Sean Liu & Grace Li
6.111 Fall 2008 Final Project
December 10, 2008

Abstract

In recent times there has been a proliferation of programs which aid in editing, coloring, and creation of art. Generally these programs utilize the traditional inputs of either a tablet or mouse. FingerArt is also an art creation program, except that it allows users to interact in the most natural way – using their fingers. With a combination of video camera recognition and paint mixing ability, FingerArt allows for an intuitive and natural paint simulation experience, allowing artists to utilize their most proficient painting tool: their hands.

Table of Contents

List of Figures

List of Tables

Appendices

1 Overview

FingerArt provides a method for artists to virtually paint using their hands and fingers, the most proficient tools available. Our goal is to simulate a natural painting style environment, in which artists can mix colors, change brushes, control saturation, and paint the canvas.

Located on one side of the screen is a color palette, which the artist can virtually dip his or her hands into much like an artist dips their brush into paints. Like a real painter, the artist can mix colors and saturate their brush with different amounts of paint depending on how long they leave their hands in the virtual paint. Saturated brushes leave dense strokes, while dry brushes leave light strokes and allow more of the underlying picture to come through. The more saturated the brush, the longer the paint on the paintbrush lasts.

A camera keeps track of the location of the artist's hand, which serves as his or her brush. The goal is to simulate a real canvas for the artist through the color palette, paint saturation, brush selection, and brush orientation. The user wears a black glove and two colored wire rings. These two rings represent the two corners of the brush. Much like brushes are often thin and wide, the user can twist their hand to control the orientation of the brush during the stroke for thick and thin strokes. A picture showcasing the capabilities of our system is shown below in Figure 1. Our hope is that artists find our system to be a fun, intuitive, and natural way to paint virtually without the hassle of dealing with the mess of real paint.

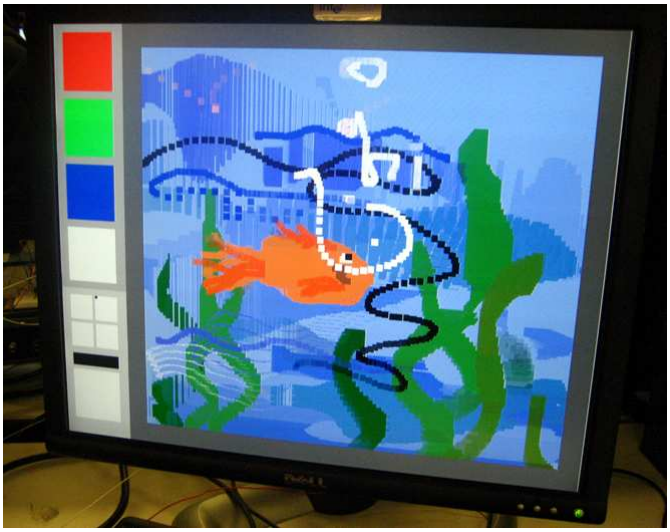

Figure 1: A showcase of the system's capabilities.

2 Video Capture, Center of Mass Calculation, and Intention

Created by Sean Liu, the purpose of the video capture, center of mass calculation, and intention modules is to identify the location, orientation, and intention of the user's hands. These modules serve as the interface for the user to control the paintbrush and indicate the actions of absorbing, painting, and switching brushes.

The user interacts with the system by wearing two colored wire rings – red and green – on a black-gloved hand, and moves his or her hand over a dark surface to control a virtual brush displayed on the screen. This setup is shown in Figure 2. The camera module tracks the colored rings by using color video and filtering out pixels below a threshold RGB value. The pixels that pass are then averaged for both a red and green center of mass. This center of mass is then scaled so that the entire area of the screen is reachable, but this has the effect of allowing only discrete changes in the center of mass. Following this, the scaled centers of mass are smoothed through averaging over the last eight values to create continuous brush strokes. Finally, a GUI is created, which allows the user to signal whether they want to absorb colors, change brushes, or paint on the screen. Figure 3 shows the modules involved in video capture, center of mass calculation, and intention.

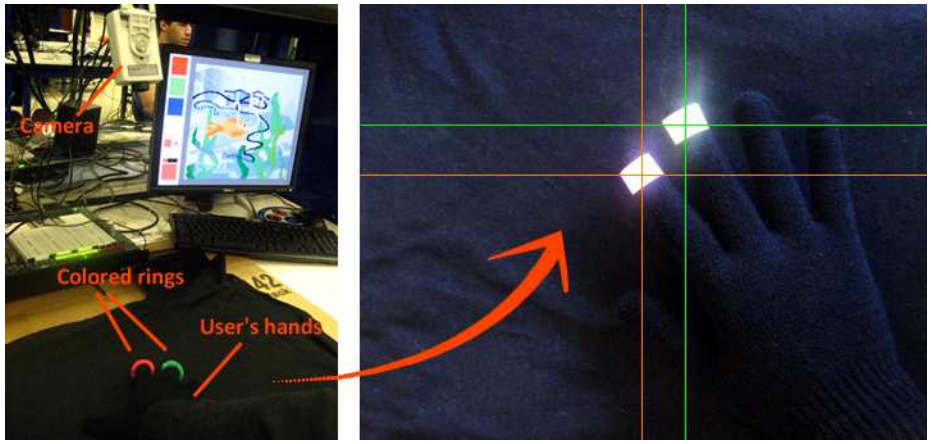

Figure 2: The user interaction setup and camera tracking of colored wires.

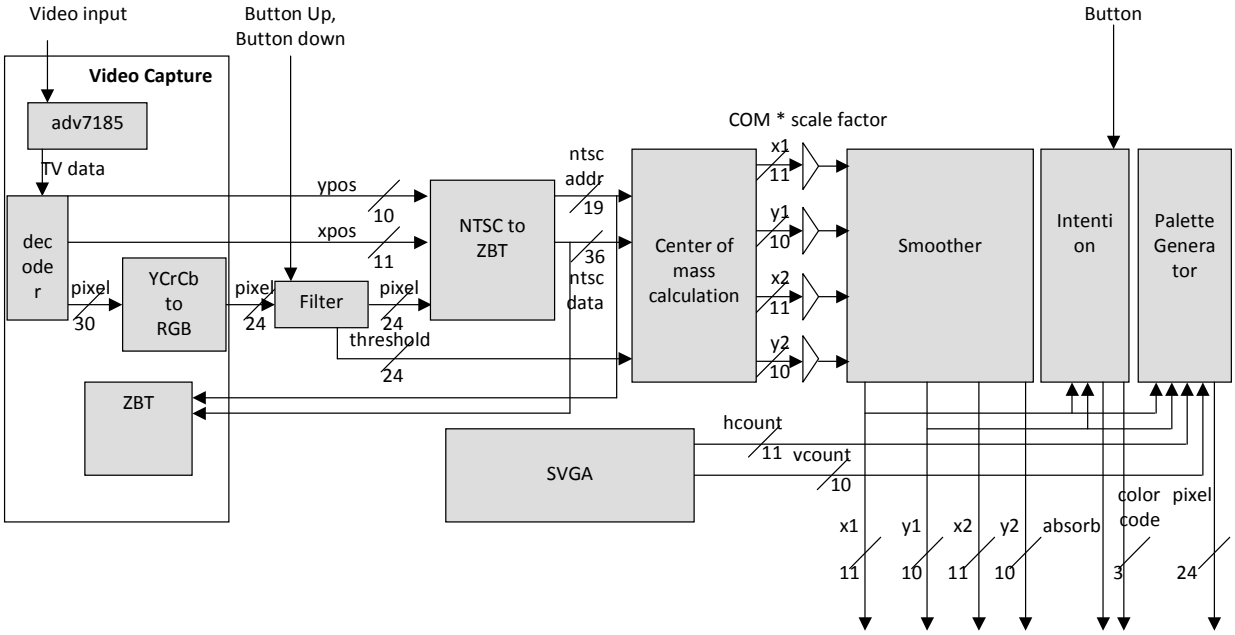

Figure 3: Block diagram of video capture, center of mass, and intention modules.

2.1 Video Capture

2.1.1 adv7185, NTSC Decode

The adv7185 and NTSC Decode modules wire the FPGA to the camera and set the appropriate TV-in signals and clock. The decoder processes data from the camera, encoded in the NTSC format, in order to extract the luminance and chrominance values, and one pixel's data is outputted per clock cycle. The 6.111 staff provided these modules and no modifications were necessary.

2.1.2 YCrCb2RGB

YCrCb2RGB takes a color in the luminance, chrominance space and maps the color to the red, green, and blue space. This process happens prior to writing to ZBT such that the ZBT read code can be reused for both the camera display and the canvas display, as the canvas process and paint genesis is done with RGB. Conversion to RGB is also required to occur early in the processing stage, as the pixel filtering and all subsequent camera processing is based on RGB values. The 6.111 staff provided this module and no modifications were necessary.

2.1.3 Pixel Filter

The pixel filter allows the user to set a threshold for pixels to accept or reject. Each clock cycle a pixel is passed in from the YCrCb2RGB module. If any of the red, green, or blue values of a given pixel has a value greater than the threshold set by the user, then the pixel is passed through unchanged. Otherwise it is outputted as black. The threshold is adjustable by the user using the up and down buttons, which enables easy testing and calibration depending on different lighting environments. The module monitors the button signals and every clock cycle looks for transitions from zero to one, which indicates the user has pressed a button. Up button presses move to a higher threshold (i.e. one that gets closer to 8'hFF which only allows truly green and truly red pixels pass), and down buttons move to a lower threshold (i.e. one that gets closer to 8'h00, which lets all pixels through).

2.1.4 NTSC to ZBT

The ntsc_to_zbt module takes in 18 of the 24 filtered pixel bits and creates ZBT words consisting of two pixels, or 36 bits. Every other clock cycle, ntsc_to_zbt outputs a completed word. The 6.111 staff provided the original code for black and white display, with ZBT words consisting of four 8-bit black and white pixels per ZBT word. The code was modified to store an expanded 18-bits per pixel and therefore two pixels per word, as described above. Although the primary focus of this module is to generate the ZBT words to be written to the RAM, it also serves the secondary purpose of synchronizing the video input clock with the system clock. For this reason, the synchronized ZBT words that are outputted by this module are also used for the center of mass calculations, which are clocked using the general system clock.

2.2 Center of Mass Calculation

2.2.1 Center of Mass Calculation (comcalc)

The comcalc module calculates the two centers of mass that represent the location of the red and green rings which are worn by the users, and indicate the two endpoints of the virtual brush. The module takes in a ZBT word from the ntsc_to_zbt module, which is actually two pixels worth of

color information at a time. The module therefore only performs calculations every other clock cycle, when the ntsc_to_zbt module provides two different outputs. First, the comcalc module verifies that the color information for both pixels in the inputted word are nonzero. A nonzero pixel indicates that its RGB values were above the threshold set by pixelfilter, and are therefore likely to represent the colored wire rings worn by the user. The requirement that both pixels are nonzero makes the system more robust against noise. However, even with the pixel filter, some speckles of noise in clumps of area greater than one pixel continue to be caught by the camera. Generally, however, the noise area is just a single pixel, and so does not fulfill the requirement that the information for both pixels be nonzero. Thus, these single noise pixels are not picked up by the comcalc algorithm.

Next, the comcalc module determines if the pixels belong to the red center of mass or the green center of mass. This is done using the threshold set by the pixelfilter. If the red in the RGB of the pixel exceeds the threshold, then the pixel is considered to contribute to the red center of mass, otherwise, it contributes to the green center of mass. Running sums are then kept for the x and y coordinates of both the red and green pixels. A divider module is then implemented for the final averaging division. Once every 1/60 of a second, the comcalc module outputs a new center of mass. The divider used in this module is generated by the Xilinx tools.

2.2.2 Center of Mass Scaling

The camera screen resolution is only 720 by 480, which makes it impossible to reach the entire screen width of 800 by 600 if we do a one to one mapping of pixels. For this reason, the centers of masses are scaled: the x coordinate is multiplied by 9/8 and the y coordinate is multiplied by 11/8. The denominator of both fractions was specifically chosen to be 8, such that the division could be a simple bit shift.

2.2.3 Center of Mass Smoother

One side effect of the center of mass scaling is that the center of mass movement appears very discrete, which is problematic if we want continuous brush strokes. To address this challenge, we use a center of mass smoother. The smoother outputs the average over the last N centers of mass (parameterized module). In our implementation, the module keeps track of the last eight centers of mass using a circular buffer. The module also stores the current sum of the centers of mass, such that a quick bit shift can output the average. Every time a new center of mass is passed into the module, the running sum is decreased by the oldest center of mass in the circular buffer, and the new value is added to the center of mass running total.

2.3 Intention and Palette Generation

2.3.1 Palette Generation

The palette_gen module generates the graphical user interface for the user. It takes in the hcount and vcount of the current pixel set by the SVGA module, and determines whether or not this pixel is part of the palette. The module is given many GUI parameters, which allow for easy and rapid adjustment of the palette. By varying the GUI parameters, the system could be quickly configured to show either more or fewer colors on the palette.

2.3.2 Intention

The intention module closely parallels the palette generation module. The primary difference is that rather than taking as input the current hcount and vcount specified by the SVGA module, it takes in the center of mass coordinates. If the center of mass coordinate is within one of the palette's color areas, then the user's intention is to absorb paint. If the center of mass is within one of the brush option areas, then the user's intention is to switch a brush. Finally, if the center of mass is within the canvas, then the intention is to paint to the screen. The fill command assigns a value to all pixels in the ZBT. This is achieved by using a counter. While the counter is smaller than a preset value, which is larger than the total number of addresses in the ZBT, it writes to the ZBT a certain color. This has the effect of filling the screen entirely with one color, and it is used to make the screen all white for initialization, or to make the screen a different color when the filling brush is used.

2.4 Camera ZBT and Display

2.4.1 ZBT 6.111

This module, written by I. Chuang, is passed a ZBT address, data to write, and a write enable signal. The module generates the appropriate signals in order to read and write from RAM. Note, however, in its original form, the module suffered errors caused by skewed clock signals being routed to the RAMs, leading to mismatches between the intended data and locations to write and the actual data and locations written. To correct this error, Gim Hom and Don Goldin recommended the use of the *ramclock* module, described in *Section 2.4.2*, and the ZBT code was modified in *labkit.v*. Every odd clock cycle, the ZBT performs a write command, and every even clock cycle the ZBT performs a read command. Two ZBTs are used in FingerArt, one of them for the camera input and the other for storing the current picture.

2.4.2 Ramclock

The ramclock module, written by Nathan Ickes, creates deskewed versions of the clock signals that are to be routed to the RAMs. The module adjusts the RAM clock phases such that the RAMs receive the clock signals at the same time as the registers in the FPGA. This corrects the error in which unexpected data locations are written to in the ZBT.

2.4.3 VRAM Display

The vram_display module is a modified version of code provided by the 6.111 staff, expanded to allow for colored coding. The vram_display module reads data from the ZBT and extracts a pixel per clock cycle to be displayed on the screen. Because each word in the ZBT contains two pixels, data is read from the ZBT every even clock cycle (odd clock cycles are used for writing to the ZBT). Every clock cycle, a pixel is extracted from the ZBT word and outputted. The module takes as inputs the SVGA hcount, vcount, and ZBT word, and outputs the ZBT address to read, in addition to the pixel color. The original 6.111 version of the vram_display module read from the ZBT every four clock cycles, since each pixel was stored in 8 bits. This was modified to accommodate 18 bits per pixel, or two pixels per 36 bit ZBT word.

2.4.4 Pixel Selection for Display

The pixel selection for display toggles between either outputting the camera data or outputting the canvas pixel. This is controlled by a switch which toggles between the two.

2.4.5 User Inputs and Control

The user can toggle between the options of a scaled center of mass, smoothed center of mass, camera display, palette display, absorbing/painting, and shedding paint. The inputs and controls to the system are shown below in Table 1.

Table 1: User input controls

Input	**Function**
Enter	"Absorb" if over palette, "paint" if over canvas
Button 0	User reset
Button Right	Same as Enter
Button Left	Clear brush of all color and saturation
Button Up	Increase the pixel filter threshold
Button Down	Decrease the pixel filter threshold
Switch 2	Show/hide palette GUI
Switch 3	Camera display/canvas display
Switch 4	Toggle center of mass smoothing
Switch 5	Toggle center of mass scaling

3 Paint Genesis

Written by Grace Li, the Paint Genesis modules perform the paint mixing, brush creation, and canvas management for FingerArt. Specifically, the paint genesis system receives four key inputs from the video detection system: two centers of masses, an absorb/paint signal, and the color which the user is trying to absorb. Paint genesis performs two primary operations: brush creation and paint color generation. The block diagram of the system is shown in Figure 4.

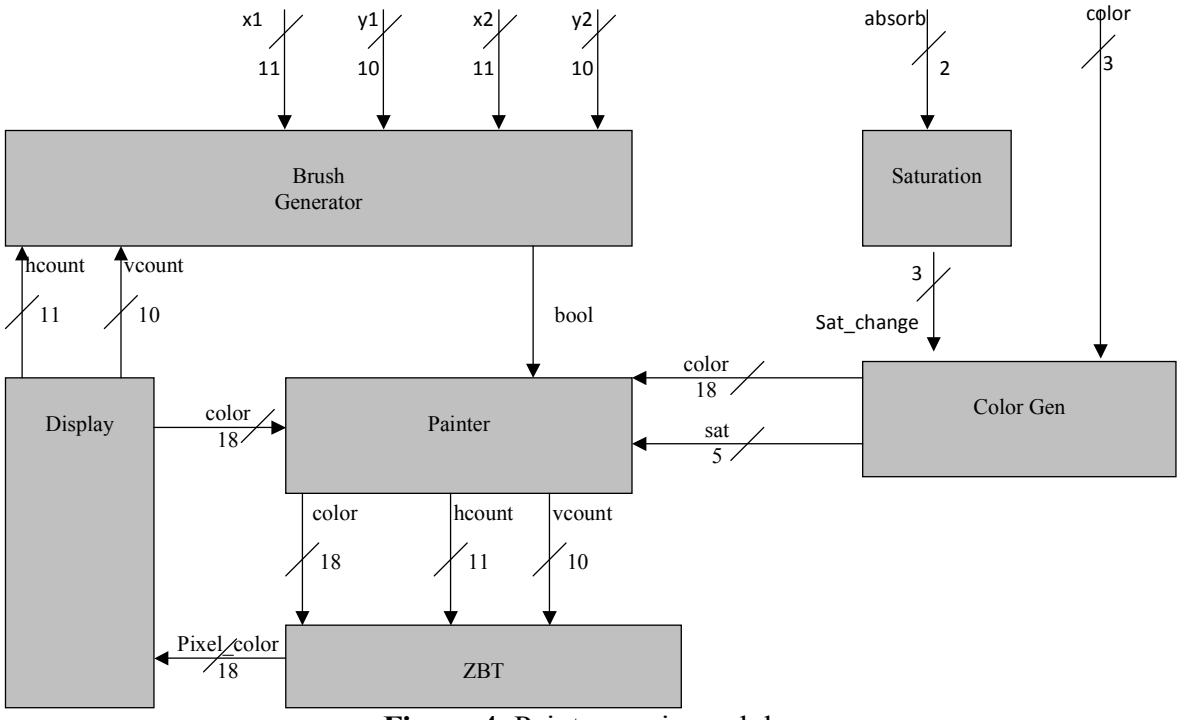

Figure 4: Paint genesis modules

The first main objective of paint genesis is brush creation, which is handled by the brush generation module, which takes an hcount, vcount pair denoting a specific pixel. The two centers of mass serve as the anchor points for the brush, and the module looks up the hcount, vcount pixel relative to these anchor points to determine if the pixel in question is part of the brush.

The second main objective, color genesis and paint saturation, takes place in the saturation and color generation modules. Both modules are passed the absorb/paint signals and the color signals from the intention module. The saturation module translates the absorb/paint signals into a timeframe that is more intuitive for the human user – rather than updating the saturation on a 40 MHz clock – and then passes the saturation change to the color generator. The color generator takes in the saturation change and the color being absorbed. These two values are processed into a newly updated color and saturation of paint for the paintbrush.

Finally, the brush information and the color information is passed to the painter module, along with the color at the same hcount, vcount location on the canvas, which has been read from the ZBT. The painter module combines both colors into a new color and writes this information back to the ZBT, along with displaying the pixel to the user on the screen.

3.1 Color and Saturation

3.1.1 Color Generator

The Color Generator tracks the paintbrush's current paint color and saturation. The module keeps an internal state of what color is currently on the brush in the form of three five-bit numbers, each of which contains the five most significant bits of red, green, and blue. In addition, the Color Generator module keeps an internal counter of what the saturation of the color is, in the form of a 5 bit number.

The Color Generator also has to change and update the paint on the paintbrush according to the user's intention. The functionality is implemented as follows:

- *Paint absorption* – when the user is picking up paint from the palette, the saturation of the paintbrush paint increases.
- *Paint application* – when the user is painting the canvas, the saturation of the paint decreases.
- *Color mixing – w*hen the user is picking up color *A* from the palette, the paint on the paintbrush changes to be more like color *A* the longer the user is picking up color *A*. Additionally, the more paint the user picks up from the palette, the darker the paintbrush paint becomes.
- *Color persistence* – when the user is painting on the canvas, the color of the paintbrush paint does not change.

