Recursive Augmented Reality Image Processing System

Logan P. Williams & José E. Cruz Serrallés

December 10, 2011

Abstract

We have implemented an augmented reality system that can overlay a digital image on a video stream of a real world environment. We read NTSC video data from a video camera and store it in external ZBT memory. A picture frame with colored markers on the corners is held in front of the camera. We then perform chroma-based object recognition to locate the coordinates of the corners. Using these coordinates, we apply a projective transformation to project an image onto the dimensions of the picture frame. From this, we generate a VGA output signal, display the original captured image, with the processed image overlayed on top of the picture frame. By using the previously displayed video frame as the image to be projected on the next frame, the system becomes "recursive."

Contents

List of Figures

1 Overview

The augmented reality image processing system, or `augreal`, was inspired by the recursive feedback effects created by pointing a video source at its own display. Unfortunately, in doing this in the real world, there is significant loss in image quality in each successive generation of recursion. Performing the same operation in digital logic provides the opportunity to achieve the same effect, in real time at a high frame rate, with minimal image quality loss. We also chose to implement object tracking functionality, to simulate the effect of moving the video monitor around within the camera's field of view. The final output of `augreal` is shown in Figure 1.

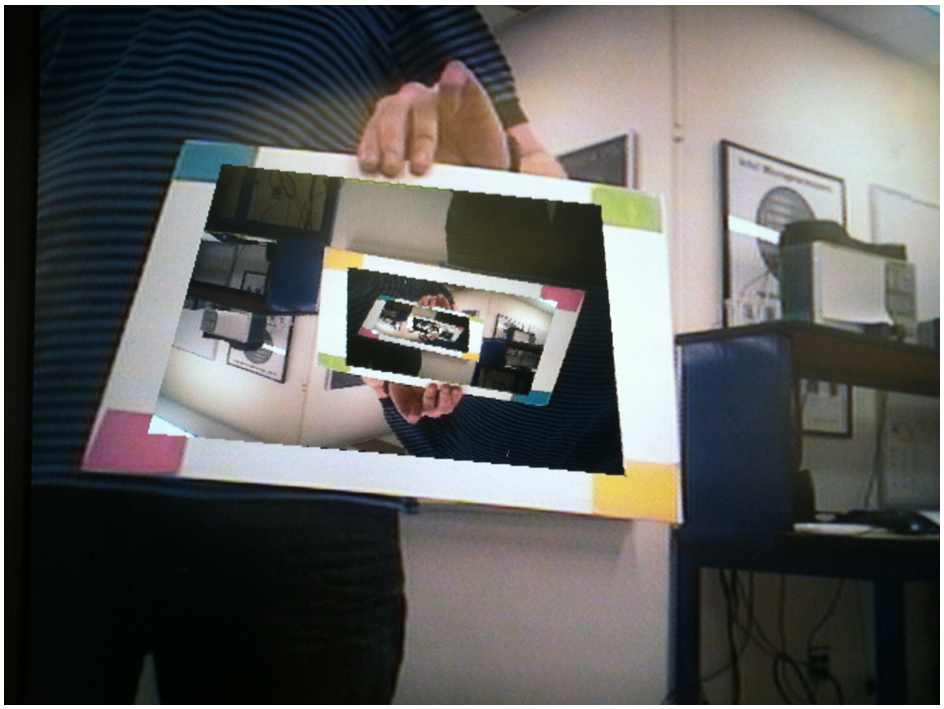

Figure 1: *A screen capture of the video output of the augmented reality image processing system.*

The `augreal` system uses a four stage buffered image processing pipeline. Images from a video camera are captured into a memory buffer, which on the next frame is partially overwritten by skewed, scaled, and rotated pixels from the last image buffer that was displayed on the VGA output. On the next frame, this overwritten buffer becomes the video output. There are seven primary modules of this system.

The `clock_gen` module is responsible for creating and phase synchronizing the clocks used by the rest of the system. Because we require a specific video clock frequency, and because of the amount of computation and memory accesses we must perform, it is necessary for `augreal` to use three distinct clock domains: a video capture clock, at approximately 27 MHz, generated externally by the video camera, a memory clock at 50 MHz, and a video output clock at 25 MHz.

The `memory_interface` module handles reading and writing from the ZBT memory, abstracting these operations away from the other modules. This module is responsible for "shifting" the buffers as described above.

Raw video is read from the video camera by the `ntsc_capture` module, which also finds recognizes pixels that match the colors of the corners of the target frame. These pixels are read by the `object_recognition` module which finds the weighted center of mass of the frame corners. The `LPF` module sends each pixel to the `projective_transform` module which then sends that pixel's value and new coordinates to the `memory_interface` module to be written to memory. The `vga_write` module reads and transmits the VGA data to be generated by the video DAC.

Raw video is read from the video camera by the `ntsc_capture` module, which also finds recognizes pixels that match the colors of the corners of the target frame. These pixels are read by the `object_recognition` module which finds the weighted center of mass of the frame corners. The `LPF` module sends each pixel to the `projective_transform` module which then sends that pixel's value and new coordinates to the `memory_interface` module to be written to memory. The `vga_write` module reads and transmits the VGA data to be generated by the video DAC.

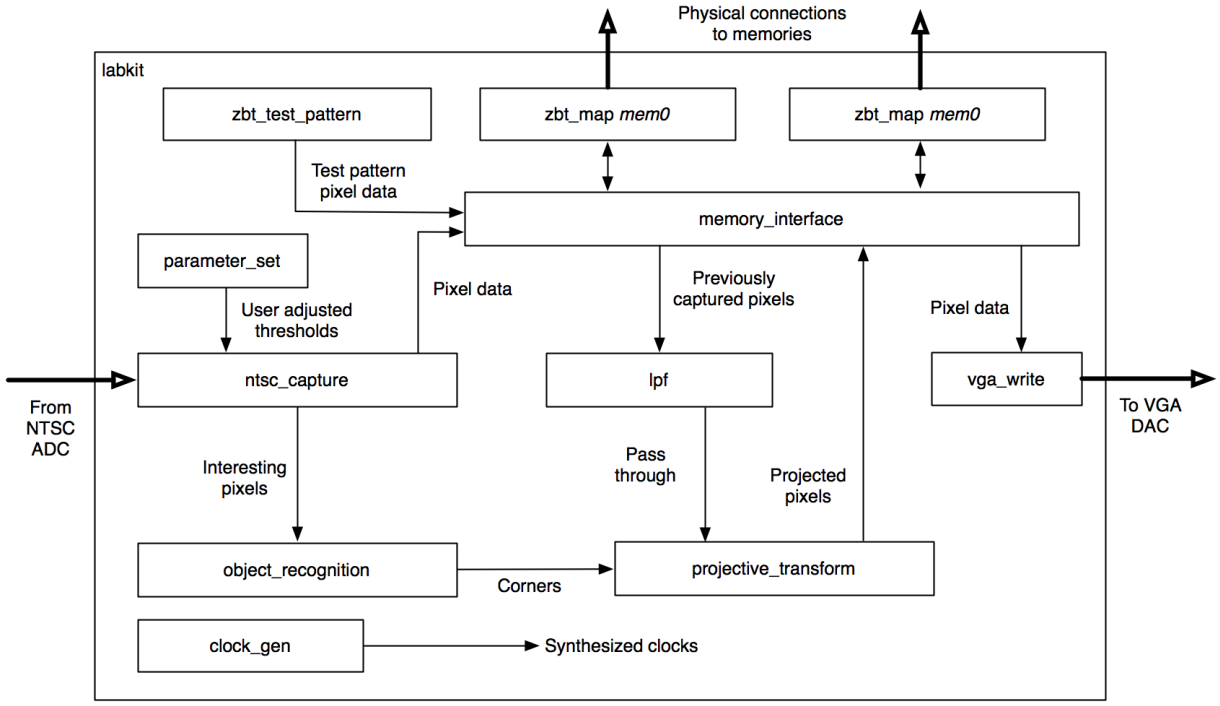

Figure 2: *An overview block diagram of the augmented reality system.*

2 Description

2.1 `ntsc_capture` (Logan)

The `ntsc_capture` module decodes NTSC Composite video using an Analog Devices ADV7185 video ADC and sends pixels in luminance/chrominance color space to `memory_interface` to be written into a ZBT memory frame buffer. Additionally, `ntsc_capture` is responsible for recognizing colors that matches the

corner targets (blue, green, pink, and orange).

`ntsc_capture` has three submodules, none of which were written by our team. They are described here with specific attention to their relation to the primary `ntsc_capture` module.

`adv7185init` This module initializes the ADV7185 video ADC. It selects the video source (in our case, the composite video input), and wires the data streams to the `labkit` module. This module was written by the 6.111 staff.

`ntsc_decode` This module takes the incoming data stream from the ADV7185, and decodes it into individual pixels. It also generates flags that indicate when its data output is valid, when the NTSC stream is in horizontal sync, when the NTSC stream is in vertical sync, and whether the current frame is an odd or even field. (Because NTSC video data is interlaced, each frame alternates even and odd lines of video.) This module was written by the 6.111 staff.

`ntf` The `ntsc_capture` module generates data at the speed of the external video clock, which is 27 Mhz. This creates problems with processing that data in modules that operate at the RAM clock speed of 50 Mhz. To solve this problem, `ntf` is a dual clock domain FIFO register, generated by the Xilinx Core Generator utility. Data is written to it on the video clock domain, and read from it on the RAM clock domain. In this way, flags and pixels are synchronized correctly before being processed by `memory_interface`.

`ntsc_capture` itself uses x and y iterator registers to tell `memory_interface` where to write the incoming pixels to. The x iterator variable increments every time the data valid output of `ntsc_decode` goes high. When x is even, the incoming pixel is stored in a buffer, and when it is odd, both the current incoming pixel and the previously buffered pixel are sent to `memory_interface`. When a horizontal sync is detected, the y iterator variable increments by two, because of the interlacing of the video stream. When a vertical sync is detected, the x iterator variable resets to 0, but the y iterator variable resets to the current field value, in order to "seed" the y iterator as even or odd. `ntsc_capture` ignores the first 23 lines of video input, and the first 10 pixels of each line, as it was found in testing that those pixels are always blanked.

In both states, comparisons are also made with threshold parameters in luminance and chrominance to check for "interesting" colors. When detected, a flag and the color detected are sent to `object_recognition` through the `ntf` FIFO memory. The threshold values for detection can be adjusted using the buttons on the labkit. These values are generated and stored by the `parameter_set` module.

2.2 `memory_interface` (José)

The `memory_interface` module handles the interactions between all of the other modules and the two ZBT Memory blocks, which house the four images that the modules use for capturing, displaying, and processing. Ideally, BRAM would have been used, but the number of pixels that we would like to store vastly exceeds BRAM capacity. Unlike BRAM, each ZBT memory block can only handle one read or write operation per cycle, causing memory access to be the main bottleneck of our system. As such, we store only 8 bits of luminance and 5 bits of each chrominance per pixel, allowing us to store two pixels per address and to reduce the number of memory accesses in our system by a factor of two. The number of required memory

acceses per module and the distribution of the images in the RAM necessitates a minimum clock frequency of 35MHz.

The inputs to `memory_interface` are (1) `frame_flag`, which signals when to shift; (2) `ntsc_pixel`, the two pixels from `ntsc_capture`; (3) four (x,y) pairs; (4) one pixel from `projective_transform`; and (5) request flags. The outputs from `memory_interface` are (1) done flags; (2) a pixel pair to `vga_write`; and (3) two pixels to `LPF`.

`memory_interface` allocates two images per memory block. These four images are (1) *capture*, the image being captured from NTSC; (2) *display*, the image being displayed in the VGA; (3) *processing*, the image that is processed by `LPF`; and (4) *next_display*, the image to which `projective_transform` writes and the next image that will be displayed. Every image refresh (1/30 seconds), the previous *next_display*, *display*, *capture*, and *processing* image locations become the next *display*, *processing*, *next_display*, and *capture* image locations, respectively. These location shifts are transparent to the other modules. Read and write requests from `vga_write` and `ntsc_capture` will be given priority over requests from other modules.

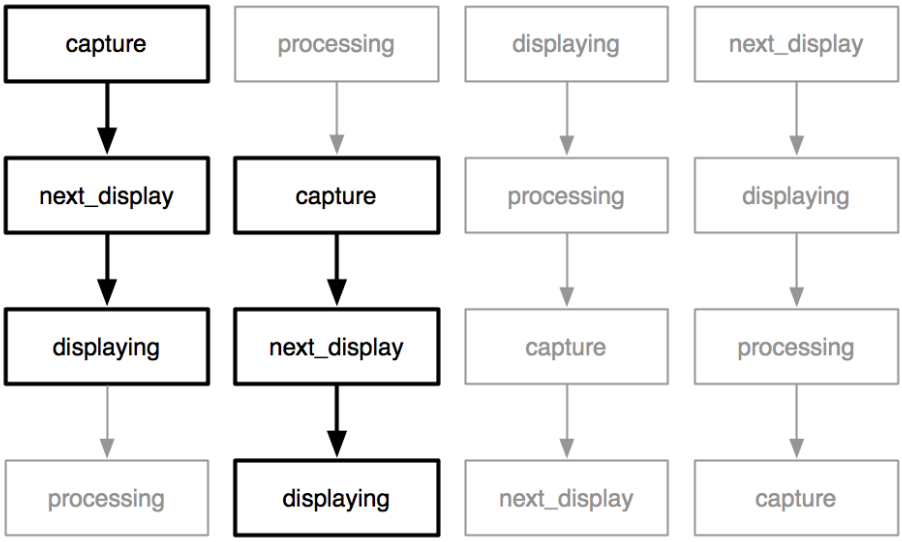

Figure 3: *The progression seen in memory locations as* `frame_flag` *is asserted. Two complete pathways from image capture to image display are highlighted.*

Due to the two-cycle latency of the ZBT RAM, care was taken in `memory_interface` to wait two cycles before assigning an output to the module that made a request. `zbt_map` is used to delay the write data to the RAM by two cycles. `zbt_map` also allows for `memory_interface` to request partial bit write enables, which proved to be necessary for the operation of `projective_transform`.

2.3 `object_recognition` (Logan)

The `object_recognition` module collects "interesting" pixels located by the `ntsc_capture` module, and calculates the center of mass of each color, to find the location of the corners of the picture frame.

It takes as inputs (1) the color of a detected pixel, (2) a flag that goes high for one clock cycle when a pixel is detected, (3) the X/Y coordinates of the pixel, and (4) a flag that goes high when a new frame is

beginning. It produces as output four sets of X/Y coordinates, one for the center of mass of each color.

Instead of performing a linear average of interesting pixels recieved in each color, we chose to implement a weighted average, in order to make the module more resistant to video noise and interference from background objects. When `object_recognition` recieves a flag from `ntsc_capture`, it multiplies that pixel's coordinates proportionally to their distance from the center of mass generated in the last frame. These modified coordinates are accumulated in seperate registers for each color. Upon receipt of a `frame_flag`, the module initializes the divide operation to perform the averaging. (The functionality of the `divider` submodule is described in detail in the `projective_transform` section below.)

After the completion of this division operation, the `object_recognition` module calculates the minimum side length in the horizontal and vertical direction. These distances were intended to be used by `lpf` for generating downsampling coefficients, but this functionality was not implemented.

2.4 `LPF` (José)

The warping of images carried out by `projective_transform` aliases the image, resulting in undesired "junk" pixels. Initially, `LPF` was devised as a way to lowpass filter the pixel data and avoid aliasing. Unfortunately, due to time constraints and unforeseen, extensive issues in getting video capture, memory interfacing, and video display to work, an `LPF` module that lowpasses pixel data was not written. The intended operation of `LPF` was to apply a lowpass filter to the *processing* image so as to avoid aliasing when `projective_transform` skews the image. Instead a simple `LPF` module was written that fetches pixels from memory when `projective_transform`'s `request` is asserted high, and four cycles later, pulses `pixel_flag` and outputs one pixel. `LPF` pulses its `lpf_flag` once every two pixels, that is, only on even pixel requests.

The inputs to `LPF` are (1) `frame_flag`, which indicates the start of a new cycle; (2) `done_lpf`, which indicates that the module's memory request was processed; (3) `lpf_pixel_read`, the pixels from `memory_interface`; and (4) `request` from projective_transform. The outputs from `LPF` to `memory_interface` are (1) `lpf_flag`, and (2) `lpf_x` and lpf_y, the (x,y) coordinates of the leftmost pixel. The outputs from `LPF` to `projective_transform` are (1) `pixel_flag`, which signals when a new pixel is available, and (2) `pixel`.

2.5 `projective_transform` (Logan)

The inputs to `projective_transform` are (1) the pixel value last produced by `LPF`, (2) a flag signal held high for one clock signal when `LPF` has processed a new pixel, (3) the four coordinates of the corners of the frame provided by the `object_recognition` module, and (4) a signal when a new frame is beginning.

The outputs from `projective_transform` are (1) a request to `LPF` for a new pixel, (2) the X/Y coordinates of the transformed pixel, (3) the transformed pixel value, and (4) a flag indicating that a new pixel is to be written.

This function maps the original rectangular image to any convex quadrilateral, provided that all sides of the destination quadrilateral are shorter than the original, which is inherent in the overall system. A graphic representation of the transformation is shown in Figure 4.

Mathematically, the algorithm works as follows:

1. Create two "iterator points," point I_A and I_B initially located at $A\prime$ and $B\prime$.
2. Create a third iterator point, I_C at the location I_A.

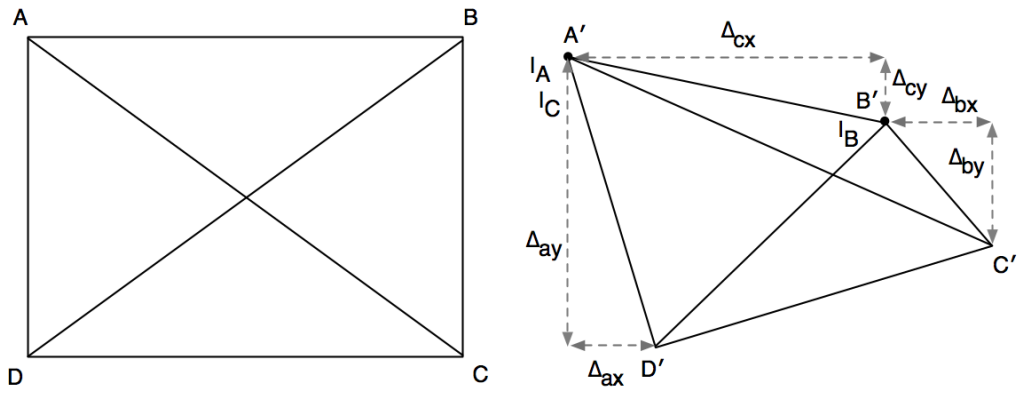

Figure 4: *A visual representation of the result of the* `projective_transform` *module. Input is on the left, a possible output, for four coordinates A′, B′, C′, and D′ is on the right.*

3. Let $o_x = 0$ and $o_y = 0$
4. Calculate the normalized incrementor value, `delta_a_x` $= \Delta_{ax}/480$.
5. Calculate the normalized incrementor value, `delta_a_y` $= \Delta_{ay}/480$.
6. Repeat steps 4 and 5 for `delta_b_x` and `delta_b_y`.
7. Calculate the normalized incrementor value, `delta_c_x` $= \Delta_{cx}/640$.
8. Calculate the normalized incrementor value, `delta_c_y` $= \Delta_{cy}/640$.
9. Assign the pixel value of I_C to pixel (o_x, o_y) in the original image.
10. Increment the coordinate of I_C by (`delta_c_x`, `delta_c_y`).
11. Increment o_x.
12. Repeat steps 9–11 until $I_C = I_B$.
13. Increment the coordinate of I_A by (`delta_a_x`, `delta_a_y`).
14. Increment the coordinate of I_B by (`delta_b_x`, `delta_b_y`).
15. Increment o_y.
16. Repeat steps 7–15 until $I_A = D\prime$ and $I_B = C\prime$.

The Verilog implementation of this module has two types of submodules. The algorithm requires no multiplications, but several divisions. To perform divisions, a divider module, named `divider`, was used that implemented a simple restoring division algorithm. [1] This algorithm takes N clock cycles to complete, where N is the width of the dividend and divisor.

The `LPF` module, which sends pixel data to the `projective_transform` module, has a five clock cycle delay on transmitted pixels. Therefore, up to five pixels can arrive at `projective_transform` while it is unable to write to memory, even after `pt` has told `lpf` to stop sending new data. Because of this, it was necessary to create a buffer to hold pixel data until it was able to be written to `memory_interface`. The `shift18` module accomplishes this by using 18 `SRL16E` 16 bit shift register primitive elements to a create an 18 bit wide shift register. By setting the address of the multiplexer associated with each `SRL16E` element, the length of the shift register can be expanded and contracted easily.

