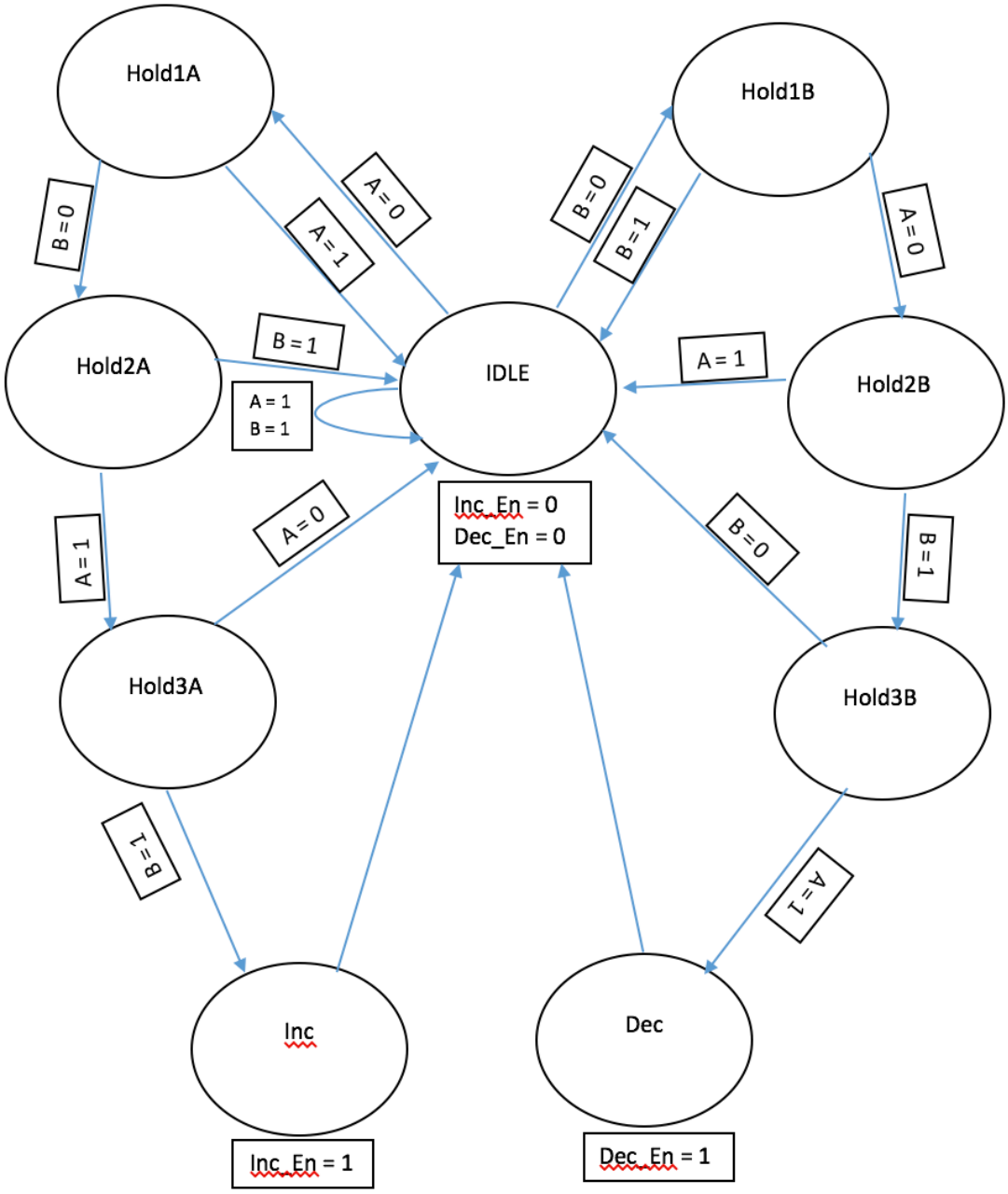
Appendix B: Top Level Block Diagram



Appendix C: Modular Block Diagram



Appendix D: Rotary Encoder State Diagram



Appendix E: Parts List

We make the assumption that the FPGA has the program written on it already – no need to download the bitstream

Reference	Quantity	Part Number	Description
Basys	1	Basys3	The FPGA Board
Digilent	2	PModEnc	The rotary encoders
VGA Screen	1	VGA Screen	Display screen
VGA Connector Cable	1	n/a	Connects the VGA to the Basys board
VGA Power Cord	1	n/a	Connects VGA to power outlet

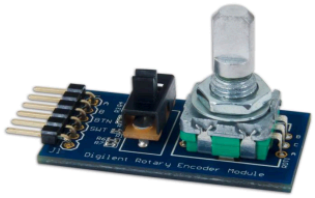
1300 Henley Court
Pullman, WA 99163
509.334.6306
www.digilentinc.com

PmodENC™ Reference Manual

Revised April 12, 2016
This manual applies to the PmodENC rev. A

Overview

The Digilent PmodENC features a rotary shaft encoder with an integral push-button to provide multiple types of outputs. The module also includes a sliding switch that is commonly used as an on/off output. An encoder is commonly used in freely rotating volume knobs to detect how many "clicks" a knob has been rotated.



The PmodENC.

Features include:

- Rotary push-button shaft encoder
- Add multiple types of user input to host board or project
- Additional static slide switch
- Small PCB size for flexible designs 1.5 in × 0.8 in (3.8 cm × 2.0 cm)
- 6-pin Pmod port with GPIO interface
- Follows Digilent Pmod Interface Specification Type 1
- Library and example code available in resource center

1 Functional Description

The PmodENC utilizes a rotary shaft encoder as a way for users to quickly switch between multiple options such as choices shown on a screen or predefined motors speeds. An integral push-button on the shaft as well as a slide switch allow for a highly configurable Pmod.

2 Interfacing with the Pmod

The PmodENC communicates with the host board via the GPIO protocol. It provides four inputs to the system board; the two buttons internal to the encoder that are in quadrature with each other as well as the integral push button on the shaft and the slide switch. A system board will read the integral push button and the slide switch at a logic low voltage in their native (or off in the case of the switch) states.