In order to implement this functionality, the Color Generator first considers the input from the Saturation Generator. The Saturation Generator passes on either a positive or a negative number. The internal saturation of the virtual paintbrush will be updated to either increase if the input

from the Saturation Generator is positive, or decrease if the input from the Saturation Generator is negative.

Next, if the module recognizes that the user is trying to pick up color *A* from the palette, it combines the color that is currently on the brush with the color that is being picked up. What seems intuitive about increasing a certain color by increasing its RGB values is actually incorrect, since the RGB system is reversed, with (0, 0, 0) being black and (256, 256, 256) being white. In order to increase the redness of a certain color, instead of increasing its R value, we instead decrease the G and the B values.

Also, after a color is fully red, we also implement the functionality that allows the user to get progressively darker shades of red if they absorb more and more red. In this case, the RGB value will be of the form (256, 0, 0) and to make the color darker, we simply decrease the R value. This can be generalized to all the colors, since we only allow the user to absorb the three preset colors of red, green, and blue. If the user absorbs enough color, then the RGB values will decrease to (0, 0, 0) and the color will be black.

Additionally, there was functionality which allowed the user to absorb white paint, to lighten the color of the paint on the brush. In order to make a certain color lighter, the RGB values should be closer to (256, 256, 256), since white is the lightest color of all. If the user ever absorbed white, the R, G, and B values were all increased by the same amount, though we were careful to ensure that the RGB values would always remain less than 256. In addition to allowing the user to absorb white paint, we allowed the user to completely reset the color of the paint on the brush. If the color reset signal was ever enable, the paintbrush saturation was set to 0, and the color of the paint on the paintbrush was set to white.

3.1.2 Saturation Generator

The saturation module keeps track of whether the saturation of the paintbrush is increasing or decreasing. If the user is trying to absorb paint, the Saturation Generator outputs a positive saturation change, and if the user is painting, the Saturation Generator outputs a negative saturation change. The number of positive and negative saturation changes that the Saturation Generator asserts is directly proportional to how many absorbs and paint signals the Saturation Generator has seen.

Our absorb and paint signals are asserted at 40MHz, but we wish the saturation to change in the same frame of time as the user, who probably can only deal with changes on the order of fractions of seconds. To this end, the Saturation Generator keeps an internal counter which increases every time a paint signal is encountered and decreases every time an absorb signal is encountered. Every time the counter has been increased or decreased sixteen million times, a saturation change is asserted, which means that the saturation of the paint on the paintbrush changes every 0.4 seconds. Each time the paintbrush is reset or the counter reaches 16 million, the internal counter of the Saturation Generator is reset to 0, and the process starts over. As the user paints on the screen, the saturation decreases proportional to the time spent painting, and similarly the saturation and color increase proportional to the time spent absorbing.

3.2 Brush Generation

The Brush Generators decide where color should be laid down on the virtual canvas. FingerArt implements three different brushes: block brush, spray-paint brush, and line brush. All brushes follow the same basic implementation of creating an anchor point based on the center of mass, and deciding color placement relative to these anchor points. As hcount, vcount pairs are input to the brush generators, the modules lookup if the hcount, vcount pixel ought to add color based on their location relative to the brush anchor. The brush generators output a Boolean flag signifying whether or not this pixel should be painted on.

3.2.1 Block Brush

The block brush forms a 9 by 9 pixel block centered on one of the two points from the video detection system. We have arbitrarily decided to center the block around the red center of mass. We only return a positive Boolean flag for adding paint if the hcount, vcount pixel pair input is within 4 pixels, in both the x and the y direction, of the red center of mass.

3.2.2 Spray Paint Brush

The spray paint brush is implemented with a bitmap. We selected certain pixels to paint, relative to a predetermined center. The spray paint brush module actually only takes as input one point passed in from the Brush Center Generator, and then virtually maps the bitmap around that one point. This differs from the other brushes, which are passed in both the red and green center of mass. The reason for this is that the spray paint brush is used in conjunction with the Brush Center Generator module, which passes the Spray Paint Brush five locations between the red and green centers of mass.

3.2.3 Line Brush

The line brush should form a line of consistent width between the two points passed in by the video detection system. We can achieve this by considering the distance, in x and y coordinates, from the point determined by hcount and vcount to one of the two points passed in by the video detection system. If the pixel we are currently considering is on the line between the two points, we know that the x and y distances must be proportional. By cross-multiplying and returning a true Boolean if the difference between the multiples is less than some preset amount, we can create a line.

3.2.4 Brush Center Generator

The Brush Center Generator takes in the two centers of masses from the video detection system and then creates three new points evenly spaced between the two original ones. We are effectively splitting up the distance between the two original points into 4 equal lines, which utilizes a division by 4 implemented with a bit shift. The Brush Center Generator then outputs the five points (two original points and three new points) to be used by the spray paint brush.

3.3 Painter

The Painter module aggregates the virtual paint from the paintbrush with the virtual paint which has already been laid down on the canvas. In doing so, it considers the previous color on the canvas, the current paintbrush color, and the saturation of the paint on the paintbrush. The

functionality should be that, the higher the saturation of the paintbrush, the more the new color should look like the paintbrush paint, and the lower the saturation of the paintbrush, the more the new color generated should look like the paint already laid on the canvas.

The Painter considers each pixel individually, taking in the Boolean of whether the pixel is on the brush, in addition to the paintbrush paint color, the previous color, and the saturation of the paint on the paintbrush. If the Boolean is true, that means that the color of the pixel needs to be updated, since it has been painted over. We update the color of the canvas by doing a weighted average of the previous color and the paintbrush color, using the following formula.

```
color_new = ( (32 - saturation) * color_old + saturation * color_brush ) / 32;
```

3.4 Paint Genesis ZBT and Display

3.4.1 ZBT Word Generation

The gen_paint_zbt module functions similarly to the ntsc_to_zbt module, with the primary difference being that it is not necessary to deal with aliasing. Every clock cycle the module takes in a pixel that is to be written and builds a ZBT word. The word is outputted with a write enable signal if the latest pixel being written has an odd hcount.

3.4.2 VRAM Display

The VRAM Display used to read data from the ZBT functions identically to the VRAM Display used for the camera ZBT and display. See section 2.4.3 for more details.

3.4.3 SVGA

The SVGA module generates the signals for a 800 by 600 pixel screen to send to the monitor. This code is based off the 1024 by 768 pixel screen found on the 6.111 site, with values modified for 800x600 display.

4 Testing and Debugging

4.1 Testing of the video capture, center of mass calculation, and intention

The video capture system behavior was verified primarily through user testing. Initially there was difficulty in getting the modules to behave properly, but after careful review of the code and minor edits, the color video functioned correctly and did not require any further special test simulations.

The center of mass calculation was implemented one component at a time. Initially we fixed the sum and division of the averaging calculation, such that a fixed center of mass was outputted. We then relaxed the constraint of the fixed sum, followed by the relaxing the fixed division.

Surprisingly, one of the most time-consuming challenges came in the center of mass smoothing calculation. This code failed to work after continual tweaks and recompiles. After several days of little progress, we created a ModelSim test simulation, which inputted fixed values into the smoother. Working with Gim, we identified an obscure small error caused by performing additions on the fly. The smoothing module utilizes a circular buffer, which stores the last N

centers of mass. A sum of all the centers of mass is also stored. When a new center of mass is inputted, the new coordinates are added to the sum, and the oldest center of mass is subtracted. To access the oldest center of mass value, we take the current pointer of the circular buffer and add one. The problem arose, however, that we did not define a new wire to store the current pointer plus one. Because of this, the compiler actually stored a new variable with a greater width instead of overflowing the variable at the appropriate times. Once this problem was identified, the appropriate changes were made.

To test the intention modules, we hooked up the selected_color signal to the display to ensure that the right selected_color signal was being sent out when we were gesturing over a certain patch of color. This completed testing of the video detection system.

4.2 Testing of paint genesis

We tested the paint genesis system's color sections individually. After hooking a count up to the display, we ensured that a sat_change was asserted every 0.4 seconds. To verify correct behavior of the color generation module, we hooked up three buttons as a red absorption signal, green absorption signal, and blue absorption signal to simulate mixing colors and confirm the appropriate blending behaviors. To test the painter module, we created a simulation program in which we mixed the color created in the color generation module with a certain fixed color using the saturation from the saturation testing.

In order to test the brush and the ZBT, we input two fixed points as our center of mass inputs and hooked up the signals to the color section testing described above. We were able to write a specified color into the ZBT using a selected brush, and were able to read the data from the ZBT and display the color in the same screen location, confirming the desired behavior.

At every clock cycle, the Gen_paint_ZBT changes one half of the two pixel word which is written into the ZBT. This means that the two pixel word is only ever complete every other clock cycle. The other half of the time, one half of the word is not going to be updated and will be from three input cycles ago. This can be seen in Figure 5: the underlined part of a word will be updated in the next clock cycle, but only the purple two-pixel words will be correct. For correct functionality, we must only enable the write enable when the word is entirely correct.

Input	A	A	B	B	A	B	B	B	A
Word			A<u>A</u>	<u>A</u>B	B<u>B</u>	<u>B</u>A	B<u>A</u>	<u>B</u>B	B<u>B</u>
we			1	0	1	0	1	0	1

Figure 5: ZBT word creation for paint genesis

We were having trouble where the ZBT was writing and reading with no delays, but when we looked at the pixels around the place where we had written, about half of the pixels wouldn't be the correct color. It wasn't that the entire paintbrush has off by one pixel; it was that fewer than half of the pixels would have the wrong color. It turned out that we were asserting the write enable at the right frequency (half the frequency of the inputs) but we were asserting them at the wrong words. In that case, one half of each word was going to be the input from 3 cycles ago,

which was causing the pixels of the wrong color to appear. Once we had switched the write enable signal, the brush was no longer distorted, and only pixels of the expected colors were displayed.

Timing delays were also an issue. Reading from the ZBT takes two clock cycles, which means that when we process a pixel at a given hcount, vcount pair, we are actually reading the canvas pixels from the ZBT that were two clock cycles old. The painter module takes the current ZBT data, calculates a new color based on the brush, and writes the new pixel back. Without delays, however, this means we actually write back a pixel from two cycles previous, causing a shift in the canvas. Creating a delayed version of the read address for the write back proved to be an adequate fix. Another issue with timing delays was the delay in the paint mixing module. This paint mixing result is output a cycle after the brush enables are set. For this reason, we delay the brush enable to align with the paint mixing output.

The final largest hurdle arose with the ZBTs. The Xilinx routing often caused the clocks going to the RAMs to be skewed. We found that our brushes were not writing to the ZBT correctly, and each compile proved to have different behavior, making it difficult to trace and tweak. Compile times ran on the order of twenty minutes. Initially we attempted to write a ModelSim simulator, but were warned by the TAs that creating a ModelSim ZBT simulator would be as big challenge of a challenge as debugging the error directly. After several days of little progress, Gim informed us of a bug caused by clock signal skews from long routing to the RAMs, and also identified a fix by locking the RAM clock. This correction proved successful and resolved our writing errors.

Once all the different sections had been tested, we put the project altogether and conducted some rigorous user testing. We changed the saturation and mixed different colors together on the screen to make sure that the ZBT was reading out and writing in the pixel colors correctly. Also, such tests made it clear that the centers of masses were being calculated correctly and that the painter module was mixing color correctly.

5 Conclusion

Our FingerArt system implements a virtual system for users to paint with their hands, as they do in real life. FingerArt offers users a palette of colors, the ability to mix paints, a canvas to hold the painting, and a variety of brush selections with which to lay down the paint. Simulating the real painting environment, artists can pick up paint from the palette with the brush, mix colors of paint, control the saturation of paint on the brush, apply paint to the canvas, and orient and angle the brush.

We suggest that in an upcoming implementation, the system be improved with the following functionality: the ability to undo the last stroke by including a second painting ZBT, allowing the user to save and print a painting, adding more palette locations for users to save premixed colors, and including a Paint Paint Revolution game in which the user mimics paint strokes in real time.

6 References

Chuang, I. ZBT Modules. November 2005. Massachusetts Institute of Technology. <http://web.mit.edu/6.111/www/f2005/handouts.html>.

Ikes, Nathan. April 2004. Massachusetts Institute of Technology. <http://www-mtl.mit.edu/Courses/6.111/labkit/verilog/extras/ramclock.v>.

Terman, Chris. 6.111. Fall. 2008. Massachusetts Institute of Technology. <http://web.mit.edu/6.111/www/f2008/index.html>.