The `projective_transform` module itself is a state machine with three states, `WAIT_FOR_CORNERS`, `WAIT_FOR_DIVIDERS`, and `WAIT_FOR_PIXEL`. The module initializes itself to the `WAIT_FOR_CORNERS` state, where

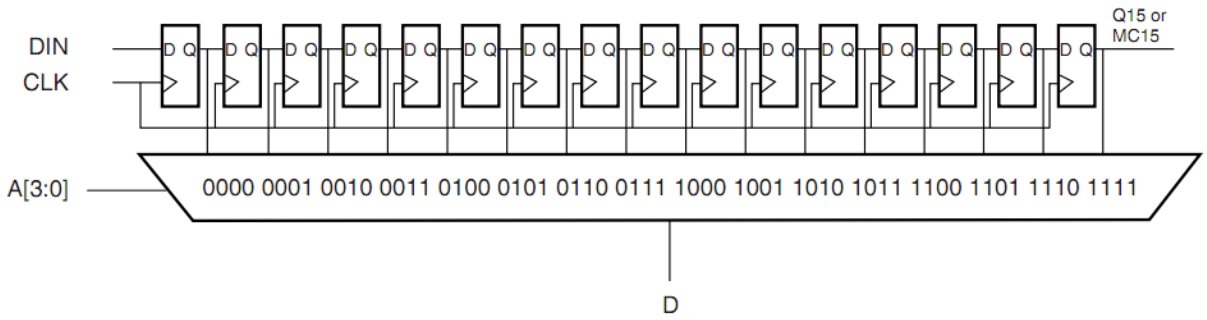

Figure 5: *The* `SRL16E` *primitive shift register element. By setting* `A[3:0]`*, the length of the buffer can be adjusted dynamically.*

it stays until `object_recognition` indicates with `corners_flag` that it has finished calculating the location of the four corners of the frame in the image. When this flag is recieved, `projective_transform` initializes the divisors and dividends of each of the six dividers, and sends a signal for the dividers to begin their operation. Then, the machine advances to the `WAIT_FOR_DIVIDERS` state.

In this state, the module waits until all of the dividers have finished their computation. When this occurs, the results of each computation are saved in the delta registers. This corresponds to steps 4–6, and the first iteration of 7–8 in the algorithm outlined above. Then the module advances to the `WAIT_FOR_PIXEL` state, where the bulk of the algorithm, steps 7–15, is implemented.

In this state, for every clock cycle in which `memory_interface` indicates that it can accept new data, `projective_transform` outputs a pixel and a coordinate from its buffer. If `memory_interface` cannot accept new data (because of a conflict with `ntsc_catpure` or `vga_display`), the module tells `lpf` to stop sending new data. If `projective_transform` receives a new pixel from `lpf`, it buffers it into `shift18`. When $o_x = 500$, `projective_transform` begins the division operators in steps 7 and 8 for the next line of the image. In this way, the divisions are pipelined so that there is no delay due to `divider`. Without this pipelining, there would be a 42 cycle delay at the beginning of each line of output. When `projective_transform` has iterated through the complete original image, it resets its iterator registers to zero and its state to `WAIT_FOR_CORNERS`.

2.6 `clock_gen` (José)

Clock synthesis is carried in the `clock_gen` module. Initially, we assumed that clocking would not be much of an issue. However, as we tested our modules, we realized that improper clock generation created persistent and pervasive setup and hold time issues. After much trial and error, we settled on using a DCM to generate a reference clock at 50MHz, using this clock to generate the `ram_clock`s and the system clock, and using the `CLKDV` output to obtain a clock for VGA video at half the frequency of the system clock, 25 Mhz. The DCM that synthesizes the system clocks and RAM clocks is located in the `ramclock` module, which is an adapted version of the module provided to us by the 6.111 staff. In the output DCM of `ramclock`, the parameter `CLKDV_DIVIDE` had to be set to 2, the parameter `CLKOUT_PHASE_SHIFT` was changed to "FIXED", and the parameter `PHASE_SHIFT` was set to 0. Setting these parameters to their respective values provided us consistent locking.

25MHz was chosen as our video display frequency because it is close to the required 25.175MHz frequency

for 640x480x60Hz display. 50MHz was chosen as our system frequency because it met our minimal frequency requirement and because 0-phase locking with the 25MHz signal could be achieved, as 50MHz is an even multiple of 25MHz. This simplified considerably the design of the `vga_display` module, which has to transfer data between these two clock domains.

2.7 `vga_write` (José)

This module reads data from `memory_interface` and displays this data on the screen. Housed inside this module are (1) `xvga`, which generates the VGA control signals necessary to display data on the screen, and (2) `ycrcb2rgb_lut`, which converts the fetched 18-bit YCrCb values to 24-bit RGB values suitable for display.

`vga_write` "flags" `memory_interface` and provides the interface with the (x, y) coordinates of the pixel to be displayed, which correspond to the hcount and vcount variables being output from the `xvga` submodule. Because the `xvga` module is clocked at 25MHz and `memory_interface` is clocked at 50MHz, care must be taken when requesting and reading pixels. Due to the implicit four-cycle delay of `memory_interface`, the outputs of `xvga` must be delayed by two video clock cycles, so that the pixels that are output to the monitor correspond the control signals at the moment of "flagging".

The fetched data consists of 36 bits, totalling two pixels. Every video clock cycle, the bits converted using `ycrcb2rgb` are alternated between the higher-order 18 bits and the lower order 18 bits. The higher order 18 bits correspond to even pixels and the lower order 18 bits correspond to odd pixels. This set of control signals and pixel values is then pipelined, so that on the next rising edge these values are fed to the VGA chip. The clock assigned to the VGA chip (vga_out_clock) is an inverted copy of the onboard 25MHz clock, due to the long path propagation delay from the FPGA to the VGA DAC.

`xvga` This submodule is an adapted version of the submodule used in the 6.111 Pong Lab. The changes include adjusting when the vsync, hsync, blank, and reset signals trigger based on the change of resolution from 1024x768 to 640x480. Operation involves a few basic steps: (1) Increment hcount by one every clock cycle, modulo 800. (2) Increment vcount by 1 when hcount has reached 799, modulo 524. (3) Pulse blank when either hcount ≥ 640 or vcount ≥ 480. (4) Pulse hsync when $656 \leq$ hcount ≤ 752. (5) Pulse vsync when $491 \leq$ vcount ≤ 493. [2]

`ycrcb2rgb_lut` This submodule maps a YCrCb pixel to an RGB pixel. The mapping from YCrCb to RGB is summarized by the following set of equations: [3]

$$
\begin{aligned}
R' &= 1.164(Y' - 16) + 1.596(Cr - 128) \\
G' &= 1.164(Y' - 16) - 0.813(Cr - 128) - 0.392(Cb - 128) \\
B' &= 1.164(Y' - 16) + 2.017(Cb - 128)
\end{aligned}
$$

The luminance and chrominances have bitwidths of 8, 5, and 5, respectively. Initially, an approach with multipliers and adders was attempted. However, even with single and dual stage pipelining, we observed setup time violations due to the comparisons necessary to account for overflow and underflow caused by

rounding error. As such, we opted to use look-up-tables to weigh Y, Cr, and Cb values and then to sum these weighted values to form the three different pixels. Because the contribution from luminance is identical for all colors, one lookup table with 2^8 entries was used for this factor. Lookup tables with 2^5 entries were used for the contributions from the chrominance values. Finally, adders were instantiated for adding all of these weighted factors.

3 Debugging and Testing

Debugging and testing Verilog-defined hardware is complicated by extremely lengthy synthesis and implementation times. Because of this, we performed as much testing as possible in the ModelSim simulation environment. While ModelSim proved very fast and effective for ensuring that behavioral logic was correct, it failed to help us determine the source of many physical timing bugs, such as setup and hold time violations on communications with the external ZBT memories.

3.1 The Video Pathway

Our testing and integration procedures began by testing that pixels could be read by `ntsc_capture`, written into memory by `memory_interface`, and finally displayed by `vga_write`.

Because the operation of `ntsc_capture` depends greatly on external input (the video camera), our testing procedure for this module began on the labkit. Initially, the logic analyzer was simply set to probe the generated pixel values and ensure that they were generating output that seemed reasonable. When the camera lens was covered for example, the magnitude of pixels generated decreased, which seemed reasonable.

However, after connecting `ntsc_capture` to `memory_interface` and to `vga_write`, we noticed severe distortion in the output video. This was ultimately caused by the aforementioned setup and hold time violation issues, both in writing and reading from ZBT memory, and in writing to the VGA output DAC. Through extensive amounts of timing analysis on the logic analyzer, pipelining of modules, and replacing multipliers by lookup ROMs, these timing problems were ultimately eliminated. As described in `clock_gen` above, adding timing constraints on our clock signals to the Xilinx UCF file helped significantly, as it told Xilinx ISE to generate useful timing reports for finding critical paths through our logic.

After the black and white video output was displayed clearly, other features, such as color processing and color recognition, were added incrementally. We avoided making large changes all at once, as it was often difficult to be certain what effects they would have on the timing of the synthesized logic.

3.2 `memory_interface`

`memory_interface` was tested in stages. Initially, this module was written and tested extensively using testbenches. The first testbench tested whether done flags pulsed two cycles after the request had been fed to the RAM. The second testbench tested whether priority was given to NTSC and VGA. The third testbench tested whether the switching of locations and blocks in memory was carried out effectively. Subsequent testbenches became progressively more complicated, combining several of these scenarios.

In lab, `memory_interface` was debugged by sending test sequences into the RAM, placing requests on the (x,y) coordinates corresponding to the locations that were directly written to, and viewing the result

on the logic analyzer. Once adequate functionality was ensured, this module was tested in conjunction with `vga_display` and `ntsc_capture`. Undesired behavior was seen in the output of `memory_interface` through `vga_display` until the clocks were synthesized correctly and the timing constraints were added to the constraints file. When the constraints were added, the ISE informed us of several critical pathways in `memory_interface`, which we were then able to pipeline, resulting in a delay of four cycles, instead of the original two. However, this new `memory_interface` could now be run at frequencies much higher than the intended 50MHz.

A very sutble bug was introduced when we edited `zbt_6111` and changed it into `zbt_map`: When writing to the RAM, the data was being delayed by one clock cycle, instead of two. This problem manifested itself in very minor distortions in the video output, which we were able to ignore until the end of the development process. Once that typo was fixed, the pathway from `ntsc_capture` to `vga_display` was completely correct.

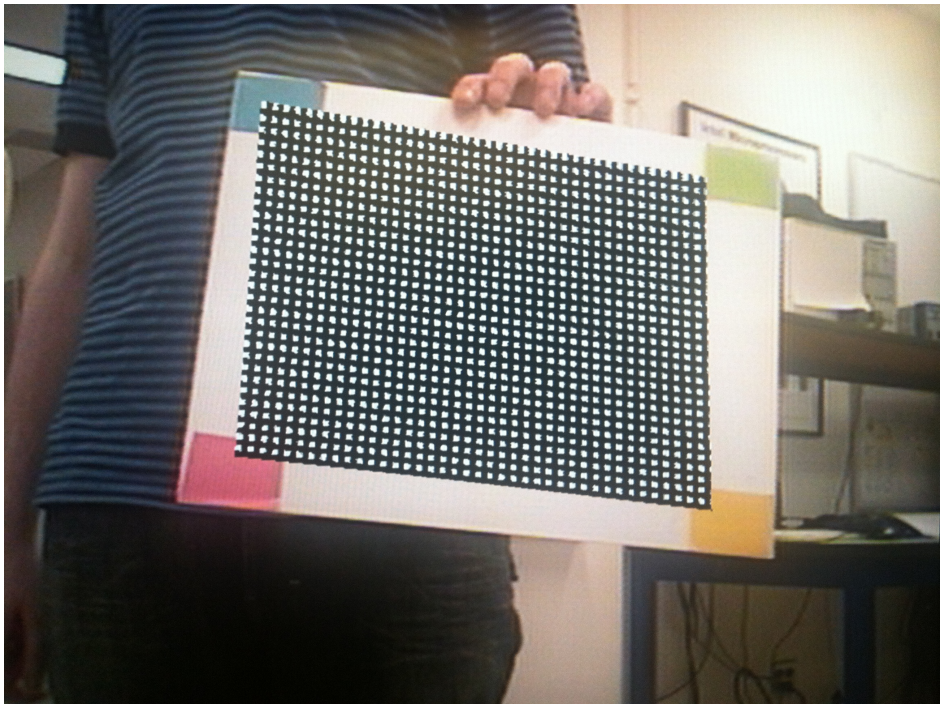

Figure 6: *Processing a test pattern instead of real image data, to decouple* `lpf` *from* `memory_interface`*, and to make bugs in* `projecting_transform` *easier to recognize and diagnose.*

3.3 `vga_display`

`vga_display` went through many iterations as the needs and structure of the project changed. When we settled on using 25MHz and 50MHz clocks, the structure of `vga_display` matured to its current state. While we were considering clocks whose positive edges were not in phase, FIFOs were considered. However, writing a `vga_display` with two FIFOs proved to be very difficult, especially in guaranteeing constant throughput that is required by the video IC.

The `xvga` module was not tested because it had been written and tested by 6.111 staff, and because all that needed to be changed was the bounds on the sync and blank regions, which are clearly defined for our

resolution and refresh rate. [2] Initially, `vga_display` was tested with a test pattern fed directly into the `vga_display` module, based on flag requests and the `hcount` and `vcount` variables. Once this test pattern was displayed appropriately, we linked the module to `memory_interface` and fed a test pattern through the RAM and into `vga_display`. When issues arose, we used the logic analyzer to discern what exactly was happening. YCrCb to RGB conversion was implemented using lookup tables, and resulted in working color almost immediately after writing the LUTs.

3.4 Corner Recognition

Initially, the behavior of `object_recognition` was tested in a simulated testbench in ModelSim. This was a straightforward process, as most of the problems found were due to mistakes in the behavior logic. Once synthesized, the majority of the modifications that needed to be made to corner recognition involved adjusting the thresholding values used in `ntsc_capture` to detect colors. To facilitate this process, we implemented several features.

The module `parameter_set` allows the thresholding parameters to be adjusted in real time, without recompiling the Verilog source code. Color highlighting was added to `ntsc_recognition` so that pixels identified as a certain color would stand out from the rest of the image, and parameters could be tuned to balance the amount of extraneous detection with the intensity of intended detections. Finally, a crosshair display was added to `vga_write`, showing the calculated corner positions on the video output.

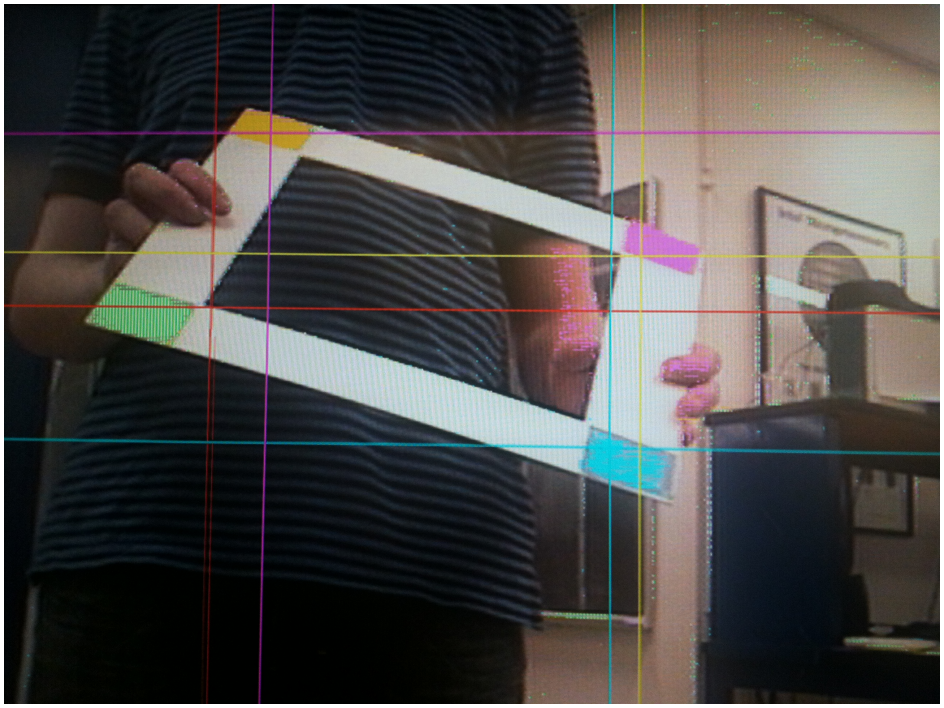

Figure 7: *Highlighting recognized colors, and displaying crosshairs on corners, so that color detection thresholds could be set more precisely.*

3.5 `projective_transform`

To test `projective_transform`, a test jig was developed that provided the module with a list of input pixels, of linearly increasing luminance. The test jig also took the output of `projective_transform`, and saved it into an external file, along with the x and y coordinates of each output pixel. A MATLAB script was then used to display the results of `projective_transform`, along with the original image. The result of this MATLAB script is shown below. In this way, the algorithm used by `projective_transform` to distort the image was known to be correct.

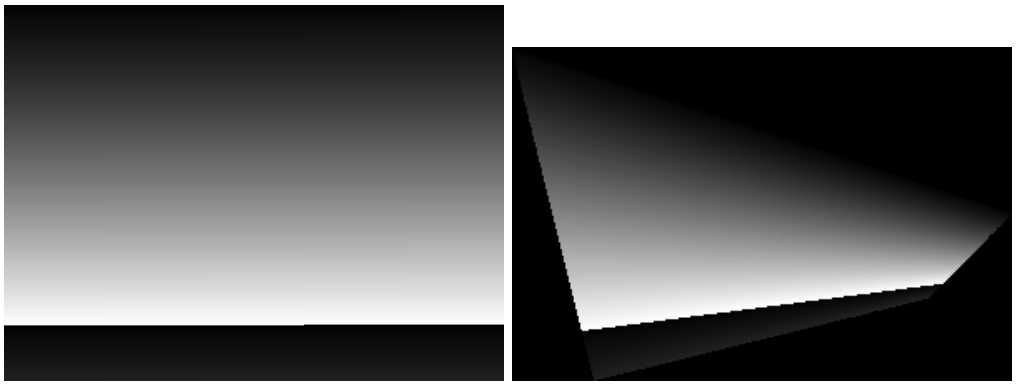

Figure 8: *On the left, the sample input image used to test* `projective_transform`. *On the right, the output image from the testbench.*

However, this behavioral simulation did not locate several problems that we had encountered with propagation delays on the signals that `projective_transform` transmits to `memory_interface`. In order to more easily diagnose these issues, we created a mode where `lpf` sent a test pattern to `projective_transform` instead of real image data. With this mode enabled, it became very clear what the pattern of misplaced and miswritten pixels was.

The logic analyzer also proved quite helpful for discovering these timing violations. However, we occasionally noticed that connecting a wire to the logic analyzer outputs was sufficient to change the behavior of the synthesized hardware, as it changed the synthesized routing path, and thus the routing delay. For this reason, using techniques such as the test pattern above for inferring the sources of bugs was necessary.

4 Conclusion

The Recursive Augmented Reality Image Processing System used digital logic to scale, skew, rotate, and translate a video image to specific coordinates in real time. It was also able to precisely locate the corners of a frame in the image by looking for specific hues. While we did not have time to implement the planned low pass filter, the output of `augreal` was already of an acceptably high quality.

Possible improvements include adding the aforementioned low pass filter, allowing arbitrary image files to be displayed, and improving object recognition to use algorithms that do not rely on hue markers. However, because `augreal` already has a very high throughput of 79 MiB of data per second, and a relatively fast clock frequency of 50 MHz, adding these modification would require extending the length of the image processing pipeline beyond four frame buffers, which would require additional RAM logic.

Overall, we were very pleased with the smoothness of the frame rate, the reliability of the corner recognition, the success of our use of ZBT memory, the appearance of the image projection, and the overall "wow" and "fun" of the system.