Copyright Digilent, Inc. All rights reserved.
Other product and company names mentioned may be trademarks of their respective owners.

The two internal buttons are both natively pulled to a logic high level through a pull-up resistor. As the two buttons are located 90 degrees from each other (i.e. in quadrature), while the shaft is rotating one button will be pulled to a low logic level voltage before the other button.

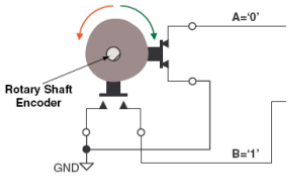


Figure 1. Rotary shaft encoder circuitry.

Users can program their system boards to determine which button was pulled low last (within a small time frame to ensure additional "clicks" are not also captured) in order to figure out which direction the shaft is being rotated.

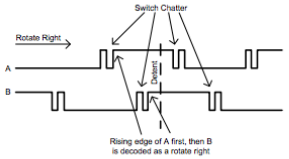


Figure 2. Timing of outputs A and B.

2.1 Pinout Description Table

Pin	Signal	Description
1	A	Output of button A in the encoder shaft
2	B	Output of button B in the encoder shaft
3	BTN	Output of the integral push button in the encoder shaft
4	SWT	Output of the on board switch
5	GND	Power Supply Ground
6	VCC	Positive Power Supply (3.3/5V)

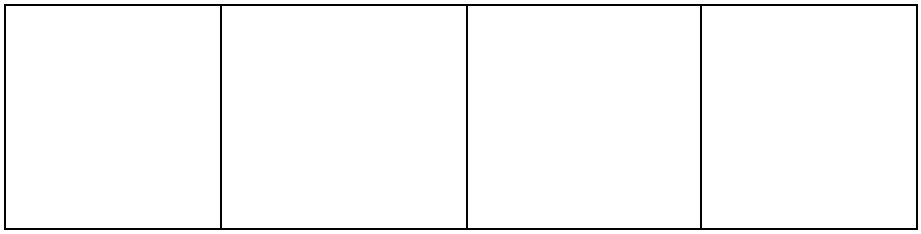
It is recommended that Pmod is operated at 3.3V or 5V, although because there are no integrated circuits on the Pmod, any voltage that your system board can handle as a digital input will work fine.

Copyright Digilent, Inc. All rights reserved.
Other product and company names mentioned may be trademarks of their respective owners.

Appendix G: Memory Map

Saved Information – Mem color

A four bit color code is saved into the address. It is dependent on the switch that is up and ranges from 0000 to 0111 each of which is decoded into a color which is fed into the VGA at the level of the constraints file.

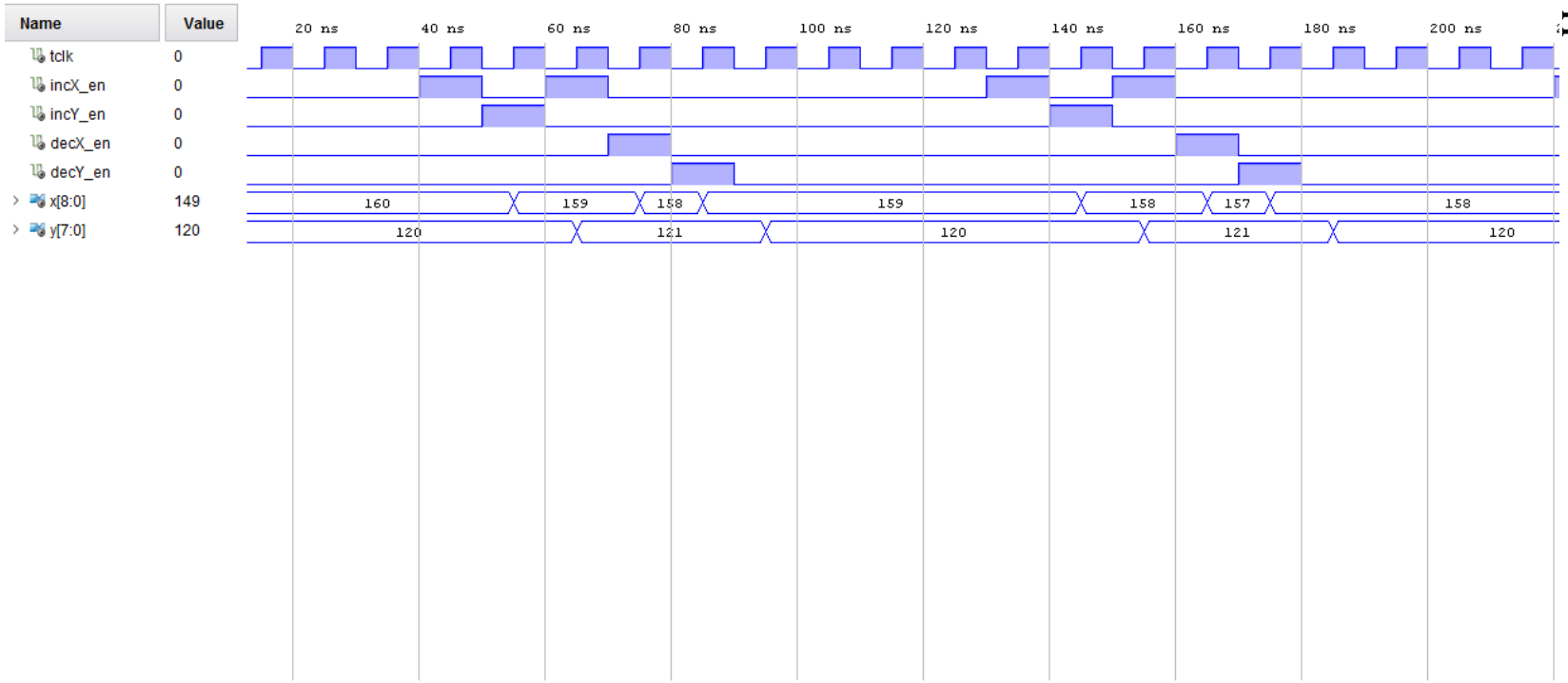


Saved Address – Location

The pixel is saved at a location in the block ram that is fed in after PixelCalc when we concatenate the x vector location and the y vector location into a 17 bit code. This is read back out from memory according to the VGA controller's concatenated x and y vectors and colors are assigned.