Appendices

A labkit.v

```
module labkit(

  // AC97
  output wire beep, audio_reset_b, ac97_synch, ac97_sdata_out,
  input wire ac97_bit_clock, ac97_sdata_in,

  // VGA
  output wire [7:0] vga_out_red, vga_out_green, vga_out_blue,
  output wire vga_out_sync_b, vga_out_blank_b,
  output wire vga_out_pixel_clock, vga_out_hsync, vga_out_vsync,

  // NTSC OUT
  output wire [9:0] tv_out_ycrcb,
  output wire tv_out_reset_b, tv_out_clock,
  output wire tv_out_i2c_clock, tv_out_i2c_data,
  output wire tv_out_pal_ntsc, tv_out_hsync_b,
  output wire tv_out_vsync_b, tv_out_blank_b,
  output wire tv_out_subcar_reset,

  // NTSC IN
  input wire [19:0] tv_in_ycrcb,
  input wire tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
  input wire tv_in_aef, tv_in_hff, tv_in_aff,
  output wire tv_in_i2c_clock, tv_in_reset_b,
  output wire tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso, tv_in_clock,
  inout wire tv_in_i2c_data,

  // ZBT RAMS
  inout wire  [35:0] ram0_data,
  output wire [18:0] ram0_address,
  output wire ram0_adv_ld,ram0_ce_b,ram0_oe_b,
  output wire ram0_clk,
  output wire ram0_cen_b,
  output wire ram0_we_b,
  output wire [3:0] ram0_bwe_b,

  inout wire  [35:0]ram1_data,
  output wire [18:0]ram1_address,
  output wire ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b,
  output wire ram1_oe_b, ram1_we_b,
  output wire [3:0] ram1_bwe_b,
  input wire  clock_feedback_in,
  output wire clock_feedback_out,


  // FLASH
  inout wire [15:0] flash_data,
  output wire [23:0] flash_address,
  output wire flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b,
flash_byte_b,
  input  wire flash_sts,
```

```
  // RS232
  output wire rs232_txd, rs232_rts,
  input wire rs232_rxd, rs232_cts,

  // PS2
  input wire mouse_clock, mouse_data,
  input wire keyboard_clock, keyboard_data,

  // FLUORESCENT DISPLAY
  output wire disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b,
  input wire disp_data_in,
  output wire disp_data_out,

  // SYSTEM ACE
  inout wire [15:0] systemace_data,
  output wire [6:0] systemace_address,
  output wire systemace_ce_b, systemace_we_b, systemace_oe_b,
  //input wire systemace_irq, systemace_mpbrdy,

  // BUTTONS, SWITCHES, LEDS
  input wire button0,
  input wire button1,
  input wire button2,
  input wire button3,
  input wire button_enter,
  input wire button_right,
  input wire button_left,
  input wire button_down,
  input wire button_up,
  input wire [7:0] switch,
  output wire [7:0] led,

  // USER CONNECTORS, DAUGHTER CARD, LOGIC ANALYZER
  inout wire [31:0] user1,
  inout wire [31:0] user2,
  inout wire [31:0] user3,
  inout wire [31:0] user4,
  inout wire [43:0] daughtercard,
  output wire [15:0] analyzer1_data, output wire analyzer1_clock,
  output wire [15:0] analyzer2_data, output wire analyzer2_clock,
  output wire [15:0] analyzer3_data, output wire analyzer3_clock,
  output wire [15:0] analyzer4_data, output wire analyzer4_clock,

  // CLOCKS
  input wire clock1,
  input wire clock2,
  input wire clock_27mhz
);

   ////////////////////////////////
   // Please note that much of the camera
   // code is based on the following file.
   // Modifications were made for color.
   //
   // File:   zbt_6111_sample.v
   // Date:   26-Nov-05
```

```
// Author: I. Chuang <ichuang@mit.edu>
////////////////////////////////

////////////////////////////////
// I/O Assignments
////////////////////////////////

// Audio Input and Output
assign beep= 1'b0;
assign audio_reset_b = 1'b0;
assign ac97_synch = 1'b0;
assign ac97_sdata_out = 1'b0;

// ac97_sdata_in is an input

// Video Output
assign tv_out_ycrcb = 10'h0;
assign tv_out_reset_b = 1'b0;
assign tv_out_clock = 1'b0;
assign tv_out_i2c_clock = 1'b0;
assign tv_out_i2c_data = 1'b0;
assign tv_out_pal_ntsc = 1'b0;
assign tv_out_hsync_b = 1'b1;
assign tv_out_vsync_b = 1'b1;
assign tv_out_blank_b = 1'b1;
assign tv_out_subcar_reset = 1'b0;

// Video Input
//assign tv_in_i2c_clock = 1'b0;
assign tv_in_fifo_read = 1'b1;
assign tv_in_fifo_clock = 1'b0;
assign tv_in_iso = 1'b1;
//assign tv_in_reset_b = 1'b0;
assign tv_in_clock = clock_27mhz;//1'b0;
//assign tv_in_i2c_data = 1'bZ;
// tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
// tv_in_aef, tv_in_hff, and tv_in_aff are inputs

// SRAMs
//assign ram0_data = 36'hZ;
//assign ram0_address = 19'h0;
//assign ram0_clk = 1'b0;
//assign ram0_we_b = 1'b1;
//assign ram0_cen_b = 1'b0;      // clock enable
assign ram0_ce_b = 1'b0;
assign ram0_oe_b = 1'b0;
assign ram0_adv_ld = 1'b0;
assign ram0_bwe_b = 4'h0;

//assign ram1_data = 36'hZ;
//assign ram1_address = 19'h0;
//assign ram1_adv_ld = 1'b0;
//assign ram1_clk = 1'b0;
//assign ram1_cen_b = 1'b1;
assign ram1_ce_b = 1'b0;
assign ram1_oe_b = 1'b0;
assign ram1_adv_ld = 1'b0;
```

```
assign ram1_bwe_b = 4'h0;

//assign clock_feedback_out = 1'b0;
// clock_feedback_in is an input

// Flash ROM
assign flash_data = 16'hZ;
assign flash_address = 24'h0;
assign flash_ce_b = 1'b1;
assign flash_oe_b = 1'b1;
assign flash_we_b = 1'b1;
assign flash_reset_b = 1'b0;
assign flash_byte_b = 1'b1;
// flash_sts is an input

// RS-232 Interface
assign rs232_txd = 1'b1;
assign rs232_rts = 1'b1;
// rs232_rxd and rs232_cts are inputs

// PS/2 Ports
// mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs

// LED Displays
//assign disp_blank = 1'b1;
//assign disp_clock = 1'b0;
//assign disp_rs = 1'b0;
//assign disp_ce_b = 1'b1;
//assign disp_reset_b = 1'b0;
//assign disp_data_out = 1'b0;
// disp_data_in is an input

// Buttons, Switches, and Individual LEDs
// assign led = 8'hFF;
// button0, button1, button2, button3, button_enter, button_right,
// button_left, button_down, button_up, and switches are inputs

// User I/Os
assign user1 = 32'hZ;
assign user2 = 32'hZ;
assign user3 = 32'hZ;
assign user4 = 32'hZ;

// Daughtercard Connectors
assign daughtercard = 44'hZ;

// SystemACE Microprocessor Port
assign systemace_data = 16'hZ;
assign systemace_address = 7'h0;
assign systemace_ce_b = 1'b1;
assign systemace_we_b = 1'b1;
assign systemace_oe_b = 1'b1;
// systemace_irq and systemace_mpbrdy are inputs

// Logic Analyzer
//assign analyzer1_data = 16'h0;
//assign analyzer1_clock = 1'b1;
```

```
   assign analyzer2_data = 16'h0;
   assign analyzer2_clock = 1'b1;
   //assign analyzer3_data = 16'h0;
   //assign analyzer3_clock = 1'b1;
   assign analyzer4_data = 16'h0;
   assign analyzer4_clock = 1'b1;

   ////////////////////////////
   // Generate clock signals
   ////////////////////////////

   // use FPGA's digital clock manager to produce a
   // 40MHz clock (actually 39.86MHz)
   wire clock_40mhz_unbuf,clock_40mhz_unphased;
   DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_40mhz_unbuf));
   // synthesis attribute CLKFX_DIVIDE of vclk1 is 21
   // synthesis attribute CLKFX_MULTIPLY of vclk1 is 31
   // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
   // synthesis attribute CLKIN_PERIOD of vclk1 is 37
   BUFG vclk2(.O(clock_40mhz_unphased),.I(clock_40mhz_unbuf));

   // Lock the clock to avoid timing delays
   wire clock_40mhz, locked;
   ramclock rc (clock_40mhz_unphased, clock_40mhz, ram0_clk, ram1_clk,
            clock_feedback_in, clock_feedback_out, locked);

   ////////////////////////////
   // Reset signals
   ////////////////////////////

   // power-on reset generation
   wire power_on_reset;    // remain high for first 16 clocks
   SRL16 reset_sr (.D(1'b0), .CLK(clock_40mhz), .Q(power_on_reset),
               .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   defparam reset_sr.INIT = 16'hFFFF;

   // ENTER button is user reset
   wire reset,user_reset,button_enter_clean;
   debounce db1(power_on_reset, clock_40mhz, ~button_enter,
button_enter_clean);
   assign reset = user_reset | power_on_reset;


   ////////////////////////////
   // debounce signals
   ////////////////////////////

   wire up,down,left,switch0, switch4,button0_clean,
switch5,switch2,switch3,button_right_clean,switch6,switch7;
   wire button1_clean, button2_clean, button3_clean;
   debounce
db2(.reset(reset),.clk(clock_40mhz),.noisy(~button_up),.clean(up));
   debounce
db3(.reset(reset),.clk(clock_40mhz),.noisy(~button_down),.clean(down));
   debounce
db4(.reset(reset),.clk(clock_40mhz),.noisy(~button_left),.clean(left));
```

```
   debounce
db5(.reset(reset),.clk(clock_40mhz),.noisy(~button0),.clean(user_reset));
   debounce
db6(.reset(reset),.clk(clock_40mhz),.noisy(switch[4]),.clean(switch4));
   debounce
db7(.reset(reset),.clk(clock_40mhz),.noisy(switch[5]),.clean(switch5));
   debounce
db8(.reset(reset),.clk(clock_40mhz),.noisy(switch[2]),.clean(switch2));
   debounce
db9(.reset(reset),.clk(clock_40mhz),.noisy(switch[3]),.clean(switch3));
   debounce
db10(.reset(reset),.clk(clock_40mhz),.noisy(~button_right),.clean(button_righ
t_clean));
   debounce
db11(.reset(reset),.clk(clock_40mhz),.noisy(switch[6]),.clean(switch6));
   debounce
db12(.reset(reset),.clk(clock_40mhz),.noisy(switch[7]),.clean(switch7));
   debounce
db13(.reset(reset),.clk(clock_40mhz),.noisy(switch[0]),.clean(switch0));
   debounce
db14(.reset(reset),.clk(clock_40mhz),.noisy(~button1),.clean(button1_clean));
   debounce
db15(.reset(reset),.clk(clock_40mhz),.noisy(~button2),.clean(button2_clean));
   debounce
db16(.reset(reset),.clk(clock_40mhz),.noisy(~button3),.clean(button3_clean));

   ////////////////////////////////
   // display module for debugging
   ////////////////////////////////

   reg [63:0] dispdata;
   display_16hex hexdisp1(reset, clock_40mhz, dispdata,
                  disp_blank, disp_clock, disp_rs, disp_ce_b,
                  disp_reset_b, disp_data_out);


   ////////////////////////////////////
   // generate basic XVGA video signals
   ////////////////////////////////////
   wire [10:0] hcount;
   wire [9:0]  vcount;
   wire hsync,vsync,blank;
   //xvga xvga1(clock_65mhz,hcount,vcount,hsync,vsync,blank); // 1024 x 768
   svga svga1(.vclock(clock_40mhz),.hcount(hcount),.vcount(vcount),
              .hsync(hsync),.vsync(vsync),.blank(blank));

   ////////////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////////
   ////////////// Sean's Modules //////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////////

   ////////////////////////////////////
   // wire up to ZBT ram 1
   ////////////////////////////////////

   // inputs
```

```
   wire [35:0] vram_write_data;
   wire [35:0] vram_read_data;
   wire [18:0] vram_addr;
   wire        vram_we;

   // Do not use zbt_6111 clock.  Use fix in order to avoid timing
   // delays that cause ZBT error.

   //zbt_6111 zbt1(clock_40mhz, 1'b1, vram_we, vram_addr,
   //    vram_write_data, vram_read_data,
   //    ram0_clk, ram0_we_b, ram0_address, ram0_data, ram0_cen_b);

   // clock enable (should be synchronous and one cycle high at a time)
   assign ram0_cen_b = 0;

   // create delayed ram_we signal: note the delay is by two cycles!
   // ie we present the data to be written two cycles after we is raised
   // this means the bus is tri-stated two cycles after we is raised.

   reg [1:0] we0_delay;

   always @(posedge clock_40mhz)
     we0_delay <= {we0_delay[0],vram_we};

   // create two-stage pipeline for write data

   reg [35:0] write_data0_old1;
   reg [35:0] write_data0_old2;
   always @(posedge clock_40mhz)
     {write_data0_old2, write_data0_old1} <= {write_data0_old1,
vram_write_data};

   // wire to ZBT RAM signals
   assign ram0_we_b = ~vram_we;
   assign ram0_address = vram_addr;
   assign ram0_data = we0_delay[1] ? write_data0_old2 : {36{1'bZ}};
   assign vram_read_data = ram0_data;

   ////////////////////////////////
   // Process read ZBT data for screen display
   ////////////////////////////////

   // generate pixel value from reading ZBT memory
   //wire [7:0]   vr_pixel; // b&w
   wire [17:0]  vr_pixel;
   wire [18:0]  vram_addr1;

   vram_display vd1(
            reset,
            clock_40mhz,
            hcount,
            vcount,
            vr_pixel,
            vram_addr1,
            vram_read_data);

   ////////////////////////
```

```
// tv decoder
//////////////////////////

// ADV7185 NTSC decoder interface code
// adv7185 initialization module
adv7185init adv7185(
         .reset(reset),
         .clock_27mhz(clock_27mhz),
         .source(1'b0),
         .tv_in_reset_b(tv_in_reset_b),
         .tv_in_i2c_clock(tv_in_i2c_clock),
         .tv_in_i2c_data(tv_in_i2c_data));

wire [29:0] ycrcb; // video data (luminance, chrominance)
wire [2:0]  fvh; // sync for field, vertical, horizontal
wire        dv;      // data valid

ntsc_decode decode (.clk(tv_in_line_clock1), .reset(reset),
                .tv_in_ycrcb(tv_in_ycrcb[19:10]),
                .ycrcb(ycrcb), .f(fvh[2]),
                .v(fvh[1]), .h(fvh[0]), .data_valid(dv));

//////////////////////////
// ycrcb to rgb conversion
//////////////////////////
// add following for color camera
wire [23:0] rgb_pixel;
YCrCb2RGB ycrcb2rgb(
  rgb_pixel[23:16],
  rgb_pixel[15:8],
  rgb_pixel[7:0],
  tv_in_line_clock1,
  reset,
  ycrcb[29:20],
  ycrcb[19:10],
  ycrcb[9:0]);

//////////////////////////
// generate zbt data signals from ntsc
//////////////////////////

wire [23:0] pf_threshold;
wire [23:0] filtered_pixel;
pixelfilter pf (
   .clk(clock_40mhz),
   .reset(reset),
   .up(up),
   .down(down),
   .rgb_pixel(rgb_pixel),
   .filtered_pixel(filtered_pixel),
   .threshold(pf_threshold)
   );

wire [18:0] ntsc_addr;
wire [35:0] ntsc_data;
wire        ntsc_we;
```

```
   //ntsc_to_zbt n2z (clock_40mhz, tv_in_line_clock1, fvh, dv, ycrcb[29:22],
// b&w
   ntsc_to_zbt n2z (
      clock_40mhz,
      tv_in_line_clock1,
      fvh,
      dv,
      {filtered_pixel[23:18], filtered_pixel[15:10], filtered_pixel[7:2]},
      ntsc_addr,
      ntsc_data,
      ntsc_we,
      //switch[6]);
      1'b0);

   ////////////////////////
   // calculate Center Of Mass (COM), scale, and smooth
   ////////////////////////
   //wire [10:0] comx1 = 11'd300;
   //wire [9:0]  comy1 = 10'd200;

   wire [10:0] com_1_x;
   wire [9:0]  com_1_y;
   wire [10:0] com_2_x;
   wire [9:0]  com_2_y;
   wire comvalid;

   comcalc comcalc1 (
      .reset(reset),
      .clk(clock_40mhz),
      .ntsc_addr(ntsc_addr),
      .ntsc_data(ntsc_data),
      .ntsc_we(ntsc_we),
      .threshold(pf_threshold),
      .com_1_x(com_1_x),
      .com_1_y(com_1_y),
      .com_2_x(com_2_x),
      .com_2_y(com_2_y),
      .comvalid(comvalid)
      );

   // scaled com
   wire [10:0] com_scale_1_x = switch5 ? com_1_x : ((com_1_x / 16) * 18);
   wire [9:0]  com_scale_1_y = switch5 ? com_1_y : ((com_1_y / 8) * 11);
   wire [10:0] com_scale_2_x = switch5 ? com_2_x : ((com_2_x / 16) * 18);
   wire [9:0]  com_scale_2_y = switch5 ? com_2_y : ((com_2_y / 8) * 11);

   // smoothed center of mass
   wire [10:0] com_smooth_1_x;
   smoother #(.HISTORY(8),.LOGHISTORY(3)) smoother1 (
      .reset(reset),
      .clk(clock_40mhz),
      .datavalue(com_scale_1_x),
      .datavalid(comvalid),
      //.datavalue(testinput),
      //.datavalid(1'b1),
      .smooth(com_smooth_1_x)
      );
```

```
   wire [10:0] com_smooth_1_y;
   smoother #(.HISTORY(8),.LOGHISTORY(3)) smoother2 (
      .reset(reset),
      .clk(clock_40mhz),
      .datavalue({1'b0,com_scale_1_y}),
      .datavalid(comvalid),
      .smooth(com_smooth_1_y)
      );

   wire [10:0] com_smooth_2_x;
   smoother #(.HISTORY(8),.LOGHISTORY(3)) smoother3 (
      .reset(reset),
      .clk(clock_40mhz),
      .datavalue(com_scale_2_x),
      .datavalid(comvalid),
      .smooth(com_smooth_2_x)
      );

   wire [10:0] com_smooth_2_y;
   smoother #(.HISTORY(8),.LOGHISTORY(3)) smoother4 (
      .reset(reset),
      .clk(clock_40mhz),
      .datavalue({1'b0,com_scale_2_y}),
      .datavalid(comvalid),
      .smooth(com_smooth_2_y)
      );

   // scaled com
   wire [10:0] com_final_1_x = switch4 ? com_smooth_1_x[10:0] :
com_scale_1_x;
   wire [9:0]  com_final_1_y = switch4 ? com_smooth_1_y[9:0]  :
com_scale_1_y;
   wire [10:0] com_final_2_x = switch4 ? com_smooth_2_x[10:0] :
com_scale_2_x;
   wire [9:0]  com_final_2_y = switch4 ? com_smooth_2_y[9:0]  :
com_scale_2_y;

   ////////////////////////
   // palette & intention modules
   ////////////////////////

   //color_brush
   wire [17:0] color_brush;
   wire [23:0] curr_brush_color = {color_brush[17:12], 2'b11,
color_brush[11:6], 2'b11, color_brush[5:0], 2'b11};

   // Parameters for the palette generation
   parameter TOTALWIDTH = 800;
   parameter TOTALHEIGHT = 600;
   parameter LEFTCOL_WIDTH = 110;
   parameter PALETTE_COLOR_WIDTH = 80;
   parameter RED_LEFT   = 15;
   parameter RED_TOP    = 10;
   parameter YEL_LEFT   = 15;
   parameter YEL_TOP    = 10 + PALETTE_COLOR_WIDTH + RED_TOP;
   parameter BLUE_LEFT  = 15;
```

```
parameter BLUE_TOP   = 10 + PALETTE_COLOR_WIDTH + YEL_TOP;
parameter WHITE_LEFT = 15;
parameter WHITE_TOP  = 10 + PALETTE_COLOR_WIDTH + BLUE_TOP;
parameter FILL_TOP = 20 + PALETTE_COLOR_WIDTH + WHITE_TOP;
parameter FILL_LEFT = 15;
parameter FILL_WIDTH = 38;
parameter SQ_TOP = 20 + PALETTE_COLOR_WIDTH + WHITE_TOP;
parameter SQ_LEFT = FILL_LEFT + FILL_WIDTH + 4;
parameter SQ_WIDTH = 38;
parameter SPECK_TOP = 4 + FILL_TOP + FILL_WIDTH;
parameter SPECK_LEFT = 15;
parameter SPECK_WIDTH = 38;
parameter LINE_TOP = 4 + FILL_TOP + FILL_WIDTH;
parameter LINE_LEFT = SPECK_LEFT + SPECK_WIDTH + 4;
parameter LINE_WIDTH = 38;
parameter SAT_LEFT = 15;
parameter SAT_TOP = 10 + SPECK_WIDTH + SPECK_TOP;
parameter SAT_HEIGHT = 20;
parameter SAT_BAR_WIDTH = PALETTE_COLOR_WIDTH/32;
parameter CURR_LEFT  = 15;
parameter CURR_TOP   = 5 + SAT_HEIGHT + SAT_TOP;
parameter CANVASBORDER = 30;
parameter XHAIR_RAD = 8;

// inputs to the palette generation
wire [4:0] brush_sat; // used to show the saturation bar

// ouput of palette generation
wire [23:0] palette_px;

// ouput of the intention module
wire [1:0]  brush_select;

wire [2:0]  absorb_color;
wire absorb_enable;
wire paint_enable;
wire fill_enable;

// generate a paint palette by checking
// what color the hcount,vcount pixel
// ought to be.
palette_gen pg1(
   .clk(clock_40mhz),
   .hcount(hcount),
   .vcount(vcount),
   .pixel(palette_px),
   .curr_rgb(curr_brush_color),
   .saturation(brush_sat),
   .com_1_x(com_final_1_x),
   .com_1_y(com_final_1_y),
   .com_2_x(com_final_2_x),
   .com_2_y(com_final_2_y),
   .brush_select(brush_select),
   .TOTALWIDTH(TOTALWIDTH)
   .TOTALHEIGHT(TOTALHEIGHT),
   .LEFTCOL_WIDTH(LEFTCOL_WIDTH),
   .PALETTE_COLOR_WIDTH(PALETTE_COLOR_WIDTH),
```

```
    .RED_LEFT(RED_LEFT),
    .RED_TOP(RED_TOP),
    .YEL_LEFT(YEL_LEFT), // actually green
    .YEL_TOP(YEL_TOP),
    .BLUE_LEFT(BLUE_LEFT),
    .BLUE_TOP(BLUE_TOP),
    .WHITE_LEFT(WHITE_LEFT),
    .WHITE_TOP(WHITE_TOP),
    .FILL_TOP(FILL_TOP),
    .FILL_LEFT(FILL_LEFT),
    .FILL_WIDTH(FILL_WIDTH),
    .SQ_TOP(SQ_TOP),
    .SQ_LEFT(SQ_LEFT),
    .SQ_WIDTH(SQ_WIDTH),
    .SPECK_TOP(SPECK_TOP),
    .SPECK_LEFT(SPECK_LEFT),
    .SPECK_WIDTH(SPECK_WIDTH),
    .LINE_TOP(LINE_TOP),
    .LINE_LEFT(LINE_LEFT),
    .LINE_WIDTH(LINE_WIDTH),
    .SAT_LEFT(SAT_LEFT),
    .SAT_TOP(SAT_TOP),
    .SAT_HEIGHT(SAT_HEIGHT),
    .SAT_BAR_WIDTH(SAT_BAR_WIDTH),
    .CURR_LEFT(CURR_LEFT),
    .CURR_TOP(CURR_TOP),
    .CANVASBORDER(CANVASBORDER),
    .XHAIR_RAD(XHAIR_RAD)
    ;

wire button_intent = button_right_clean | button_enter_clean;
intention_v2 intv2(
    .reset(reset),
    .clk(clock_40mhz),
    .button(button_intent),
    .com_1_x(com_final_1_x),
    .com_1_y(com_final_1_y),
    .brush_select(brush_select),
    .absorb_color(absorb_color),
    .absorb_enable(absorb_enable),
    .paint_enable(paint_enable),
    .fill_enable(fill_enable),
    .TOTALWIDTH(TOTALWIDTH),
    .TOTALHEIGHT(TOTALHEIGHT),
    .LEFTCOL_WIDTH(LEFTCOL_WIDTH),
    .PALETTE_COLOR_WIDTH(PALETTE_COLOR_WIDTH),
    .RED_LEFT(RED_LEFT),
    .RED_TOP(RED_TOP),
    .YEL_LEFT(YEL_LEFT), // actually green
    .YEL_TOP(YEL_TOP),
    .BLUE_LEFT(BLUE_LEFT),
    .BLUE_TOP(BLUE_TOP),
    .WHITE_LEFT(WHITE_LEFT),
    .WHITE_TOP(WHITE_TOP),
    .FILL_TOP(FILL_TOP),
    .FILL_LEFT(FILL_LEFT),
    .FILL_WIDTH(FILL_WIDTH),
```

```
      .SQ_TOP(SQ_TOP),
      .SQ_LEFT(SQ_LEFT),
      .SQ_WIDTH(SQ_WIDTH),
      .SPECK_TOP(SPECK_TOP),
      .SPECK_LEFT(SPECK_LEFT),
      .SPECK_WIDTH(SPECK_WIDTH),
      .LINE_TOP(LINE_TOP),
      .LINE_LEFT(LINE_LEFT),
      .LINE_WIDTH(LINE_WIDTH),
      .SAT_LEFT(SAT_LEFT),
      .SAT_TOP(SAT_TOP),
      .SAT_HEIGHT(SAT_HEIGHT),
      .SAT_BAR_WIDTH(SAT_BAR_WIDTH),
      .CURR_LEFT(CURR_LEFT),
      .CURR_TOP(CURR_TOP),
      .CANVASBORDER(CANVASBORDER),
      .XHAIR_RAD(XHAIR_RAD)
      );

   ////////////////////////
   // debug pattern 1
   ////////////////////////

   // code to write pattern to ZBT memory
   reg [31:0]     count;
   always @(posedge clock_40mhz) count <= reset ? 0 : count + 1;

   wire [18:0] vram_addr2 = count[0+18:0];
   wire [35:0] vpat = ( switch[1] ? {4{count[3+3:3],4'b0}} :
{4{count[3+4:4],4'b0}} );

   ////////////////////////
   // read / write to zbt 1
  ////////////////////////

   // mux selecting read/write to memory based on which write-enable is
chosen

   wire sw_ntsc = ~switch[7];
   //wire my_we = sw_ntsc ? (hcount[1:0]==2'd2) : blank; // b&w
   wire my_we = sw_ntsc ? (hcount[0]==1'd1) : blank;
   wire [18:0] write_addr = sw_ntsc ? ntsc_addr : vram_addr2;
   wire [35:0] write_data = sw_ntsc ? ntsc_data : vpat;

   assign vram_addr = my_we ? write_addr : vram_addr1;
   assign vram_we = my_we; // also goes to the com calculator
   assign vram_write_data = write_data;

   // select output pixel data

   //reg [7:0] pixel; // b&w
   reg [23:0] pixel;
   wire b,hs,vs;

   delayN #(.NDELAY(3)) dn1(clock_40mhz,hsync,hs); // delay by cycles to sync
with ZBT read
   delayN #(.NDELAY(3)) dn2(clock_40mhz,vsync,vs);
```

```
   delayN #(.NDELAY(3)) dn3(clock_40mhz,blank,b);

   ////////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////
   ////////////// Grace's Modules /////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////

   ///////////////////////////////////
   // wire up grace ZBT to ram
   ///////////////////////////////////

   reg [20:0] zbt_clear_counter = 0; // used to fill ZBT with a color, for
instance white at reset

   reg  [35:0] grace_write_data;
   reg  [35:0] grace_read_data;
   wire [35:0] grace_preprocess_read;
   reg  [18:0] grace_addr; // need to toggle this between read and write
   reg         grace_we;

   // Alternate on clock cycles between read and write

   // clock enable (should be synchronous and one cycle high at a time)
   assign ram1_cen_b = 0;

   // create delayed ram_we signal: note the delay is by two cycles!
   // ie we present the data to be written two cycles after we is raised
   // this means the bus is tri-stated two cycles after we is raised.

   reg [1:0] we1_delay;

   always @(posedge clock_40mhz)
      we1_delay <= {we1_delay[0],grace_we};

   // create two-stage pipeline for write data

   reg [35:0] write_data1_old1;
   reg [35:0] write_data1_old2;
   always @(posedge clock_40mhz)
      {write_data1_old2, write_data1_old1} <= {write_data1_old1,
grace_write_data};

   // wire to ZBT RAM signals
   assign ram1_we_b = ~grace_we;
   assign ram1_address = grace_addr;
   assign ram1_data = we1_delay[1] ? write_data1_old2 : {36{1'bZ}};
   assign grace_preprocess_read = ram1_data;

   ////////////////////////////////////
   // generate pixel to write to zbt
   ////////////////////////////////////

   wire [17:0] paint_to_zbt_rgb;
   wire [35:0] paint_zbt_data;
   wire paint_zbt_we;
```

```
gen_paint_zbt gen_paint_zbt1 (
   .clk(clock_40mhz),
   .reset(reset),
   .x(hcount),
   .y(vcount),
   .data(paint_to_zbt_rgb),
   .ntsc_addr(), // not needed, same as grace_addr1
   .ntsc_data(paint_zbt_data),
   .ntsc_we(paint_zbt_we)
   );

//////////////////////////////////
// Generate brushes
//////////////////////////////////

wire brushenable;
wire [10:0] x1;
wire [10:0] x2;
wire [10:0] x3;
wire [10:0] x4;
wire [10:0] x5;
wire [9:0] y1;
wire [9:0] y2;
wire [9:0] y3;
wire [9:0] y4;
wire [9:0] y5;

// Generate five points between centers of mass
brushcentergen brushgen(
   .ix1(com_final_1_x),
   .iy1(com_final_1_y),
   .ix2(com_final_2_x),
   .iy2(com_final_2_y),
   .clk(clock_40mhz),
   .brushenable(brushenable),
   .x1(x1),
   .y1(y1),
   .x2(x2),
   .y2(y2),
   .x3(x3),
   .y3(y3),
   .x4(x4),
   .y4(y4),
   .x5(x5),
   .y5(y5));

wire bool;
wire bool1;
wire bool2;
wire bool3;
wire bool4;
wire bool5;
wire boolline;
wire boolsq;

// scritchy scratchy brush
brush1 myrect1(.x(x1),
```

```
   .hcount(hcount),
   .y(y1),
   .vcount(vcount),
   .bool(bool1));
brush1 myrect2(.x(x2),
   .hcount(hcount),
   .y(y2),
   .vcount(vcount),
   .bool(bool2));
brush1 myrect3(.x(x3),
   .hcount(hcount),
   .y(y3),
   .vcount(vcount),
   .bool(bool3));
brush1 myrect4(.x(x4),
   .hcount(hcount),
   .y(y4),
   .vcount(vcount),
   .bool(bool4));
brush1 myrect5(.x(x5),
   .hcount(hcount),
   .y(y5),
   .vcount(vcount),
   .bool(bool5));

// square brush
brushsquare mybrushsq(
   .x(x1),
   .hcount(hcount),
   .y(y1),
   .vcount(vcount),
   .bool(boolsq));

brushline myline (
   .x1(com_final_1_x),
   .x2(com_final_2_x),
   .hcount(hcount),
   .y1(com_final_1_y),
   .y2(com_final_2_y),
   .vcount(vcount)
   .bool(boolline));

assign bool = (brush_select == 2'b11)
         ? boolline
         : ((brush_select == 2'b10)
               ? (bool1 | bool2 | bool3 | bool4 | bool5)
               : boolsq
               );

//////////////////////////////////
// generate pixel value from reading ZBT memory
//////////////////////////////////
wire color_reset;
wire [17:0] grace_vr_pixel;
wire [2:0] sel_color = absorb_color;
wire [1:0] intention = {paint_enable, absorb_enable};
wire signed [4:0] sel_sat;
```

```
   wire [18:0] grace_addr1;
   wire paint_enable_bool = paint_enable & bool; // intend to write & brush
px

   assign color_reset = left;

   painter mypaint(
      .vclock(clock_40mhz),
      .bool(bool),
      .color_in(grace_vr_pixel),
      .color_brush(color_brush),
      .sat(brush_sat),
      .color_out(paint_to_zbt_rgb));

   colorgen mycolorgen(
      .clk(clock_40mhz),
      .sel_color(sel_color),
      .sel_sat(sel_sat),
      .reset(reset),
      .color_reset(color_reset),
      .color(color_brush),
      .sat(brush_sat));

   saturation mysat(
      .clock(clock_40mhz),
      .intention(intention),
      .reset(reset),
      .sat_change(sel_sat));

   grace_vram_display vd_grace(
      .reset(reset),
      .clk(clock_40mhz),
      .hcount(hcount),
      .vcount(vcount),
      .vram_read_data(grace_read_data),
      .vr_pixel(grace_vr_pixel), // output pixel to show to screen
      .vram_addr(grace_addr1) // output addr to read/write from zbt
      );

   reg [18:0] grace_write_addr; // will be a delayed version of the read
address

   ////////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////
   ////////////// VGA Assignment //////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////
   ////////////////////////////////////////////////////////////////////////

   ///////////////////////
   // assign vga pixel value
   ///////////////////////

   reg [17:0] rgb; // graceli
   reg delayedbool1;
   reg delayedbool2;
   reg delayedbool3;
   reg delayedbool4;
```

```
   reg [18:0] grace_addr_delay1;
   reg [18:0] grace_addr_delay2;
   reg [18:0] grace_addr_delay3;
   reg [18:0] grace_addr_delay4;
   reg [18:0] grace_addr_delay5;
   reg [18:0] grace_addr_delay6;
   reg [18:0] grace_addr_delay7;
   reg [18:0] grace_addr_delay8;
   wire zbt_we_with_delaybools = paint_zbt_we & (delayedbool1 |
delayedbool2);

   reg [35:0] latched_write_data;

   reg [35:0] fill_color;

   always @(posedge clock_40mhz) begin
      ////////////////////
      // Grace pixel
      ////////////////////

      // Code for setting ZBT to be all 1's, or some fill color
      if (reset || fill_enable) begin
            zbt_clear_counter <= 0;

            if (fill_enable) begin
                  grace_write_data <= {color_brush, color_brush};
                  fill_color <= {color_brush, color_brush};
            end
            else begin
                  grace_write_data <= 36'hFFFFFFFFF;
                  fill_color <= 36'hFFFFFFFFF;
            end

            grace_we <= 1;
            grace_read_data <= 0;
            grace_addr <= grace_addr1; // do not delay white write
            grace_addr_delay1 <= 0;
            grace_addr_delay2 <= 0;
            grace_addr_delay3 <= 0;
            grace_addr_delay4 <= 0;
            grace_addr_delay5 <= 0;
            grace_addr_delay6 <= 0;
            grace_addr_delay7 <= 0;
            grace_addr_delay8 <= 0;
            grace_write_addr <= 0; // seanyliuchange
      end
      else if (zbt_clear_counter == 1024*1024) begin // arbitrary big number
            grace_write_data <= paint_zbt_data;
            grace_we <= zbt_we_with_delaybools; // only write if the brush
needs to write
            grace_addr <= zbt_we_with_delaybools ? grace_write_addr :
grace_addr1; // toggle between read and write
            if (!grace_we) grace_read_data <= grace_preprocess_read;
      end
      else begin
            zbt_clear_counter <= zbt_clear_counter + 1;
            grace_write_data <= fill_color;
```

```
            grace_we <= 1;
            grace_addr <= grace_addr1; // do not delay white write
      end
      delayedbool1 <= paint_enable_bool; // delay by 1 cycle because
paint_zbt_data delayed 1 cycle
      delayedbool2 <= delayedbool1;
      delayedbool3 <= delayedbool2;
      delayedbool4 <= delayedbool3;
      rgb <= button1_clean ? paint_to_zbt_rgb : grace_vr_pixel;

      grace_addr_delay1 <= grace_addr1;
      grace_addr_delay2 <= grace_addr_delay1;
      grace_addr_delay3 <= grace_addr_delay2;
      grace_addr_delay4 <= grace_addr_delay3;
      grace_addr_delay5 <= grace_addr_delay4;
      grace_addr_delay6 <= grace_addr_delay5;
      grace_addr_delay7 <= grace_addr_delay6;
      grace_addr_delay8 <= grace_addr_delay7;
      grace_write_addr <= grace_addr_delay7;

      if (grace_we) latched_write_data <= grace_write_data;

      ////////////////////
      // Sean pixel
      ////////////////////

      pixel <= switch[0]
            ? {hcount[8:6],5'b0,hcount[8:6],5'b0,hcount[8:6],5'b0}
            : {vr_pixel[17:12],2'b0, vr_pixel[11:6],2'b0,
vr_pixel[5:0],2'b0};

      // dispdata <= {vram_read_data,9'b0,vram_addr}; // 19+36+9 = 45 54
      // dispdata <= {ntsc_data,com_1_x[8:0],ntsc_addr}; // seanyliu
      dispdata <= {
            16'b0,
            1'b0,ntsc_addr, // 19+1 = 20
            2'b0,ntsc_data[13:0], // 14bits+2 = 16
            1'b0,absorb_color, // 3bits + 1 = 4
            3'b0,brush_sat // 5bits + 3 = 8
            };
      //dispdata <= {17'h0,{pixel[17:12], 2'b1}, {pixel[11:6], 2'b1},
{pixel[5:0], 2'b1}};
   end

   // b&w:
   // VGA Output.  In order to meet the setup and hold times of the
   // AD7125, we send it ~clock_40mhz.
   //assign vga_out_red = pixel;
   //assign vga_out_green = pixel;
   //assign vga_out_blue = pixel;

   wire [23:0] com_1_pixel;
   xhair_red com1(
      .xcom(com_final_1_x),
      .ycom(com_final_1_y),
      .hcount(hcount),
      .vcount(vcount),
```

```
      .pixel(com_1_pixel)
      );

   wire [23:0] com_2_pixel;
   xhair_green com2(
      .xcom(com_final_2_x),
      .ycom(com_final_2_y),
      .hcount(hcount),
      .vcount(vcount),
      .pixel(com_2_pixel)
      );

   ////////////////////////////
   // pixel assignments
   ////////////////////////////

   wire [7:0] sean_out_red;
   wire [7:0] sean_out_green;
   wire [7:0] sean_out_blue;

   wire sean_enable = switch3; // toggle between sean and grace module
displays

   assign sean_out_red = (((palette_px != 0) && switch2) ? palette_px[23:16]
: pixel[23:16])
                  | com_1_pixel[23:16] | com_2_pixel[23:16];

   assign sean_out_green = (((palette_px != 0) && switch2) ? palette_px[15:8]
: pixel[15:8])
                   | com_1_pixel[15:8] | com_2_pixel[15:8];

   assign sean_out_blue = (((palette_px != 0) && switch2) ? palette_px[7:0] :
pixel[7:0])
                  | com_1_pixel[7:0] | com_2_pixel[7:0];

   wire [7:0] grace_out_red;
   wire [7:0] grace_out_green;
   wire [7:0] grace_out_blue;

   assign grace_out_red =
      ((palette_px != 0) && switch2)
      ? palette_px[23:16]
      : {rgb[17:12], 2'b1};
   assign grace_out_green =
      ((palette_px != 0) && switch2)
      ? palette_px[15:8]
      : {rgb[11:6], 2'b1};
   assign grace_out_blue =
      ((palette_px != 0) && switch2)
      ? palette_px[7:0]
      : {rgb[5:0], 2'b1};

   assign vga_out_red = sean_enable ? sean_out_red : grace_out_red;
   assign vga_out_green = sean_enable ? sean_out_green : grace_out_green;
   assign vga_out_blue = sean_enable ? sean_out_blue : grace_out_blue;

   assign vga_out_sync_b = 1'b1;    // not used
```

```
   assign vga_out_pixel_clock = ~clock_40mhz;
   assign vga_out_blank_b = ~b;
   assign vga_out_hsync = hs;
   assign vga_out_vsync = vs;

   /////////////////
   // debugging
   /////////////////
   assign led = ~{1'd0, bool, button_right_clean, paint_enable,
absorb_enable, fill_enable, switch4, button1_clean};
   assign analyzer1_data = {bool, grace_addr[5:0], grace_we,
grace_write_data[35:28]};
   assign analyzer1_clock = {clock_40mhz};
   assign analyzer3_data = {grace_preprocess_read[35:28],
grace_read_data[35:28]};
   assign analyzer3_clock = {clock_40mhz};

endmodule
```