References

[1] Malek, Miroslaw. 2002. *Division algorithms.* Available: `http://devel-rok.informatik.hu-berlin.de/svn/TI2/2006/folien/pdf/eng_ca12.pdf`

[2] Ickes, Nathan. 2007. *VGA Video Output* Available: `http://www-mtl.mit.edu/Courses/6.111/labkit/vga.shtml`

[3] Pillai, Latha. 2001. *XAPP283: Color Space Converter* Available: `http://www-mtl.mit.edu/Courses/6.111/labkit/appnotes/xapp283.pdf`

A Project Source Code

A.1 labkit.v

```
1  `default_nettype none
   `include "params.v"
3
   ////////////////////////////////////////////////////////////////////////////
5  //
   // 6.111 FPGA Labkit --- Template Toplevel Module
7  //
   // For Labkit Revision 004
9  //
   //
11 // Created: October 31, 2004, from revision 003 file
   // Author: Nathan Ickes
13 //
   ////////////////////////////////////////////////////////////////////////////
15 //
   // CHANGES FOR BOARD REVISION 004
17 //
   // 1) Added signals for logic analyzer pods 2-4.
19 // 2) Expanded "tv_in_ycrcb" to 20 bits.
   // 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
21 //    "tv_out_i2c_clock".
   // 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
23 //    output of the FPGA, and "in" is an input.
   //
25 // CHANGES FOR BOARD REVISION 003
   //
27 // 1) Combined flash chip enables into a single signal, flash_ce_b.
   //
29 // CHANGES FOR BOARD REVISION 002
   //
31 // 1) Added SRAM clock feedback path input and output
   // 2) Renamed "mousedata" to "mouse_data"
33 // 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
   //    the data bus, and the byte write enables have been combined into the
35 //    4-bit ram#_bwe_b bus.
   // 4) Removed the "systemace_clock" net, since the SystemACE clock is now
37 //    hardwired on the PCB to the oscillator.
   //
39 ////////////////////////////////////////////////////////////////////////////
   //
41 // Complete change history (including bug fixes)
   //
43 // 2006-Mar-08: Corrected default assignments to "vga_out_red", "vga_out_green"
   //              and "vga_out_blue". (Was 10'h0, now 8'h0.)
45 //
   // 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
47 //              "disp_data_out", "analyzer[2-3]_clock" and
```

```
//                "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//                actually populated on the boards. (The boards support up to
//                256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//                value. (Previous versions of this file declared this port to
//                be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
//                actually populated on the boards. (The boards support up to
//                72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
//////////////////////////////////////////////////////////////////////////////

module labkit(beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
              ac97_bit_clock,

              vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
              vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
              vga_out_vsync,

              tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
              tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
              tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

              tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
              tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
              tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
              tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

              ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
              ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

              ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
              ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

              clock_feedback_out, clock_feedback_in,

              flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
              flash_reset_b, flash_sts, flash_byte_b,

              rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

              mouse_clock, mouse_data, keyboard_clock, keyboard_data,

              clock_27mhz, clock1, clock2,
```

```
101                 disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
                    disp_reset_b, disp_data_in,
103
                    button0, button1, button2, button3, button_enter, button_right,
105                 button_left, button_down, button_up,

107                 switch,

109                 led,

111                 user1, user2, user3, user4,

113                 daughtercard,

115                 systemace_data, systemace_address, systemace_ce_b,
                    systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,
117
                    analyzer1_data, analyzer1_clock,
119                 analyzer2_data, analyzer2_clock,
                    analyzer3_data, analyzer3_clock,
121                 analyzer4_data, analyzer4_clock);

123     output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
        input  ac97_bit_clock, ac97_sdata_in;
125
        output [7:0] vga_out_red, vga_out_green, vga_out_blue;
127     output       vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
                     vga_out_hsync, vga_out_vsync;
129
        output [9:0] tv_out_ycrcb;
131     output       tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
                     tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
133                  tv_out_subcar_reset;

135     input [19:0] tv_in_ycrcb;
        input        tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
137                  tv_in_hff, tv_in_aff;
        output       tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
139                  tv_in_reset_b, tv_in_clock;
        inout        tv_in_i2c_data;
141
        inout [35:0] ram0_data;
143     output [18:0] ram0_address;
        output        ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
145     output [3:0]  ram0_bwe_b;

147     inout [35:0]  ram1_data;
        output [18:0] ram1_address;
149     output        ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
        output [3:0]  ram1_bwe_b;
151
```

```
    input           clock_feedback_in;
153 output          clock_feedback_out;

155 inout [15:0]    flash_data;
    output [23:0]   flash_address;
157 output          flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
    input           flash_sts;
159
    output          rs232_txd, rs232_rts;
161 input           rs232_rxd, rs232_cts;

163 input           mouse_clock, mouse_data, keyboard_clock, keyboard_data;

165 input           clock_27mhz, clock1, clock2;

167 output          disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
    input           disp_data_in;
169 output          disp_data_out;

171 input           button0, button1, button2, button3, button_enter, button_right,
                    button_left, button_down, button_up;
173 input [7:0]     switch;
    output [7:0]    led;
175
    inout [31:0]    user1, user2, user3, user4;
177
    inout [43:0]    daughtercard;
179
    inout [15:0]    systemace_data;
181 output [6:0]    systemace_address;
    output          systemace_ce_b, systemace_we_b, systemace_oe_b;
183 input           systemace_irq, systemace_mpbrdy;

185 output [15:0]   analyzer1_data, analyzer2_data, analyzer3_data,
                    analyzer4_data;
187 output          analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

189 //////////////////////////////////////////////////////////////////////
    //
191 // Clock Assignments
    //
193 //////////////////////////////////////////////////////////////////////
    wire            clock_50mhz, clock_50mhz_90, clock_50mhz_270;
195 wire            clock_25mhz, locked_ram, locked_25mhz;
    clock_gen cgen(.reset_button(button0), .clock_27mhz(clock_27mhz),
197                 .clock_feedback_in(clock_feedback_in),
                    .clock_feedback_out(clock_feedback_out),
199                 .clock_50mhz(clock_50mhz), .clock_25mhz(clock_25mhz),
                    .clock_50mhz_90(clock_50mhz_90),
201                 .clock_50mhz_270(clock_50mhz_270),
                    .ram0_clk(ram0_clk), .ram1_clk(ram1_clk),
203                 .locked_ram(locked_ram), .locked_25mhz(locked_25mhz));
```

```
    assign led[0] = ~locked_ram;
    assign led[1] = ~locked_25mhz;

    ////////////////////////////////////////////////////////////////////////
    //
    // 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;

    // SRAMs
    // 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


    // Buttons, Switches, and Individual LEDs
    assign led[7:2] = 6'b111111;
```

```
    // button0, button1, button2, button3, button_enter, button_right,
257 // button_left, button_down, button_up, and switches are inputs

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

265 // Daughtercard Connectors
    assign daughtercard = 44'hZ;
267
    // SystemACE Microprocessor Port
269 assign systemace_data = 16'hZ;
    assign systemace_address = 7'h0;
271 assign systemace_ce_b = 1'b1;
    assign systemace_we_b = 1'b1;
273 assign systemace_oe_b = 1'b1;
    // systemace_irq and systemace_mpbrdy are inputs
275
    ////////////////////////////////////////////////////////////////////////
277 //
    // Reset Generation
279 //
    // A shift register primitive is used to generate an active-high reset
281 // signal that remains high for 16 clock cycles after configuration finishes
    // and the FPGA's internal clocks begin toggling.
283 //
    ////////////////////////////////////////////////////////////////////////
285 wire          reset;
    SRL16 reset_sr(.D(1'b0), .CLK(clock_27mhz), .Q(reset),
287                .A0(1'b1), .A1(1'b1), .A2(1'b1), .A3(1'b1));
    defparam reset_sr.INIT = 16'hFFFF;
289
    ////////////////////////////////////////////////////////////////////////
291 //
    // OUR MODULES: ntsc_capture
293 //                          memory_interface
    //                          lpf
295 //                          projective_transform,
    //
297 ////////////////////////////////////////////////////////////////////////

299 wire          frame_flag;
    wire          done_ntsc;
301 wire          done_pt;
    wire          done_lpf;
303 wire          done_vga;
    wire          ntsc_flag;
305 wire          pt_flag;
    wire          lpf_flag;
307 wire          vga_flag;
```

```
    wire [35:0]    ntsc_pixels;
309 wire [35:0]    vga_pixel;
    wire           vwr;
311
    /***********************************
313  ******* NTSC BLOCK ******************
     ***********************************/
315
    assign tv_in_fifo_read = 1'b0;
317 assign tv_in_fifo_clock = 1'b0;
    assign tv_in_iso = 1'b0;
319 assign tv_in_clock = clock_27mhz;

321 wire           dv;
    wire [2:0]     fvh;
323 wire [9:0]     nx;
    wire [8:0]     ny;
325
    wire           ntsc_flag_cleaned;
327 wire           frame_flag_cleaned;
    wire           debug_state;
329
    wire [9:0]     ntsc_x;
331 wire [8:0]     ntsc_y;

333 wire [3:0]     nr;
    wire [9:0]     midcr, midcb, midy;
335 wire [1:0]     color;
    wire           i_flag;
337 wire           nwr;

339 // PARAMETER SELECTION AND SETTING
    wire [9:0]     GREEN_LUM_MAX;
341 wire [9:0]     GREEN_LUM_MIN;
    wire [9:0]     GREEN_CR_MAX;
343 wire [9:0]     GREEN_CR_MIN;
    wire [9:0]     GREEN_CB_MAX;
345 wire [9:0]     GREEN_CB_MIN;

347 wire [9:0]     ORANGE_LUM_MAX;
    wire [9:0]     ORANGE_LUM_MIN;
349 wire [9:0]     ORANGE_CR_MAX;
    wire [9:0]     ORANGE_CR_MIN;
351 wire [9:0]     ORANGE_CB_MAX;
    wire [9:0]     ORANGE_CB_MIN;
353
    wire [9:0]     PINK_LUM_MAX;
355 wire [9:0]     PINK_LUM_MIN;
    wire [9:0]     PINK_CR_MAX;
357 wire [9:0]     PINK_CR_MIN;
    wire [9:0]     PINK_CB_MAX;
359 wire [9:0]     PINK_CB_MIN;
```

```
361     wire [9:0]     BLUE_LUM_MAX;
        wire [9:0]     BLUE_LUM_MIN;
363     wire [9:0]     BLUE_CR_MAX;
        wire [9:0]     BLUE_CR_MIN;
365     wire [9:0]     BLUE_CB_MAX;
        wire [9:0]     BLUE_CB_MIN;
367
        wire [63:0]    hex_output;
369
        parameter_set pset(
371                       .clock(clock_50mhz), .reset(reset),
                          .switch(switch[7:3]) ,.hex_output(hex_output),
373                       .GREEN_LUM_MAX(GREEN_LUM_MAX),
                          .GREEN_LUM_MIN(GREEN_LUM_MIN),
375                       .GREEN_CR_MAX(GREEN_CR_MAX),
                          .GREEN_CR_MIN(GREEN_CR_MIN),
377                       .GREEN_CB_MAX(GREEN_CB_MAX),
                          .GREEN_CB_MIN(GREEN_CB_MIN),
379
                          .ORANGE_LUM_MAX(ORANGE_LUM_MAX),
381                       .ORANGE_LUM_MIN(ORANGE_LUM_MIN),
                          .ORANGE_CR_MAX(ORANGE_CR_MAX),
383                       .ORANGE_CR_MIN(ORANGE_CR_MIN),
                          .ORANGE_CB_MAX(ORANGE_CB_MAX),
385                       .ORANGE_CB_MIN(ORANGE_CB_MIN),

387                       .PINK_LUM_MAX(PINK_LUM_MAX),
                          .PINK_LUM_MIN(PINK_LUM_MIN),
389                       .PINK_CR_MAX(PINK_CR_MAX),
                          .PINK_CR_MIN(PINK_CR_MIN),
391                       .PINK_CB_MAX(PINK_CB_MAX),
                          .PINK_CB_MIN(PINK_CB_MIN),
393
                          .BLUE_LUM_MAX(BLUE_LUM_MAX),
395                       .BLUE_LUM_MIN(BLUE_LUM_MIN),
                          .BLUE_CR_MAX(BLUE_CR_MAX),
397                       .BLUE_CR_MIN(BLUE_CR_MIN),
                          .BLUE_CB_MAX(BLUE_CB_MAX),
399                       .BLUE_CB_MIN(BLUE_CB_MIN));

401     display_16hex ds(reset, clock_27mhz, hex_output,
                         disp_blank, disp_clock, disp_rs, disp_ce_b,
403                      disp_reset_b, disp_data_out);
        // END PARAMETER_SET
405
        wire           enable_highlighting_and_xhairs;
407     debounce db6(
                    .clock(clock_50mhz), .reset(reset), .noisy(switch[2]),
409                 .clean(enable_highlighting_and_xhairs));


411
```

```
    ntsc_capture ntsc(
413                         .clock_50mhz(clock_50mhz_90),
                            .clock_27mhz(clock_27mhz),
415                         .reset(reset),
                            .tv_in_reset_b(tv_in_reset_b),
417                         .tv_in_i2c_clock(tv_in_i2c_clock),
                            .tv_in_i2c_data(tv_in_i2c_data),
419                         .tv_in_line_clock1(tv_in_line_clock1),
                            .tv_in_ycrcb(tv_in_ycrcb),
421                         .ntsc_pixels(ntsc_pixels),
                            .ntsc_flag(ntsc_flag_cleaned),
423                         .o_frame_flag(frame_flag_cleaned),
                            .o_x(ntsc_x),
425                         .o_y(ntsc_y),
                            .read_state_out(wr_ack),
427                         .wr_en(wr_en),
                            .empty(empty),
429                         .ntsc_raw(nr),
                            .midcr(midcr),
431                         .midcb(midcb),
                            .midy(midy),
433                         .o_i_flag(i_flag),
                            .o_color(color),
435                         .ntsc_will_request(nwr),
                            .GREEN_LUM_MAX(GREEN_LUM_MAX),
437                         .GREEN_LUM_MIN(GREEN_LUM_MIN),
                            .GREEN_CR_MAX(GREEN_CR_MAX),
439                         .GREEN_CR_MIN(GREEN_CR_MIN),
                            .GREEN_CB_MAX(GREEN_CB_MAX),
441                         .GREEN_CB_MIN(GREEN_CB_MIN),

443                         .ORANGE_LUM_MAX(ORANGE_LUM_MAX),
                            .ORANGE_LUM_MIN(ORANGE_LUM_MIN),
445                         .ORANGE_CR_MAX(ORANGE_CR_MAX),
                            .ORANGE_CR_MIN(ORANGE_CR_MIN),
447                         .ORANGE_CB_MAX(ORANGE_CB_MAX),
                            .ORANGE_CB_MIN(ORANGE_CB_MIN),
449
                            .PINK_LUM_MAX(PINK_LUM_MAX),
451                         .PINK_LUM_MIN(PINK_LUM_MIN),
                            .PINK_CR_MAX(PINK_CR_MAX),
453                         .PINK_CR_MIN(PINK_CR_MIN),
                            .PINK_CB_MAX(PINK_CB_MAX),
455                         .PINK_CB_MIN(PINK_CB_MIN),

457                         .BLUE_LUM_MAX(BLUE_LUM_MAX),
                            .BLUE_LUM_MIN(BLUE_LUM_MIN),
459                         .BLUE_CR_MAX(BLUE_CR_MAX),
                            .BLUE_CR_MIN(BLUE_CR_MIN),
461                         .BLUE_CB_MAX(BLUE_CB_MAX),
                            .BLUE_CB_MIN(BLUE_CB_MIN)
463                         );
```

```
465     /***********************************************
         ************ OBJECT_RECOGNITION BLOCK **************
467      ***********************************************/

469     wire [9:0]     a_x, b_x, c_x, d_x;
        wire [8:0]     a_y, b_y, c_y, d_y;
471     wire           corners_flag;

473     object_recognition objr(
                            .clk(clock_50mhz),
475                         .reset(reset),
                            .color(color),
477                         .interesting_x(ntsc_x),
                            .interesting_y(ntsc_y),
479                         .frame_flag(frame_flag_cleaned),
                            .interesting_flag(i_flag),
481                         .a_x(a_x), .a_y(a_y),
                            .b_x(b_x), .b_y(b_y),
483                         .c_x(c_x), .c_y(c_y),
                            .d_x(d_x), .d_y(d_y),
485                         .corners_flag(corners_flag));

487     /***************************************
         ************ LPF & PT BLOCK **************
489      ***************************************/

491     wire           lpf_wr;
        wire [`LOG_WIDTH-1:0] lpf_x;
493     wire [`LOG_HEIGHT-1:0] lpf_y;
        wire [`LOG_MEM-1:0]    lpf_pixel_write;
495     wire [`LOG_MEM-1:0]    lpf_pixel_read;

497     wire [`LOG_TRUNC-1:0]  pixel_out_lpf;
        wire                   pixel_flag;
499     wire                   request;
        wire [`LOG_TRUNC-1:0]  pt_pixel;
501     wire [`LOG_WIDTH-1:0]  pt_x;
        wire [`LOG_HEIGHT-1:0] pt_y;
503     wire                   pt_wr;
        wire                   ready_pt;
505
        // for writing the test pattern
507     wire                   lpf_testing;
        debounce db4(
509                 .clock(clock_50mhz), .reset(reset), .noisy(~switch[0]),
                    .clean(lpf_testing));
511
        lpf dlpf(
513             .clock(clock_50mhz),
                .reset(reset),
515             .frame_flag(frame_flag_cleaned),
```

```
            .done_lpf(done_lpf),
            .lpf_flag(lpf_flag),
            .lpf_wr(lpf_wr),
            .lpf_x(lpf_x),
            .lpf_y(lpf_y),
            .lpf_pixel_write(lpf_pixel_write),
            .lpf_pixel_read(lpf_pixel_read),
            .request(request),
            .pixel(pixel_out_lpf),
            .pixel_flag(pixel_flag),
            .testing(lpf_testing));

    projective_transform_srl pt(
                                .clk(clock_50mhz),
                                .frame_flag(frame_flag_cleaned),
                                .pixel(pixel_out_lpf),
                                .pixel_flag(pixel_flag),
                                .done_pt(done_pt),
                                .a_x(a_x), .a_y(a_y),
                                .b_x(b_x), .b_y(b_y),
                                .c_x(c_x), .c_y(c_y),
                                .d_x(d_x), .d_y(d_y),
                                .corners_flag(corners_flag),
                                .ptflag(ready_pt),
                                .pt_pixel_write(pt_pixel),
                                .pt_x(pt_x), .pt_y(pt_y),
                                .pt_wr(pt_wr),
                                .request_pixel(request));


    /*************************************************
     ********** MEMORY_INTERFACE BLOCK ******************
     *************************************************/

    // default values
    assign ram0_ce_b = 1'b0;
    assign ram0_oe_b = 1'b0;
    assign ram0_adv_ld = 1'b0;

    assign ram1_ce_b = 1'b0;
    assign ram1_oe_b = 1'b0;
    assign ram1_adv_ld = 1'b0;

    // memory_interface
    wire                    mem0_wr;
    wire                    mem1_wr;
    wire [3:0]              mem0_bwe;
    wire [3:0]              mem1_bwe;
    wire [`LOG_ADDR-1:0]    mem0_addr;
    wire [`LOG_ADDR-1:0]    mem1_addr;
    wire [`LOG_MEM-1:0]     mem0_read;
    wire [`LOG_MEM-1:0]     mem1_read;
```

```
    wire [`LOG_MEM-1:0]     mem0_write;
569 wire [`LOG_MEM-1:0]     mem1_write;
    wire [7:0]              debug_locs;
571 wire [3:0]              debug_blocks;

573 wire [`LOG_HCOUNT-1:0] hcount;
    wire [`LOG_VCOUNT-1:0] vcount;
575
    wire                    mem0_wrt, mem1_wrt, mem0_wrr, mem1_wrr;
577 wire [3:0]              mem0_bwer;
    wire [3:0]              mem1_bwer;
579 wire [35:0]             mem0_writet, mem1_writet, mem0_writer, mem1_writer;
    wire [`LOG_ADDR-1:0]    mem0_addrr, mem1_addrr, mem0_addrt, mem1_addrt;
581
    wire                    enable_lpf_pt;
583 debounce db5(
              .clock(clock_50mhz), .reset(reset), .noisy(~switch[1]),
585           .clean(enable_lpf_pt));

587 memory_interface mi(
                    .clock(clock_50mhz), .reset(reset),
589                 .frame_flag(frame_flag_cleaned), .ntsc_flag(ntsc_flag_cleaned),
                    .ntsc_pixel(ntsc_pixels),.done_ntsc(done_ntsc),
591                 .vga_flag(vga_flag),.done_vga(done_vga),.vga_pixel(vga_pixel),
                    .vcount(vcount), .hcount(hcount), .vsync(vga_out_vsync),
593                 .mem0_addr(mem0_addrr),.mem1_addr(mem1_addrr),
                    .mem0_read(mem0_read),.mem1_read(mem1_read),
595                 .mem0_write(mem0_writer),.mem1_write(mem1_writer),
                    .mem0_wr(mem0_wrr),.mem1_wr(mem1_wrr),
597                 .mem0_bwe(mem0_bwer),.mem1_bwe(mem1_bwer),
                    .ntsc_x(ntsc_x), .ntsc_y(ntsc_y),
599                 .ready_pt(ready_pt),
                    .lpf_flag(lpf_flag & enable_lpf_pt), .lpf_wr(lpf_wr),
601                 .lpf_pixel_read(lpf_pixel_read),
                    .lpf_pixel_write(lpf_pixel_write),
603                 .done_lpf(done_lpf),
                    .lpf_x(lpf_x), .lpf_y(lpf_y),
605                 .pt_x(pt_x), .pt_y(pt_y),
                    .pt_flag(pt_wr & enable_lpf_pt), .pt_pixel(pt_pixel),
607                 .nwr(nwr), .vwr(vwr), .done_pt(done_pt));

609 wire                    enter_clean;
    debounce db3(
611           .clock(clock_50mhz), .reset(reset), .noisy(~button_enter),
              .clean(enter_clean));
613
    // TEST PATTERN FOR TESTING MEMORY_INTERFACE AND OTHER MODULES
615 zbt_test_pattern ztp(
                    .clock(clock_50mhz), .reset(reset), .start(enter_clean),
617                 .mem0_addr(mem0_addrt), .mem1_addr(mem1_addrt),
                    .mem0_write(mem0_writet), .mem1_write(mem1_writet),
619                 .mem0_wr(mem0_wrt), .mem1_wr(mem1_wrt));
```

```
621     assign mem0_wr = (enter_clean) ? mem0_wrt : mem0_wrr;
        assign mem0_bwe = (enter_clean) ? 4'b1111 : mem0_bwer;
623     assign mem0_addr = (enter_clean) ? mem0_addrt : mem0_addrr;
        assign mem0_write = (enter_clean) ? mem0_writet : mem0_writer;
625     assign mem1_wr = (enter_clean) ? mem1_wrt : mem1_wrr;
        assign mem1_bwe = (enter_clean) ? 4'b1111 : mem1_bwer;
627     assign mem1_addr = (enter_clean) ? mem1_addrt : mem1_addrr;
        assign mem1_write = (enter_clean) ? mem1_writet : mem1_writer;
629
        zbt_map mem0(
631                 .clock(clock_50mhz), .cen(1'b1),
                    .we(mem0_wr), .bwe(mem0_bwe), .addr(mem0_addr),
633                 .write_data(mem0_write), .read_data(mem0_read),
                    .ram_we_b(ram0_we_b), .ram_bwe_b(ram0_bwe_b),
635                 .ram_address(ram0_address), .ram_data(ram0_data),
                    .ram_cen_b(ram0_cen_b));
637
        zbt_map mem1(
639                 .clock(clock_50mhz), .cen(1'b1),
                    .we(mem1_wr), .bwe(mem1_bwe), .addr(mem1_addr),
641                 .write_data(mem1_write), .read_data(mem1_read),
                    .ram_we_b(ram1_we_b), .ram_bwe_b(ram1_bwe_b),
643                 .ram_address(ram1_address), .ram_data(ram1_data),
                    .ram_cen_b(ram1_cen_b));
645
        /**********************************
647      *******  VGA BLOCK ******************
         **********************************/
649     vga_write vga(
                    .clock(clock_50mhz), .vclock(clock_25mhz),
651                 .reset(reset), .frame_flag(frame_flag_cleaned),
                    .vga_pixel(vga_pixel),
653                 .done_vga(done_vga), .vga_flag(vga_flag),
                    .vga_out_red(vga_out_red),
655                 .vga_out_green(vga_out_green),
                    .vga_out_blue(vga_out_blue),
657                 .vga_out_sync_b(vga_out_sync_b),
                    .vga_out_blank_b(vga_out_blank_b),
659                 .vga_out_pixel_clock(vga_out_pixel_clock),
                    .vga_out_hsync(vga_out_hsync),
661                 .vga_out_vsync(vga_out_vsync),
                    .clocked_hcount(hcount),
663                 .clocked_vcount(vcount),
                    .a_x(a_x), .a_y(a_y),
665                 .b_x(b_x), .b_y(b_y),
                    .c_x(c_x), .c_y(c_y),
667                 .d_x(d_x), .d_y(d_y),
                    .vga_will_request(vwr),
669                 .enable_xhairs(enable_highlighting_and_xhairs));

671     /**********************************
```

```
    ******* LOGIC_ANALYZER ***************
673 *************************************/

675   assign analyzer1_clock = 1'b1;
    assign analyzer2_clock = 1'b1;
677   assign analyzer3_clock = 1'b1;
    assign analyzer4_clock = 1'b1;
679   assign analyzer1_data = 16'h0;
    assign analyzer2_data = 16'h0;
681   assign analyzer3_data = 16'h0;
    assign analyzer4_data = 16'h0;
683 endmodule

685 module debounce (input reset, clock, noisy,
                   output reg clean);
687
    reg [19:0]                count;
689   reg                       new;

691   always @(posedge clock)
      if (reset) begin new <= noisy; clean <= noisy; count <= 0; end
693   else if (noisy != new) begin new <= noisy; count <= 0; end
      else if (count == 650000) clean <= new;
695   else count <= count+1;

697 endmodule
```