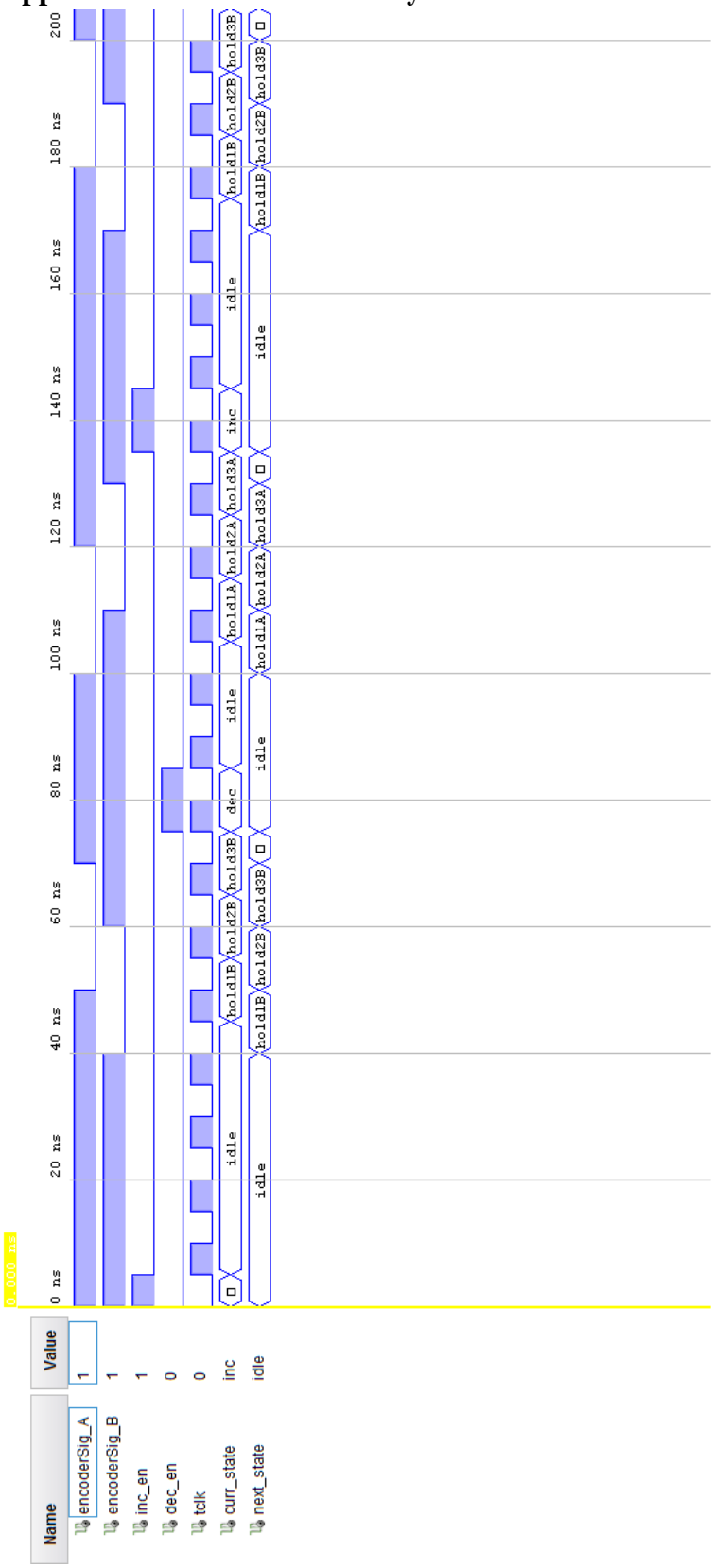
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**y**	**y**	**y**	**y**	**y**	**y**	**y**	**y**

Appendix H: Waveform – PixelCalc Module



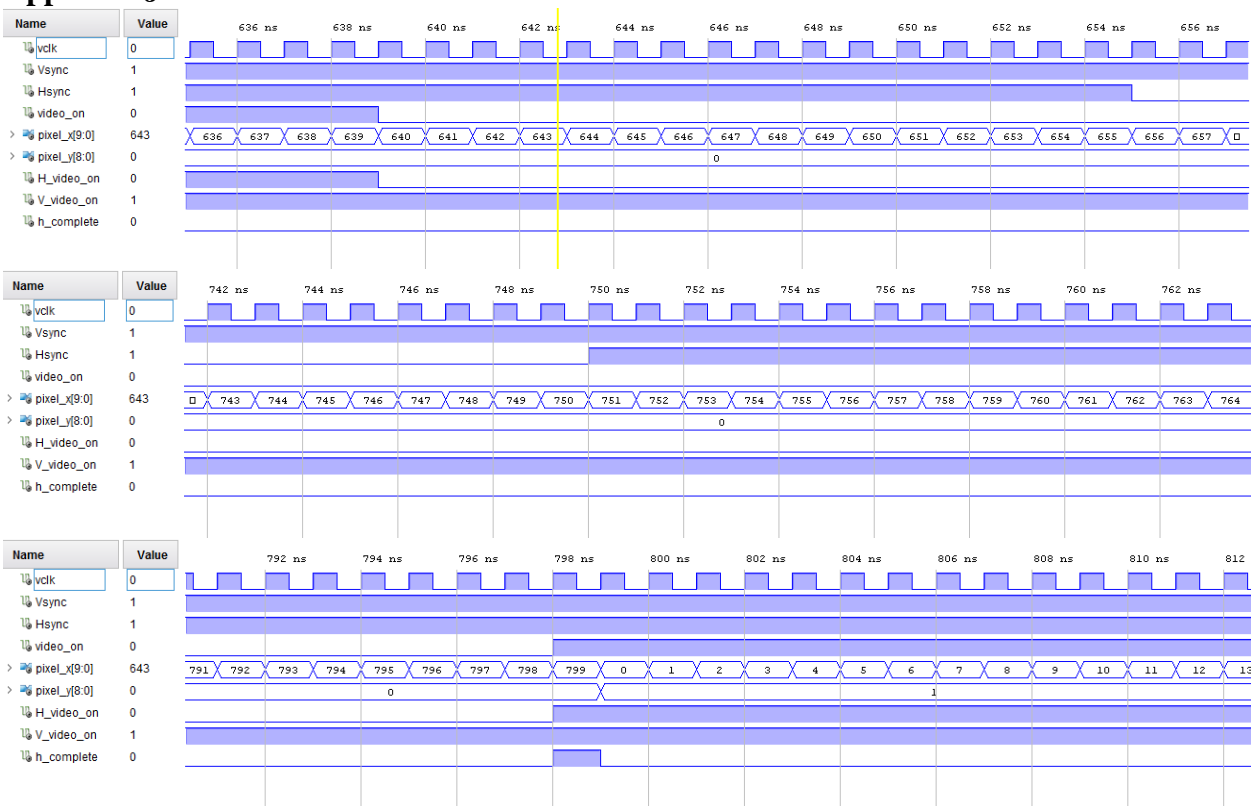
In the following waveform, we are able to see the proper functionality of PixelCalc. The counts increment or decrement only when the inc and dec enable signals are inputted.