B video_decoder.v

```
//
// File:   video_decoder.v
// Date:   31-Oct-05
// Author: J. Castro (MIT 6.111, fall 2005)
//
// This file contains the ntsc_decode and adv7185init modules
//
// These modules are used to grab input NTSC video data from the RCA
// phono jack on the right hand side of the 6.111 labkit (connect
// the camera to the LOWER jack).
//

//////////////////////////////////////////////////////////////////////////
//
// NTSC decode - 16-bit CCIR656 decoder
// By Javier Castro
// This module takes a stream of LLC data from the adv7185
// NTSC/PAL video decoder and generates the corresponding pixels,
// that are encoded within the stream, in YCrCb format.

// Make sure that the adv7185 is set to run in 16-bit LLC2 mode.

module ntsc_decode(clk, reset, tv_in_ycrcb, ycrcb, f, v, h, data_valid);

   // clk - line-locked clock (in this case, LLC1 which runs at 27Mhz)
   // reset - system reset
   // tv_in_ycrcb - 10-bit input from chip. should map to pins [19:10]
   // ycrcb - 24 bit luminance and chrominance (8 bits each)
   // f - field: 1 indicates an even field, 0 an odd field
   // v - vertical sync: 1 means vertical sync
   // h - horizontal sync: 1 means horizontal sync

   input clk;
   input reset;
   input [9:0] tv_in_ycrcb; // modified for 10 bit input - should be P[19:10]
```

```
   output [29:0] ycrcb;
   output   f;
   output   v;
   output   h;
   output   data_valid;
   // output [4:0] state;

   parameter      SYNC_1 = 0;
   parameter      SYNC_2 = 1;
   parameter      SYNC_3 = 2;
   parameter      SAV_f1_cb0 = 3;
   parameter      SAV_f1_y0 = 4;
   parameter      SAV_f1_cr1 = 5;
   parameter      SAV_f1_y1 = 6;
   parameter      EAV_f1 = 7;
   parameter      SAV_VBI_f1 = 8;
   parameter      EAV_VBI_f1 = 9;
   parameter      SAV_f2_cb0 = 10;
   parameter      SAV_f2_y0 = 11;
   parameter      SAV_f2_cr1 = 12;
   parameter      SAV_f2_y1 = 13;
   parameter      EAV_f2 = 14;
   parameter      SAV_VBI_f2 = 15;
   parameter      EAV_VBI_f2 = 16;

   // In the start state, the module doesn't know where
   // in the sequence of pixels, it is looking.

   // Once we determine where to start, the FSM goes through a normal
   // sequence of SAV process_YCrCb EAV... repeat

   // The data stream looks as follows
   // SAV_FF | SAV_00 | SAV_00 | SAV_XY | Cb0 | Y0 | Cr1 | Y1 | Cb2 | Y2 |
... | EAV sequence
   // There are two things we need to do:
   //   1. Find the two SAV blocks (stands for Start Active Video perhaps?)
   //   2. Decode the subsequent data

   reg [4:0]      current_state = 5'h00;
   reg [9:0]      y = 10'h000;  // luminance
   reg [9:0]      cr = 10'h000; // chrominance
   reg [9:0]      cb = 10'h000; // more chrominance

   assign   state = current_state;

   always @ (posedge clk)
     begin
      if (reset)
        begin

        end
      else
        begin
           // these states don't do much except allow us to know where we are
in the stream.
           // whenever the synchronization code is seen, go back to the
sync_state before
```

```
            // transitioning to the new state
            case (current_state)
              SYNC_1: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_2 :
SYNC_1;
              SYNC_2: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_3 :
SYNC_1;
              SYNC_3: current_state <= (tv_in_ycrcb == 10'h200) ? SAV_f1_cb0 :
                              (tv_in_ycrcb == 10'h274) ? EAV_f1 :
                              (tv_in_ycrcb == 10'h2ac) ? SAV_VBI_f1 :
                              (tv_in_ycrcb == 10'h2d8) ? EAV_VBI_f1 :
                              (tv_in_ycrcb == 10'h31c) ? SAV_f2_cb0 :
                              (tv_in_ycrcb == 10'h368) ? EAV_f2 :
                              (tv_in_ycrcb == 10'h3b0) ? SAV_VBI_f2 :
                              (tv_in_ycrcb == 10'h3c4) ? EAV_VBI_f2 : SYNC_1;

              SAV_f1_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f1_y0;
              SAV_f1_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f1_cr1;
              SAV_f1_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f1_y1;
              SAV_f1_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f1_cb0;

              SAV_f2_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f2_y0;
              SAV_f2_y0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f2_cr1;
              SAV_f2_cr1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f2_y1;
              SAV_f2_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 :
SAV_f2_cb0;

              // These states are here in the event that we want to cover
these signals
              // in the future. For now, they just send the state machine back
to SYNC_1
              EAV_f1: current_state <= SYNC_1;
              SAV_VBI_f1: current_state <= SYNC_1;
              EAV_VBI_f1: current_state <= SYNC_1;
              EAV_f2: current_state <= SYNC_1;
              SAV_VBI_f2: current_state <= SYNC_1;
              EAV_VBI_f2: current_state <= SYNC_1;

            endcase
          end
      end // always @ (posedge clk)

   // implement our decoding mechanism

   wire y_enable;
   wire cr_enable;
   wire cb_enable;

   // if y is coming in, enable the register
   // likewise for cr and cb
   assign y_enable = (current_state == SAV_f1_y0) ||
```

```
                  (current_state == SAV_f1_y1) ||
                  (current_state == SAV_f2_y0) ||
                  (current_state == SAV_f2_y1);
   assign cr_enable = (current_state == SAV_f1_cr1) ||
                   (current_state == SAV_f2_cr1);
   assign cb_enable = (current_state == SAV_f1_cb0) ||
                   (current_state == SAV_f2_cb0);

   // f, v, and h only go high when active
   assign {v,h} = (current_state == SYNC_3) ? tv_in_ycrcb[7:6] : 2'b00;

   // data is valid when we have all three values: y, cr, cb
   assign data_valid = y_enable;
   assign ycrcb = {y,cr,cb};

   reg        f = 0;

   always @ (posedge clk)
     begin
      y <= y_enable ? tv_in_ycrcb : y;
      cr <= cr_enable ? tv_in_ycrcb : cr;
      cb <= cb_enable ? tv_in_ycrcb : cb;
      f <= (current_state == SYNC_3) ? tv_in_ycrcb[8] : f;
     end

endmodule



//////////////////////////////////////////////////////////////////////////////
//
//
// 6.111 FPGA Labkit -- ADV7185 Video Decoder Configuration Init
//
// Created:
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////
//

//////////////////////////////////////////////////////////////////////////////
//
// Register 0
//////////////////////////////////////////////////////////////////////////////
//

`define INPUT_SELECT                            4'h0
  // 0: CVBS on AIN1 (composite video in)
  // 7: Y on AIN2, C on AIN5 (s-video in)
  // (These are the only configurations supported by the 6.111 labkit
hardware)
`define INPUT_MODE                              4'h0
  // 0: Autodetect: NTSC or PAL (BGHID), w/o pedestal
  // 1: Autodetect: NTSC or PAL (BGHID), w/pedestal
  // 2: Autodetect: NTSC or PAL (N), w/o pedestal
  // 3: Autodetect: NTSC or PAL (N), w/pedestal
  // 4: NTSC w/o pedestal
```

```
  // 5: NTSC w/pedestal
  // 6: NTSC 4.43 w/o pedestal
  // 7: NTSC 4.43 w/pedestal
  // 8: PAL BGHID w/o pedestal
  // 9: PAL N w/pedestal
  // A: PAL M w/o pedestal
  // B: PAL M w/pedestal
  // C: PAL combination N
  // D: PAL combination N w/pedestal
  // E-F: [Not valid]

`define ADV7185_REGISTER_0 {`INPUT_MODE, `INPUT_SELECT}

////////////////////////////////////////////////////////////////////////////
//
// Register 1
////////////////////////////////////////////////////////////////////////////
//

`define VIDEO_QUALITY                          2'h0
  // 0: Broadcast quality
  // 1: TV quality
  // 2: VCR quality
  // 3: Surveillance quality
`define SQUARE_PIXEL_IN_MODE                   1'b0
  // 0: Normal mode
  // 1: Square pixel mode
`define DIFFERENTIAL_INPUT                     1'b0
  // 0: Single-ended inputs
  // 1: Differential inputs
`define FOUR_TIMES_SAMPLING                    1'b0
  // 0: Standard sampling rate
  // 1: 4x sampling rate (NTSC only)
`define BETACAM                                1'b0
  // 0: Standard video input
  // 1: Betacam video input
`define AUTOMATIC_STARTUP_ENABLE               1'b1
  // 0: Change of input triggers reacquire
  // 1: Change of input does not trigger reacquire

`define ADV7185_REGISTER_1 {`AUTOMATIC_STARTUP_ENABLE, 1'b0, `BETACAM,
`FOUR_TIMES_SAMPLING, `DIFFERENTIAL_INPUT, `SQUARE_PIXEL_IN_MODE,
`VIDEO_QUALITY}

////////////////////////////////////////////////////////////////////////////
//
// Register 2
////////////////////////////////////////////////////////////////////////////
//

`define Y_PEAKING_FILTER                       3'h4
  // 0: Composite =  4.5dB,  s-video =  9.25dB
  // 1: Composite =  4.5dB,  s-video =  9.25dB
  // 2: Composite =  4.5dB,  s-video =  5.75dB
  // 3: Composite =  1.25dB, s-video =  3.3dB
  // 4: Composite =  0.0dB,  s-video =  0.0dB
  // 5: Composite = -1.25dB, s-video = -3.0dB
```

```
  // 6: Composite = -1.75dB, s-video = -8.0dB
  // 7: Composite = -3.0dB,  s-video = -8.0dB
`define CORING                                  2'h0
  // 0: No coring
  // 1: Truncate if Y < black+8
  // 2: Truncate if Y < black+16
  // 3: Truncate if Y < black+32

`define ADV7185_REGISTER_2 {3'b000, `CORING, `Y_PEAKING_FILTER}

////////////////////////////////////////////////////////////////////////
//
// Register 3
////////////////////////////////////////////////////////////////////////
//

`define INTERFACE_SELECT                        2'h0
  // 0: Philips-compatible
  // 1: Broktree API A-compatible
  // 2: Broktree API B-compatible
  // 3: [Not valid]
`define OUTPUT_FORMAT                           4'h0
  // 0: 10-bit @ LLC, 4:2:2 CCIR656
  // 1: 20-bit @ LLC, 4:2:2 CCIR656
  // 2: 16-bit @ LLC, 4:2:2 CCIR656
  // 3: 8-bit @ LLC, 4:2:2 CCIR656
  // 4: 12-bit @ LLC, 4:1:1
  // 5-F: [Not valid]
  // (Note that the 6.111 labkit hardware provides only a 10-bit interface to
  // the ADV7185.)
`define TRISTATE_OUTPUT_DRIVERS                 1'b0
  // 0: Drivers tristated when ~OE is high
  // 1: Drivers always tristated
`define VBI_ENABLE                              1'b0
  // 0: Decode lines during vertical blanking interval
  // 1: Decode only active video regions

`define ADV7185_REGISTER_3 {`VBI_ENABLE, `TRISTATE_OUTPUT_DRIVERS,
`OUTPUT_FORMAT, `INTERFACE_SELECT}

////////////////////////////////////////////////////////////////////////
//
// Register 4
////////////////////////////////////////////////////////////////////////
//

`define OUTPUT_DATA_RANGE                       1'b0
  // 0: Output values restricted to CCIR-compliant range
  // 1: Use full output range
`define BT656_TYPE                              1'b0
  // 0: BT656-3-compatible
  // 1: BT656-4-compatible

`define ADV7185_REGISTER_4 {`BT656_TYPE, 3'b000, 3'b110, `OUTPUT_DATA_RANGE}

////////////////////////////////////////////////////////////////////////
//
```

```
// Register 5
////////////////////////////////////////////////////////////////////////////
//


`define GENERAL_PURPOSE_OUTPUTS                 4'b0000
`define GPO_0_1_ENABLE                          1'b0
  // 0: General purpose outputs 0 and 1 tristated
  // 1: General purpose outputs 0 and 1 enabled
`define GPO_2_3_ENABLE                          1'b0
  // 0: General purpose outputs 2 and 3 tristated
  // 1: General purpose outputs 2 and 3 enabled
`define BLANK_CHROMA_IN_VBI                     1'b1
  // 0: Chroma decoded and output during vertical blanking
  // 1: Chroma blanked during vertical blanking
`define HLOCK_ENABLE                            1'b0
  // 0: GPO 0 is a general purpose output
  // 1: GPO 0 shows HLOCK status

`define ADV7185_REGISTER_5 {`HLOCK_ENABLE, `BLANK_CHROMA_IN_VBI,
`GPO_2_3_ENABLE, `GPO_0_1_ENABLE, `GENERAL_PURPOSE_OUTPUTS}

////////////////////////////////////////////////////////////////////////////
//
// Register 7
////////////////////////////////////////////////////////////////////////////
//

`define FIFO_FLAG_MARGIN                        5'h10
  // Sets the locations where FIFO almost-full and almost-empty flags are set
`define FIFO_RESET                              1'b0
  // 0: Normal operation
  // 1: Reset FIFO. This bit is automatically cleared
`define AUTOMATIC_FIFO_RESET                    1'b0
  // 0: No automatic reset
  // 1: FIFO is autmatically reset at the end of each video field
`define FIFO_FLAG_SELF_TIME                     1'b1
  // 0: FIFO flags are synchronized to CLKIN
  // 1: FIFO flags are synchronized to internal 27MHz clock

`define ADV7185_REGISTER_7 {`FIFO_FLAG_SELF_TIME, `AUTOMATIC_FIFO_RESET,
`FIFO_RESET, `FIFO_FLAG_MARGIN}

////////////////////////////////////////////////////////////////////////////
//
// Register 8
////////////////////////////////////////////////////////////////////////////
//

`define INPUT_CONTRAST_ADJUST                       8'h80

`define ADV7185_REGISTER_8 {`INPUT_CONTRAST_ADJUST}

////////////////////////////////////////////////////////////////////////////
//
// Register 9
```

```
////////////////////////////////////////////////////////////////////////////
//

`define INPUT_SATURATION_ADJUST                         8'h8C

`define ADV7185_REGISTER_9 {`INPUT_SATURATION_ADJUST}

////////////////////////////////////////////////////////////////////////////
//
// Register A
////////////////////////////////////////////////////////////////////////////
//

`define INPUT_BRIGHTNESS_ADJUST                         8'h00

`define ADV7185_REGISTER_A {`INPUT_BRIGHTNESS_ADJUST}

////////////////////////////////////////////////////////////////////////////
//
// Register B
////////////////////////////////////////////////////////////////////////////
//

`define INPUT_HUE_ADJUST                                8'h00

`define ADV7185_REGISTER_B {`INPUT_HUE_ADJUST}

////////////////////////////////////////////////////////////////////////////
//
// Register C
////////////////////////////////////////////////////////////////////////////
//

`define DEFAULT_VALUE_ENABLE                        1'b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values
`define DEFAULT_VALUE_AUTOMATIC_ENABLE          1'b0
  // 0: Use programmed Y, Cr, and Cb values
  // 1: Use default values if lock is lost
`define DEFAULT_Y_VALUE                             6'h0C
  // Default Y value

`define ADV7185_REGISTER_C {`DEFAULT_Y_VALUE,
`DEFAULT_VALUE_AUTOMATIC_ENABLE, `DEFAULT_VALUE_ENABLE}

////////////////////////////////////////////////////////////////////////////
//
// Register D
////////////////////////////////////////////////////////////////////////////
//

`define DEFAULT_CR_VALUE                            4'h8
  // Most-significant four bits of default Cr value
`define DEFAULT_CB_VALUE                            4'h8
  // Most-significant four bits of default Cb value

`define ADV7185_REGISTER_D {`DEFAULT_CB_VALUE, `DEFAULT_CR_VALUE}
```

```
////////////////////////////////////////////////////////////////////////////
//
// Register E
////////////////////////////////////////////////////////////////////////////
//

`define TEMPORAL_DECIMATION_ENABLE              1'b0
  // 0: Disable
  // 1: Enable
`define TEMPORAL_DECIMATION_CONTROL             2'h0
  // 0: Supress frames, start with even field
  // 1: Supress frames, start with odd field
  // 2: Supress even fields only
  // 3: Supress odd fields only
`define TEMPORAL_DECIMATION_RATE                4'h0
  // 0-F: Number of fields/frames to skip

`define ADV7185_REGISTER_E {1'b0, `TEMPORAL_DECIMATION_RATE,
`TEMPORAL_DECIMATION_CONTROL, `TEMPORAL_DECIMATION_ENABLE}

////////////////////////////////////////////////////////////////////////////
//
// Register F
////////////////////////////////////////////////////////////////////////////
//

`define POWER_SAVE_CONTROL                      2'h0
  // 0: Full operation
  // 1: CVBS only
  // 2: Digital only
  // 3: Power save mode
`define POWER_DOWN_SOURCE_PRIORITY              1'b0
  // 0: Power-down pin has priority
  // 1: Power-down control bit has priority
`define POWER_DOWN_REFERENCE                    1'b0
  // 0: Reference is functional
  // 1: Reference is powered down
`define POWER_DOWN_LLC_GENERATOR                1'b0
  // 0: LLC generator is functional
  // 1: LLC generator is powered down
`define POWER_DOWN_CHIP                         1'b0
  // 0: Chip is functional
  // 1: Input pads disabled and clocks stopped
`define TIMING_REACQUIRE                        1'b0
  // 0: Normal operation
  // 1: Reacquire video signal (bit will automatically reset)
`define RESET_CHIP                              1'b0
  // 0: Normal operation
  // 1: Reset digital core and I2C interface (bit will automatically reset)

`define ADV7185_REGISTER_F {`RESET_CHIP, `TIMING_REACQUIRE, `POWER_DOWN_CHIP,
`POWER_DOWN_LLC_GENERATOR, `POWER_DOWN_REFERENCE,
`POWER_DOWN_SOURCE_PRIORITY, `POWER_SAVE_CONTROL}

////////////////////////////////////////////////////////////////////////////
//
```

```
// Register 33
////////////////////////////////////////////////////////////////////////////
//

`define PEAK_WHITE_UPDATE                       1'b1
  // 0: Update gain once per line
  // 1: Update gain once per field
`define AVERAGE_BIRIGHTNESS_LINES               1'b1
  // 0: Use lines 33 to 310
  // 1: Use lines 33 to 270
`define MAXIMUM_IRE                             3'h0
  // 0: PAL: 133, NTSC: 122
  // 1: PAL: 125, NTSC: 115
  // 2: PAL: 120, NTSC: 110
  // 3: PAL: 115, NTSC: 105
  // 4: PAL: 110, NTSC: 100
  // 5: PAL: 105, NTSC: 100
  // 6-7: PAL: 100, NTSC: 100
`define COLOR_KILL                              1'b1
  // 0: Disable color kill
  // 1: Enable color kill

`define ADV7185_REGISTER_33 {1'b1, `COLOR_KILL, 1'b1, `MAXIMUM_IRE,
`AVERAGE_BIRIGHTNESS_LINES, `PEAK_WHITE_UPDATE}

`define ADV7185_REGISTER_10 8'h00
`define ADV7185_REGISTER_11 8'h00
`define ADV7185_REGISTER_12 8'h00
`define ADV7185_REGISTER_13 8'h45
`define ADV7185_REGISTER_14 8'h18
`define ADV7185_REGISTER_15 8'h60
`define ADV7185_REGISTER_16 8'h00
`define ADV7185_REGISTER_17 8'h01
`define ADV7185_REGISTER_18 8'h00
`define ADV7185_REGISTER_19 8'h10
`define ADV7185_REGISTER_1A 8'h10
`define ADV7185_REGISTER_1B 8'hF0
`define ADV7185_REGISTER_1C 8'h16
`define ADV7185_REGISTER_1D 8'h01
`define ADV7185_REGISTER_1E 8'h00
`define ADV7185_REGISTER_1F 8'h3D
`define ADV7185_REGISTER_20 8'hD0
`define ADV7185_REGISTER_21 8'h09
`define ADV7185_REGISTER_22 8'h8C
`define ADV7185_REGISTER_23 8'hE2
`define ADV7185_REGISTER_24 8'h1F
`define ADV7185_REGISTER_25 8'h07
`define ADV7185_REGISTER_26 8'hC2
`define ADV7185_REGISTER_27 8'h58
`define ADV7185_REGISTER_28 8'h3C
`define ADV7185_REGISTER_29 8'h00
`define ADV7185_REGISTER_2A 8'h00
`define ADV7185_REGISTER_2B 8'hA0
`define ADV7185_REGISTER_2C 8'hCE
`define ADV7185_REGISTER_2D 8'hF0
`define ADV7185_REGISTER_2E 8'h00
`define ADV7185_REGISTER_2F 8'hF0
```

```
`define ADV7185_REGISTER_30 8'h00
`define ADV7185_REGISTER_31 8'h70
`define ADV7185_REGISTER_32 8'h00
`define ADV7185_REGISTER_34 8'h0F
`define ADV7185_REGISTER_35 8'h01
`define ADV7185_REGISTER_36 8'h00
`define ADV7185_REGISTER_37 8'h00
`define ADV7185_REGISTER_38 8'h00
`define ADV7185_REGISTER_39 8'h00
`define ADV7185_REGISTER_3A 8'h00
`define ADV7185_REGISTER_3B 8'h00

`define ADV7185_REGISTER_44 8'h41
`define ADV7185_REGISTER_45 8'hBB

`define ADV7185_REGISTER_F1 8'hEF
`define ADV7185_REGISTER_F2 8'h80


module adv7185init (reset, clock_27mhz, source, tv_in_reset_b,
                tv_in_i2c_clock, tv_in_i2c_data);

   input reset;
   input clock_27mhz;
   output tv_in_reset_b; // Reset signal to ADV7185
   output tv_in_i2c_clock; // I2C clock output to ADV7185
   output tv_in_i2c_data; // I2C data line to ADV7185
   input source; // 0: composite, 1: s-video

   initial begin
      $display("ADV7185 Initialization values:");
      $display("  Register 0:  0x%X", `ADV7185_REGISTER_0);
      $display("  Register 1:  0x%X", `ADV7185_REGISTER_1);
      $display("  Register 2:  0x%X", `ADV7185_REGISTER_2);
      $display("  Register 3:  0x%X", `ADV7185_REGISTER_3);
      $display("  Register 4:  0x%X", `ADV7185_REGISTER_4);
      $display("  Register 5:  0x%X", `ADV7185_REGISTER_5);
      $display("  Register 7:  0x%X", `ADV7185_REGISTER_7);
      $display("  Register 8:  0x%X", `ADV7185_REGISTER_8);
      $display("  Register 9:  0x%X", `ADV7185_REGISTER_9);
      $display("  Register A:  0x%X", `ADV7185_REGISTER_A);
      $display("  Register B:  0x%X", `ADV7185_REGISTER_B);
      $display("  Register C:  0x%X", `ADV7185_REGISTER_C);
      $display("  Register D:  0x%X", `ADV7185_REGISTER_D);
      $display("  Register E:  0x%X", `ADV7185_REGISTER_E);
      $display("  Register F:  0x%X", `ADV7185_REGISTER_F);
      $display("  Register 33: 0x%X", `ADV7185_REGISTER_33);
   end

   //
   // Generate a 1MHz for the I2C driver (resulting I2C clock rate is 250kHz)
   //

   reg [7:0] clk_div_count, reset_count;
   reg clock_slow;
   wire reset_slow;
```

```
   initial
     begin
      clk_div_count <= 8'h00;
      // synthesis attribute init of clk_div_count is "00";
      clock_slow <= 1'b0;
      // synthesis attribute init of clock_slow is "0";
     end

   always @(posedge clock_27mhz)
     if (clk_div_count == 26)
       begin
        clock_slow <= ~clock_slow;
        clk_div_count <= 0;
       end
     else
       clk_div_count <= clk_div_count+1;

   always @(posedge clock_27mhz)
     if (reset)
       reset_count <= 100;
     else
       reset_count <= (reset_count==0) ? 0 : reset_count-1;

   assign reset_slow = reset_count != 0;

   //
   // I2C driver
   //

   reg load;
   reg [7:0] data;
   wire ack, idle;

   i2c i2c(.reset(reset_slow), .clock4x(clock_slow), .data(data),
.load(load),
         .ack(ack), .idle(idle), .scl(tv_in_i2c_clock),
         .sda(tv_in_i2c_data));

   //
   // State machine
   //

   reg [7:0] state;
   reg tv_in_reset_b;
   reg old_source;

   always @(posedge clock_slow)
      if (reset_slow)
      begin
         state <= 0;
         load <= 0;
         tv_in_reset_b <= 0;
         old_source <= 0;
      end
      else
      case (state)
        8'h00:
```

```
  begin
     // Assert reset
     load <= 1'b0;
     tv_in_reset_b <= 1'b0;
     if (!ack)
     state <= state+1;
  end
8'h01:
  state <= state+1;
8'h02:
  begin
     // Release reset
     tv_in_reset_b <= 1'b1;
     state <= state+1;
              end
8'h03:
  begin
     // Send ADV7185 address
     data <= 8'h8A;
     load <= 1'b1;
     if (ack)
     state <= state+1;
  end
8'h04:
  begin
     // Send subaddress of first register
     data <= 8'h00;
     if (ack)
     state <= state+1;
  end
8'h05:
  begin
     // Write to register 0
     data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
     if (ack)
     state <= state+1;
  end
8'h06:
  begin
     // Write to register 1
     data <= `ADV7185_REGISTER_1;
     if (ack)
     state <= state+1;
  end
8'h07:
  begin
     // Write to register 2
     data <= `ADV7185_REGISTER_2;
     if (ack)
     state <= state+1;
  end
8'h08:
  begin
     // Write to register 3
     data <= `ADV7185_REGISTER_3;
     if (ack)
     state <= state+1;
```

```
  end
8'h09:
  begin
     // Write to register 4
     data <= `ADV7185_REGISTER_4;
     if (ack)
     state <= state+1;
  end
8'h0A:
  begin
     // Write to register 5
     data <= `ADV7185_REGISTER_5;
     if (ack)
     state <= state+1;
  end
8'h0B:
  begin
     // Write to register 6
     data <= 8'h00; // Reserved register, write all zeros
     if (ack)
     state <= state+1;
  end
8'h0C:
  begin
     // Write to register 7
     data <= `ADV7185_REGISTER_7;
     if (ack)
     state <= state+1;
  end
8'h0D:
  begin
     // Write to register 8
     data <= `ADV7185_REGISTER_8;
     if (ack)
     state <= state+1;
  end
8'h0E:
  begin
     // Write to register 9
     data <= `ADV7185_REGISTER_9;
     if (ack)
     state <= state+1;
  end
8'h0F: begin
   // Write to register A
   data <= `ADV7185_REGISTER_A;
 if (ack)
   state <= state+1;
end
8'h10:
  begin
     // Write to register B
     data <= `ADV7185_REGISTER_B;
     if (ack)
     state <= state+1;
  end
8'h11:
```

```
  begin
     // Write to register C
     data <= `ADV7185_REGISTER_C;
     if (ack)
     state <= state+1;
  end
8'h12:
  begin
     // Write to register D
     data <= `ADV7185_REGISTER_D;
     if (ack)
     state <= state+1;
  end
8'h13:
  begin
     // Write to register E
     data <= `ADV7185_REGISTER_E;
     if (ack)
     state <= state+1;
  end
8'h14:
  begin
     // Write to register F
     data <= `ADV7185_REGISTER_F;
     if (ack)
     state <= state+1;
  end
8'h15:
  begin
     // Wait for I2C transmitter to finish
     load <= 1'b0;
     if (idle)
     state <= state+1;
  end
8'h16:
  begin
     // Write address
     data <= 8'h8A;
     load <= 1'b1;
     if (ack)
     state <= state+1;
  end
8'h17:
  begin
     data <= 8'h33;
     if (ack)
     state <= state+1;
  end
8'h18:
  begin
     data <= `ADV7185_REGISTER_33;
     if (ack)
     state <= state+1;
  end
8'h19:
  begin
     load <= 1'b0;
```

```
      if (idle)
      state <= state+1;
  end

8'h1A: begin
   data <= 8'h8A;
   load <= 1'b1;
   if (ack)
     state <= state+1;
end
8'h1B:
  begin
     data <= 8'h33;
     if (ack)
     state <= state+1;
  end
8'h1C:
  begin
     load <= 1'b0;
     if (idle)
     state <= state+1;
  end
8'h1D:
  begin
     load <= 1'b1;
     data <= 8'h8B;
     if (ack)
     state <= state+1;
  end
8'h1E:
  begin
     data <= 8'hFF;
     if (ack)
     state <= state+1;
  end
8'h1F:
  begin
     load <= 1'b0;
     if (idle)
     state <= state+1;
  end
8'h20:
  begin
     // Idle
     if (old_source != source) state <= state+1;
     old_source <= source;
  end
8'h21: begin
   // Send ADV7185 address
   data <= 8'h8A;
   load <= 1'b1;
   if (ack) state <= state+1;
end
8'h22: begin
   // Send subaddress of register 0
   data <= 8'h00;
   if (ack) state <= state+1;
```

```
        end
        8'h23: begin
           // Write to register 0
           data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
           if (ack) state <= state+1;
        end
        8'h24: begin
           // Wait for I2C transmitter to finish
           load <= 1'b0;
           if (idle) state <= 8'h20;
        end
       endcase

endmodule

// i2c module for use with the ADV7185

module i2c (reset, clock4x, data, load, idle, ack, scl, sda);

   input reset;
   input clock4x;
   input [7:0] data;
   input load;
   output ack;
   output idle;
   output scl;
   output sda;

   reg [7:0] ldata;
   reg ack, idle;
   reg scl;
   reg sdai;

   reg [7:0] state;

   assign sda = sdai ? 1'bZ : 1'b0;

   always @(posedge clock4x)
     if (reset)
       begin
        state <= 0;
        ack <= 0;
       end
     else
       case (state)
       8'h00: // idle
         begin
            scl <= 1'b1;
            sdai <= 1'b1;
            ack <= 1'b0;
            idle <= 1'b1;
            if (load)
            begin
               ldata <= data;
               ack <= 1'b1;
               state <= state+1;
            end
```

```
  end
8'h01: // Start
  begin
     ack <= 1'b0;
     idle <= 1'b0;
     sdai <= 1'b0;
     state <= state+1;
  end
8'h02:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h03: // Send bit 7
  begin
     ack <= 1'b0;
     sdai <= ldata[7];
     state <= state+1;
  end
8'h04:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h05:
  begin
     state <= state+1;
  end
8'h06:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h07:
  begin
     sdai <= ldata[6];
     state <= state+1;
  end
8'h08:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h09:
  begin
     state <= state+1;
  end
8'h0A:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h0B:
  begin
     sdai <= ldata[5];
     state <= state+1;
  end
```

```
8'h0C:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h0D:
  begin
     state <= state+1;
  end
8'h0E:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h0F:
  begin
     sdai <= ldata[4];
     state <= state+1;
  end
8'h10:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h11:
  begin
     state <= state+1;
  end
8'h12:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h13:
  begin
     sdai <= ldata[3];
     state <= state+1;
  end
8'h14:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h15:
  begin
     state <= state+1;
  end
8'h16:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h17:
  begin
     sdai <= ldata[2];
     state <= state+1;
  end
```

```
8'h18:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h19:
  begin
     state <= state+1;
  end
8'h1A:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h1B:
  begin
     sdai <= ldata[1];
     state <= state+1;
  end
8'h1C:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h1D:
  begin
     state <= state+1;
  end
8'h1E:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h1F:
  begin
     sdai <= ldata[0];
     state <= state+1;
  end
8'h20:
  begin
     scl <= 1'b1;
     state <= state+1;
  end
8'h21:
  begin
     state <= state+1;
  end
8'h22:
  begin
     scl <= 1'b0;
     state <= state+1;
  end
8'h23: // Acknowledge bit
  begin
     state <= state+1;
  end
8'h24:
```

```
          begin
             scl <= 1'b1;
             state <= state+1;
          end
        8'h25:
          begin
             state <= state+1;
          end
        8'h26:
          begin
             scl <= 1'b0;
             if (load)
             begin
                ldata <= data;
                ack <= 1'b1;
                state <= 3;
             end
             else
             state <= state+1;
          end
        8'h27:
          begin
             sdai <= 1'b0;
             state <= state+1;
          end
        8'h28:
          begin
             scl <= 1'b1;
             state <= state+1;
          end
        8'h29:
          begin
             sdai <= 1'b1;
             state <= 0;
          end
        endcase

endmodule
```