A.2 ntsc_capture.v

```
1 `default_nettype none
    // comment out when testing
3 `include "params.v"

5   module ntsc_capture(
                        input             clock_50mhz, // the main system clock
7                       input             clock_27mhz,
                        input             reset, // reset line
9
                        output            tv_in_reset_b, // these are all labkit wires
11                      output            tv_in_i2c_clock, //   |
                        inout             tv_in_i2c_data, //    |
13                      input             tv_in_line_clock1, // |
                        input [19:0]      tv_in_ycrcb, //       |
15
                         // outputs two sets of pixels in Y/Cr/Cb/Y/Cr/Cb format
17                      output reg [35:0] ntsc_pixels,
                         // a flag that goes high when a pixel is being output
19                      output reg        ntsc_flag,
                        // these outputs are for object_recognition. this indicates the color
                            of the recognized pixel
21                      output reg [1:0]  o_color,
```

```
                        // a flag that indicates the data is good
23                      output reg          o_i_flag,
                        output reg          o_frame_flag,
25                      output reg [9:0]    o_x,
                        output reg [8:0]    o_y,
27
                        // indicates that ntsc will request
29                      // to write on the next clock cycle
                        output reg          ntsc_will_request,
31
                        // thresholding parameters
33                      input [9:0]         GREEN_LUM_MAX,
                        input [9:0]         GREEN_LUM_MIN,
35                      input [9:0]         GREEN_CR_MAX,
                        input [9:0]         GREEN_CR_MIN,
37                      input [9:0]         GREEN_CB_MAX,
                        input [9:0]         GREEN_CB_MIN,
39
                        input [9:0]         ORANGE_LUM_MAX,
41                      input [9:0]         ORANGE_LUM_MIN,
                        input [9:0]         ORANGE_CR_MAX,
43                      input [9:0]         ORANGE_CR_MIN,
                        input [9:0]         ORANGE_CB_MAX,
45                      input [9:0]         ORANGE_CB_MIN,

47                      input [9:0]         PINK_LUM_MAX,
                        input [9:0]         PINK_LUM_MIN,
49                      input [9:0]         PINK_CR_MAX,
                        input [9:0]         PINK_CR_MIN,
51                      input [9:0]         PINK_CB_MAX,
                        input [9:0]         PINK_CB_MIN,
53
                        input [9:0]         BLUE_LUM_MAX,
55                      input [9:0]         BLUE_LUM_MIN,
                        input [9:0]         BLUE_CR_MAX,
57                      input [9:0]         BLUE_CR_MIN,
                        input [9:0]         BLUE_CB_MAX,
59                      input [9:0]         BLUE_CB_MIN
                    ); // a flag that indicates when a new frame begins
61
    // initialize the adv7185 video ADC
63  adv7185init adv7185(.reset(reset), .clock_27mhz(clock_27mhz), .source(1'b0),
                        .tv_in_reset_b(tv_in_reset_b),
65                      .tv_in_i2c_clock(tv_in_i2c_clock),
                        .tv_in_i2c_data(tv_in_i2c_data));
67
    // this module decodes the data and outputs the ycrcb pair
69  ntsc_decode decode(.clk(tv_in_line_clock1), .reset(reset),
                       .tv_in_ycrcb(tv_in_ycrcb[19:10]), .ycrcb(ycrcb),
71                     .v(fvh[1]), .h(fvh[0]), .data_valid(dv), .f(fvh[2]));

73  // variables for communicating with ntsc decode
```

```
    wire [29:0]                         ycrcb;
75  wire [2:0]                          fvh;
    wire                                dv;
77
    // variables for communicating with the fifo
79  reg [63:0]                          din;
    reg                                 wr_en = 0;
81  wire                                rd_en;
    wire [63:0]                         dout;
83  wire                                full, valid;

85  reg [17:0]                          pixel_buffer;

87  reg                                 read_state;

89  // this is a FIFO module for sending data between the 27 mhz video
    // clock domain and the 50 mhz ram clock domain
91  ntf n2f(.din(din), .rd_clk(clock_50mhz), .rd_en(rd_en),
            .rst(reset), .wr_clk(tv_in_line_clock1), .wr_en(wr_en),
93          .dout(dout), .empty(empty), .full(full), .valid(valid)
            );
95
    reg                                 state = 0;
97  reg [9:0]                           x = 0;
    reg [8:0]                           y = 0;
99
    reg                                 sv;
101 reg                                 sh;
    reg                                 rh;
103 reg                                 rv;

105 reg                                 pulseonce;

107 // create some convenience variables
    wire [9:0]                          cr, cb, lum;
109 wire                                f, v, h;

111 assign cr = ycrcb[19:10];
    assign cb = ycrcb[9:0];
113 assign lum = ycrcb[29:20];

115 assign f = fvh[2];
    assign v = fvh[1];
117 assign h = fvh[0];

119 wire [8:0]                          corrected_y;
    wire [9:0]                          corrected_x;
121
    // create some corrected variables for output coordinates
123 assign corrected_y = y - 23;
    assign corrected_x = x - 10;
125
```

```
    reg                                     orange_match, green_match, pink_match, blue_match;
127
    // latches very short hsync and vsync flags
129 always @(*) begin
       sh <= (~rh & sh) | h;
131    sv <= (~rv & sv) | v;
    end
133
    always @(posedge tv_in_line_clock1) begin
135    if (dv) begin
          // assign match variables based on current chrominance
137       // and luminance data
          orange_match <= ((cb < ORANGE_CB_MAX) &
139                           (cr > ORANGE_CR_MIN) &
                             (lum > ORANGE_LUM_MIN));
141
          green_match <= ((lum < GREEN_LUM_MAX) &
143                       (lum > GREEN_LUM_MIN) &
                         (cr > GREEN_CR_MIN) &
145                      (cr < GREEN_CR_MAX) &
                         (cb > GREEN_CB_MIN) &
147                      (cb < GREEN_CB_MAX) &
                         (y < 480));
149
          blue_match <=  ((cb > BLUE_CB_MIN) &
151                       (cr < BLUE_CR_MAX) &
                         (lum > BLUE_LUM_MIN));
153
          pink_match <= ((cb > PINK_CB_MIN) &
155                      (cr > PINK_CR_MIN) &
                        (lum > PINK_LUM_MIN));
157
       end else if (sh | sv) begin // if (dv)
159       orange_match <= 0;
          green_match <= 0;
161       blue_match <= 0;
          pink_match <= 0;
163    end // else: !if(dv)
    end // always @ (posedge tv_in_line_clock1)
165
    // synchronize to the external video line clock
167 always @ (posedge tv_in_line_clock1) begin

169    // on hsync, increment y by two (because of interlacing) and reset
       // the x variable
171    if (sh) begin
          y <= y + 2;
173       x <= 0;
          rh <= 1; // reset the latch
175    end else if (x > 1) begin
          rh <= 0;
177    end
```

```
179         // on vsync, set the seed y value to the current field (even or odd)
            // reset x
181         if (sv) begin
               y <= f;
183            x <= 0;
               rv <= 1;
185         end else if (y > 1) begin
               rv <= 0;
187         end

189         // at the end of every other frame (interlacing), send a frame_flag
            if (((y > 502) | sv) & f & ~pulseonce) begin
191            din[26] <= 1;
               din[27] <= 0;
193
               wr_en <= 1;
195
               pulseonce <= 1;
197         end else begin
               wr_en <= 0;
199         end

201         // set a state variable so that we don't send multiple frame flags
            if (~f) begin
203            pulseonce <= 0;
            end
205
            if (dv) begin
207            if (y >= 23 && y < 503 && x < 650 && x > 9) begin // above 480 lines are blanked
                  if (state == 0) begin
209
                     // ORANGE
211                  if (orange_match) begin
                        // we have detected a pixel, so spit out an interesting flag, and
213                     // the current coordinates
                        din[27] <= 0; // ntsc_flag
215                     din[4] <= 1; // interesting_flag
                        din[6:5] <= 2'b00; // color
217                     din[25:16] <= corrected_x;
                        din[15:7] <= corrected_y;
219                     wr_en <= 1; // and write it into the FIFO

221                  // GREEN (same as above)
                     end else if (green_match)  begin
223                     din[27] <= 0;
                        din[4] <= 1;
225                     din[6:5] <= 2'b11;
                        din[25:16] <= corrected_x;
227                     din[15:7] <= corrected_y;
                        wr_en <= 1;
229
```

```
            // PINK (same as above)
        end else if (pink_match) begin
            din[27] <= 0;
            din[4] <= 1;
            din[6:5] <= 2'b01;
            din[25:16] <= corrected_x;
            din[15:7] <= corrected_y;
            wr_en <= 1;

        // BLUE (same as above)
        end else if (blue_match) begin
            din[27] <= 0;
            din[4] <= 1;
            din[6:5] <=2'b10;
            din[25:16] <= corrected_x;
            din[15:7] <= corrected_y;
            wr_en <= 1;

        end else begin
            // we only output on state 0 if interesting pixels have been
            // detected

            wr_en <= 0;

            // if we didn't detect a pixel, don't write anything
            // in this state
        end

        // save this pixel to be written in the next state
        pixel_buffer[17:10] <= ycrcb[29:22];
        pixel_buffer[9:5] <= ycrcb[19:15];
        pixel_buffer[4:0] <= ycrcb[9:5];
        state <= 1; // advance the state


    end else begin
        state <= 0;

        din[63:46] <= pixel_buffer;


        // ORANGE
        if (orange_match) begin
            din[4] <= 1; // output an interesting flag if an interesting
            din[6:5] <= 2'b00; // color is detected

        // GREEN
        end else if (green_match) begin
            din[4] <= 1;
            din[6:5] <= 2'b11;

        // PINK
```

```
                end else if (pink_match)  begin
283                 din[4] <= 1;
                    din[6:5] <= 2'b01;
285
                // BLUE
287             end else if (blue_match) begin
                    din[4] <= 1;
289                 din[6:5] <= 2'b10;

291             end else begin // if ((cb > BLUE_CB_MIN) &...
                    din[6:5] <= 2'b00;
293                 din[4] <= 1'b0;

295             end // else: !if((cb > BLUE_CB_MIN) &...

297             din[45:38] <= ycrcb[29:22];
                din[37:33] <= ycrcb[19:15];
299             din[32:28] <= ycrcb[9:5];
                din[27] <= 1; // ntsc_flag
301             din[26] <= 0;
                din[25:16] <= corrected_x;
303             din[15:7] <= corrected_y;

305             wr_en <= 1;

307          end // else: !if(state == 0)

309       end // if (y < 480 && x < 640)

311       x <= x + 1; // increment the x variable

313    end // if (dv)


315
       if (reset) begin // on reset
317       state <= 0; // reset everything
          x<= 0;
319       y <= 0;
       end
321 end // always @ (posedge tv_in_line_clock1)

323 // read from the FIFO whenever it is not empty
    assign rd_en = ~empty;
325
    // Synchronize outputs to main system clock
327 always @ (posedge clock_50mhz) begin
       ntsc_will_request <= 0;
329
       // if the FIFO is not empty
331    if (rd_en) begin
          read_state <= 1;
333       ntsc_will_request <= 1;
```

```
      end else begin
         read_state <= 0;
      end

      // we have data
      if (read_state) begin
         ntsc_pixels <= dout[63: 28];
         ntsc_flag <= dout[27];
         o_frame_flag <= dout[26];
         o_x <= dout[25:16];
         o_y <= dout[15:7];
         o_color <= dout[6:5];
         o_i_flag <= dout[4];
         ntsc_will_request <= 1;

      end else begin
         ntsc_pixels <= 0;
         ntsc_flag <= 0;
         o_frame_flag <= 0;
         o_x <= 0;
         o_y <= 0;
         o_color <= 0;
         o_i_flag <= 0;
      end
   end // always @ (posedge clock_50mhz)

endmodule // ntsc_capture

// 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
```

```
387     input clk;
        input reset;
389     input [9:0] tv_in_ycrcb; // modified for 10 bit input - should be P[19:10]
        output [29:0] ycrcb;
391     output        f;
        output        v;
393     output        h;
        output        data_valid;
395     // output [4:0] state;

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


415


417
        // In the start state, the module doesn't know where
419     // in the sequence of pixels, it is looking.

421     // Once we determine where to start, the FSM goes through a normal
        // sequence of SAV process_YCrCb EAV... repeat
423
        // The data stream looks as follows
425     // SAV_FF | SAV_00 | SAV_00 | SAV_XY | Cb0 | Y0 | Cr1 | Y1 | Cb2 | Y2 | ... | EAV
            sequence
        // There are two things we need to do:
427     //   1. Find the two SAV blocks (stands for Start Active Video perhaps?)
        //   2. Decode the subsequent data
429
        reg [4:0]     current_state = 5'h00;
431     reg [9:0]     y = 10'h000;  // luminance
        reg [9:0]     cr = 10'h000; // chrominance
433     reg [9:0]     cb = 10'h000; // more chrominance

435     //assign      state = current_state;
```

```
437     always @ (posedge clk)
        begin
439         if (reset)
            begin
441
            end
443         else
            begin
445             // 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
447             // transitioning to the new state
                case (current_state)
449               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;
451               SYNC_3: current_state <= (tv_in_ycrcb == 10'h200) ? SAV_f1_cb0 :
                                           (tv_in_ycrcb == 10'h274) ? EAV_f1 :
453                                        (tv_in_ycrcb == 10'h2ac) ? SAV_VBI_f1 :
                                           (tv_in_ycrcb == 10'h2d8) ? EAV_VBI_f1 :
455                                        (tv_in_ycrcb == 10'h31c) ? SAV_f2_cb0 :
                                           (tv_in_ycrcb == 10'h368) ? EAV_f2 :
457                                        (tv_in_ycrcb == 10'h3b0) ? SAV_VBI_f2 :
                                           (tv_in_ycrcb == 10'h3c4) ? EAV_VBI_f2 : SYNC_1;
459
                  SAV_f1_cb0: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_y0;
461               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;
463               SAV_f1_y1: current_state <= (tv_in_ycrcb == 10'h3ff) ? SYNC_1 : SAV_f1_cb0;

465               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;
467               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;
469
                  // These states are here in the event that we want to cover these signals
471               // in the future. For now, they just send the state machine back to SYNC_1
                  EAV_f1: current_state <= SYNC_1;
473               SAV_VBI_f1: current_state <= SYNC_1;
                  EAV_VBI_f1: current_state <= SYNC_1;
475               EAV_f2: current_state <= SYNC_1;
                  SAV_VBI_f2: current_state <= SYNC_1;
477               EAV_VBI_f2: current_state <= SYNC_1;

479             endcase
            end
481     end // always @ (posedge clk)

483 // implement our decoding mechanism

485 wire y_enable;
    wire cr_enable;
487 wire cb_enable;
```

```
489     // if y is coming in, enable the register
        // likewise for cr and cb
491     assign y_enable = (current_state == SAV_f1_y0) ||
                          (current_state == SAV_f1_y1) ||
493                       (current_state == SAV_f2_y0) ||
                          (current_state == SAV_f2_y1);
495     assign cr_enable = (current_state == SAV_f1_cr1) ||
                           (current_state == SAV_f2_cr1);
497     assign cb_enable = (current_state == SAV_f1_cb0) ||
                           (current_state == SAV_f2_cb0);
499
        // f, v, and h only go high when active
501     assign {v,h} = (current_state == SYNC_3) ? tv_in_ycrcb[7:6] : 2'b00;

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

507     reg  f = 0;

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

517 endmodule


519


521 ////////////////////////////////////////////////////////////////////////////////
    //
523 // 6.111 FPGA Labkit -- ADV7185 Video Decoder Configuration Init
    //
525 // Created:
    // Author: Nathan Ickes
527 //
    ////////////////////////////////////////////////////////////////////////////////
529
    ////////////////////////////////////////////////////////////////////////////////
531 // Register 0
    ////////////////////////////////////////////////////////////////////////////////
533
    `define INPUT_SELECT                            4'h0
535 // 0: CVBS on AIN1 (composite video in)
    // 7: Y on AIN2, C on AIN5 (s-video in)
537 // (These are the only configurations supported by the 6.111 labkit hardware)
    `define INPUT_MODE                              4'h0
539 // 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
```

```
591 // 2: Composite =  4.5dB,  s-video =  5.75dB
    // 3: Composite =  1.25dB, s-video =  3.3dB
593 // 4: Composite =  0.0dB,  s-video =  0.0dB
    // 5: Composite = -1.25dB, s-video = -3.0dB
595 // 6: Composite = -1.75dB, s-video = -8.0dB
    // 7: Composite = -3.0dB,  s-video = -8.0dB
597 `define CORING                                  2'h0
    // 0: No coring
599 // 1: Truncate if Y < black+8
    // 2: Truncate if Y < black+16
601 // 3: Truncate if Y < black+32

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

605 ////////////////////////////////////////////////////////////////////////////
    // Register 3
607 ////////////////////////////////////////////////////////////////////////////

609 `define INTERFACE_SELECT                       2'h0
    // 0: Philips-compatible
611 // 1: Broktree API A-compatible
    // 2: Broktree API B-compatible
613 // 3: [Not valid]
    `define OUTPUT_FORMAT                        4'h0
615 // 0: 10-bit @ LLC, 4:2:2 CCIR656
    // 1: 20-bit @ LLC, 4:2:2 CCIR656
617 // 2: 16-bit @ LLC, 4:2:2 CCIR656
    // 3: 8-bit @ LLC, 4:2:2 CCIR656
619 // 4: 12-bit @ LLC, 4:1:1
    // 5-F: [Not valid]
621 // (Note that the 6.111 labkit hardware provides only a 10-bit interface to
    // the ADV7185.)
623 `define TRISTATE_OUTPUT_DRIVERS                1'b0
    // 0: Drivers tristated when ~OE is high
625 // 1: Drivers always tristated
    `define VBI_ENABLE                           1'b0
627 // 0: Decode lines during vertical blanking interval
    // 1: Decode only active video regions
629
    `define ADV7185_REGISTER_3 {`VBI_ENABLE, `TRISTATE_OUTPUT_DRIVERS, `OUTPUT_FORMAT,
        `INTERFACE_SELECT}
631
    ////////////////////////////////////////////////////////////////////////////
633 // Register 4
    ////////////////////////////////////////////////////////////////////////////
635
    `define OUTPUT_DATA_RANGE                    1'b0
637 // 0: Output values restricted to CCIR-compliant range
    // 1: Use full output range
639 `define BT656_TYPE                           1'b0
    // 0: BT656-3-compatible
641 // 1: BT656-4-compatible
```

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

645 ////////////////////////////////////////////////////////////////////////////
    // Register 5
647 ////////////////////////////////////////////////////////////////////////////