Appendix I: Waveform – Rotary Encoder Controller



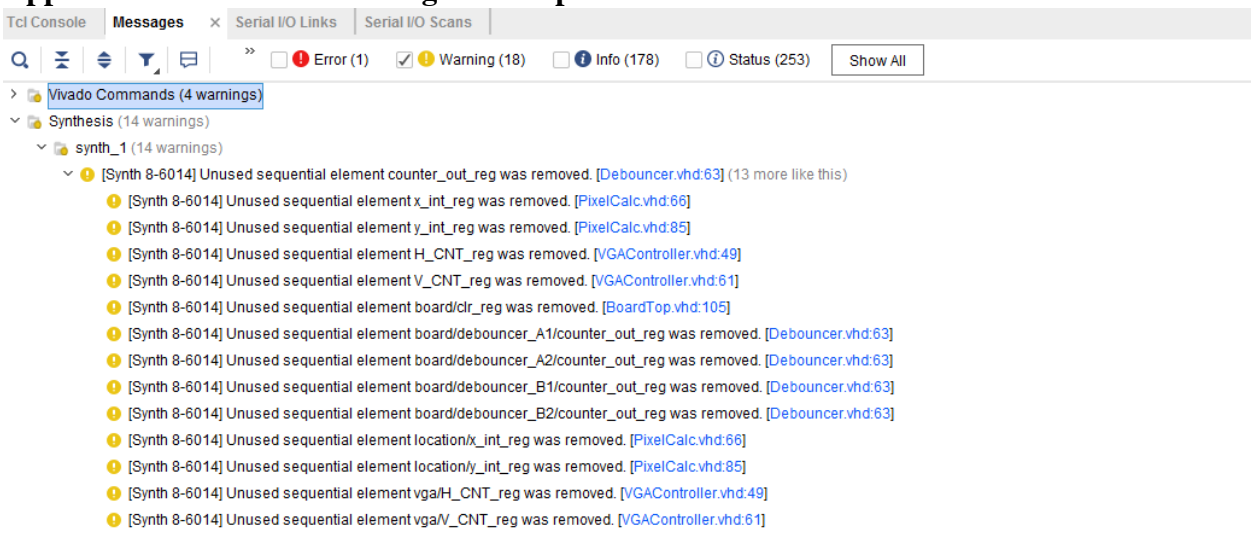
This shows the proper transition of states in the Rotary Encoder Controller module. The output signals are monopulsed and they are not triggered until the A and B signals go through appropriate hold states.

Appendix J: Waveform – VGA Controller



These are the waveforms from our VGA Controller testbench. As we can see, they match the waveform in the notes about the VGA controller as Hsync and Vsync correspond to the proper values and video_on is accurate and dependent on H and V video_on signals.

Appendix K: Residual Warnings and Explanations



The only warnings that were left were the above warnings about unused registers being removed. After consulting with Professor Hansen, we were able to confirm that these were not affecting our code and that the values were still being used and incrementing properly even though Vivado claimed to have removed the registers.

Appendix L: Usage Report

```
1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs              |  261 |     0 |     20800 |  1.25 |
|   LUT as Logic          |  261 |     0 |     20800 |  1.25 |
|   LUT as Memory         |    0 |     0 |      9600 |  0.00 |
| Slice Registers         |  185 |     0 |     41600 |  0.44 |
|   Register as Flip Flop |  185 |     0 |     41600 |  0.44 |
|   Register as Latch     |    0 |     0 |     41600 |  0.00 |
| F7 Muxes                |    0 |     0 |     16300 |  0.00 |
| F8 Muxes                |    0 |     0 |      8150 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

```
3. Memory
---------

+-------------------+------+-------+-----------+-------+
|     Site Type     | Used | Fixed | Available | Util% |
+-------------------+------+-------+-----------+-------+
| Block RAM Tile    |   10 |     0 |         50 | 20.00 |
|   RAMB36/FIFO*    |   10 |     0 |         50 | 20.00 |
|     RAMB36E1 only |   10 |       |            |       |
|   RAMB18          |    0 |     0 |        100 |  0.00 |
+-------------------+------+-------+-----------+-------+
```