C ycrcb2rgb.v

```
module YCrCb2RGB ( R, G, B, clk, rst, Y, Cr, Cb );

output [7:0]  R, G, B;

input clk,rst;
input[9:0] Y, Cr, Cb;

wire [7:0] R,G,B;
reg [20:0] R_int,G_int,B_int,X_int,A_int,B1_int,B2_int,C_int;
reg [9:0] const1,const2,const3,const4,const5;
reg[9:0] Y_reg, Cr_reg, Cb_reg;

//registering constants
always @ (posedge clk)
begin
```

```
 const1 = 10'b 0100101010; //1.164 = 01.00101010
 const2 = 10'b 0110011000; //1.596 = 01.10011000
 const3 = 10'b 0011010000; //0.813 = 00.11010000
 const4 = 10'b 0001100100; //0.392 = 00.01100100
 const5 = 10'b 1000000100; //2.017 = 10.00000100
end

always @ (posedge clk or posedge rst)
   if (rst)
      begin
      Y_reg <= 0; Cr_reg <= 0; Cb_reg <= 0;
      end
   else
      begin
        Y_reg <= Y; Cr_reg <= Cr; Cb_reg <= Cb;
      end

always @ (posedge clk or posedge rst)
   if (rst)
      begin
       A_int <= 0; B1_int <= 0; B2_int <= 0; C_int <= 0; X_int <= 0;
      end
   else
     begin
     X_int <= (const1 * (Y_reg - 'd64)) ;
     A_int <= (const2 * (Cr_reg - 'd512));
     B1_int <= (const3 * (Cr_reg - 'd512));
     B2_int <= (const4 * (Cb_reg - 'd512));
     C_int <= (const5 * (Cb_reg - 'd512));
     end

always @ (posedge clk or posedge rst)
   if (rst)
      begin
       R_int <= 0; G_int <= 0; B_int <= 0;
      end
   else
     begin
     R_int <= X_int + A_int;
     G_int <= X_int - B1_int - B2_int;
     B_int <= X_int + C_int;
     end

assign R = (R_int[20]) ? 0 : (R_int[19:18] == 2'b0) ? R_int[17:10] :
8'b11111111;
assign G = (G_int[20]) ? 0 : (G_int[19:18] == 2'b0) ? G_int[17:10] :
8'b11111111;
assign B = (B_int[20]) ? 0 : (B_int[19:18] == 2'b0) ? B_int[17:10] :
8'b11111111;

endmodule
```