649
    `define GENERAL_PURPOSE_OUTPUTS                    4'b0000
651 `define GPO_0_1_ENABLE                              1'b0
    // 0: General purpose outputs 0 and 1 tristated
653 // 1: General purpose outputs 0 and 1 enabled
    `define GPO_2_3_ENABLE                             1'b0
655 // 0: General purpose outputs 2 and 3 tristated
    // 1: General purpose outputs 2 and 3 enabled
657 `define BLANK_CHROMA_IN_VBI                        1'b1
    // 0: Chroma decoded and output during vertical blanking
659 // 1: Chroma blanked during vertical blanking
    `define HLOCK_ENABLE                               1'b0
661 // 0: GPO 0 is a general purpose output
    // 1: GPO 0 shows HLOCK status
663
    `define ADV7185_REGISTER_5 {`HLOCK_ENABLE, `BLANK_CHROMA_IN_VBI, `GPO_2_3_ENABLE,
        `GPO_0_1_ENABLE, `GENERAL_PURPOSE_OUTPUTS}
665
    ////////////////////////////////////////////////////////////////////////////
667 // Register 7
    ////////////////////////////////////////////////////////////////////////////
669
    `define FIFO_FLAG_MARGIN                           5'h10
671 // Sets the locations where FIFO almost-full and almost-empty flags are set
    `define FIFO_RESET                                 1'b0
673 // 0: Normal operation
    // 1: Reset FIFO. This bit is automatically cleared
675 `define AUTOMATIC_FIFO_RESET                       1'b0
    // 0: No automatic reset
677 // 1: FIFO is autmatically reset at the end of each video field
    `define FIFO_FLAG_SELF_TIME                        1'b1
679 // 0: FIFO flags are synchronized to CLKIN
    // 1: FIFO flags are synchronized to internal 27MHz clock
681
    `define ADV7185_REGISTER_7 {`FIFO_FLAG_SELF_TIME, `AUTOMATIC_FIFO_RESET, `FIFO_RESET,
        `FIFO_FLAG_MARGIN}
683
    ////////////////////////////////////////////////////////////////////////////
685 // Register 8
    ////////////////////////////////////////////////////////////////////////////
687
    `define INPUT_CONTRAST_ADJUST                          8'h80
689
    `define ADV7185_REGISTER_8 {`INPUT_CONTRAST_ADJUST}
691
```

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

```
793 `define PEAK_WHITE_UPDATE                          1'b1
    // 0: Update gain once per line
795 // 1: Update gain once per field
    `define AVERAGE_BIRIGHTNESS_LINES                 1'b1
797 // 0: Use lines 33 to 310
    // 1: Use lines 33 to 270
799 `define MAXIMUM_IRE                               3'h0
    // 0: PAL: 133, NTSC: 122
801 // 1: PAL: 125, NTSC: 115
    // 2: PAL: 120, NTSC: 110
803 // 3: PAL: 115, NTSC: 105
    // 4: PAL: 110, NTSC: 100
805 // 5: PAL: 105, NTSC: 100
    // 6-7: PAL: 100, NTSC: 100
807 `define COLOR_KILL                                1'b1
    // 0: Disable color kill
809 // 1: Enable color kill

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

813 `define ADV7185_REGISTER_10 8'h00
    `define ADV7185_REGISTER_11 8'h00
815 `define ADV7185_REGISTER_12 8'h00
    `define ADV7185_REGISTER_13 8'h45
817 `define ADV7185_REGISTER_14 8'h18
    `define ADV7185_REGISTER_15 8'h60
819 `define ADV7185_REGISTER_16 8'h00
    `define ADV7185_REGISTER_17 8'h01
821 `define ADV7185_REGISTER_18 8'h00
    `define ADV7185_REGISTER_19 8'h10
823 `define ADV7185_REGISTER_1A 8'h10
    `define ADV7185_REGISTER_1B 8'hF0
825 `define ADV7185_REGISTER_1C 8'h16
    `define ADV7185_REGISTER_1D 8'h01
827 `define ADV7185_REGISTER_1E 8'h00
    `define ADV7185_REGISTER_1F 8'h3D
829 `define ADV7185_REGISTER_20 8'hD0
    `define ADV7185_REGISTER_21 8'h09
831 `define ADV7185_REGISTER_22 8'h8C
    `define ADV7185_REGISTER_23 8'hE2
833 `define ADV7185_REGISTER_24 8'h1F
    `define ADV7185_REGISTER_25 8'h07
835 `define ADV7185_REGISTER_26 8'hC2
    `define ADV7185_REGISTER_27 8'h58
837 `define ADV7185_REGISTER_28 8'h3C
    `define ADV7185_REGISTER_29 8'h00
839 `define ADV7185_REGISTER_2A 8'h00
    `define ADV7185_REGISTER_2B 8'hA0
841 `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

   //
```

```
895   // Generate a 1MHz for the I2C driver (resulting I2C clock rate is 250kHz)
      //
897
      reg [7:0] clk_div_count, reset_count;
899   reg       clock_slow;
      wire      reset_slow;
901
      initial
903     begin
          clk_div_count <= 8'h00;
905       // synthesis attribute init of clk_div_count is "00";
          clock_slow <= 1'b0;
907       // synthesis attribute init of clock_slow is "0";
        end
909
      always @(posedge clock_27mhz)
911     if (clk_div_count == 26)
          begin
913         clock_slow <= ~clock_slow;
            clk_div_count <= 0;
915       end
        else
917       clk_div_count <= clk_div_count+1;

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

925   assign reset_slow = reset_count != 0;

927   //
      // I2C driver
929   //

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

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

939   //
      // State machine
941   //

943   reg [7:0] state;
      reg       tv_in_reset_b;
945   reg       old_source;
```

```
947     always @(posedge clock_slow)
          if (reset_slow)
949         begin
              state <= 0;
951           load <= 0;
              tv_in_reset_b <= 0;
953           old_source <= 0;
            end
955       else
            case (state)
957           8'h00:
                begin
959               // Assert reset
                  load <= 1'b0;
961               tv_in_reset_b <= 1'b0;
                  if (!ack)
963                 state <= state+1;
                end
965           8'h01:
                state <= state+1;
967           8'h02:
                begin
969               // Release reset
                  tv_in_reset_b <= 1'b1;
971               state <= state+1;
                end
973           8'h03:
                begin
975               // Send ADV7185 address
                  data <= 8'h8A;
977               load <= 1'b1;
                  if (ack)
979                 state <= state+1;
                end
981           8'h04:
                begin
983               // Send subaddress of first register
                  data <= 8'h00;
985               if (ack)
                    state <= state+1;
987             end
              8'h05:
989             begin
                  // Write to register 0
991               data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
                  if (ack)
993                 state <= state+1;
                end
995           8'h06:
                begin
997               // Write to register 1
                  data <= `ADV7185_REGISTER_1;
```

```
999                 if (ack)
                      state <= state+1;
1001              end
                  8'h07:
1003              begin
                    // Write to register 2
1005                data <= `ADV7185_REGISTER_2;
                    if (ack)
1007                  state <= state+1;
                  end
1009              8'h08:
                  begin
1011                // Write to register 3
                    data <= `ADV7185_REGISTER_3;
1013                if (ack)
                      state <= state+1;
1015              end
                  8'h09:
1017              begin
                    // Write to register 4
1019                data <= `ADV7185_REGISTER_4;
                    if (ack)
1021                  state <= state+1;
                  end
1023              8'h0A:
                  begin
1025                // Write to register 5
                    data <= `ADV7185_REGISTER_5;
1027                if (ack)
                      state <= state+1;
1029              end
                  8'h0B:
1031              begin
                    // Write to register 6
1033                data <= 8'h00; // Reserved register, write all zeros
                    if (ack)
1035                  state <= state+1;
                  end
1037              8'h0C:
                  begin
1039                // Write to register 7
                    data <= `ADV7185_REGISTER_7;
1041                if (ack)
                      state <= state+1;
1043              end
                  8'h0D:
1045              begin
                    // Write to register 8
1047                data <= `ADV7185_REGISTER_8;
                    if (ack)
1049                  state <= state+1;
                  end
```

```
1051            8'h0E:
                  begin
1053                 // Write to register 9
                     data <= `ADV7185_REGISTER_9;
1055                 if (ack)
                       state <= state+1;
1057             end
                 8'h0F: begin
1059                // Write to register A
                    data <= `ADV7185_REGISTER_A;
1061                if (ack)
                      state <= state+1;
1063            end
                 8'h10:
1065               begin
                     // Write to register B
1067                 data <= `ADV7185_REGISTER_B;
                     if (ack)
1069                   state <= state+1;
                 end
1071             8'h11:
                   begin
1073                 // Write to register C
                     data <= `ADV7185_REGISTER_C;
1075                 if (ack)
                       state <= state+1;
1077             end
                 8'h12:
1079               begin
                     // Write to register D
1081                 data <= `ADV7185_REGISTER_D;
                     if (ack)
1083                   state <= state+1;
                 end
1085             8'h13:
                   begin
1087                 // Write to register E
                     data <= `ADV7185_REGISTER_E;
1089                 if (ack)
                       state <= state+1;
1091             end
                 8'h14:
1093               begin
                     // Write to register F
1095                 data <= `ADV7185_REGISTER_F;
                     if (ack)
1097                   state <= state+1;
                 end
1099             8'h15:
                   begin
1101                 // Wait for I2C transmitter to finish
                     load <= 1'b0;
```

```
1103                if (idle)
                      state <= state+1;
1105            end
              8'h16:
1107            begin
                   // Write address
1109               data <= 8'h8A;
                   load <= 1'b1;
1111               if (ack)
                      state <= state+1;
1113            end
              8'h17:
1115            begin
                   data <= 8'h33;
1117               if (ack)
                      state <= state+1;
1119            end
              8'h18:
1121            begin
                   data <= `ADV7185_REGISTER_33;
1123               if (ack)
                      state <= state+1;
1125            end
              8'h19:
1127            begin
                   load <= 1'b0;
1129               if (idle)
                      state <= state+1;
1131            end

1133          8'h1A: begin
                  data <= 8'h8A;
1135              load <= 1'b1;
                  if (ack)
1137                state <= state+1;
              end
1139          8'h1B:
                begin
1141               data <= 8'h33;
                   if (ack)
1143                  state <= state+1;
                end
1145          8'h1C:
                begin
1147               load <= 1'b0;
                   if (idle)
1149                  state <= state+1;
                end
1151          8'h1D:
                begin
1153               load <= 1'b1;
                   data <= 8'h8B;
```

```
1155                if (ack)
                      state <= state+1;
1157            end
              8'h1E:
1159            begin
                   data <= 8'hFF;
1161               if (ack)
                      state <= state+1;
1163            end
              8'h1F:
1165            begin
                   load <= 1'b0;
1167               if (idle)
                      state <= state+1;
1169            end
              8'h20:
1171            begin
                   // Idle
1173               if (old_source != source) state <= state+1;
                   old_source <= source;
1175            end
              8'h21: begin
1177              // Send ADV7185 address
                  data <= 8'h8A;
1179              load <= 1'b1;
                  if (ack) state <= state+1;
1181          end
              8'h22: begin
1183              // Send subaddress of register 0
                  data <= 8'h00;
1185              if (ack) state <= state+1;
              end
1187          8'h23: begin
                  // Write to register 0
1189              data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
                  if (ack) state <= state+1;
1191          end
              8'h24: begin
1193              // Wait for I2C transmitter to finish
                  load <= 1'b0;
1195              if (idle) state <= 8'h20;
              end
1197        endcase

1199  endmodule

1201  // i2c module for use with the ADV7185

1203  module i2c (reset, clock4x, data, load, idle, ack, scl, sda);

1205     input reset;
      input clock4x;
```

```
1207    input [7:0] data;
        input       load;
1209    output      ack;
        output      idle;
1211    output      scl;
        output      sda;
1213
        reg [7:0]   ldata;
1215    reg         ack, idle;
        reg         scl;
1217    reg         sdai;

1219    reg [7:0]   state;

1221    assign sda = sdai ? 1'bZ : 1'b0;

1223    always @(posedge clock4x)
          if (reset)
1225        begin
              state <= 0;
1227          ack <= 0;
            end
1229      else
            case (state)
1231          8'h00: // idle
                begin
1233              scl <= 1'b1;
                  sdai <= 1'b1;
1235              ack <= 1'b0;
                  idle <= 1'b1;
1237              if (load)
                    begin
1239                  ldata <= data;
                      ack <= 1'b1;
1241                  state <= state+1;
                    end
1243            end
              8'h01: // Start
1245            begin
                  ack <= 1'b0;
1247              idle <= 1'b0;
                  sdai <= 1'b0;
1249              state <= state+1;
                end
1251          8'h02:
                begin
1253              scl <= 1'b0;
                  state <= state+1;
1255            end
              8'h03: // Send bit 7
1257            begin
                  ack <= 1'b0;
```

```
1259                sdai <= ldata[7];
                    state <= state+1;
1261              end
                8'h04:
1263              begin
                    scl <= 1'b1;
1265                state <= state+1;
                  end
1267            8'h05:
                  begin
1269                state <= state+1;
                  end
1271            8'h06:
                  begin
1273                scl <= 1'b0;
                    state <= state+1;
1275              end
                8'h07:
1277              begin
                    sdai <= ldata[6];
1279                state <= state+1;
                  end
1281            8'h08:
                  begin
1283                scl <= 1'b1;
                    state <= state+1;
1285              end
                8'h09:
1287              begin
                    state <= state+1;
1289              end
                8'h0A:
1291              begin
                    scl <= 1'b0;
1293                state <= state+1;
                  end
1295            8'h0B:
                  begin
1297                sdai <= ldata[5];
                    state <= state+1;
1299              end
                8'h0C:
1301              begin
                    scl <= 1'b1;
1303                state <= state+1;
                  end
1305            8'h0D:
                  begin
1307                state <= state+1;
                  end
1309            8'h0E:
                  begin
```

```
1311                scl <= 1'b0;
                    state <= state+1;
1313              end
                8'h0F:
1315              begin
                    sdai <= ldata[4];
1317                state <= state+1;
                  end
1319            8'h10:
                  begin
1321                scl <= 1'b1;
                    state <= state+1;
1323              end
                8'h11:
1325              begin
                    state <= state+1;
1327              end
                8'h12:
1329              begin
                    scl <= 1'b0;
1331                state <= state+1;
                  end
1333            8'h13:
                  begin
1335                sdai <= ldata[3];
                    state <= state+1;
1337              end
                8'h14:
1339              begin
                    scl <= 1'b1;
1341                state <= state+1;
                  end
1343            8'h15:
                  begin
1345                state <= state+1;
                  end
1347            8'h16:
                  begin
1349                scl <= 1'b0;
                    state <= state+1;
1351              end
                8'h17:
1353              begin
                    sdai <= ldata[2];
1355                state <= state+1;
                  end
1357            8'h18:
                  begin
1359                scl <= 1'b1;
                    state <= state+1;
1361              end
                8'h19:
```

```
1363            begin
                    state <= state+1;
1365            end
              8'h1A:
1367            begin
                    scl <= 1'b0;
1369                state <= state+1;
                end
1371          8'h1B:
                begin
1373                sdai <= ldata[1];
                    state <= state+1;
1375            end
              8'h1C:
1377            begin
                    scl <= 1'b1;
1379                state <= state+1;
                end
1381          8'h1D:
                begin
1383                state <= state+1;
                end
1385          8'h1E:
                begin
1387                scl <= 1'b0;
                    state <= state+1;
1389            end
              8'h1F:
1391            begin
                    sdai <= ldata[0];
1393                state <= state+1;
                end
1395          8'h20:
                begin
1397                scl <= 1'b1;
                    state <= state+1;
1399            end
              8'h21:
1401            begin
                    state <= state+1;
1403            end
              8'h22:
1405            begin
                    scl <= 1'b0;
1407                state <= state+1;
                end
1409          8'h23: // Acknowledge bit
                begin
1411                state <= state+1;
                end
1413          8'h24:
                begin
```

```
1415                scl <= 1'b1;
                    state <= state+1;
1417              end
                8'h25:
1419              begin
                    state <= state+1;
1421              end
                8'h26:
1423              begin
                    scl <= 1'b0;
1425                if (load)
                      begin
1427                    ldata <= data;
                        ack <= 1'b1;
1429                    state <= 3;
                      end
1431                else
                      state <= state+1;
1433              end
                8'h27:
1435              begin
                    sdai <= 1'b0;
1437                state <= state+1;
                  end
1439            8'h28:
                  begin
1441                scl <= 1'b1;
                    state <= state+1;
1443              end
                8'h29:
1445              begin
                    sdai <= 1'b1;
1447                state <= 0;
                  end
1449          endcase

1451  endmodule
```

A.3 memory_interface.v

```
1  `default_nettype none
   // comment out when testing
3  `include "params.v"
   // memory_interface
5  // handles EVERYTHING ram related
   // actual ram modules are instantiated in top module
7
   module memory_interface
9          (
                   // STANDARD SIGNALS
11                 input clock,
```

```
                input reset,
13              // NTSC_CAPTURE
                input frame_flag,
15              input ntsc_flag,
                input [`LOG_MEM-1:0] ntsc_pixel,
17              output reg done_ntsc,
                input [`LOG_WIDTH-1:0] ntsc_x,
19              input [`LOG_HEIGHT-1:0] ntsc_y,
                // LPF
21              input lpf_flag,
                input lpf_wr,
23              input [`LOG_WIDTH-1:0] lpf_x,
                input [`LOG_HEIGHT-1:0] lpf_y,
25              input [`LOG_MEM-1:0] lpf_pixel_write,
                output reg done_lpf,
27              output reg [`LOG_MEM-1:0] lpf_pixel_read,
                // PROJECTIVE_TRANSFORM
29              input pt_flag,
                input [`LOG_WIDTH-1:0] pt_x,
31              input [`LOG_HEIGHT-1:0] pt_y,
                input [`LOG_TRUNC-1:0] pt_pixel,
33              output reg done_pt,
                output reg ready_pt,
35              // VGA_WRITE
                input vga_flag,
37              output reg done_vga,
                output reg [`LOG_MEM-1:0] vga_pixel,
39              input [`LOG_VCOUNT-1:0] vcount,
                input [`LOG_HCOUNT-1:0] hcount,
41              input vsync,
                // MEMORY
43              // MEM ADDRESSES
                output [`LOG_ADDR-1:0] mem0_addr,
45              output [`LOG_ADDR-1:0] mem1_addr,
                // MEM READ
47              input [`LOG_MEM-1:0] mem0_read,
                input [`LOG_MEM-1:0] mem1_read,
49              // MEM WRITE
                output reg [`LOG_MEM-1:0] mem0_write,
51              output reg [`LOG_MEM-1:0] mem1_write,
                // WR FLAGS
53              output reg mem0_wr,
                output reg mem1_wr,
55              // BWE FLAGS
                output reg [3:0] mem0_bwe,
57              output reg [3:0] mem1_bwe,

59              input nwr,
                input vwr,
61
                // TESTING
63              output [3:0] debug_blocks,
```

```
            output [7:0] debug_locs,

            output pt_conflict,
            output vga_lpf_conflict
        );

        /******** PARAMETERS ********/
        // MODULE ORDINALS
        parameter NTSC = 4'b1000;
        parameter VGA  = 4'b0100;
        parameter LPF  = 4'b0010;
        parameter PT   = 4'b0001;
        parameter NONE = 4'b0000;
        parameter LOG_ORD = 4;
        /***************************/

        // BLOCK OF SRAM IMAGE IS IN
        reg capt_mem_block;
        reg proc_mem_block;
        reg nexd_mem_block;
        reg disp_mem_block;

        // LOCATIONS OF IMAGES IN EACH BLOCK
        reg [1:0] capt_mem_loc;
        reg [1:0] proc_mem_loc;
        reg [1:0] nexd_mem_loc;
        reg [1:0] disp_mem_loc;

        // ADDRESSES
        wire [`LOG_ADDR-1:0] ntsc_addr;
        wire [`LOG_ADDR-1:0] vga_addr;
        wire [`LOG_ADDR-1:0] lpf_addr;
        wire [`LOG_ADDR-1:0] pt_addr;

        // PARTIAL DONE FLAGS
        reg [3:0] mem0_done;
        reg [3:0] mem1_done;

        // READ QUEUES
        reg [LOG_ORD-1:0] mem0_read_queue;
        reg [LOG_ORD-1:0] mem1_read_queue;
        // ELEMENT TO BE OUTPUT FROM MEM AT NEXT CYCLE
        reg [LOG_ORD-1:0] mem0_next_read;
        reg [LOG_ORD-1:0] mem1_next_read;

        // PREVIOUS LPF, VGA, AND PTF READ VALUES
        // (for stable vga_pixel, lpf_pixel_read, and ptf_pixel_read)
        reg [`LOG_MEM-1:0] prev_vga_pixel;
        reg [`LOG_MEM-1:0] prev_lpf_pixel_read;

        // DEBUG
        assign debug_blocks = {capt_mem_block, proc_mem_block, nexd_mem_block, disp_mem_block};
```

```
    assign debug_locs = {capt_mem_loc, proc_mem_loc, nexd_mem_loc, disp_mem_loc};

    assign pt_conflict = (capt_mem_block == nexd_mem_block);

    // ADDRESSING
    wire [18:0] ntsc_loc_offset;
    wire [18:0] ntsc_y_offset;
    wire [18:0] ntsc_x_offset;

    wire [18:0] vga_loc_offset;
    wire [18:0] vga_y_offset;
    wire [18:0] vga_x_offset;

    wire [18:0] lpf_loc_offset;
    wire [18:0] lpf_y_offset;
    wire [18:0] lpf_x_offset;

    wire [18:0] pt_loc_offset;
    wire [18:0] pt_y_offset;
    wire [18:0] pt_x_offset;

    reg [18:0] mem0_loc_offset;
    reg [18:0] mem0_y_offset;
    reg [18:0] mem0_x_offset;

    reg [18:0] mem1_loc_offset;
    reg [18:0] mem1_y_offset;
    reg [18:0] mem1_x_offset;