```
8. Primitives
-------------

+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+
| LUT2     |  231 |                 LUT |
| FDRE     |  183 |        Flop & Latch |
| CARRY4   |   52 |          CarryLogic |
| LUT5     |   34 |                 LUT |
| LUT4     |   29 |                 LUT |
| LUT1     |   24 |                 LUT |
| LUT3     |   23 |                 LUT |
| LUT6     |   22 |                 LUT |
| OBUF     |   14 |                  IO |
| IBUF     |   14 |                  IO |
| RAMB36E1 |   10 |        Block Memory |
| FDCE     |    2 |        Flop & Latch |
| BUFG     |    2 |               Clock |
+----------+------+---------------------+
```

Appendix M: VHDL Code – Rotary Encoder Controller

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer: himadri narasimhamurthy and ali hagen
--
-- Create Date: 08/13/2018 01:55:29 PM
-- Design Name:
-- Module Name: RotaryEncoderController - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


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 RotaryEncoderController is
    Port (
        clk : in std_logic;
        encoderSig_A : in std_logic;
        encoderSig_b : in std_logic;
        inc_en : out std_logic;
        dec_en : out std_logic
        );
    end RotaryEncoderController;

architecture Behavioral of RotaryEncoderController is

--state types
type state_type is (inc, dec, idle, hold1A, hold1B, hold2A, hold2B, hold3A, hold3B);
signal curr_state, next_state: state_type;

begin

--controller
state_update : process (clk)
begin
    if rising_edge(clk) then
        curr_state<= next_state;
    end if;
end process;

comb_logic : process (clk, encoderSig_A, encoderSig_B, curr_state)
```

```
begin

    next_state<=curr_state;
    dec_en <= '0';
    inc_en <= '0';

    case curr_state is
        when idle =>
            inc_en <= '0';
            dec_en <= '0';
            if encoderSig_A = '0' then
                next_state <= hold1A;
            elsif encoderSig_B = '0' then
                next_state <= hold1B;
            end if;
        when inc => inc_en <= '1';
            next_state<= idle;
        when dec => dec_en <= '1';
            next_state<= idle;
        when hold1A =>
            if encoderSig_A = '1' then
                next_state <= idle;
            elsif encoderSig_B = '0' then
                 next_state <= hold2A;
            end if;
        when hold1B =>
            if encoderSig_B = '1' then
                next_state <= idle;
            elsif encoderSig_A = '0' then
                next_state <= hold2B;
            end if;
        when hold2A =>
            if encoderSig_A = '1' then
                next_state <= hold3A;
            elsif encoderSig_B = '1' then --blip
                next_state<= idle;
            end if;
        when hold2B =>
            if encoderSig_B = '1' then
                next_state <= hold3B;
            elsif encoderSig_A = '1' then --blip
                next_state<= idle;
            end if;
        when hold3A =>
            if encoderSig_B = '1' then
                next_state <= inc;
            elsif encoderSig_A = '0' then
                next_state <= idle;
            end if;
         when hold3B =>
            if encoderSig_A = '1' then
                next_state <= dec;
            elsif encoderSig_B = '0' then
                next_state <= idle;
            end if;
    end case;
end process;

end Behavioral;
```

Appendix N: VHDL Code – Debouncer Module

```
----------------------------------------------------------------------------------
-- Engineer: modified by Himadri Narasimhamurthy and Ali Hagen
--
-- modified an online debouncer from Yourigh on Github
--
-- Create Date: 08/16/2018 08:45:33 PM
-- Design Name:
-- Module Name: Debouncer - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.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 Debouncer is
PORT ( clk : in std_logic;
       initial : in std_logic;     --non debounced inputs
       db_result : out std_logic
       );  --debounced outputs
end Debouncer;

architecture Behavioral of Debouncer is

    signal flipflop: std_logic_vector(1 downto 0);

    signal counter_set: std_logic;
    constant counter_size : integer := 10;
    signal counter_out : std_logic_vector(counter_size downto 0) := (others => '0');
begin

counter_set <= flipflop(0) xor flipflop(1);

debounce : process(clk)

begin
    if (clk'event and clk = '1') then
        flipflop(0) <= initial;
        flipflop(1) <= flipflop(0);
```

```
        if (counter_set = '1') then
            counter_out <= (others => '0');
        elsif (counter_out(counter_size) = '0') then
            counter_out <= counter_out + 1;
        else
            db_result <= flipflop(1);
        end if;
    end if;

end process;



end Behavioral;
```

Appendix O: VHDL Code – Board Top Level

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer: Ali Hagen
--
-- Create Date: 08/16/2018 10:32:31 PM
-- Design Name:
-- Module Name: TOP - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use  IEEE.numeric_std.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 BoardTop is
  Port ( clk : in std_logic;
         RL_inc : in std_logic;
         RL_dec : in std_logic;
         UD_inc : in std_logic;
         UD_dec : 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 BoardTop;

architecture Behavioral of BoardTop is

Component Debouncer is
Port ( clk : in std_logic;
        initial  : in std_logic;
        db_result : out std_logic );
end component;

Component RotaryEncoderController is
    Port ( clk : in std_logic;
        encoderSig_A : in std_logic;
```

```
        encoderSig_b : in std_logic;
        inc_en : out std_logic;
        dec_en : out std_logic );
end component;

-- intermediate signals
signal dbA1, dbA2, dbB1, dbB2 : std_logic;

begin

-- debounce the signals rom all of the rotary encoders
debouncer_A1 : Debouncer port map (
        clk => clk,
        initial => RL_inc,
        db_result => dbA1 );
debouncer_A2 : Debouncer port map (
         clk => clk,
         initial => RL_dec,
         db_result => dbA2 );
debouncer_B1 : Debouncer port map (
         clk => clk,
         initial => UD_inc,
         db_result => dbB1 );
debouncer_B2 : Debouncer port map (
           clk => clk,
           initial => UD_dec,
           db_result => dbB2 );

encoder_A : RotaryEncoderController port map (
        clk => clk,
        encoderSig_A => dbA1,
        encoderSig_b => dbA2,
        inc_en => right,
        dec_en => left );
encoder_B : RotaryEncoderController port map (
                clk => clk,
                encoderSig_A => dbB1,
                encoderSig_b => dbB2,
                inc_en => up,
                dec_en => down );

clear_proc: process(clk)
begin
    if (rising_edge(clk)) then
        clr <= '0';
        if rst = '1' then
            clr <= '1';
        end if;
    end if;
end process clear_proc;