D ntsc2zbt.v

```
//
// File:   ntsc2zbt.v
```

```
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Example for MIT 6.111 labkit showing how to prepare NTSC data
// (from Javier's decoder) to be loaded into the ZBT RAM for video
// display.
//
// The ZBT memory is 36 bits wide; we only use 32 bits of this, to
// store 4 bytes of black-and-white intensity data from the NTSC
// video input.

//////////////////////////////////////////////////////////////////////////
// Prepare data and address values to fill ZBT memory with NTSC data

module ntsc_to_zbt(clk, vclk, fvh, dv, din, ntsc_addr, ntsc_data, ntsc_we,
sw);

   input            clk;            // system clock
   input         vclk;        // video clock from camera
   input  [2:0]  fvh;
   input            dv;
   //input[7:0]  din;         // b&w

      input  [17:0] din;
   output [18:0] ntsc_addr;
   output [35:0] ntsc_data;
   output           ntsc_we;  // write enable for NTSC data
   input            sw;             // switch which determines mode (for
debugging)

   //parameter     COL_START = 10'd30;
   //parameter     ROW_START = 10'd30;

      parameter    COL_START = 10'd00;
   parameter       ROW_START = 10'd00;

   // here put the luminance data from the ntsc decoder into the ram
   // this is for 1024 x 768 XGA display

   reg [9:0]  col = 0;
   reg [9:0]  row = 0;
   //reg [7:0]vdata = 0; // b&w

      reg [17:0] vdata = 0;
   reg              vwe;
   reg              old_dv;
   reg              old_frame;      // frames are even / odd interlaced
   reg              even_odd; // decode interlaced frame to this wire

   wire frame = fvh[2];
   wire frame_edge = frame & ~old_frame;

   always @ (posedge vclk) begin
            old_dv <= dv;
            vwe <= dv && !fvh[2] & ~old_dv; // if data valid, write it
            old_frame <= frame;
            even_odd = frame_edge ? ~even_odd : even_odd;
```

```
            if (!fvh[2]) begin
                  col <= fvh[0] ? COL_START :

                        //(!fvh[2] && !fvh[1] && dv && (col < 1024)) ? col +
1 : col;

                        (!fvh[2] && !fvh[1] && dv && (col < 800)) ? col + 1 :
col;
                  row <= fvh[1] ? ROW_START :

                        //(!fvh[2] && fvh[0] && (row < 768)) ? row + 1 : row;

                        (!fvh[2] && fvh[0] && (row < 600)) ? row + 1 : row;
                  //vdata <= (dv && !fvh[2]) ? din : vdata; // b&w

                  vdata <= (dv && !fvh[2]) ? din : vdata;
            end
   end

   // synchronize with system clock

   reg [9:0]  x[1:0],y[1:0];
   //reg [7:0]data[1:0]; // b&w

      reg [17:0] data[1:0];
   reg        we[1:0];
   reg           eo[1:0];

   always @(posedge clk) begin
            {x[1],x[0]} <= {x[0],col};
            {y[1],y[0]} <= {y[0],row};
            {data[1],data[0]} <= {data[0],vdata};
            {we[1],we[0]} <= {we[0],vwe};
            {eo[1],eo[0]} <= {eo[0],even_odd};
   end

   // edge detection on write enable signal

   reg old_we;
   wire we_edge = we[1] & ~old_we;
   always @(posedge clk) old_we <= we[1];

   // shift each set of four bytes into a large register for the ZBT

      // reg [31:0] mydata; // b&w

   reg [35:0] mydata;
   always @(posedge clk)

            if (we_edge)

                  //mydata <= { mydata[23:0], data[1] }; // b&w

                  mydata <= { mydata[17:0], data[1] };

   // compute address to store data in
```

```
   //wire [18:0] myaddr = {1'b0, y[1][8:0], eo[1], x[1][9:2]}; // b&w

      wire [18:0] myaddr = {y[1][8:0], eo[1], x[1][9:1]};

   // alternate (256x192) image data and address
   //wire [31:0] mydata2 = {data[1],data[1],data[1],data[1]}; //b&w

      wire [35:0] mydata2 = {data[1],data[1]};
   //wire [18:0] myaddr2 = {1'b0, y[1][8:0], eo[1], x[1][7:0]}; // b&w

      wire [18:0] myaddr2 = {y[1][8:0], eo[1], x[1][8:0]};

   // update the output address and data only when four bytes ready

   reg [18:0] ntsc_addr;
   reg [35:0] ntsc_data;
   //wire ntsc_we = sw ? we_edge : (we_edge & (x[1][1:0]==2'b00)); // b&w

      wire ntsc_we = sw ? we_edge : (we_edge & (x[1][0]==0));

   always @(posedge clk)

            if ( ntsc_we ) begin
                  ntsc_addr <= sw ? myaddr2 : myaddr; // normal and expanded
modes
                  //ntsc_data <= sw ? {4'b0,mydata2} : {4'b0,mydata}; // b&w

                  ntsc_data <= sw ? mydata2 : mydata;
      end

endmodule // ntsc_to_zbt
```

E comcalc.v

```
module comcalc(
      input               reset,
   input         clk, // video clock from camera
      input [18:0]  ntsc_addr, // {vcount[9:0], hcount[9:1]}
      input [35:0]  ntsc_data,
      input         ntsc_we,
      input [23:0]  threshold,
      output reg [10:0] com_1_x,
      output reg [9:0]  com_1_y,
      output reg [10:0] com_2_x,
      output reg [9:0]  com_2_y,
      output reg comvalid
      );

      // Every cycle we read in two
      // pixels, because that is how
      // the ntsc2zbt outputs data.
      // We use the ntsc2zbt data
      // because it has been synchronized
      // with the clock.

      ////////////////////////
```

```
// incoming pixel information
//////////////////////
reg [18:0] last_ntsc_addr;
reg [10:0] pixel_1_x; // use 11 bits for expansion to 1024x768
reg [17:0] pixel_1_data;
reg [9:0]  pixel_1_y;
reg [10:0] pixel_2_x;
reg [9:0]  pixel_2_y;
reg [17:0] pixel_2_data;

//////////////////////
// center of mass calculation vars
//////////////////////
wire [10:0] quotient_1_x; // com 1
wire [9:0]  quotient_1_y; // com 1
reg  [18:0] count_1; // number of pixels accepted
reg  [29:0] sum_1_x; // (800*801/2) * 600 // com 1
reg  [29:0] sum_1_y; // (600*601/2) * 800 // com 2

wire [10:0] quotient_2_x;
wire [9:0]  quotient_2_y;
reg  [18:0] count_2;
reg  [29:0] sum_2_x;
reg  [29:0] sum_2_y;

always @ (posedge clk) begin
      if (reset) begin
            /////////////////
            // reset state
            /////////////////

            last_ntsc_addr <= 19'h00000;

            // pixel information
            pixel_1_x <= 11'd0;
            pixel_1_y <= 10'd0;
            pixel_1_data <= 18'd0;
            pixel_2_x <= 11'd0;
            pixel_2_y <= 10'd0;
            pixel_2_data <= 18'd0;

            // com variables
            com_1_x <= 11'd0;
            com_1_y <= 10'd0;
            count_1 <= 19'd0;
            com_2_x <= 11'd0;
            com_2_y <= 10'd0;
            count_2 <= 19'd0;

            comvalid <= 0;

      end
      else if (ntsc_we) begin
            last_ntsc_addr <= ntsc_addr;
            if (ntsc_addr != last_ntsc_addr) begin
                  // we are looking at new pixels
```

```
                        //////////////////////
                        // extract incoming pixel info
                        //////////////////////
                        pixel_1_x <= {ntsc_addr[8:0],1'b0};
                        pixel_1_y <= ntsc_addr[18:9];
                        pixel_1_data <= ntsc_data[17+18:0+18];
                        pixel_2_x <= {ntsc_addr[8:0],1'b1};
                        pixel_2_y <= ntsc_addr[18:9];
                        pixel_2_data <= ntsc_data[17:0];

                        // doesn't write to (0,0), so choose some arbitrary
                        // pixel 100,100 inside display region
                        if ((pixel_1_x == 11'd100) && (pixel_1_y == 10'd100))
begin
                              //////////////////////
                              // new frame
                              //////////////////////
                              if (count_1 > 19'd10) begin
                                    // don't move because of noise
                                    com_1_x <= quotient_1_x[10:0];
                                    com_1_y <= quotient_1_y[9:0];
                                    com_2_x <= quotient_2_x[10:0];
                                    com_2_y <= quotient_2_y[9:0];
                              end

                              sum_1_x <= 30'd0;
                              sum_1_y <= 30'd0;
                              count_1 <= 19'd0;

                              sum_2_x <= 30'd0;
                              sum_2_y <= 30'd0;
                              count_2 <= 19'd0;

                              comvalid <= 1;

                        end
                        else begin
                              //////////////////////
                              // add pixel to sum if hasn't been rejected
                              //////////////////////

                              comvalid <= 0;

                              if ((pixel_1_data != 17'd0) && (pixel_2_data !=
17'd0)) begin
                                    // require two non-black pixels in a row
to avoid noise

                                    if ({pixel_1_data[17:12]} >
threshold[23:18]) begin // detect red
                                          sum_1_x <= sum_1_x + pixel_1_x +
pixel_2_x;
                                          sum_1_y <= sum_1_y + pixel_1_y +
pixel_2_y;
                                          count_1 <= count_1 + 2;
                                    end
```

```
                                    else if ({pixel_1_data[11:6]} >
threshold[15:10]) begin
                                          sum_2_x <= sum_2_x + pixel_1_x +
pixel_2_x;
                                          sum_2_y <= sum_2_y + pixel_1_y +
pixel_2_y;
                                          count_2 <= count_2 + 2;
                                    end

                              end
                        end

                  end
            end
            else comvalid <= 0;
      end


      // DEBUG:
      /*
      assign quotient_1_x = 400;
      assign quotient_1_y = 200;
      assign quotient_2_x = 300;
      assign quotient_2_y = 400;
      */

      divider_sean xdiv1(
            .clk(clk),
            .dividend(sum_1_x[27:0]),
            .divisor(count_1[18:0]),
            .quotient(quotient_1_x),
            .remainder(),
            .rfd()
            );

      divider_sean ydiv1(
            .clk(clk),
            .dividend(sum_1_y[27:0]),
            .divisor(count_1[18:0]),
            .quotient(quotient_1_y),
            .remainder(),
            .rfd()
            );

      divider_sean xdiv2(
            .clk(clk),
            .dividend(sum_2_x[27:0]),
            .divisor(count_2[18:0]),
            .quotient(quotient_2_x),
            .remainder(),
            .rfd()
            );

      divider_sean ydiv2(
            .clk(clk),
            .dividend(sum_2_y[27:0]),
            .divisor(count_2[18:0]),
```

```
            .quotient(quotient_2_y),
            .remainder(),
            .rfd()
            );

endmodule
```

F divider_sean.v

```
/*****************************************************************************
*     This file is owned and controlled by Xilinx and must be used          *
*     solely for design, simulation, implementation and creation of         *
*     design files limited to Xilinx devices or technologies. Use           *
*     with non-Xilinx devices or technologies is expressly prohibited       *
*     and immediately terminates your license.                              *
*                                                                           *
*     XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"         *
*     SOLELY FOR USE IN DEVELOPING PROGRAMS AND SOLUTIONS FOR               *
*     XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION       *
*     AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION           *
*     OR STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS             *
*     IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,               *
*     AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE      *
*     FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY              *
*     WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE               *
*     IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR        *
*     REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF       *
*     INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS       *
*     FOR A PARTICULAR PURPOSE.                                             *
*                                                                           *
*     Xilinx products are not intended for use in life support              *
*     appliances, devices, or systems. Use in such applications are         *
*     expressly prohibited.                                                 *
*                                                                           *
*     (c) Copyright 1995-2006 Xilinx, Inc.                                  *
*     All rights reserved.                                                  *
*****************************************************************************/
// The synopsys directives "translate_off/translate_on" specified below are
// supported by XST, FPGA Compiler II, Mentor Graphics and Synplicity
synthesis
// tools. Ensure they are correct for your synthesis tool(s).

// You must compile the wrapper file divider_sean.v when simulating
// the core, divider_sean. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library. For detailed
// instructions, please refer to the "CORE Generator Help".

`timescale 1ns/1ps

module divider_sean(
      clk,
      dividend,
      divisor,
      quotient,
      remainder,
      rfd);
```

```
input clk;
input [27 : 0] dividend;
input [18 : 0] divisor;
output [27 : 0] quotient;
output [18 : 0] remainder;
output rfd;

// synopsys translate_off

      DIV_GEN_V1_0 #(
            1,    // algorithm_type
            0,    // bias
            0,    // c_has_aclr
            0,    // c_has_ce
            0,    // c_has_sclr
            0,    // c_sync_enable
            1,    // divclk_sel
            28,   // dividend_width
            19,   // divisor_width
            8,    // exponent_width
            0,    // fractional_b
            19,   // fractional_width
            1,    // latency
            8,    // mantissa_width
            0)    // signed_b
      inst (
            .CLK(clk),
            .DIVIDEND(dividend),
            .DIVISOR(divisor),
            .QUOTIENT(quotient),
            .REMAINDER(remainder),
            .RFD(rfd),
            .CE(),
            .ACLR(),
            .SCLR(),
            .DIVIDEND_MANTISSA(),
            .DIVIDEND_SIGN(),
            .DIVIDEND_EXPONENT(),
            .DIVISOR_MANTISSA(),
            .DIVISOR_SIGN(),
            .DIVISOR_EXPONENT(),
            .QUOTIENT_MANTISSA(),
            .QUOTIENT_SIGN(),
            .QUOTIENT_EXPONENT(),
            .OVERFLOW(),
            .UNDERFLOW());


// synopsys translate_on

// FPGA Express black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of divider_sean is "true"

// XST black box declaration
```

```
// box_type "black_box"
// synthesis attribute box_type of divider_sean is "black_box"

endmodule
```

G smoother.v

```
module smoother
      #(parameter HISTORY=16, LOGHISTORY=4)
      (
      input wire reset,
      input wire clk,
      input wire [10:0] datavalue,
      input wire datavalid,
      output reg [10:0] smooth
      );

      // This modules averages the data
      // over the last 16 entries.

      reg [10:0] samples [HISTORY-1:0]; // 16, 11 bit registers
      reg [LOGHISTORY-1:0]  pointer; // points into circular buffer
      reg [10+LOGHISTORY:0] sum;
      reg [LOGHISTORY-1:0]  datacounter; // how many valid data entries
inputted
      reg delay; // for delaying one clock cycle

      always @ (posedge clk) begin
            if (reset) begin
                  pointer <= 0;
                  sum <= 0;
                  datacounter <= 0;
                  delay <= 0;
            end
            else begin
                  if (datavalid) begin
                        // Only work on datavalid.  This is because you may
only
                        // get valid data every X number of clock cycles.

                        pointer <= pointer + 1;
                        samples[pointer] <= datavalue;
                        smooth <= sum / HISTORY;
                        if (delay == 1'b1) begin
                              // really want datacounter == all 1's plus 1.
                              // Only then will entire circular buffer fill.

                              // need datacounter, otherwise you'll get a
                              // horrendous initial value for sum and
                              // will have this value locked in place.

                              sum <= sum + datavalue - samples[pointer];

                              //sum <= sum + samples[(pointer+15)& 4'hF];
                              // NOTE: cannot do pointer-1, because this
actually
```

```
                              // makes a new imaginary wire, which then can
take on
                              // a width that is larger than 4 bits...
                        end
                        else begin
                              sum <= sum + datavalue;

                              datacounter <= datacounter + 1;
                              if (&(datacounter) == 1'b1) delay <= 1;
                        end
                  end
            end
      end

endmodule
```