    // get offsets (summands) from (x,y,loc)
    offset_calculator ntsc_oc(
            .x(ntsc_x), .y(ntsc_y), .loc(capt_mem_loc),
            .x_offset(ntsc_x_offset),
            .y_offset(ntsc_y_offset),
            .loc_offset(ntsc_loc_offset));
    offset_calculator vga_oc(
            .x(hcount), .y(vcount), .loc(disp_mem_loc),
            .x_offset(vga_x_offset),
            .y_offset(vga_y_offset),
            .loc_offset(vga_loc_offset));
    offset_calculator lpf_oc(
            .x(lpf_x), .y(lpf_y), .loc(proc_mem_loc),
            .x_offset(lpf_x_offset),
            .y_offset(lpf_y_offset),
            .loc_offset(lpf_loc_offset));
    offset_calculator pt_oc(
            .x(pt_x), .y(pt_y), .loc(nexd_mem_loc),
            .x_offset(pt_x_offset),
            .y_offset(pt_y_offset),
            .loc_offset(pt_loc_offset));

    assign mem0_addr = mem0_loc_offset + mem0_x_offset + mem0_y_offset;
```

```
        assign mem1_addr = mem1_loc_offset + mem1_x_offset + mem1_y_offset;
169
        always @(posedge clock) begin
171             // set address & write & done flags
                // assign write value to mem0 & mem1 based on who's writing
173             // FIRST BLOCK OF RAM
                if (!capt_mem_block && ntsc_flag) begin
175                     mem0_x_offset <= ntsc_x_offset;
                        mem0_y_offset <= ntsc_y_offset;
177                     mem0_loc_offset <= ntsc_loc_offset;

179                     mem0_write      <= ntsc_pixel;
                        mem0_wr         <= 1;
181                     mem0_bwe        <= 4'b1111;
                        mem0_done       <= NTSC;
183             end
                else if (!disp_mem_block && vga_flag) begin
185                     mem0_x_offset <= vga_x_offset;
                        mem0_y_offset <= vga_y_offset;
187                     mem0_loc_offset <= vga_loc_offset;

189                     mem0_write      <= mem0_write;
                        mem0_wr         <= 0;
191                     mem0_bwe        <= 4'b1111;
                        mem0_done       <= VGA;
193             end
                else if (!proc_mem_block && lpf_flag) begin
195                     mem0_x_offset <= lpf_x_offset;
                        mem0_y_offset <= lpf_y_offset;
197                     mem0_loc_offset <= lpf_loc_offset;

199                     mem0_write      <= lpf_pixel_write;
                        mem0_wr         <= lpf_wr;
201                     mem0_bwe        <= 4'b1111;
                        mem0_done       <= LPF;
203             end
                else if (!nexd_mem_block && pt_flag) begin
205                     mem0_x_offset <= pt_x_offset;
                        mem0_y_offset <= pt_y_offset;
207                     mem0_loc_offset <= pt_loc_offset;

209                     mem0_done       <= PT;
                        mem0_wr         <= 1;
211                     if (pt_x[0] == 1'b0) begin
                                mem0_write      <= {pt_pixel, 18'd0};
213                             mem0_bwe        <= 4'b1100;
                        end
215                     else begin // pt_x[0] == 1'b1
                                mem0_write      <= {18'd0, pt_pixel};
217                             mem0_bwe        <= 4'b0011;
                        end
219             end
```

```
                else begin // nothing's happening
221                     mem0_x_offset <= mem0_x_offset;
                        mem0_y_offset <= mem0_y_offset;
223                     mem0_loc_offset <= mem0_loc_offset;

225                     mem0_write      <= 0;
                        mem0_wr         <= 0;
227                     mem0_bwe        <= 4'b1111;
                        mem0_done       <= NONE;
229             end

231             // SECOND BLOCK OF RAM
                if (capt_mem_block && ntsc_flag) begin
233                     mem1_x_offset <= ntsc_x_offset;
                        mem1_y_offset <= ntsc_y_offset;
235                     mem1_loc_offset <= ntsc_loc_offset;

237                     mem1_write      <= ntsc_pixel;
                        mem1_wr         <= 1;
239                     mem1_bwe        <= 4'b1111;
                        mem1_done       <= NTSC;
241             end
                else if (disp_mem_block && vga_flag) begin
243                     mem1_x_offset <= vga_x_offset;
                        mem1_y_offset <= vga_y_offset;
245                     mem1_loc_offset <= vga_loc_offset;

247                     mem1_write      <= mem1_write;
                        mem1_wr         <= 0;
249                     mem1_bwe        <= 4'b1111;
                        mem1_done       <= VGA;
251             end
                else if (proc_mem_block && lpf_flag) begin
253                     mem1_x_offset <= lpf_x_offset;
                        mem1_y_offset <= lpf_y_offset;
255                     mem1_loc_offset <= lpf_loc_offset;

257                     mem1_write      <= lpf_pixel_write;
                        mem1_wr         <= lpf_wr;
259                     mem1_bwe        <= 4'b1111;
                        mem1_done       <= LPF;
261             end
                else if (nexd_mem_block && pt_flag) begin
263                     mem1_x_offset <= pt_x_offset;
                        mem1_y_offset <= pt_y_offset;
265                     mem1_loc_offset <= pt_loc_offset;

267                     mem1_done       <= PT;
                        mem1_wr         <= 1;
269                     if (pt_x[0] == 1'b0) begin
                                mem1_write      <= {pt_pixel, 18'd0};
271                             mem1_bwe        <= 4'b1100;
```

```
                end
                else begin // pt_x[0] == 1'b1
                        mem1_write      <= {18'd0, pt_pixel};
                        mem1_bwe        <= 4'b0011;
                end
        end
        else begin // nothing's happening
                mem1_x_offset <= mem1_x_offset;
                mem1_y_offset <= mem1_y_offset;
                mem1_loc_offset <= mem1_loc_offset;

                mem1_write      <= 0;
                mem1_wr         <= 0;
                mem1_bwe        <= 4'b1111;
                mem1_done       <= NONE;
        end

        // add new queue members, if any
        if (mem0_done == VGA) mem0_read_queue[LOG_ORD-1:0] <= VGA;
        else if (mem0_done == LPF && !lpf_wr) mem0_read_queue[LOG_ORD-1:0] <= LPF;
        else mem0_read_queue[LOG_ORD-1:0] <= NONE;

        if (mem1_done == VGA) mem1_read_queue[LOG_ORD-1:0] <= VGA;
        else if (mem1_done == LPF && !lpf_wr) mem1_read_queue[LOG_ORD-1:0] <= LPF;
        else mem1_read_queue[LOG_ORD-1:0] <= NONE;

        // assign read value to corresponding member of queue
        mem0_next_read <= mem0_read_queue[LOG_ORD-1:0];
        mem1_next_read <= mem1_read_queue[LOG_ORD-1:0];

        // LPF's turn in the queue
        if (mem0_next_read == LPF) lpf_pixel_read <= mem0_read;
        else if (mem1_next_read == LPF) lpf_pixel_read <= mem1_read;
        //else lpf_pixel_read <= prev_lpf_pixel_read;

        // VGA's turn
        if (mem0_next_read == VGA) vga_pixel <= mem0_read;
        else if (mem1_next_read == VGA) vga_pixel <= mem1_read;
        //else vga_pixel <= prev_vga_pixel;
        // this should be it
end

always @(*) begin
        // set done flags
        done_ntsc = (mem0_done == NTSC) || (mem1_done == NTSC);
        done_vga  = (mem0_done == VGA)  || (mem1_done == VGA);
        done_lpf  = (mem0_done == LPF)  || (mem1_done == LPF);
        done_pt   = (mem0_done == PT)   || (mem1_done == PT);

        if (nexd_mem_block == capt_mem_block) ready_pt = ~nwr;
        else if (nexd_mem_block == disp_mem_block) ready_pt = ~vwr;
        else ready_pt = 1'b0;
```

```
        end

        always @(posedge clock) begin
                // update blocks and locations of images in RAM
                if (reset) begin
                        capt_mem_block  <= 1'b0;
                        capt_mem_loc    <= 2'b00;
                        proc_mem_block  <= 1'b0;
                        proc_mem_loc    <= 2'b01;
                        nexd_mem_block  <= 1'b1;
                        nexd_mem_loc    <= 2'b00;
                        disp_mem_block  <= 1'b1;
                        disp_mem_loc    <= 2'b01;
                end
                else if (frame_flag) begin
                        capt_mem_block  <= proc_mem_block;
                        capt_mem_loc    <= proc_mem_loc;
                        proc_mem_block  <= disp_mem_block;
                        proc_mem_loc    <= disp_mem_loc;
                        nexd_mem_block  <= capt_mem_block;
                        nexd_mem_loc    <= capt_mem_loc;
                        disp_mem_block  <= nexd_mem_block;
                        disp_mem_loc    <= nexd_mem_loc;
                end
                else begin
                        capt_mem_block  <= capt_mem_block;
                        capt_mem_loc    <= capt_mem_loc;
                        proc_mem_block  <= proc_mem_block;
                        proc_mem_loc    <= proc_mem_loc;
                        nexd_mem_block  <= nexd_mem_block;
                        nexd_mem_loc    <= nexd_mem_loc;
                        disp_mem_block  <= disp_mem_block;
                        disp_mem_loc    <= disp_mem_loc;
                end

                // retain previous output pixel values
                prev_vga_pixel <= vga_pixel;
                prev_lpf_pixel_read <= lpf_pixel_read;
        end
endmodule

// maps outputs of memory interface to the inputs, outputs, and inouts
// of the ram modules themselves
// delays we, write_data, and bwe by 2 clock cycles
// modified version of zbt_6111
module zbt_map(
        input clock, // system clock
        input cen, // clock enable
        input we, // write enable (active HIGH)
        input [3:0] bwe, // byte write enable (active HIGH)
        input [18:0] addr, // memory address
        input [35:0] write_data, // data to write
```

```
        output [35:0] read_data, // data read from memory
        output ram_we_b, // physical line to ram we_b
        output [3:0] ram_bwe_b, // physical line to ram bwe_b
        output [18:0] ram_address, // physical line to ram address
        inout [35:0] ram_data, // physical line to ram data
        output ram_cen_b); // physical line to ram clock enable

        // to memory_interface
        assign read_data = ram_data;

        // delaying of signals associated to writing
        reg [71:0] delayed_write_data;
        reg [1:0] delayed_we;

        always @(posedge clock) begin
                delayed_write_data[71:36] <= delayed_write_data[35:0];
                delayed_write_data[35:0] <= write_data[35:0];
                delayed_we[1] <= delayed_we[0];
                delayed_we[0] <= we;
        end

        // to ram itself
        assign ram_cen_b = ~cen;
        assign ram_address = addr;
        assign ram_we_b = ~we;
        assign ram_bwe_b[3:0] = ~bwe[3:0];

        // delay write data
        assign ram_data = delayed_we[1] ? delayed_write_data[71:36] : {36{1'bZ}};
endmodule

module offset_calculator(
                input [`LOG_WIDTH-1:0] x,
                input [`LOG_HEIGHT-1:0] y,
                input [1:0] loc,
                output [18:0] x_offset,
                output [18:0] y_offset,
                output [18:0] loc_offset
        );

        assign x_offset[18:0] = {9'd0, x[`LOG_WIDTH-1:1]};
        loc_lut llut(.loc(loc), .addr_off(loc_offset));
        wire [17:0] y_off_trunc;
        y_lut ylut(.y(y), .addr_off(y_off_trunc));
        assign y_offset[18:0] = {1'b0, y_off_trunc[17:0]};
endmodule

module loc_lut(
        input [1:0] loc,
        output reg [18:0] addr_off
        );

```

```
        always @(*) begin
                case (loc)
                        2'd0: addr_off = 19'd0;
                        2'd1: addr_off = 19'd153600;
                        2'd2: addr_off = 19'd307200;
                        default: addr_off = 19'd0;
                endcase
        end
endmodule

module y_lut(
        input [8:0] y,
        output reg [17:0] addr_off
        );

        always @(*) begin
                case (y)
                        9'd0: addr_off = 18'd0;
                        9'd1: addr_off = 18'd320;
                        9'd2: addr_off = 18'd640;
                        9'd3: addr_off = 18'd960;
                        9'd4: addr_off = 18'd1280;
                        9'd5: addr_off = 18'd1600;
                        9'd6: addr_off = 18'd1920;
                        9'd7: addr_off = 18'd2240;
                        9'd8: addr_off = 18'd2560;
                        9'd9: addr_off = 18'd2880;
                        9'd10: addr_off = 18'd3200;
                        9'd11: addr_off = 18'd3520;
                        9'd12: addr_off = 18'd3840;
                        9'd13: addr_off = 18'd4160;
                        9'd14: addr_off = 18'd4480;
                        9'd15: addr_off = 18'd4800;
                        9'd16: addr_off = 18'd5120;
                        9'd17: addr_off = 18'd5440;
                        9'd18: addr_off = 18'd5760;
                        9'd19: addr_off = 18'd6080;
                        9'd20: addr_off = 18'd6400;
                        9'd21: addr_off = 18'd6720;
                        9'd22: addr_off = 18'd7040;
                        9'd23: addr_off = 18'd7360;
                        9'd24: addr_off = 18'd7680;
                        9'd25: addr_off = 18'd8000;
                        9'd26: addr_off = 18'd8320;
                        9'd27: addr_off = 18'd8640;
                        9'd28: addr_off = 18'd8960;
                        9'd29: addr_off = 18'd9280;
                        9'd30: addr_off = 18'd9600;
                        9'd31: addr_off = 18'd9920;
                        9'd32: addr_off = 18'd10240;
                        9'd33: addr_off = 18'd10560;
                        9'd34: addr_off = 18'd10880;
```

```
                9'd35: addr_off = 18'd11200;
481             9'd36: addr_off = 18'd11520;
                9'd37: addr_off = 18'd11840;
483             9'd38: addr_off = 18'd12160;
                9'd39: addr_off = 18'd12480;
485             9'd40: addr_off = 18'd12800;
                9'd41: addr_off = 18'd13120;
487             9'd42: addr_off = 18'd13440;
                9'd43: addr_off = 18'd13760;
489             9'd44: addr_off = 18'd14080;
                9'd45: addr_off = 18'd14400;
491             9'd46: addr_off = 18'd14720;
                9'd47: addr_off = 18'd15040;
493             9'd48: addr_off = 18'd15360;
                9'd49: addr_off = 18'd15680;
495             9'd50: addr_off = 18'd16000;
                9'd51: addr_off = 18'd16320;
497             9'd52: addr_off = 18'd16640;
                9'd53: addr_off = 18'd16960;
499             9'd54: addr_off = 18'd17280;
                9'd55: addr_off = 18'd17600;
501             9'd56: addr_off = 18'd17920;
                9'd57: addr_off = 18'd18240;
503             9'd58: addr_off = 18'd18560;
                9'd59: addr_off = 18'd18880;
505             9'd60: addr_off = 18'd19200;
                9'd61: addr_off = 18'd19520;
507             9'd62: addr_off = 18'd19840;
                9'd63: addr_off = 18'd20160;
509             9'd64: addr_off = 18'd20480;
                9'd65: addr_off = 18'd20800;
511             9'd66: addr_off = 18'd21120;
                9'd67: addr_off = 18'd21440;
513             9'd68: addr_off = 18'd21760;
                9'd69: addr_off = 18'd22080;
515             9'd70: addr_off = 18'd22400;
                9'd71: addr_off = 18'd22720;
517             9'd72: addr_off = 18'd23040;
                9'd73: addr_off = 18'd23360;
519             9'd74: addr_off = 18'd23680;
                9'd75: addr_off = 18'd24000;
521             9'd76: addr_off = 18'd24320;
                9'd77: addr_off = 18'd24640;
523             9'd78: addr_off = 18'd24960;
                9'd79: addr_off = 18'd25280;
525             9'd80: addr_off = 18'd25600;
                9'd81: addr_off = 18'd25920;
527             9'd82: addr_off = 18'd26240;
                9'd83: addr_off = 18'd26560;
529             9'd84: addr_off = 18'd26880;
                9'd85: addr_off = 18'd27200;
531             9'd86: addr_off = 18'd27520;
```

```
                9'd87: addr_off = 18'd27840;
533             9'd88: addr_off = 18'd28160;
                9'd89: addr_off = 18'd28480;
535             9'd90: addr_off = 18'd28800;
                9'd91: addr_off = 18'd29120;
537             9'd92: addr_off = 18'd29440;
                9'd93: addr_off = 18'd29760;
539             9'd94: addr_off = 18'd30080;
                9'd95: addr_off = 18'd30400;
541             9'd96: addr_off = 18'd30720;
                9'd97: addr_off = 18'd31040;
543             9'd98: addr_off = 18'd31360;
                9'd99: addr_off = 18'd31680;
545             9'd100: addr_off = 18'd32000;
                9'd101: addr_off = 18'd32320;
547             9'd102: addr_off = 18'd32640;
                9'd103: addr_off = 18'd32960;
549             9'd104: addr_off = 18'd33280;
                9'd105: addr_off = 18'd33600;
551             9'd106: addr_off = 18'd33920;
                9'd107: addr_off = 18'd34240;
553             9'd108: addr_off = 18'd34560;
                9'd109: addr_off = 18'd34880;
555             9'd110: addr_off = 18'd35200;
                9'd111: addr_off = 18'd35520;
557             9'd112: addr_off = 18'd35840;
                9'd113: addr_off = 18'd36160;
559             9'd114: addr_off = 18'd36480;
                9'd115: addr_off = 18'd36800;
561             9'd116: addr_off = 18'd37120;
                9'd117: addr_off = 18'd37440;
563             9'd118: addr_off = 18'd37760;
                9'd119: addr_off = 18'd38080;
565             9'd120: addr_off = 18'd38400;
                9'd121: addr_off = 18'd38720;
567             9'd122: addr_off = 18'd39040;
                9'd123: addr_off = 18'd39360;
569             9'd124: addr_off = 18'd39680;
                9'd125: addr_off = 18'd40000;
571             9'd126: addr_off = 18'd40320;
                9'd127: addr_off = 18'd40640;
573             9'd128: addr_off = 18'd40960;
                9'd129: addr_off = 18'd41280;
575             9'd130: addr_off = 18'd41600;
                9'd131: addr_off = 18'd41920;
577             9'd132: addr_off = 18'd42240;
                9'd133: addr_off = 18'd42560;
579             9'd134: addr_off = 18'd42880;
                9'd135: addr_off = 18'd43200;
581             9'd136: addr_off = 18'd43520;
                9'd137: addr_off = 18'd43840;
583             9'd138: addr_off = 18'd44160;
```

```
                9'd139: addr_off = 18'd44480;
585             9'd140: addr_off = 18'd44800;
                9'd141: addr_off = 18'd45120;
587             9'd142: addr_off = 18'd45440;
                9'd143: addr_off = 18'd45760;
589             9'd144: addr_off = 18'd46080;
                9'd145: addr_off = 18'd46400;
591             9'd146: addr_off = 18'd46720;
                9'd147: addr_off = 18'd47040;
593             9'd148: addr_off = 18'd47360;
                9'd149: addr_off = 18'd47680;
595             9'd150: addr_off = 18'd48000;
                9'd151: addr_off = 18'd48320;
597             9'd152: addr_off = 18'd48640;
                9'd153: addr_off = 18'd48960;
599             9'd154: addr_off = 18'd49280;
                9'd155: addr_off = 18'd49600;
601             9'd156: addr_off = 18'd49920;
                9'd157: addr_off = 18'd50240;
603             9'd158: addr_off = 18'd50560;
                9'd159: addr_off = 18'd50880;
605             9'd160: addr_off = 18'd51200;
                9'd161: addr_off = 18'd51520;
607             9'd162: addr_off = 18'd51840;
                9'd163: addr_off = 18'd52160;
609             9'd164: addr_off = 18'd52480;
                9'd165: addr_off = 18'd52800;
611             9'd166: addr_off = 18'd53120;
                9'd167: addr_off = 18'd53440;
613             9'd168: addr_off = 18'd53760;
                9'd169: addr_off = 18'd54080;
615             9'd170: addr_off = 18'd54400;
                9'd171: addr_off = 18'd54720;
617             9'd172: addr_off = 18'd55040;
                9'd173: addr_off = 18'd55360;
619             9'd174: addr_off = 18'd55680;
                9'd175: addr_off = 18'd56000;
621             9'd176: addr_off = 18'd56320;
                9'd177: addr_off = 18'd56640;
623             9'd178: addr_off = 18'd56960;
                9'd179: addr_off = 18'd57280;
625             9'd180: addr_off = 18'd57600;
                9'd181: addr_off = 18'd57920;
627             9'd182: addr_off = 18'd58240;
                9'd183: addr_off = 18'd58560;
629             9'd184: addr_off = 18'd58880;
                9'd185: addr_off = 18'd59200;
631             9'd186: addr_off = 18'd59520;
                9'd187: addr_off = 18'd59840;
633             9'd188: addr_off = 18'd60160;
                9'd189: addr_off = 18'd60480;
635             9'd190: addr_off = 18'd60800;
```