end Behavioral;
```

Appendix P: VHDL Code – Colors Module

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer: Himadri Narasimhamurthy
--
-- Create Date: 08/18/2018 03:14:17 PM
-- Design Name:
-- Module Name: Colors - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


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 Colors is
    Port (
         sw_red: in std_logic;
         sw_yel: in std_logic;
         sw_grn: in std_logic;
         sw_blu: in std_logic;
         sw_cyn: in std_logic;
         sw_mag: in std_logic;
         sw_wht: in std_logic;
         sw_blk: in std_logic;
         --final_color: out std_logic_vector(11 downto 0);
         mem_col: out std_logic_vector(3 downto 0)
    );
end Colors;

architecture Behavioral of Colors is

begin

color_proc: process(sw_red, sw_yel,sw_grn, sw_blu, sw_cyn, sw_mag, sw_wht, sw_blk)
begin
    if sw_red = '1' then
        --final_color<= "111100000000";
        mem_col <= "0000";
    elsif sw_yel = '1' then
        --final_color<= "111111110000";
        mem_col <= "0001";
```

```
    elsif sw_grn = '1' then
        --final_color<= "000011110000";
        mem_col <= "0010";
    elsif sw_blu = '1' then
        --final_color<= "000000001111";
        mem_col <= "0011";
    elsif sw_cyn = '1' then
        --final_color<= "000011111111";
        mem_col <= "0100";
    elsif sw_mag = '1' then
        --final_color<= "111100001111";
        mem_col <= "0101";
    elsif sw_wht = '1' then
        --final_color<= "111111111111";
        mem_col <= "0110";
    elsif sw_blk = '1' then
        --final_color <= "000000000000";
        mem_col <= "0111";
    else
        --final_color<= "111111111111";
        mem_col <= "0110";
    end if;

end process color_proc;

end Behavioral;
```

Appendix Q: VHDL Code – PixelCalc Module

```
----------------------------------------------------------------------------------
-- Company: EtchASketch
-- Engineer: Himadri Narasimhamurthy
--
-- Create Date: 08/16/2018 09:49:52 PM
-- Design Name:
-- Module Name: PixelCalc - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use IEEE.std_logic_arith.all;
use IEEE.STD_LOGIC_UNSIGNED.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 PixelCalc is
  Port (
    clk: in std_logic;
    incX_en: in std_logic;
    incY_en: in std_logic;
    decX_en: in std_logic;
    decY_EN: in std_logic;

    -- 2 pixel width
    x: out std_logic_vector(8 downto 0);      --binary of 320 (101000000)
    y: out std_logic_vector(7 downto 0));     --binary of 240 (11110000)
end PixelCalc;

architecture Behavioral of PixelCalc is



--intermediate location signals
signal x_int: std_logic_vector(8 downto 0) := "010100000"; -- 160
signal y_int: std_logic_vector(7 downto 0) := "01111000";  -- 120

begin

calculateX: process(clk)
```

```
begin

    if rising_edge(clk) then
        --if (vid_on = '1') then
            if ((decX_en) = '1') then
                if (x_int+2 < 320) then
                    x_int <= x_int + 1;
                end if;
            elsif (incX_en) = '1' then
                if (x_int > 0) then
                    x_int <= x_int - 1;
                end if;
            end if;

            x<= std_logic_vector(unsigned(x_int));
        --end if;
   end if;
end process calculateX;

calculateY: process(clk)
begin
    if(rising_edge(clk)) then
        --if (vid_on = '1') then
            if ((incY_en) = '1') then
                if (y_int+2 < 240) then
                    y_int <= y_int + 1;
                end if;
            elsif ((decY_en) = '1') then
                if (y_int > 0) then
                    y_int <= y_int - 1;
                end if;
            end if;

            y<= std_logic_vector(unsigned(y_int));
       -- end if;
    end if;
end process calculateY;

end Behavioral;
```

Appendix R: VHDL Code – VGA Controller

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

ENTITY VGAController IS
PORT (     vclk      :  in STD_LOGIC; --100 MHz clock
      Vsync  :  out    STD_LOGIC;
      Hsync  :  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 VGAController;


architecture behavior of VGAController 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(vclk)
begin
if rising_edge(vclk) then
    if (H_CNT < HSCAN) then
        H_CNT <= H_CNT + 1;
    else
        H_CNT <= 0;
    end if;
end if;
end process hCounter;

vCounter: process(vclk)
```

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

H_Logic: process (H_CNT)
begin

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

       if (H_CNT>=  h_display ) and (H_CNT <HSCAN) then
         H_video_on <= '0';
       else H_video_on <= '1';
       end if;

       if (H_CNT >= h_display + right_border) and (H_CNT <HSCAN-left_border) then
             Hsync <= '0';
       else Hsync <= '1';
       end if;

       if (H_CNT = HSCAN) then
             H_complete <= '1';
       else H_complete <= '0';
       end if;

end process H_Logic;


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

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

      if (V_CNT >=  v_display) and (V_CNT<VSCAN) then
        V_video_on <= '0';
      else V_video_on <= '1';
      end if;

      if (V_CNT >= v_display + bottom_border) and (V_CNT < VSCAN-top_border) then
            Vsync <= '0';
      else Vsync <= '1';
      end if;

end process Vert_logic;

video_on <= H_video_on AND V_video_on;

end behavior;
```

Appendix S: VHDL Code – Top Level

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer: Himadri Narasimhamurthy and Ali Hagen
--
-- Create Date: 08/20/2018 01:54:30 PM
-- Design Name:
-- Module Name: Top - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use  IEEE.numeric_std.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;                    -- needed for the BUFG component
use UNISIM.Vcomponents.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 is
    Port (
        clk: in std_logic;
        right_top, left_top, up_top, down_top : in std_logic;
        switch_red: in std_logic;
        switch_yel: in std_logic;
        switch_grn: in std_logic;
        switch_blu: in std_logic;
        switch_cyn: in std_logic;
        switch_mag: in std_logic;
        switch_wht: in std_logic;
        switch_blk: in std_logic;
        clr_top : in std_logic;
        Hsync: out std_logic;
        Vsync: out std_logic;
        color: out std_logic_vector(11 downto 0)
     );
end Top;

architecture Behavioral of Top is

COMPONENT blk_mem_gen_0 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(3 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    clkb : IN STD_LOGIC;
    web : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addrb : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
    dinb : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    doutb : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
  );
  END COMPONENT;