H palette_gen.v

```
module palette_gen(
      input wire clk,
      input wire [10:0] hcount,
   input wire [9:0]  vcount,
      input wire [24:0] curr_rgb,
      input wire [4:0]  saturation,
      input wire [10:0] com_1_x,
      input wire [9:0]  com_1_y,
      input wire [10:0] com_2_x,
      input wire [9:0]  com_2_y,
      input wire [1:0]  brush_select,

      input wire [10:0] TOTALWIDTH,
      input wire [9:0]  TOTALHEIGHT,
      input wire [9:0] LEFTCOL_WIDTH,
      input wire [9:0] PALETTE_COLOR_WIDTH,
      input wire [9:0] RED_LEFT,
      input wire [9:0] RED_TOP,
      input wire [9:0] YEL_LEFT,
      input wire [9:0] YEL_TOP,
      input wire [9:0] BLUE_LEFT,
      input wire [9:0] BLUE_TOP,
      input wire [9:0] WHITE_LEFT,
      input wire [9:0] WHITE_TOP,
      input wire [9:0] FILL_TOP,
      input wire [9:0] FILL_LEFT,
      input wire [9:0] FILL_WIDTH,
      input wire [9:0] SQ_TOP,
      input wire [9:0] SQ_LEFT,
      input wire [9:0] SQ_WIDTH,
      input wire [9:0] SPECK_TOP,
      input wire [9:0] SPECK_LEFT,
      input wire [9:0] SPECK_WIDTH,
      input wire [9:0] LINE_TOP,
      input wire [9:0] LINE_LEFT,
      input wire [9:0] LINE_WIDTH,
      input wire [9:0] SAT_LEFT,
      input wire [9:0] SAT_TOP,
```

```
    input wire [9:0] SAT_HEIGHT,
    input wire [9:0] SAT_BAR_WIDTH,
    input wire [9:0] CURR_LEFT,
    input wire [9:0] CURR_TOP,
    input wire [9:0] CANVASBORDER,
    input wire [9:0] XHAIR_RAD,

output reg [23:0] pixel
    );

    parameter SEL_ICON_WIDTH = 5;

    //parameter com_1_x = 400;
    //parameter com_1_y = 300;
    //parameter com_2_x = 200;
    //parameter com_2_y = 400;
    /*
    parameter TOTALWIDTH = 800;
    parameter TOTALHEIGHT = 600;
    parameter LEFTCOL_WIDTH = 110;
    parameter PALETTE_COLOR_WIDTH = 80;
    parameter RED_LEFT   = 15;
    parameter RED_TOP    = 10;
    parameter YEL_LEFT   = 15;
    parameter YEL_TOP    = 10 + PALETTE_COLOR_WIDTH + RED_TOP;
    parameter BLUE_LEFT  = 15;
    parameter BLUE_TOP   = 10 + PALETTE_COLOR_WIDTH + YEL_TOP;
    //parameter GREEN_LEFT = 15;
    //parameter GREEN_TOP  = 15 + PALETTE_COLOR_WIDTH + BLUE_TOP;
    parameter WHITE_LEFT = 15;
    //parameter WHITE_TOP  = 15 + PALETTE_COLOR_WIDTH + GREEN_TOP;
    parameter WHITE_TOP  = 10 + PALETTE_COLOR_WIDTH + BLUE_TOP;

    parameter FILL_TOP = 20 + PALETTE_COLOR_WIDTH + WHITE_TOP;
    parameter FILL_LEFT = 15;
    parameter FILL_WIDTH = 38;

    parameter SQ_TOP = 20 + PALETTE_COLOR_WIDTH + WHITE_TOP;
    parameter SQ_LEFT = FILL_LEFT + FILL_WIDTH + 4;
    parameter SQ_WIDTH = 38;

    parameter SPECK_TOP = 4 + FILL_TOP + FILL_WIDTH;
    parameter SPECK_LEFT = 15;
    parameter SPECK_WIDTH = 38;

    parameter SAT_LEFT = 15;
    parameter SAT_TOP = 10 + SPECK_WIDTH + SPECK_TOP;
    parameter SAT_HEIGHT = 20;
    parameter SAT_BAR_WIDTH = PALETTE_COLOR_WIDTH/5'b11111;

    parameter CURR_LEFT  = 15;
    parameter CURR_TOP   = 5 + SAT_HEIGHT + SAT_TOP;

    parameter CANVASBORDER = 30;

    parameter XHAIR_RAD = 8;
    */
```

```
always @(posedge clk) begin
	// xhair
	if (
			(
				(hcount > com_1_x - XHAIR_RAD)
				&& (hcount < com_1_x + XHAIR_RAD)
				&& (vcount == com_1_y)
			)
			||
			(
				(vcount > com_1_y - XHAIR_RAD)
				&& (vcount < com_1_y + XHAIR_RAD)
				&& (hcount == com_1_x)
			)
		) begin
		pixel <= 24'hAA2222;
	end
	else if (
			(
				(hcount > com_2_x - XHAIR_RAD)
				&& (hcount < com_2_x + XHAIR_RAD)
				&& (vcount == com_2_y)
			)
			||
			(
				(vcount > com_2_y - XHAIR_RAD)
				&& (vcount < com_2_y + XHAIR_RAD)
				&& (hcount == com_2_x)
			)
		) begin
		pixel <= 24'h22AA22;
	end
	else if (hcount <= LEFTCOL_WIDTH) begin
		// left column
		if (
			(RED_LEFT < hcount)
			&& (hcount < (RED_LEFT + PALETTE_COLOR_WIDTH))
			&& (RED_TOP < vcount)
			&& (vcount < (RED_TOP + PALETTE_COLOR_WIDTH))
			) begin
			pixel <= 24'hFF0000;
		end
		else if (
			(YEL_LEFT < hcount)
			&& (hcount < (YEL_LEFT + PALETTE_COLOR_WIDTH))
			&& (YEL_TOP < vcount)
			&& (vcount < (YEL_TOP + PALETTE_COLOR_WIDTH))
			) begin
			pixel <= 24'h00FF00;
		end
		else if (
			(BLUE_LEFT < hcount)
			&& (hcount < (BLUE_LEFT + PALETTE_COLOR_WIDTH))
			&& (BLUE_TOP < vcount)
			&& (vcount < (BLUE_TOP + PALETTE_COLOR_WIDTH))
			) begin
```

```
                        pixel <= 24'h0000FF;
                  end
                  else if (
                        (WHITE_LEFT < hcount)
                        && (hcount < (WHITE_LEFT + PALETTE_COLOR_WIDTH))
                        && (WHITE_TOP < vcount)
                        && (vcount < (WHITE_TOP + PALETTE_COLOR_WIDTH))
                        ) begin
                        pixel <= 24'hFFFFFF;
                  end
                  else if (
                        (CURR_LEFT < hcount)
                        && (hcount < (CURR_LEFT + PALETTE_COLOR_WIDTH))
                        && (CURR_TOP < vcount)
                        && (vcount < (CURR_TOP + PALETTE_COLOR_WIDTH))
                        ) begin
                        pixel <= curr_rgb;
                  end
                  else if (
                        (SAT_LEFT < hcount)
                        && (hcount <= (SAT_LEFT + (saturation *
SAT_BAR_WIDTH)))
                        && (SAT_TOP < vcount)
                        && (vcount < (SAT_TOP + SAT_HEIGHT))
                        ) begin
                        pixel <= curr_rgb;
                  end
                  else if (
                        ((SAT_LEFT + (saturation * SAT_BAR_WIDTH)) < hcount)
                        && (hcount < (SAT_LEFT + PALETTE_COLOR_WIDTH))
                        && (SAT_TOP < vcount)
                        && (vcount < (SAT_TOP + SAT_HEIGHT))
                        ) begin
                        pixel <= 24'h111111;
                  end
                  else if (
                        (FILL_LEFT < hcount)
                        && (hcount < (FILL_LEFT + FILL_WIDTH))
                        && (FILL_TOP < vcount)
                        && (vcount < (FILL_TOP + FILL_WIDTH))
                        ) begin
                        if (
                              // brush selected icon
                              (brush_select == 2'b00)
                              && (hcount < (FILL_LEFT + SEL_ICON_WIDTH))
                              && (vcount < (FILL_TOP + SEL_ICON_WIDTH))
                              ) begin
                              pixel <= 24'b111111;
                        end
                        else pixel <= curr_rgb;
                  end
                  else if (
                        (SQ_LEFT < hcount)
                        && (hcount < (SQ_LEFT + SQ_WIDTH))
                        && (SQ_TOP < vcount)
                        && (vcount < (SQ_TOP + SQ_WIDTH))
                        ) begin
```

```
                        if (
                              // brush selected icon
                              (brush_select == 2'b01)
                              && (hcount < (SQ_LEFT + SEL_ICON_WIDTH))
                              && (vcount < (SQ_TOP + SEL_ICON_WIDTH))
                              ) begin
                              pixel <= 24'b111111;
                        end
                        else if (
                              (SQ_LEFT + 10 < hcount)
                              && (hcount < (SQ_LEFT + SQ_WIDTH - 10))
                              && (SQ_TOP + 10 < vcount)
                              && (vcount < (SQ_TOP + SQ_WIDTH - 10))
                              ) begin
                              pixel <= curr_rgb;
                        end
                        else begin
                              pixel <= 24'hFFFFFF;
                        end
                  end
                  else if (
                        (SPECK_LEFT < hcount)
                        && (hcount < (SPECK_LEFT + SPECK_WIDTH))
                        && (SPECK_TOP < vcount)
                        && (vcount < (SPECK_TOP + SPECK_WIDTH))
                        ) begin
                        if (
                              // brush selected icon
                              (brush_select == 2'b10)
                              && (hcount < (SPECK_LEFT + SEL_ICON_WIDTH))
                              && (vcount < (SPECK_TOP + SEL_ICON_WIDTH))
                              ) begin
                              pixel <= 24'b111111;
                        end
                        else if (hcount[0] ^ vcount[0] == 1) pixel <=
curr_rgb;
                        else pixel <= 24'hFFFFFF;
                  end
                  else if (
                        (LINE_LEFT < hcount)
                        && (hcount < (LINE_LEFT + LINE_WIDTH))
                        && (LINE_TOP < vcount)
                        && (vcount < (LINE_TOP + LINE_WIDTH))
                        ) begin
                        if (
                              // brush selected icon
                              (brush_select == 2'b11)
                              && (hcount < (LINE_LEFT + SEL_ICON_WIDTH))
                              && (vcount < (LINE_TOP + SEL_ICON_WIDTH))
                              ) begin
                              pixel <= 24'b111111;
                        end
                        else if (hcount == (2 * LINE_LEFT + LINE_WIDTH)/2)
pixel <= curr_rgb;
                        else pixel <= 24'hFFFFFF;
                  end
                  else begin
```

```
                        if (vcount > (WHITE_TOP + PALETTE_COLOR_WIDTH + 10))
begin
                              // brush section bg
                              pixel <= 24'hCCCCCC;
                        end
                        else begin
                              // palette section bg
                              pixel <= 24'hAAAAAA;
                        end
                  end

            end
            else if (hcount <= TOTALWIDTH) begin
                  // main body
                  if (
                        (CANVASBORDER < vcount)
                        && (vcount < (TOTALHEIGHT - CANVASBORDER))
                        && ((LEFTCOL_WIDTH + CANVASBORDER) < hcount)
                        && (hcount < (TOTALWIDTH - CANVASBORDER))
                        ) begin
                        pixel <= 24'h000000;
                  end
                  else begin
                        pixel <= 24'h555555;
                  end
            end
            else begin
                  pixel <= 24'hAAAAAA; // make same as palette column bg,
because 1 cycle delay wrap around
            end
      end

endmodule
```

I intention_v2.v

```
module intention_v2(
      input wire reset,
      input wire clk,
      input wire button,
      input wire [10:0] com_1_x,
      input wire [9:0]  com_1_y,
      input wire [10:0] TOTALWIDTH,
      input wire [9:0]  TOTALHEIGHT,
      input wire [9:0] LEFTCOL_WIDTH,
      input wire [9:0] PALETTE_COLOR_WIDTH,
      input wire [9:0] RED_LEFT,
      input wire [9:0] RED_TOP,
      input wire [9:0] YEL_LEFT,
      input wire [9:0] YEL_TOP,
      input wire [9:0] BLUE_LEFT,
      input wire [9:0] BLUE_TOP,
      input wire [9:0] WHITE_LEFT,
      input wire [9:0] WHITE_TOP,
      input wire [9:0] FILL_TOP,
      input wire [9:0] FILL_LEFT,
```

```
input wire [9:0] FILL_WIDTH,
input wire [9:0] SQ_TOP,
input wire [9:0] SQ_LEFT,
input wire [9:0] SQ_WIDTH,
input wire [9:0] SPECK_TOP,
input wire [9:0] SPECK_LEFT,
input wire [9:0] SPECK_WIDTH,
input wire [9:0] LINE_TOP,
input wire [9:0] LINE_LEFT,
input wire [9:0] LINE_WIDTH,
input wire [9:0] SAT_LEFT,
input wire [9:0] SAT_TOP,
input wire [9:0] SAT_HEIGHT,
input wire [9:0] SAT_BAR_WIDTH,
input wire [9:0] CURR_LEFT,
input wire [9:0] CURR_TOP,
input wire [9:0] CANVASBORDER,
input wire [9:0] XHAIR_RAD,

output reg [1:0]  brush_select,
output reg [2:0]  absorb_color,
output reg absorb_enable,
output reg paint_enable,
output reg fill_enable
);

always @(posedge clk) begin
      if (reset) begin
            brush_select <= 2'b01;
            absorb_color <= 3'b000;
            absorb_enable <= 0;
            paint_enable <= 0;
            fill_enable <= 0;
      end
      else if (com_1_x <= LEFTCOL_WIDTH) begin
            // left column
            if (!button) begin
                  absorb_color <= 3'b000;
                  absorb_enable <= 0;
                  paint_enable <= 0;
                  fill_enable <= 0;
            end
            else if (
                  (RED_LEFT < com_1_x)
                  && (com_1_x < (RED_LEFT + PALETTE_COLOR_WIDTH))
                  && (RED_TOP < com_1_y)
                  && (com_1_y < (RED_TOP + PALETTE_COLOR_WIDTH))
                  ) begin
                  absorb_color <= 3'b100;
                  absorb_enable <= 1;
                  paint_enable <= 0;
                  fill_enable <= 0;
            end
            else if (
                  (YEL_LEFT < com_1_x)
                  && (com_1_x < (YEL_LEFT + PALETTE_COLOR_WIDTH))
                  && (YEL_TOP < com_1_y)
```

```
      && (com_1_y < (YEL_TOP + PALETTE_COLOR_WIDTH))
      ) begin
      absorb_color <= 3'b010;
      absorb_enable <= 1;
      paint_enable <= 0;
      fill_enable <= 0;
end
else if (
      (BLUE_LEFT < com_1_x)
      && (com_1_x < (BLUE_LEFT + PALETTE_COLOR_WIDTH))
      && (BLUE_TOP < com_1_y)
      && (com_1_y < (BLUE_TOP + PALETTE_COLOR_WIDTH))
      ) begin
      absorb_color <= 3'b001;
      absorb_enable <= 1;
      paint_enable <= 0;
      fill_enable <= 0;
end
else if (
      (WHITE_LEFT < com_1_x)
      && (com_1_x < (WHITE_LEFT + PALETTE_COLOR_WIDTH))
      && (WHITE_TOP < com_1_y)
      && (com_1_y < (WHITE_TOP + PALETTE_COLOR_WIDTH))
      ) begin
      absorb_color <= 3'b111;
      absorb_enable <= 1;
      paint_enable <= 0;
      fill_enable <= 0;
end
else if (
      (CURR_LEFT < com_1_x)
      && (com_1_x < (CURR_LEFT + PALETTE_COLOR_WIDTH))
      && (CURR_TOP < com_1_y)
      && (com_1_y < (CURR_TOP + PALETTE_COLOR_WIDTH))
      ) begin
      absorb_color <= 3'b000;
      absorb_enable <= 1;
      paint_enable <= 0;
      fill_enable <= 0;
end
else if (
      (FILL_LEFT < com_1_x)
      && (com_1_x < (FILL_LEFT + FILL_WIDTH))
      && (FILL_TOP < com_1_y)
      && (com_1_y < (FILL_TOP + FILL_WIDTH))
      ) begin
      absorb_color <= 3'b000;
      absorb_enable <= 0;
      paint_enable <= 0;
      fill_enable <= 1;
end
else if (
      (SQ_LEFT < com_1_x)
      && (com_1_x < (SQ_LEFT + SQ_WIDTH))
      && (SQ_TOP < com_1_y)
      && (com_1_y < (SQ_TOP + SQ_WIDTH))
      ) begin
```

```
                        absorb_color <= 3'b000;
                        absorb_enable <= 0;
                        paint_enable <= 0;
                        fill_enable <= 0;
                        brush_select <= 2'b01;
                  end
                  else if (
                        (SPECK_LEFT < com_1_x)
                        && (com_1_x < (SPECK_LEFT + SPECK_WIDTH))
                        && (SPECK_TOP < com_1_y)
                        && (com_1_y < (SPECK_TOP + SPECK_WIDTH))
                        ) begin
                        absorb_color <= 3'b000;
                        absorb_enable <= 0;
                        paint_enable <= 0;
                        fill_enable <= 0;
                        brush_select <= 2'b10;
                  end
                  else if (
                        (LINE_LEFT < com_1_x)
                        && (com_1_x < (LINE_LEFT + LINE_WIDTH))
                        && (LINE_TOP < com_1_y)
                        && (com_1_y < (LINE_TOP + LINE_WIDTH))
                        ) begin
                        absorb_color <= 3'b000;
                        absorb_enable <= 0;
                        paint_enable <= 0;
                        fill_enable <= 0;
                        brush_select <= 2'b11;
                  end
                  else begin
                        absorb_color <= 3'b000;
                        absorb_enable <= 0;
                        paint_enable <= 0;
                        fill_enable <= 0;
                  end
            end
            else if ((com_1_x <= TOTALWIDTH) && button) begin
                  // main body
                  paint_enable <= 1;
                  absorb_enable <= 0;
                  fill_enable <= 0;
            end
            else begin
                  paint_enable <= 0;
                  absorb_enable <= 0;
                  fill_enable <= 0;
            end
      end

endmodule
```

J ramclock.v

```
//////////////////////////////////////////////////////////////////////////////////
//
```

```
//
// 6.111 FPGA Labkit -- ZBT RAM clock generation
//
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////
//
//
// This module generates deskewed clocks for driving the ZBT SRAMs and FPGA
// registers. A special feedback trace on the labkit PCB (which is length
// matched to the RAM traces) is used to adjust the RAM clock phase so that
// rising clock edges reach the RAMs at exactly the same time as rising clock
// edges reach the registers in the FPGA.
//
// The RAM clock signals are driven by DDR output buffers, which further
// ensures that the clock-to-pad delay is the same for the RAM clocks as it
is
// for any other registered RAM signal.
//
// When the FPGA is configured, the DCMs are enabled before the chip-level
I/O
// drivers are released from tristate. It is therefore necessary to
// artificially hold the DCMs in reset for a few cycles after configuration.
// This is done using a 16-bit shift register. When the DCMs have locked, the
// <lock> output of this mnodule will go high. Until the DCMs are locked, the
// ouput clock timings are not guaranteed, so any logic driven by the
// <fpga_clock> should probably be held inreset until <locked> is high.
//
//////////////////////////////////////////////////////////////////////////////
//

module ramclock(ref_clock, fpga_clock, ram0_clock, ram1_clock,
              clock_feedback_in, clock_feedback_out, locked);

   input ref_clock;                 // Reference clock input
   output fpga_clock;               // Output clock to drive FPGA logic
   output ram0_clock, ram1_clock;   // Output clocks for each RAM chip
   input  clock_feedback_in;        // Output to feedback trace
   output clock_feedback_out;       // Input from feedback trace
   output locked;                   // Indicates that clock outputs are
stable

   wire  ref_clk, fpga_clk, ram_clk, fb_clk, lock1, lock2, dcm_reset;


//////////////////////////////////////////////////////////////////////////////

   assign ref_clk = ref_clock;

   BUFG int_buf (.O(fpga_clock), .I(fpga_clk));

   DCM int_dcm (.CLKFB(fpga_clock),
            .CLKIN(ref_clk),
            .RST(dcm_reset),
            .CLK0(fpga_clk),
```

```
            .LOCKED(lock1));
   // synthesis attribute DLL_FREQUENCY_MODE of int_dcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of int_dcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of int_dcm is "FALSE"
   // synthesis attribute DFS_FREQUENCY_MODE of int_dcm is "LOW"
   // synthesis attribute CLK_FEEDBACK of int_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of int_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of int_dcm is 0

   BUFG ext_buf (.O(ram_clock), .I(ram_clk));

   IBUFG fb_buf (.O(fb_clk), .I(clock_feedback_in));

   DCM ext_dcm (.CLKFB(fb_clk),
                .CLKIN(ref_clk),
                .RST(dcm_reset),
                .CLK0(ram_clk),
                .LOCKED(lock2));
   // synthesis attribute DLL_FREQUENCY_MODE of ext_dcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of ext_dcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of ext_dcm is "FALSE"
   // synthesis attribute DFS_FREQUENCY_MODE of ext_dcm is "LOW"
   // synthesis attribute CLK_FEEDBACK of ext_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of ext_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of ext_dcm is 0

   SRL16 dcm_rst_sr (.D(1'b0), .CLK(ref_clk), .Q(dcm_reset),
                 .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
   // synthesis attribute init of dcm_rst_sr is "000F";


   OFDDRRSE ddr_reg0 (.Q(ram0_clock), .C0(ram_clock), .C1(~ram_clock),
                   .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
   OFDDRRSE ddr_reg1 (.Q(ram1_clock), .C0(ram_clock), .C1(~ram_clock),
                   .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));
   OFDDRRSE ddr_reg2 (.Q(clock_feedback_out), .C0(ram_clock),
.C1(~ram_clock),
                   .CE (1'b1), .D0(1'b1), .D1(1'b0), .R(1'b0), .S(1'b0));

   assign locked = lock1 && lock2;

endmodule
```