```
                9'd191: addr_off = 18'd61120;
637             9'd192: addr_off = 18'd61440;
                9'd193: addr_off = 18'd61760;
639             9'd194: addr_off = 18'd62080;
                9'd195: addr_off = 18'd62400;
641             9'd196: addr_off = 18'd62720;
                9'd197: addr_off = 18'd63040;
643             9'd198: addr_off = 18'd63360;
                9'd199: addr_off = 18'd63680;
645             9'd200: addr_off = 18'd64000;
                9'd201: addr_off = 18'd64320;
647             9'd202: addr_off = 18'd64640;
                9'd203: addr_off = 18'd64960;
649             9'd204: addr_off = 18'd65280;
                9'd205: addr_off = 18'd65600;
651             9'd206: addr_off = 18'd65920;
                9'd207: addr_off = 18'd66240;
653             9'd208: addr_off = 18'd66560;
                9'd209: addr_off = 18'd66880;
655             9'd210: addr_off = 18'd67200;
                9'd211: addr_off = 18'd67520;
657             9'd212: addr_off = 18'd67840;
                9'd213: addr_off = 18'd68160;
659             9'd214: addr_off = 18'd68480;
                9'd215: addr_off = 18'd68800;
661             9'd216: addr_off = 18'd69120;
                9'd217: addr_off = 18'd69440;
663             9'd218: addr_off = 18'd69760;
                9'd219: addr_off = 18'd70080;
665             9'd220: addr_off = 18'd70400;
                9'd221: addr_off = 18'd70720;
667             9'd222: addr_off = 18'd71040;
                9'd223: addr_off = 18'd71360;
669             9'd224: addr_off = 18'd71680;
                9'd225: addr_off = 18'd72000;
671             9'd226: addr_off = 18'd72320;
                9'd227: addr_off = 18'd72640;
673             9'd228: addr_off = 18'd72960;
                9'd229: addr_off = 18'd73280;
675             9'd230: addr_off = 18'd73600;
                9'd231: addr_off = 18'd73920;
677             9'd232: addr_off = 18'd74240;
                9'd233: addr_off = 18'd74560;
679             9'd234: addr_off = 18'd74880;
                9'd235: addr_off = 18'd75200;
681             9'd236: addr_off = 18'd75520;
                9'd237: addr_off = 18'd75840;
683             9'd238: addr_off = 18'd76160;
                9'd239: addr_off = 18'd76480;
685             9'd240: addr_off = 18'd76800;
                9'd241: addr_off = 18'd77120;
687             9'd242: addr_off = 18'd77440;
```

```
                9'd243: addr_off = 18'd77760;
689             9'd244: addr_off = 18'd78080;
                9'd245: addr_off = 18'd78400;
691             9'd246: addr_off = 18'd78720;
                9'd247: addr_off = 18'd79040;
693             9'd248: addr_off = 18'd79360;
                9'd249: addr_off = 18'd79680;
695             9'd250: addr_off = 18'd80000;
                9'd251: addr_off = 18'd80320;
697             9'd252: addr_off = 18'd80640;
                9'd253: addr_off = 18'd80960;
699             9'd254: addr_off = 18'd81280;
                9'd255: addr_off = 18'd81600;
701             9'd256: addr_off = 18'd81920;
                9'd257: addr_off = 18'd82240;
703             9'd258: addr_off = 18'd82560;
                9'd259: addr_off = 18'd82880;
705             9'd260: addr_off = 18'd83200;
                9'd261: addr_off = 18'd83520;
707             9'd262: addr_off = 18'd83840;
                9'd263: addr_off = 18'd84160;
709             9'd264: addr_off = 18'd84480;
                9'd265: addr_off = 18'd84800;
711             9'd266: addr_off = 18'd85120;
                9'd267: addr_off = 18'd85440;
713             9'd268: addr_off = 18'd85760;
                9'd269: addr_off = 18'd86080;
715             9'd270: addr_off = 18'd86400;
                9'd271: addr_off = 18'd86720;
717             9'd272: addr_off = 18'd87040;
                9'd273: addr_off = 18'd87360;
719             9'd274: addr_off = 18'd87680;
                9'd275: addr_off = 18'd88000;
721             9'd276: addr_off = 18'd88320;
                9'd277: addr_off = 18'd88640;
723             9'd278: addr_off = 18'd88960;
                9'd279: addr_off = 18'd89280;
725             9'd280: addr_off = 18'd89600;
                9'd281: addr_off = 18'd89920;
727             9'd282: addr_off = 18'd90240;
                9'd283: addr_off = 18'd90560;
729             9'd284: addr_off = 18'd90880;
                9'd285: addr_off = 18'd91200;
731             9'd286: addr_off = 18'd91520;
                9'd287: addr_off = 18'd91840;
733             9'd288: addr_off = 18'd92160;
                9'd289: addr_off = 18'd92480;
735             9'd290: addr_off = 18'd92800;
                9'd291: addr_off = 18'd93120;
737             9'd292: addr_off = 18'd93440;
                9'd293: addr_off = 18'd93760;
739             9'd294: addr_off = 18'd94080;
```

```
                9'd295: addr_off = 18'd94400;
741             9'd296: addr_off = 18'd94720;
                9'd297: addr_off = 18'd95040;
743             9'd298: addr_off = 18'd95360;
                9'd299: addr_off = 18'd95680;
745             9'd300: addr_off = 18'd96000;
                9'd301: addr_off = 18'd96320;
747             9'd302: addr_off = 18'd96640;
                9'd303: addr_off = 18'd96960;
749             9'd304: addr_off = 18'd97280;
                9'd305: addr_off = 18'd97600;
751             9'd306: addr_off = 18'd97920;
                9'd307: addr_off = 18'd98240;
753             9'd308: addr_off = 18'd98560;
                9'd309: addr_off = 18'd98880;
755             9'd310: addr_off = 18'd99200;
                9'd311: addr_off = 18'd99520;
757             9'd312: addr_off = 18'd99840;
                9'd313: addr_off = 18'd100160;
759             9'd314: addr_off = 18'd100480;
                9'd315: addr_off = 18'd100800;
761             9'd316: addr_off = 18'd101120;
                9'd317: addr_off = 18'd101440;
763             9'd318: addr_off = 18'd101760;
                9'd319: addr_off = 18'd102080;
765             9'd320: addr_off = 18'd102400;
                9'd321: addr_off = 18'd102720;
767             9'd322: addr_off = 18'd103040;
                9'd323: addr_off = 18'd103360;
769             9'd324: addr_off = 18'd103680;
                9'd325: addr_off = 18'd104000;
771             9'd326: addr_off = 18'd104320;
                9'd327: addr_off = 18'd104640;
773             9'd328: addr_off = 18'd104960;
                9'd329: addr_off = 18'd105280;
775             9'd330: addr_off = 18'd105600;
                9'd331: addr_off = 18'd105920;
777             9'd332: addr_off = 18'd106240;
                9'd333: addr_off = 18'd106560;
779             9'd334: addr_off = 18'd106880;
                9'd335: addr_off = 18'd107200;
781             9'd336: addr_off = 18'd107520;
                9'd337: addr_off = 18'd107840;
783             9'd338: addr_off = 18'd108160;
                9'd339: addr_off = 18'd108480;
785             9'd340: addr_off = 18'd108800;
                9'd341: addr_off = 18'd109120;
787             9'd342: addr_off = 18'd109440;
                9'd343: addr_off = 18'd109760;
789             9'd344: addr_off = 18'd110080;
                9'd345: addr_off = 18'd110400;
791             9'd346: addr_off = 18'd110720;
```

```
                9'd347: addr_off = 18'd111040;
793             9'd348: addr_off = 18'd111360;
                9'd349: addr_off = 18'd111680;
795             9'd350: addr_off = 18'd112000;
                9'd351: addr_off = 18'd112320;
797             9'd352: addr_off = 18'd112640;
                9'd353: addr_off = 18'd112960;
799             9'd354: addr_off = 18'd113280;
                9'd355: addr_off = 18'd113600;
801             9'd356: addr_off = 18'd113920;
                9'd357: addr_off = 18'd114240;
803             9'd358: addr_off = 18'd114560;
                9'd359: addr_off = 18'd114880;
805             9'd360: addr_off = 18'd115200;
                9'd361: addr_off = 18'd115520;
807             9'd362: addr_off = 18'd115840;
                9'd363: addr_off = 18'd116160;
809             9'd364: addr_off = 18'd116480;
                9'd365: addr_off = 18'd116800;
811             9'd366: addr_off = 18'd117120;
                9'd367: addr_off = 18'd117440;
813             9'd368: addr_off = 18'd117760;
                9'd369: addr_off = 18'd118080;
815             9'd370: addr_off = 18'd118400;
                9'd371: addr_off = 18'd118720;
817             9'd372: addr_off = 18'd119040;
                9'd373: addr_off = 18'd119360;
819             9'd374: addr_off = 18'd119680;
                9'd375: addr_off = 18'd120000;
821             9'd376: addr_off = 18'd120320;
                9'd377: addr_off = 18'd120640;
823             9'd378: addr_off = 18'd120960;
                9'd379: addr_off = 18'd121280;
825             9'd380: addr_off = 18'd121600;
                9'd381: addr_off = 18'd121920;
827             9'd382: addr_off = 18'd122240;
                9'd383: addr_off = 18'd122560;
829             9'd384: addr_off = 18'd122880;
                9'd385: addr_off = 18'd123200;
831             9'd386: addr_off = 18'd123520;
                9'd387: addr_off = 18'd123840;
833             9'd388: addr_off = 18'd124160;
                9'd389: addr_off = 18'd124480;
835             9'd390: addr_off = 18'd124800;
                9'd391: addr_off = 18'd125120;
837             9'd392: addr_off = 18'd125440;
                9'd393: addr_off = 18'd125760;
839             9'd394: addr_off = 18'd126080;
                9'd395: addr_off = 18'd126400;
841             9'd396: addr_off = 18'd126720;
                9'd397: addr_off = 18'd127040;
843             9'd398: addr_off = 18'd127360;
```

```
                9'd399: addr_off = 18'd127680;
845             9'd400: addr_off = 18'd128000;
                9'd401: addr_off = 18'd128320;
847             9'd402: addr_off = 18'd128640;
                9'd403: addr_off = 18'd128960;
849             9'd404: addr_off = 18'd129280;
                9'd405: addr_off = 18'd129600;
851             9'd406: addr_off = 18'd129920;
                9'd407: addr_off = 18'd130240;
853             9'd408: addr_off = 18'd130560;
                9'd409: addr_off = 18'd130880;
855             9'd410: addr_off = 18'd131200;
                9'd411: addr_off = 18'd131520;
857             9'd412: addr_off = 18'd131840;
                9'd413: addr_off = 18'd132160;
859             9'd414: addr_off = 18'd132480;
                9'd415: addr_off = 18'd132800;
861             9'd416: addr_off = 18'd133120;
                9'd417: addr_off = 18'd133440;
863             9'd418: addr_off = 18'd133760;
                9'd419: addr_off = 18'd134080;
865             9'd420: addr_off = 18'd134400;
                9'd421: addr_off = 18'd134720;
867             9'd422: addr_off = 18'd135040;
                9'd423: addr_off = 18'd135360;
869             9'd424: addr_off = 18'd135680;
                9'd425: addr_off = 18'd136000;
871             9'd426: addr_off = 18'd136320;
                9'd427: addr_off = 18'd136640;
873             9'd428: addr_off = 18'd136960;
                9'd429: addr_off = 18'd137280;
875             9'd430: addr_off = 18'd137600;
                9'd431: addr_off = 18'd137920;
877             9'd432: addr_off = 18'd138240;
                9'd433: addr_off = 18'd138560;
879             9'd434: addr_off = 18'd138880;
                9'd435: addr_off = 18'd139200;
881             9'd436: addr_off = 18'd139520;
                9'd437: addr_off = 18'd139840;
883             9'd438: addr_off = 18'd140160;
                9'd439: addr_off = 18'd140480;
885             9'd440: addr_off = 18'd140800;
                9'd441: addr_off = 18'd141120;
887             9'd442: addr_off = 18'd141440;
                9'd443: addr_off = 18'd141760;
889             9'd444: addr_off = 18'd142080;
                9'd445: addr_off = 18'd142400;
891             9'd446: addr_off = 18'd142720;
                9'd447: addr_off = 18'd143040;
893             9'd448: addr_off = 18'd143360;
                9'd449: addr_off = 18'd143680;
895             9'd450: addr_off = 18'd144000;
```

```
                        9'd451: addr_off = 18'd144320;
                        9'd452: addr_off = 18'd144640;
                        9'd453: addr_off = 18'd144960;
                        9'd454: addr_off = 18'd145280;
                        9'd455: addr_off = 18'd145600;
                        9'd456: addr_off = 18'd145920;
                        9'd457: addr_off = 18'd146240;
                        9'd458: addr_off = 18'd146560;
                        9'd459: addr_off = 18'd146880;
                        9'd460: addr_off = 18'd147200;
                        9'd461: addr_off = 18'd147520;
                        9'd462: addr_off = 18'd147840;
                        9'd463: addr_off = 18'd148160;
                        9'd464: addr_off = 18'd148480;
                        9'd465: addr_off = 18'd148800;
                        9'd466: addr_off = 18'd149120;
                        9'd467: addr_off = 18'd149440;
                        9'd468: addr_off = 18'd149760;
                        9'd469: addr_off = 18'd150080;
                        9'd470: addr_off = 18'd150400;
                        9'd471: addr_off = 18'd150720;
                        9'd472: addr_off = 18'd151040;
                        9'd473: addr_off = 18'd151360;
                        9'd474: addr_off = 18'd151680;
                        9'd475: addr_off = 18'd152000;
                        9'd476: addr_off = 18'd152320;
                        9'd477: addr_off = 18'd152640;
                        9'd478: addr_off = 18'd152960;
                        9'd479: addr_off = 18'd153280;
                        default: addr_off = 18'd0;
                endcase
        end
endmodule
/* FOR REFERENCE
        // SRAMs
    assign ram0_data = 36'hZ;
    assign ram0_address = 19'h0;
    assign ram0_adv_ld = 1'b0;
    assign ram0_clk = 1'b0;
    assign ram0_cen_b = 1'b1;
    assign ram0_ce_b = 1'b1;
    assign ram0_oe_b = 1'b1;
    assign ram0_we_b = 1'b1;
    assign ram0_bwe_b = 4'hF;
    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'b1;
    assign ram1_oe_b = 1'b1;
    assign ram1_we_b = 1'b1;
```

```
    assign ram1_bwe_b = 4'hF;
    assign clock_feedback_out = 1'b0;
*/
```

A.3.1 memory_interface_testbench.v

```
        // STANDARD SIGNALS
        reg clock;
        reg reset;
        // NTSC_CAPTURE
        reg frame_flag;
        reg ntsc_flag;
        reg [`LOG_MEM-1:0] ntsc_pixel;
        wire done_ntsc;
        // LPF
        reg lpf_flag;
        reg lpf_wr;
        reg [`LOG_WIDTH-1:0] lpf_x;
        reg [`LOG_HEIGHT-1:0] lpf_y;
        reg [`LOG_MEM-1:0] lpf_pixel_write;
        wire done_lpf;
        wire [`LOG_MEM-1:0] lpf_pixel_read;
        // PROJECTIVE_TRANSFORM
        reg pt_flag;
        reg pt_wr;
        reg [`LOG_WIDTH-1:0] pt_x;
        reg [`LOG_HEIGHT-1:0] pt_y;
        reg [`LOG_TRUNC-1:0] pt_pixel_write;
        wire done_pt;
        // VGA_WRITE
        reg vga_flag;
        wire done_vga;
        wire [`LOG_FULL-1:0] vga_pixel;
        // MEMORY
        // MEM ADDRESSES
        wire [`LOG_ADDR-1:0] mem0_addr;
        wire [`LOG_ADDR-1:0] mem1_addr;
        // MEM READ
        wire [`LOG_MEM-1:0] mem0_read;
        wire [`LOG_MEM-1:0] mem1_read;
        // MEM WRITE
        wire [`LOG_MEM-1:0] mem0_write;
        wire [`LOG_MEM-1:0] mem1_write;
        // WR FLAGS
        wire mem0_wr;
        wire mem1_wr;

        memory_interface mem_int (
                .clock                          (clock),
                .reset                          (reset),
                .frame_flag             (frame_flag),
```

```
46                  .ntsc_flag                  (ntsc_flag),
                    .ntsc_pixel                 (ntsc_pixel),
48                  .done_ntsc                  (done_ntsc),
                    .lpf_flag                   (lpf_flag),
50                  .lpf_wr                     (lpf_wr),
                    .lpf_x                      (lpf_x),
52                  .lpf_y                      (lpf_y),
                    .lpf_pixel_write        (lpf_pixel_write),
54                  .done_lpf                   (done_lpf),
                    .lpf_pixel_read         (lpf_pixel_read),
56                  .pt_flag                    (pt_flag),
                    .pt_wr                      (pt_wr),
58                  .pt_x                       (pt_x),
                    .pt_y                       (pt_y),
60                  .pt_pixel_write         (pt_pixel_write),
                    .done_pt                    (done_pt),
62                  .vga_flag                   (vga_flag),
                    .done_vga                   (done_vga),
64                  .vga_pixel                  (vga_pixel),
                    .mem0_addr                  (mem0_addr),
66                  .mem1_addr                  (mem1_addr),
                    .mem0_read                  (mem0_read),
68                  .mem1_read                  (mem1_read),
                    .mem0_write                 (mem0_write),
70                  .mem1_write                 (mem1_write),
                    .mem0_wr                    (mem0_wr),
72                  .mem1_wr                    (mem1_wr)
        );
74
        initial begin
76              clock = 0;
                reset = 1;
78              #10 reset = 0;
        end
80
        initial begin
82              $dumpvars;
        end
84
        always #5 clock = !clock;
```

```
1   `include "../params.v"
    `include "../memory_interface.v"
3
    module tb1;
5       `include "tb_template.v"

7       initial begin
                frame_flag = 0;
9               #1000 frame_flag = 1;
                #10 frame_flag = 0;
11      end
```

```
13          integer i;
            initial begin
15                  vga_flag = 0;
                    ntsc_flag = 0;
17                  #100
                    for (i = 0;i < 150;i=i+1) begin
19                          #10 vga_flag = ~vga_flag;
                            ntsc_flag = ~ntsc_flag;
21                  end
            end
23
            initial #2000 $finish;
25  endmodule
```

```
1   `include "../params.v"
    `include "../memory_interface.v"
3
    module tb2;
5           `include "tb_template.v"

7           initial begin
                    frame_flag = 0;
9                   #1000 frame_flag = 1;
                    #10 frame_flag = 0;
11                  #1000000 frame_flag = 1;
                    #10 frame_flag = 0;
13          end

15          integer i;
            integer j;
17          initial begin
                    lpf_x = 0;
19                  lpf_y = 0;
                    pt_x = 0;
21                  pt_y = 0;

23                  for (i = 0;i < `IMAGE_HEIGHT;i = i+1) begin
                            for (j=0;j < `IMAGE_WIDTH;j = j+1) begin
25                                  #10
                                    lpf_x = j;
27                                  lpf_y = i;
                                    pt_x = j;
29                                  pt_y = i;
                            end
31                  end
            end
33
            initial begin
35                  #6000000 $stop;
            end
37  endmodule
```

```
1  `include "../params.v"
   `include "../memory_interface.v"
3
   module tb3;
5          `include "tb_template.v"

7          dummy_zbt mem0(.clock(clock),.reset(reset),.wr(mem0_wr),.addr(mem0_addr),.write(
               mem0_write),.data(mem0_read));
           dummy_zbt mem1(.clock(clock),.reset(reset),.wr(mem1_wr),.addr(mem1_addr),.write(
               mem1_write),.data(mem1_read));
9
           integer i;
11         initial begin
                   frame_flag = 0;
13                 ntsc_flag = 0;
                   ntsc_pixel = 0;
15                 vga_flag = 0;
                   // continuous stream
17                 for (i = 0;i < 2000;i=i+1) begin
                           #10
19                         ntsc_pixel = i*7;
                           ntsc_flag = 1;
21                 end

23                 // non-continuous stream
                   for (i = 2000;i < 4000;i=i+1) begin
25                         #10
                           ntsc_pixel = i*8;
27                         ntsc_flag = 1;
                           #10
29                         ntsc_flag = 0;
                   end
31                 #10 frame_flag = 1;
                   #10 frame_flag = 0;
33                 #10 frame_flag = 1;
                   #10 frame_flag = 0;
35                 // capt pixels should now be disp pixels
                   // continuous stream
37                 for (i = 0;i < 2000;i=i+1) begin
                           #10
39                         vga_flag = 1;
                   end
41
                   // non-continuous stream
43                 for (i = 2000;i < 4000;i=i+1) begin
                           #10
45                         vga_flag = 1;
                           #10
47                         vga_flag = 0;
                   end
49         end
```

```
51          initial begin
                #100000 $stop;
53          end
endmodule
```

```
`default_nettype none
2 `include "../params.v"
`include "../memory_interface.v"
4
module tb4;
6           // STANDARD SIGNALS
            reg clock;
8           reg reset;
            // NTSC_CAPTURE
10          reg frame_flag;
            wire ntsc_flag;
12          wire [`LOG_MEM-1:0] ntsc_pixel;
            wire done_ntsc;
14          // LPF
            reg lpf_flag;
16          reg lpf_wr;
            reg [`LOG_WIDTH-1:0] lpf_x;
18          reg [`LOG_HEIGHT-1:0] lpf_y;
            reg [`LOG_MEM-1:0] lpf_pixel_write;
20          wire done_lpf;
            wire [`LOG_MEM-1:0] lpf_pixel_read;
22          // PROJECTIVE_TRANSFORM
            wire pt_flag;
24          reg pt_wr;
            wire [`LOG_WIDTH-1:0] pt_x;
26          wire [`LOG_HEIGHT-1:0] pt_y;
            wire [`LOG_TRUNC-1:0] pt_pixel_write;
28          wire done_pt;
            // VGA_WRITE
30          reg vga_flag;
            wire done_vga;
32          wire [`LOG_FULL-1:0] vga_pixel;
            // MEMORY
34          // MEM ADDRESSES
            wire [`LOG_ADDR-1:0] mem0_addr;
36          wire [`LOG_ADDR-1:0] mem1_addr;
            // MEM READ
38          wire [`LOG_MEM-1:0] mem0_read;
            wire [`LOG_MEM-1:0] mem1_read;
40          // MEM WRITE
            wire [`LOG_MEM-1:0] mem0_write;
42          wire [`LOG_MEM-1:0] mem1_write;
            // WR FLAGS
44          wire mem0_wr;
            wire mem1_wr;
46
```