Component BoardTop is
    Port(
        clk : in std_logic;
        RL_inc : in std_logic;
        RL_dec : in std_logic;
        UD_inc : in std_logic;
        UD_dec : 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 Colors is
    Port(
        sw_red: in std_logic;
        sw_yel: in std_logic;
        sw_grn: in std_logic;
        sw_blu: in std_logic;
        sw_cyn: in std_logic;
        sw_mag: in std_logic;
        sw_wht: in std_logic;
        sw_blk: in std_logic;
        --final_color: out std_logic_vector(11 downto 0);
        mem_col: out std_logic_vector(3 downto 0)
    );
end component;

Component PixelCalc is
    Port(
        clk: in std_logic;
        incX_en: in std_logic;
        incY_en: in std_logic;
        decX_en: in std_logic;
        decY_EN: in std_logic;

        -- 2 pixel width
        x: out std_logic_vector(8 downto 0);      --binary of 320 (101000000)
        y: out std_logic_vector(7 downto 0)     --binary of 240 (11110000)
    );
end component;

Component VGAController is
    Port(
```

```
        vclk : in STD_LOGIC; --25 MHz clock
        Vsync: out STD_LOGIC;
        Hsync: 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 u_int, d_int, r_int, l_int: std_logic;
signal loc_x_int: std_logic_vector(8 downto 0);
signal loc_y_int: std_logic_vector(7 downto 0);
signal pixloc: std_logic_vector(16 downto 0);

signal x: std_logic_vector(9 downto 0);
signal y: std_logic_vector(8 downto 0);
signal vgaloc: std_logic_vector(16 downto 0);
signal video_on: std_logic;

signal clr_int: std_logic;

--signal color_int: std_logic_vector(11 downto 0);
signal mem_color_int: std_logic_vector(3 downto 0);
signal mem_color_fin: std_logic_vector(3 downto 0) := "0111"; -- changed
signal mem_color_t: std_logic_vector(3 downto 0);

signal wea_int : std_logic;
signal addra_int, addrb_int : std_logic_vector(16 downto 0);
signal dia_int, dob_int : std_logic_vector(3 downto 0);

constant SCLK_DIVIDER_VALUE: integer := 4/2; --4/2
--constant CLOCK_DIVIDER_VALUE: integer := 5;     -- for simulation
signal sclkdiv: unsigned(1 downto 0) := (others => '0');  -- clock divider counter
signal sclk_unbuf: std_logic := '0';    -- unbuffered serial clock
signal divclk: std_logic := '0';          -- internal serial clock

signal num_int1 : integer := 0;
signal num_int2 : integer := 0;

signal video_on_vect : std_logic_vector(0 downto 0) := "0";
signal clr_vect : std_logic_vector(0 downto 0) := "0";

begin

-- Clock buffer for sclk
-- The BUFG component puts the signal onto the FPGA clocking network
Slow_clock_buffer: BUFG
   port map (I => sclk_unbuf,
             O => divclk );

-- Divide the 100 MHz clock down to 2 MHz, then toggling a flip flop gives the final
-- 1 MHz system clock
Serial_clock_divider: process(clk)
begin
   if rising_edge(clk) then
          if sclkdiv = SCLK_DIVIDER_VALUE-1 then
         sclkdiv <= (others => '0');
         sclk_unbuf <= NOT(sclk_unbuf);
      else
         sclkdiv <= sclkdiv + 1;
      end if;
```

```
   end if;
end process Serial_clock_divider;

board: BoardTop port map(
        clk => divclk,
        RL_inc => right_top,
        RL_dec =>  left_top,
        UD_inc => up_top,
        UD_dec =>  down_top,
        rst => clr_top,
        up => u_int,
        down => d_int,
        left => l_int,
        right => r_int,
        clr => clr_int
);
location: PixelCalc port map(
        clk => divclk,
        incX_en => r_int,
        incY_en => u_int,       --swapped r and l
        decX_en => l_int,
        decY_EN => d_int,

        -- 2 pixel width
        x => loc_x_int,      --binary of 320 (101000000)
        y => loc_y_int
);
--pixel combined address for mem
pixloc <= std_logic_vector((unsigned(loc_x_int) )) &
std_logic_vector((unsigned(loc_y_int)));

color_switches: Colors port map(
        sw_red => switch_red,
        sw_yel => switch_yel,
        sw_grn => switch_grn,
        sw_blu => switch_blu,
        sw_cyn => switch_cyn,
        sw_mag => switch_mag,
        sw_wht => switch_wht,
        sw_blk => switch_blk,
        --final_color => color_int,
        mem_col=> mem_color_int
);

vga: VGAController port map(
        vclk => divclk,
        Vsync => Vsync,
        Hsync => Hsync,
        video_on => video_on,
        pixel_x => x,
        pixel_y => y );

vgaloc <= x(9 downto 1) & y(8 downto 1);

--extra signal to convert video_on to std_logic_vector
video_on_vect(0) <= video_on;
clr_vect(0) <= clr_top;

BRAM : blk_mem_gen_0
  PORT MAP (
    clka => divclk,
```

```
    wea => video_on_vect,
    addra => pixloc,        --write address
    dina => mem_color_int,
    douta => mem_color_t,
    clkb => divclk,
    web => clr_vect,
    addrb => vgaloc,     --todo
    dinb => "0000",
    doutb => mem_color_fin
  );

color_proc : process(clk, video_on, mem_color_fin)
begin
if rising_edge(clk) then
    if video_on = '1' then
        if mem_color_fin = "0000" then
            color <= "000000000000";     -- black
        elsif mem_color_fin = "0001" then
            color <= "111100000000";     --yellow
        elsif mem_color_fin = "0010" then
            color<= "000011110000";      -- green
        elsif mem_color_fin = "0011" then
            color<= "000000001111";      --blue
        elsif  mem_color_fin = "0100" then
            color<= "000011111111";         --cyan
        elsif  mem_color_fin = "0101" then
            color<= "111100001111";        --magenta
         elsif  mem_color_fin = "0110" then
            color<= "111111111111";         -- white
         elsif  mem_color_fin = "0111" then
            color <= "111111110000";        --red
         else color <= "111111111111";
         end if;
    else color <= "000000000000";
    end if;
end if;
end process color_proc;
end Behavioral;
```

Appendix T: VHDL Code – Constraints File