K vram_display.v

```
//////////////////////////////////////////////////////////////////////////////
// generate display pixels from reading the ZBT ram
// note that the ZBT ram has 2 cycles of read (and write) latency
//
// We take care of that by latching the data at an appropriate time.
//
// Note that the ZBT stores 36 bits per word; we use only 32 bits here,
// decoded into four bytes of pixel data.

module vram_display(reset,clk,hcount,vcount,vr_pixel,
                vram_addr,vram_read_data);
```

```
   input reset, clk;
   input [10:0] hcount;
   input [9:0]    vcount;
   //output [7:0] vr_pixel; // b&w
      output [17:0] vr_pixel;
   output [18:0] vram_addr;
   input [35:0]  vram_read_data;

   //wire [18:0]   vram_addr = {1'b0, vcount, hcount[9:2]}; // b&w
      wire [18:0]        vram_addr = {vcount, hcount[9:1]}; // seanyliu

//   wire [1:0]    hc4 = hcount[1:0]; // b&w
        wire      hc2 = hcount[0];
   //reg [7:0]     vr_pixel; // b&w
        reg [17:0]      vr_pixel;
   reg [35:0]      vr_data_latched;
   reg [35:0]      last_vr_data;

   //always @(posedge clk)
     //last_vr_data <= (hc4==2'd3) ? vr_data_latched : last_vr_data;

   //always @(posedge clk)
     //vr_data_latched <= (hc4==2'd1) ? vram_read_data : vr_data_latched;

      always @(posedge clk)
     last_vr_data <= (hc2) ? vr_data_latched : last_vr_data;

   always @(posedge clk)
     vr_data_latched <= (!hc2) ? vram_read_data : vr_data_latched;


/*
   always @(*)          // each 36-bit word from RAM is decoded to 4 bytes
     case (hc4)
       2'd3: vr_pixel = last_vr_data[7:0];
       2'd2: vr_pixel = last_vr_data[7+8:0+8];
       2'd1: vr_pixel = last_vr_data[7+16:0+16];
       2'd0: vr_pixel = last_vr_data[7+24:0+24];
     endcase
*/

      // seanyliu
      always @(*)       // each 36-bit word from RAM is decoded to 4 bytes
     case (hc2)
       1'd1: vr_pixel = last_vr_data[17:0];
       1'd0: vr_pixel = last_vr_data[17+18:0+18];
     endcase
endmodule // vram_display
```

L colorgen.v

```
module colorgen(  input wire clk,
                                    input wire [2:0] sel_color,
                                    input signed [4:0] sel_sat,
                                    input wire reset,
```

```
                                    input wire color_reset,
                                    output wire [17:0] color,
                                    output wire [4:0] sat);

      reg [5:0] color_r;
      reg [5:0] color_g;
      reg [5:0] color_b;
      reg [4:0] c_sat;
      reg [5:0] count;

      parameter change = 4;
      parameter total = 63;
      parameter maxsat = 32;

      assign color = {color_r, color_g, color_b};
      assign sat = c_sat;

      always @(posedge clk) begin
            if (reset || color_reset) begin
                  color_r <= total;
                  color_g <= total;
                  color_b <= total;
                  c_sat <= 0;
                  //count <= 0;
            end
            else if ((c_sat > 0) & (sel_sat < 0)) begin
                  c_sat <= c_sat + sel_sat;
            end
            else if (sel_sat > 0) begin
                  //if ((count == 40) & (c_sat < maxsat - 1)) begin
                  if (c_sat < maxsat - 1) begin
                        c_sat <= c_sat + 1;
                  end
                  //else count <= count + 1;
                  // red: (256, 0, 0)
      //11111_00000_00000
                  // green: (0, 256, 0)               //00000_11111_00000
                  // blue: (0, 0, 256)
      //00000_00000_11111
                  // white: (256, 256, 256)           //11111_11111_11111
                  // previous color: (0, 0, 0)  //00000_00000_00000
                  if ((sel_color[2] == 1) & (sel_color[1] == 0) &
(sel_color[0] == 0)) begin
                  //if ((sel_color[17:12] == total) & (sel_color[11:6] == 0)
& (sel_color[5:0] == 0)) begin
                        //red
                        if ((color_g < change) & (color_b < change)) begin
                              color_g <= 0;
                              color_b <= 0;
                              if (color_r >= change) color_r <= color_r -
change;
                              else color_r <= 0;
                        end
                        else if ((color_g >= change) & (color_b < change))
begin
                              color_b <= 0;
                              color_g <= color_g - change;
```

```
                        end
                        else if ((color_g < change) & (color_b >= change))
begin
                              color_b <= color_b - change;
                              color_g <= 0;
                        end
                        else if ((color_g >= change) & (color_b >= change))
begin
                              color_g <= color_g - change;
                              color_b <= color_b - change;
                        end
                  end
                  else if ((sel_color[2] == 0) & (sel_color[1] == 1) &
(sel_color[0] == 0)) begin
                  //else if ((sel_color[17:12] == 0) & (sel_color[11:6] ==
total) & (sel_color[5:0] == 0)) begin
                        //green
                        if ((color_r < change) & (color_b < change)) begin
                              color_r <= 0;
                              color_b <= 0;
                              if (color_g >= change) color_g <= color_g -
change;
                              else color_g <= 0;
                        end
                        else if ((color_r >= change) & (color_b < change))
begin
                              color_b <= 0;
                              color_r <= color_r - change;
                        end
                        else if ((color_r < change) & (color_b >= change))
begin
                              color_r <= 0;
                              color_b <= color_b - change;
                        end
                        else if ((color_r >= change) & (color_b >= change))
begin
                              color_r <= color_r - change;
                              color_b <= color_b - change;
                        end
                  end
                  else if ((sel_color[2] == 0) & (sel_color[1] == 0) &
(sel_color[0] == 1)) begin
                  //else if ((sel_color[17:12] == 0) & (sel_color[11:6] == 0)
& (sel_color[5:0] == total)) begin
                        //blue
                        if ((color_r < change) & (color_g < change)) begin
                              color_r <= 0;
                              color_g <= 0;
                              if (color_b >= change) color_b <= color_b -
change;
                              else color_b <= 0;
                        end
                        else if ((color_r >= change) & (color_g < change))
begin
                              color_g <= 0;
                              color_r <= color_r - change;
                        end
```

```
                        else if ((color_r < change) & (color_g >= change))
begin
                              color_r <= 0;
                              color_g <= color_g - change;
                        end
                        else if ((color_r >= change) & (color_g >= change))
begin
                              color_r <= color_r - change;
                              color_g <= color_g - change;
                        end
                  end
                  else if ((sel_color[2] == 1) & (sel_color[1] == 1) &
(sel_color[0] == 0)) begin
                        if (color_b >= change) color_b <= color_b - change;
                        else begin
                              color_b <= 0;
                              if ((color_r < change) & (color_g < change))
begin
                                    color_r <= 0;
                                    color_g <= 0;
                              end
                              else if ((color_r >= change) & (color_g <
change)) begin
                                    color_r <= color_r - change;
                                    color_g <= 0;
                              end
                              else if ((color_r < change) & (color_g >=
change)) begin
                                    color_r <= 0;
                                    color_g <= color_g - change;
                              end
                              else if ((color_r >= change) & (color_g >=
change)) begin
                                    color_r <= color_r - change;
                                    color_g <= color_g - change;
                              end
                        end
                  end
                  else if ((sel_color[2] == 1) & (sel_color[1] == 1) &
(sel_color[0] == 1)) begin
                  //else if ((sel_color[17:12] == total) & (sel_color[11:6]
== total) & (sel_color[5:0] == total)) begin
                        //white
                        if (color_r <= total - change)      color_r <=
color_r + change;
                        else color_r <= total;
                        if (color_g <= total - change) color_g <= color_g +
change;
                        else color_g <= total;
                        if (color_b <= total - change) color_b <= color_b +
change;
                        else color_b <= total;
                  end
            end
      end
endmodule
```

M saturation.v

```
//paint: intention[1]
//suck: intention[0]
module saturation(input clock,
                                    input [1:0] intention,
                                    input reset,
                                    output reg signed [2:0] sat_change);
      reg signed [28:0] count;
      always @(posedge clock) begin
            if (reset) begin
                  count <= 0;
                  sat_change <= 0;
            end
            else begin
                  if (count == 16_000_000) begin // graceli change to make
bigger
                        count <= 0;
                        sat_change <= -1;
                  end
                  else if (count == -16_000_000) begin
                        count <= 0;
                        sat_change <= 1;
                  end
                  else if (intention[1]) begin
                        count <= count + 1;
                        sat_change <= 0;
                  end
                  else if (intention[0]) begin
                        count <= count - 1;
                        sat_change <= 0;
                  end
            end
      end
endmodule
```

N rect.v (Brush Generation)

```
module brush1 (input [10:0] x,hcount,
                              input [9:0] y,vcount,
                              output bool);

      assign bool = (   ((hcount - x == 5) && (vcount - y == 0)) ||
                                          ((hcount - x == 3) && (vcount - y
== 1)) ||
                                          ((hcount - x == 3) && (vcount - y
== 0)) ||
                                          ((hcount - x == 2) && (vcount - y
== 0)) ||
                                          ((hcount - x == 0) && (vcount - y
== 0)) ||
                                          ((hcount - x == -1) && (vcount - y
== 0)) ||
```

```
                                        ((hcount - x == -2) && (vcount - y
== 0)) ||
                                        ((hcount - x == -2) && (vcount - y
== -1)) ||
                                        ((hcount - x == -4) && (vcount - y
== 0)) ||
                                        ((hcount - x == -5) && (vcount - y
== 0))      );

endmodule

module brushsquare (    input [10:0] x,hcount,
                                        input [9:0] y,vcount,
                                        output bool);

      assign bool = (   (hcount <= 4 + x) &&
                                        (x <= 4 + hcount) &&
                                        (vcount <= 4 + y) &&
                                        (y <= 4 + vcount) );

endmodule

module brushcrazy (input [10:0] x1, x2, hcount,
                                    input [9:0] y1, y2, vcount,
                                    output bool);

      wire signed [9:0]       diffx1 = x1 - hcount;
      wire signed [9:0]       diffy1 = vcount - y2;
      wire signed [9:0] diffx2 = hcount - x2;
      wire signed [9:0]       diffy2 = y1 - vcount;
      wire signed [10:0] diff1 = diffx1 * diffy1;
      wire signed [10:0] diff2 = diffx2 * diffy2;
      assign bool = (   (diff1 - diff2 < 20) && (diff2 - diff1 < 20)    );

endmodule

module brushtest (input [10:0] x1, x2, hcount,
                                    input [9:0] y1, y2, vcount,
                                    output bool);
      assign bool = (   (     ((hcount >= x1) && (hcount <= x2)) || ((hcount
>= x2) && (hcount <= x1))     ) &&
                                        (     ((vcount >= y1) && (vcount <=
y2)) || ((vcount >= y2) && (vcount <= y1))      )           );
endmodule

module brushline (input [10:0] x1, x2, hcount,
                                    input [9:0] y1, y2, vcount,
                                    output bool);

      wire signed [10:0]      diffx1 = x1 - hcount;
      wire signed [9:0]       diffy1 = y1 - vcount;
      wire signed [10:0]      diffx2 = x1 - x2;
      wire signed [9:0]       diffy2 = y1 - y2;
      wire signed [21:0] diff1 = diffx1 * diffy2;
      wire signed [21:0] diff2 = diffx2 * diffy1;
      assign bool = (   (diff1 < 800 + diff2)   &&
                                        (diff2 < 800 + diff1)   &&
```

```
                                        (     ((hcount >= x1) && (hcount <=
x2)) || ((hcount >= x2) && (hcount <= x1))      ) &&
                                        (     ((vcount >= y1) && (vcount <=
y2)) || ((vcount >= y2) && (vcount <= y1))      )           );

endmodule

module brushdotlines (input [10:0] x1, x2, hcount,
                                   input [9:0] y1, y2, vcount,
                                   output bool);

      wire [9:0]  diffx1 = x1 - hcount;
      wire [9:0]  diffy1 = vcount - y2;
      wire [9:0]  diffx2 = hcount - x2;
      wire [9:0]  diffy2 = y1 - vcount;
      wire [10:0] diff1 = diffx1 * diffy1;
      wire [10:0] diff2 = diffx2 * diffy2;
      assign bool = (   (diff1 - diff2 < 20) && (diff2 - diff2 < 20)    );

endmodule

module brush2 (input [10:0] x,hcount,
                            input [9:0] y,vcount,
                            output bool);

      assign bool = (   ((hcount - x == 1) && (vcount - y == 7)) ||
                                        ((hcount - x == 1) && (vcount - y
== -1)) ||
                                        ((hcount - x == 1) && (vcount - y
== -2)) ||
                                        ((hcount - x == 1) && (vcount - y
== -3)) ||
                                        ((hcount - x == 2) && (vcount - y
== -1)) ||
                                        ((hcount - x == 2) && (vcount - y
== -2)) ||
                                        ((hcount - x == 2) && (vcount - y
== -3)) ||
                                        ((hcount - x == 3) && (vcount - y
== -1)) ||
                                        ((hcount - x == 3) && (vcount - y
== -2)) ||
                                        ((hcount - x == 3) && (vcount - y
== 4)) ||
                                        ((hcount - x == 4) && (vcount - y
== 4)) ||
                                        ((hcount - x == 5) && (vcount - y
== 4)) ||
                                        ((hcount - x == 0) && (vcount - y
== 4)) ||
                                        ((hcount - x == 0) && (vcount - y
== 0)) ||
                                        ((hcount - x == 0) && (vcount - y
== -1)) ||
                                        ((hcount - x == 0) && (vcount - y
== -2)) ||
```

```
((hcount - x == 0) && (vcount - y == -3)) ||
((hcount - x == -1) && (vcount - y == 4)) ||
((hcount - x == -1) && (vcount - y == 3)) ||
((hcount - x == -1) && (vcount - y == 2)) ||
((hcount - x == -1) && (vcount - y == 1)) ||
((hcount - x == -1) && (vcount - y == 0)) ||
((hcount - x == -1) && (vcount - y == -1)) ||
((hcount - x == -1) && (vcount - y == -2)) ||
((hcount - x == -2) && (vcount - y == 4)) ||
((hcount - x == -2) && (vcount - y == 3)) ||
((hcount - x == -2) && (vcount - y == 2)) ||
((hcount - x == -2) && (vcount - y == 1)) ||
((hcount - x == -2) && (vcount - y == 0)) ||
((hcount - x == -2) && (vcount - y == -1)) ||
((hcount - x == -2) && (vcount - y == -2)) ||
((hcount - x == -3) && (vcount - y == -1)) ||
((hcount - x == -3) && (vcount - y == -2)) ||
((hcount - x == -4) && (vcount - y == -1)) ||
((hcount - x == -4) && (vcount - y == -2)) ||
((hcount - x == -5) && (vcount - y == 1))      );
endmodule
```

O brushcentergen.v

```
`timescale 1ns / 1ps
module brushcentergen(  input [10:0] ix1,
                        input [9:0] iy1,
                        input [10:0] ix2,
                        input [9:0] iy2,
                        input clk,
                        output brushenable,
                        output reg [10:0] x1,
                        output reg [9:0] y1,
                        output reg [10:0] x2,
                        output reg [9:0] y2,
                        output reg [10:0] x3,
```

```
                                                output reg [9:0] y3,
                                                output reg [10:0] x4,
                                                output reg [9:0] y4,
                                                output reg [10:0] x5,
                                                output reg [9:0] y5);
      assign brushenable = 1;

      always @(posedge clk) begin
            x1 <= ix1;
            y1 <= iy1;
            x2 <= ix2;
            y2 <= iy2;
            x3 <= (ix1[10:0] + ix2[10:0]) / 2;
            y3 <= (iy1[9:0] + iy2[9:0]) / 2;
            x4 <= (3 * ix1[10:0] + ix2[10:0]) / 4;
            y4 <= (3 * iy1[9:0] + iy2[9:0]) / 4;
            x5 <= (ix1[10:0] + 3 * ix2[10:0]) / 4;
            y5 <= (iy1[9:0] + 3 * iy2[9:0]) / 4;
      end
endmodule
```

P painter.v

```
module painter(input vclock,
                              input bool,
                              input [17:0] color_in,
                              input [17:0] color_brush,
                              input [4:0] sat,
                              output [17:0] color_out);

      parameter maxsat = 32;
      wire [5:0] red_out, green_out, blue_out;

      wire [5:0] red_in = color_in[17:12];
      wire [5:0] green_in = color_in[11:6];
      wire [5:0] blue_in = color_in[5:0];

      wire [5:0] red_brush = color_brush[17:12];
      wire [5:0] green_brush = color_brush[11:6];
      wire [5:0] blue_brush = color_brush[5:0];

      assign red_out = ((red_in * (maxsat - sat)) + (red_brush * sat)) /
maxsat;
      assign green_out = ((green_in * (maxsat - sat)) + (green_brush * sat))
/ maxsat;
      assign blue_out = ((blue_in * (maxsat - sat)) + (blue_brush * sat)) /
maxsat;

      assign color_out = bool ? {red_out, green_out, blue_out} : color_in;
endmodule
```

Q gen_paint_zbt.v

```
module gen_paint_zbt(
   input        clk,
```

```
      input           reset,
      input [10:0] x,
      input [9:0]  y,
      input [17:0] data,
   output reg [18:0] ntsc_addr,
   output reg [35:0] ntsc_data,
   output reg        ntsc_we
      );

      always @ (posedge clk) begin
            if (reset) begin
                  ntsc_addr <= 19'd0;
                  ntsc_data <= 36'd0;
                  ntsc_we <= 0;
            end
            else begin
                  ntsc_addr <= {y[9:0], x[9:1]};
                  if (x[0]) begin // ends 1'b1
                        ntsc_data[17:0] <= data;
                        ntsc_we <= 1;
                  end
                  else begin // ends 1'b0
                        ntsc_data[35:18] <= data;
                        ntsc_we <= 0;
                  end
            end
      end

endmodule // ntsc_to_zbt
```

R grace_vram_display.v

```
module grace_vram_display(
      input reset,
      input clk,
      input [10:0] hcount,
      input [9:0] vcount,
      input [35:0] vram_read_data,
      output reg  [17:0] vr_pixel,
      output wire [18:0] vram_addr
      );

      assign vram_addr = {vcount, hcount[9:1]};

   wire hc2 = hcount[0];
   reg  [35:0] vr_data_latched;
   reg  [35:0] last_vr_data;

      always @(posedge clk)
     last_vr_data <= (hc2) ? vr_data_latched : last_vr_data;

   always @(posedge clk)
     vr_data_latched <= (!hc2) ? vram_read_data : vr_data_latched;

      always @ (*) begin // each 36-bit word from RAM is decoded to 4 bytes
            case (hc2)
```

```
                1'd1: vr_pixel = last_vr_data[17:0];
                1'd0: vr_pixel = last_vr_data[17+18:0+18];
            endcase
      end
endmodule
```

S svga.v

```
module svga(input vclock,
            output reg [10:0] hcount,    // pixel number on current line
            output reg [9:0] vcount,       // line number
            output reg vsync,hsync,blank);

   // horizontal: 1056 pixels total
   // display 800 pixels per line
   reg hblank,vblank;
   wire hsyncon,hsyncoff,hreset,hblankon;
   assign hblankon = (hcount == 799);
   assign hsyncon = (hcount == 839);
   assign hsyncoff = (hcount == 967);
   assign hreset = (hcount == 1055);

   // vertical: 628 lines total
   // display 600 lines
   wire vsyncon,vsyncoff,vreset,vblankon;
   assign vblankon = hreset & (vcount == 599);
   assign vsyncon = hreset & (vcount == 600);
   assign vsyncoff = hreset & (vcount == 604);
   assign vreset = hreset & (vcount == 627);

   // sync and blanking
   wire next_hblank,next_vblank;
   assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
   assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
   always @(posedge vclock) begin
      hcount <= hreset ? 0 : hcount + 1;
      hblank <= next_hblank;
      hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync;  // active low

      vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
      vblank <= next_vblank;
      vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync;  // active low

      blank <= next_vblank | (next_hblank & ~hreset);
   end
endmodule
```