```
## 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

# Clock signal
#Bank = 34, Pin name = CLK,                Sch name = CLK100MHZ
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports
clk]

# Switches
set_property PACKAGE_PIN V2 [get_ports switch_blk]
set_property IOSTANDARD LVCMOS33 [get_ports switch_blk]
set_property PACKAGE_PIN T3 [get_ports switch_wht]
set_property IOSTANDARD LVCMOS33 [get_ports switch_wht]
set_property PACKAGE_PIN T2 [get_ports switch_mag]
set_property IOSTANDARD LVCMOS33 [get_ports switch_mag]
set_property PACKAGE_PIN R3 [get_ports switch_cyn]
set_property IOSTANDARD LVCMOS33 [get_ports switch_cyn]
set_property PACKAGE_PIN W2 [get_ports switch_blu]
set_property IOSTANDARD LVCMOS33 [get_ports switch_blu]
set_property PACKAGE_PIN U1 [get_ports switch_grn]
set_property IOSTANDARD LVCMOS33 [get_ports switch_grn]
set_property PACKAGE_PIN T1 [get_ports switch_yel]
set_property IOSTANDARD LVCMOS33 [get_ports switch_yel]
set_property PACKAGE_PIN R2 [get_ports switch_red]
set_property IOSTANDARD LVCMOS33 [get_ports switch_red]

#Buttons
#Bank = 14, Pin name = ,               Sch name = BTNC
set_property PACKAGE_PIN U18 [get_ports clr_top]
set_property IOSTANDARD LVCMOS33 [get_ports clr_top]


##Pmod Header JA - RL
##Bank = 15, Pin name = IO_L1N_T0_AD0N_15,             Sch name = JA1
set_property PACKAGE_PIN J1 [get_ports {right_top}]
set_property IOSTANDARD LVCMOS33 [get_ports {right_top}]
##Bank = 15, Pin name = IO_L5N_T0_AD9N_15,gh         Sch name = JA2
set_property PACKAGE_PIN L2 [get_ports {left_top}]
set_property IOSTANDARD LVCMOS33 [get_ports {left_top}]

#Pmod Header JB - UD
#Bank = 15, Pin name = IO_L15N_T2_DQS_ADV_B_15,             Sch name = JB1
set_property PACKAGE_PIN A14 [get_ports up_top]
set_property IOSTANDARD LVCMOS33 [get_ports up_top]
###Bank = 14, Pin name = IO_L13P_T2_MRCC_14,               Sch name = JB2
set_property PACKAGE_PIN A16 [get_ports down_top]
set_property IOSTANDARD LVCMOS33 [get_ports down_top]


##VGA Connector
##Bank = 14, Pin name = ,              Sch name = VGA_R0
set_property PACKAGE_PIN G19 [get_ports color[0]]
set_property IOSTANDARD LVCMOS33 [get_ports color[0]]
###Bank = 14, Pin name = ,             Sch name = VGA_R1
```

```
set_property PACKAGE_PIN H19 [get_ports color[1]]
set_property IOSTANDARD LVCMOS33 [get_ports color[1]]
###Bank = 14, Pin name = ,             Sch name = VGA_R2
set_property PACKAGE_PIN J19 [get_ports color[2]]
set_property IOSTANDARD LVCMOS33 [get_ports color[2]]
###Bank = 14, Pin name = ,             Sch name = VGA_R3
set_property PACKAGE_PIN N19 [get_ports color[3]]
set_property IOSTANDARD LVCMOS33 [get_ports color[3]]
###Bank = 14, Pin name = ,             Sch name = VGA_B0
set_property PACKAGE_PIN N18 [get_ports color[4]]
set_property IOSTANDARD LVCMOS33 [get_ports color[4]]
###Bank = 14, Pin name = ,                Sch name = VGA_B1
set_property PACKAGE_PIN L18 [get_ports color[5]]
set_property IOSTANDARD LVCMOS33 [get_ports color[5]]
###Bank = 14, Pin name = ,             Sch name = VGA_B2
set_property PACKAGE_PIN K18 [get_ports color[6]]
set_property IOSTANDARD LVCMOS33 [get_ports color[6]]
###Bank = 14, Pin name = ,                Sch name = VGA_B3
set_property PACKAGE_PIN J18 [get_ports color[7]]
set_property IOSTANDARD LVCMOS33 [get_ports color[7]]
##Bank = 14, Pin name = ,              Sch name = VGA_G0
set_property PACKAGE_PIN J17 [get_ports color[8]]
set_property IOSTANDARD LVCMOS33 [get_ports color[8]]
###Bank = 14, Pin name = ,          Sch name = VGA_G1
set_property PACKAGE_PIN H17 [get_ports color[9]]
set_property IOSTANDARD LVCMOS33 [get_ports color[9]]
###Bank = 14, Pin name = ,             Sch name = VGA_G2
set_property PACKAGE_PIN G17 [get_ports color[10]]
set_property IOSTANDARD LVCMOS33 [get_ports color[10]]
###Bank = 14, Pin name = ,          Sch name = VGA_G3
set_property PACKAGE_PIN D17 [get_ports color[11]]
set_property IOSTANDARD LVCMOS33 [get_ports color[11]]
###Bank = 14, Pin name = ,                Sch name = VGA_HS
set_property PACKAGE_PIN P19 [get_ports Hsync]
set_property IOSTANDARD LVCMOS33 [get_ports Hsync]
##Bank = 14, Pin name = ,           Sch name = VGA_VS
set_property PACKAGE_PIN R19 [get_ports Vsync]
set_property IOSTANDARD LVCMOS33 [get_ports Vsync]


set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]

set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]

set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]
```