```
        memory_interface mem_int (
48              .clock                  (clock),
                .reset                  (reset),
50              .frame_flag             (frame_flag),
                .ntsc_flag              (ntsc_flag),
52              .ntsc_pixel             (ntsc_pixel),
                .done_ntsc              (done_ntsc),
54              .lpf_flag               (lpf_flag),
                .lpf_wr                 (lpf_wr),
56              .lpf_x                  (lpf_x),
                .lpf_y                  (lpf_y),
58              .lpf_pixel_write        (lpf_pixel_write),
                .done_lpf               (done_lpf),
60              .lpf_pixel_read         (lpf_pixel_read),
                .pt_flag                (pt_flag),
62              .pt_wr                  (pt_wr),
                .pt_x                   (pt_x),
64              .pt_y                   (pt_y),
                .pt_pixel_write         (pt_pixel_write),
66              .done_pt                (done_pt),
                .vga_flag               (vga_flag),
68              .done_vga               (done_vga),
                .vga_pixel              (vga_pixel),
70              .mem0_addr              (mem0_addr),
                .mem1_addr              (mem1_addr),
72              .mem0_read              (mem0_read),
                .mem1_read              (mem1_read),
74              .mem0_write             (mem0_write),
                .mem1_write             (mem1_write),
76              .mem0_wr                (mem0_wr),
                .mem1_wr                (mem1_wr)
78      );

80      initial begin
                clock = 0;
82              reset = 1;
                #10 reset = 0;
84      end

86      initial begin
                $dumpvars;
88      end

90      always #5 clock = !clock;

92      dummy_zbt mem0(.clock(clock),.reset(reset),.wr(mem0_wr),.addr(mem0_addr),.write(
            mem0_write),.data(mem0_read));
        dummy_zbt mem1(.clock(clock),.reset(reset),.wr(mem1_wr),.addr(mem1_addr),.write(
            mem1_write),.data(mem1_read));
94
        reg start;
96
```

```
        writer #(.DELAY(2), .REPS(32'd1000), .START(32'd1), .DEL(1)) ntsc(.clock(clock), .
            reset(reset), .start(start), .done(done_ntsc), .flag(ntsc_flag), .pixel(
            ntsc_pixel));
98      writer #(.DELAY(3), .REPS(32'd1000), .START(32'd3000), .DEL(4)) pt(.clock(clock), .
            reset(reset), .start(start), .done(done_pt), .flag(pt_flag), .pixel(
            pt_pixel_write), .x(pt_x), .y(pt_y));

100     integer i;
        initial begin
102             frame_flag = 0;
                vga_flag = 0;
104             lpf_flag = 0;
                pt_wr = 1;
106             #10 frame_flag = 1;
                #10 frame_flag = 0;
108             #10 start = 1;
                #10 start = 0;
110             #20000 frame_flag = 1;
                #10 frame_flag = 0;
112             // capt pixels should now be disp pixels
                // continuous stream
114             for (i = 0;i < 1500;i=i+1) begin
                        #10
116                     vga_flag = 1;
                end
118             #10 frame_flag = 1;
                #10 frame_flag = 0;
120             for (i = 0;i < 1500;i=i+1) begin
                        #10
122                     vga_flag = 1;
                end
124     end

126     initial begin
                #80000 $stop;
128     end
endmodule
130

module writer #(parameter DELAY=2,REPS=1000,START=1,DEL=1)
132     (
                input clock,
134             input reset,
                input start,
136             input done,
                output reg flag,
138             output reg [`LOG_MEM-1:0] pixel,
                output reg [`LOG_WIDTH-1:0] x,
140             output reg [`LOG_HEIGHT-1:0] y
        );
142
        parameter IDLE      = 3'b001;
144     parameter WRITING   = 3'b010;
```

```
        parameter COUNTING = 3'b100;
146     reg [2:0] state;
        reg [31:0] count;
148     reg [31:0] rep_count;

150     always @(*) begin
                flag = (state == WRITING);
152     end

154     always @(posedge clock) begin
                if (reset) begin
156                     state <= IDLE;
                        count <= 32'b0;
158                     rep_count <= 0;
                        pixel <= START;
160                     x <= 0;
                        y <= 0;
162             end
                else begin
164                     case (state)
                                IDLE: begin
166                                     if (start) state <= WRITING;
                                        else state <= state;
168                                     count <= 0;
                                        rep_count <= 0;
170                                     pixel <= START;
                                        x <= 0;
172                                     y <= 0;
                                end
174                             WRITING: begin
                                        if (done) begin
176                                             state <= COUNTING;
                                                count <= 1;
178                                             rep_count <= rep_count+1;
                                                pixel <= pixel+DEL;
180                                             if (x == `IMAGE_WIDTH-1) begin
                                                        x <= 0;
182                                                     y <= y+1;
                                                end
184                                             else begin
                                                        x <= x+1;
186                                                     y <= y;
                                                end
188                                     end
                                        else begin
190                                             state <= state;
                                                count <= count;
192                                             rep_count <= rep_count;
                                                pixel <= pixel;
194                                             x <= x;
                                                y <= y;
196                                     end
```

```
                                end
                                COUNTING: begin
                                        if (rep_count == REPS-1) begin
                                                state <= IDLE;
                                                count <= 0;
                                        end
                                        else if (count == DELAY-1) begin
                                                state <= WRITING;
                                                count <= 0;
                                        end
                                        else begin
                                                state <= COUNTING;
                                                count <= count+1;
                                        end
                                        rep_count <= rep_count;
                                        pixel <= pixel;
                                        x <= x;
                                        y <= y;
                                end
                                default: begin
                                        state <= state;
                                        count <= count;
                                        rep_count <= rep_count;
                                        pixel <= pixel;
                                        x <= x;
                                        y <= y;
                                end
                        endcase
                end
        end
endmodule
```

```
`default_nettype none
`include "../params.v"
`include "../memory_interface.v"

module tb4;
        // STANDARD SIGNALS
        reg clock;
        reg vclock;
        reg reset;
        // NTSC_CAPTURE
        reg frame_flag;
        wire ntsc_flag;
        wire [`LOG_MEM-1:0] ntsc_pixel;
        wire done_ntsc;
        // LPF
        reg lpf_flag;
        reg lpf_wr;
        reg [`LOG_WIDTH-1:0] lpf_x;
        reg [`LOG_HEIGHT-1:0] lpf_y;
        reg [`LOG_MEM-1:0] lpf_pixel_write;
```

```
21          wire done_lpf;
            wire [`LOG_MEM-1:0] lpf_pixel_read;
23          // PROJECTIVE_TRANSFORM
            wire pt_flag;
25          reg pt_wr;
            wire [`LOG_WIDTH-1:0] pt_x;
27          wire [`LOG_HEIGHT-1:0] pt_y;
            wire [`LOG_TRUNC-1:0] pt_pixel_write;
29          wire done_pt;
            // VGA_WRITE
31          wire vga_flag;
            wire done_vga;
33          wire [`LOG_MEM-1:0] vga_pixel;
            // MEMORY
35          // MEM ADDRESSES
            wire [`LOG_ADDR-1:0] mem0_addr;
37          wire [`LOG_ADDR-1:0] mem1_addr;
            // MEM READ
39          wire [`LOG_MEM-1:0] mem0_read;
            wire [`LOG_MEM-1:0] mem1_read;
41          // MEM WRITE
            wire [`LOG_MEM-1:0] mem0_write;
43          wire [`LOG_MEM-1:0] mem1_write;
            // WR FLAGS
45          wire mem0_wr;
            wire mem1_wr;
47
            memory_interface mem_int (
49                  .clock                  (clock),
                    .reset                  (reset),
51                  .frame_flag             (frame_flag),
                    .ntsc_flag              (ntsc_flag),
53                  .ntsc_pixel             (ntsc_pixel),
                    .done_ntsc              (done_ntsc),
55                  .lpf_flag               (lpf_flag),
                    .lpf_wr                 (lpf_wr),
57                  .lpf_x                  (lpf_x),
                    .lpf_y                  (lpf_y),
59                  .lpf_pixel_write        (lpf_pixel_write),
                    .done_lpf               (done_lpf),
61                  .lpf_pixel_read         (lpf_pixel_read),
                    .pt_flag                (pt_flag),
63                  //.pt_wr                        (pt_wr),
                    .pt_x                   (pt_x),
65                  .pt_y                   (pt_y),
                    .pt_pixel               (pt_pixel_write),
67                  .done_pt                (done_pt),
                    .vga_flag               (vga_flag),
69                  .done_vga               (done_vga),
                    .vga_pixel              (vga_pixel),
71                  .mem0_addr              (mem0_addr),
                    .mem1_addr              (mem1_addr),
```

```
73              .mem0_read                      (mem0_read),
                .mem1_read                      (mem1_read),
75              .mem0_write                     (mem0_write),
                .mem1_write                     (mem1_write),
77              .mem0_wr                        (mem0_wr),
                .mem1_wr                        (mem1_wr)
79      );

81      initial begin
                clock = 0;
83              vclock = 0;
                reset = 1;
85              #15 reset = 0;
        end
87
        initial begin
89              $dumpvars;
        end
91
        always #5 clock = !clock;
93      always #10 vclock = !vclock;

95      dummy_zbt mem0(.clock(clock),.reset(reset),.wr(mem0_wr),.addr(mem0_addr),.write(
            mem0_write),.data(mem0_read));
        dummy_zbt mem1(.clock(clock),.reset(reset),.wr(mem1_wr),.addr(mem1_addr),.write(
            mem1_write),.data(mem1_read));
97
        reg start;
99
        writer #(.DELAY(2), .REPS(32'd1000), .START(32'd1), .DEL(1)) ntsc(.clock(clock), .
            reset(reset), .start(start), .done(done_ntsc), .flag(ntsc_flag), .pixel(
            ntsc_pixel));
101     // writer #(.DELAY(3), .REPS(32'd1000), .START(32'd3000), .DEL(4)) pt(.clock(clock),
             .reset(reset), .start(start), .done(done_pt), .flag(pt_flag), .pixel(
            pt_pixel_write), .x(pt_x), .y(pt_y));
        //
103     vga_write vga(.clock(clock), .vclock(vclock), .reset(reset), .frame_flag(frame_flag)
            , .vga_pixel(vga_pixel), .done_vga(done_vga), .vga_flag(vga_flag));

105     integer i;
        initial begin
107             frame_flag = 0;
                lpf_flag = 0;
109             pt_wr = 1;
                #10 frame_flag = 1;
111             #10 frame_flag = 0;
                #10 start = 1;
113             #10 start = 0;
                #20000 frame_flag = 1;
115             #10 frame_flag = 0;
                // capt pixels should now be disp pixels
117             // continuous stream
```

```
        end

        initial begin
                #80000 $stop;
        end
endmodule

module writer #(parameter DELAY=2,REPS=1000,START=1,DEL=1)
        (
                input clock,
                input reset,
                input start,
                input done,
                output reg flag,
                output reg [`LOG_MEM-1:0] pixel,
                output reg [`LOG_WIDTH-1:0] x,
                output reg [`LOG_HEIGHT-1:0] y
        );

        parameter IDLE      = 3'b001;
        parameter WRITING   = 3'b010;
        parameter COUNTING = 3'b100;
        reg [2:0] state;
        reg [31:0] count;
        reg [31:0] rep_count;

        always @(*) begin
                flag = (state == WRITING);
        end

        always @(posedge clock) begin
                if (reset) begin
                        state <= IDLE;
                        count <= 32'b0;
                        rep_count <= 0;
                        pixel <= START;
                        x <= 0;
                        y <= 0;
                end
                else begin
                        case (state)
                                IDLE: begin
                                        if (start) state <= WRITING;
                                        else state <= state;
                                        count <= 0;
                                        rep_count <= 0;
                                        pixel <= START;
                                        x <= 0;
                                        y <= 0;
                                end
                                WRITING: begin
                                        if (done) begin
```

```
                                        state <= COUNTING;
                                        count <= 1;
                                        rep_count <= rep_count+1;
                                        pixel <= pixel+DEL;
                                        if (x == `IMAGE_WIDTH-1) begin
                                                x <= 0;
                                                y <= y+1;
                                        end
                                        else begin
                                                x <= x+1;
                                                y <= y;
                                        end
                                end
                                else begin
                                        state <= state;
                                        count <= count;
                                        rep_count <= rep_count;
                                        pixel <= pixel;
                                        x <= x;
                                        y <= y;
                                end
                        end
                        COUNTING: begin
                                if (rep_count == REPS-1) begin
                                        state <= IDLE;
                                        count <= 0;
                                end
                                else if (count == DELAY-1) begin
                                        state <= WRITING;
                                        count <= 0;
                                end
                                else begin
                                        state <= COUNTING;
                                        count <= count+1;
                                end
                                rep_count <= rep_count;
                                pixel <= pixel;
                                x <= x;
                                y <= y;
                        end
                        default: begin
                                state <= state;
                                count <= count;
                                rep_count <= rep_count;
                                pixel <= pixel;
                                x <= x;
                                y <= y;
                        end
                endcase
        end
    end
endmodule
```

A.4 object_recognition.v

```
// Logan Williams


module object_recognition(
                          input             clk,
                          input             reset,
                          input [1:0]       color,
                          input [9:0]       interesting_x,
                          input [8:0]       interesting_y,
                          input             interesting_flag,
                          input             frame_flag,
                          output reg [9:0] m_x,
                          output reg [8:0] m_y,
                          output reg [9:0] a_x,
                          output reg [8:0] a_y,
                          output reg [9:0] b_x,
                          output reg [8:0] b_y,
                          output reg [9:0] c_x,
                          output reg [8:0] c_y,
                          output reg [9:0] d_x,
                          output reg [8:0] d_y,
                          output reg        corners_flag);

   // delayed registers for pipelining the weighting
   reg [1:0]                                delayed_color;
   reg [9:0]                                delayed_interesting_x;
   reg [8:0]                                delayed_interesting_y;
   reg                                      delayed_interesting_flag;

   // a state register
   reg [1:0]                                state;

   // accumulators
   reg [63:0]                               sumx [0:3];
   reg [63:0]                               sumy [0:3];
   reg [63:0]                               num [0:3];
   wire [63:0]                              averagex [0:3];
   wire [63:0]                              averagey [0:3];

   reg [20:0]                               top;
   reg [20:0]                               bottom;
   reg [20:0]                               left;
   reg [20:0]                               right;

   wire [10:0]                              topd;
   wire [10:0]                              bottomd;
   wire [10:0]                              leftd;
   wire [10:0]                              rightd;

   reg                                      startdivs;
```

```
51      wire [7:0]                              divsready;
        reg                                     calcdone;
53      wire [3:0]                              sqrtdone;
        reg                                     sqrtstart;
55
        // distance deltas
57      reg [9:0]                               dif_x[0:3];
        reg [8:0]                               dif_y[0:3];
59
        // STATE PARAMETERS
61      parameter COUNTING = 2'b00;
        parameter WAITING_FOR_DIVS = 2'b01;
63      parameter STARTSQRTS = 2'b10;
        parameter WAITING_FOR_SQRT = 2'b11;
65
        // parallelized dividers
67      divider #(.WIDTH(64)) diva(.clk(clk), .ready(divsready[0]), .dividend(sumx[0]),
                                   .divider(num[0]), .quotient(averagex[0]), .sign(1'b0), .start(
                                       startdivs));
69      divider #(.WIDTH(64)) divb(.clk(clk), .ready(divsready[1]), .dividend(sumy[0]),
                                   .divider(num[0]), .quotient(averagey[0]), .sign(1'b0), .start(
                                       startdivs));
71      divider #(.WIDTH(64)) divc(.clk(clk), .ready(divsready[2]), .dividend(sumx[1]),
                                   .divider(num[1]), .quotient(averagex[1]), .sign(1'b0), .start(
                                       startdivs));
73      divider #(.WIDTH(64)) divd(.clk(clk), .ready(divsready[3]), .dividend(sumy[1]),
                                   .divider(num[1]), .quotient(averagey[1]), .sign(1'b0), .start(
                                       startdivs));
75      divider #(.WIDTH(64)) dive(.clk(clk), .ready(divsready[4]), .dividend(sumx[2]),
                                   .divider(num[2]), .quotient(averagex[2]), .sign(1'b0), .start(
                                       startdivs));
77      divider #(.WIDTH(64)) divf(.clk(clk), .ready(divsready[5]), .dividend(sumy[2]),
                                   .divider(num[2]), .quotient(averagey[2]), .sign(1'b0), .start(
                                       startdivs));
79      divider #(.WIDTH(64)) divg(.clk(clk), .ready(divsready[6]), .dividend(sumx[3]),
                                   .divider(num[3]), .quotient(averagex[3]), .sign(1'b0), .start(
                                       startdivs));
81      divider #(.WIDTH(64)) divh(.clk(clk), .ready(divsready[7]), .dividend(sumy[3]),
                                   .divider(num[3]), .quotient(averagey[3]), .sign(1'b0), .start(
                                       startdivs));
83
        // parallelized square rooters
85      sqrt #(.NBITS(21)) sqrta(.clk(clk), .start(sqrtstart), .data(top),
                                 .answer(topd), .done(sqrtdone[0]));
87      sqrt #(.NBITS(21)) sqrtb(.clk(clk), .start(sqrtstart), .data(bottom),
                                 .answer(bottomd), .done(sqrtdone[1]));
89      sqrt #(.NBITS(21)) sqrtc(.clk(clk), .start(sqrtstart), .data(left),
                                 .answer(leftd), .done(sqrtdone[2]));
91            sqrt #(.NBITS(21)) sqrtd(.clk(clk), .start(sqrtstart), .data(right),
                                       .answer(rightd), .done(sqrtdone[3]));
93
        always @(posedge clk) begin
```

```
95          // intialize output
            corners_flag <= 0;
97          startdivs <= 0;
            sqrtstart <= 0;
99
            // on reset, reset accumulators, reset state to COUNTING
101         if (reset) begin
               sumx[0] <= 0;
103            sumy[0] <= 0;
               num[0] <= 0;
105            sumx[1] <= 0;
               sumy[1] <= 0;
107            num[1] <= 0;
               sumx[2] <= 0;
109            sumy[2] <= 0;
               num[2] <= 0;
111            sumx[3] <= 0;
               sumy[3] <= 0;
113            num[3] <= 0;
               state <= COUNTING;
115         end


117
            case (state)
119           COUNTING: begin
                 // reset corners flag
121              corners_flag <= 0;

123              // delay incoming flags, so that distance calculations are pipelined
                 delayed_interesting_flag <= interesting_flag;
125              delayed_interesting_x <= interesting_x;
                 delayed_interesting_y <= interesting_y;
127              delayed_color <= color;

129              // if a new pixel came in
                 if (interesting_flag) begin
131                 // calculate distances between incoming pixels and the previously generated
                        pixel

133                 dif_x[0] <= (interesting_x > a_x) ? interesting_x-a_x : a_x - interesting_x;
                    dif_y[0] <= (interesting_y > a_y) ? interesting_y-a_y : a_y - interesting_y;
135                 dif_x[1] <= (interesting_x > b_x) ? interesting_x-b_x : b_x - interesting_x;
                    dif_y[1] <= (interesting_y > b_y) ? interesting_y-b_y : b_y - interesting_y;
137                 dif_x[2] <= (interesting_x > c_x) ? interesting_x-c_x : c_x - interesting_x;
                    dif_y[2] <= (interesting_y > c_y) ? interesting_y-c_y : c_y - interesting_y;
139                 dif_x[3] <= (interesting_x > d_x) ? interesting_x-d_x : d_x - interesting_x;
                    dif_y[3] <= (interesting_y > d_y) ? interesting_y-d_y : d_y - interesting_y;
141              end

143              // if a new pixel came in one clock cycle ago
                 if (delayed_interesting_flag) begin
```

```
145             // weight this pixel based on its distance from the previously calculated
                    center of mass

147             if (dif_x[delayed_color] < 16 && dif_y[delayed_color] < 16) begin
                   sumx[delayed_color] <= sumx[delayed_color] + {delayed_interesting_x, 5'b0};
149                sumy[delayed_color] <= sumy[delayed_color] + {delayed_interesting_y, 5'b0};
                   num[delayed_color] <= num[delayed_color] + 32;
151             end
                else if (dif_x[delayed_color] < 32 && dif_y[delayed_color] < 32) begin
153                sumx[delayed_color] <= sumx[delayed_color] + {delayed_interesting_x,3'b0};
                   sumy[delayed_color] <= sumy[delayed_color] + {delayed_interesting_y,3'b0};
155                num[delayed_color] <= num[delayed_color] + 8;
                end
157             else if (dif_x[delayed_color] < 64 && dif_y[delayed_color] < 64) begin
                   sumx[delayed_color] <= sumx[delayed_color] + {delayed_interesting_x,2'b0};
159                sumy[delayed_color] <= sumy[delayed_color] + {delayed_interesting_y,2'b0};
                   num[delayed_color] <= num[delayed_color] + 4;
161             end
                else if (dif_x[delayed_color] < 128 && dif_y[delayed_color] < 128) begin
163                sumx[delayed_color] <= sumx[delayed_color] + {delayed_interesting_x, 1'b0};
                   sumy[delayed_color] <= sumy[delayed_color] + {delayed_interesting_y, 1'b0};
165                num[delayed_color] <= num[delayed_color] + 2;
                end
167             else begin
                   sumx[delayed_color] <= sumx[delayed_color] + delayed_interesting_x;
169                sumy[delayed_color] <= sumy[delayed_color] + delayed_interesting_y;
                   num[delayed_color] <= num[delayed_color] + 1;
171             end
             end
173
             // if the frame is over, begin calculating the new center of mass
175          if (frame_flag) begin
                startdivs <= 1;
177             state <= WAITING_FOR_DIVS;
             end
179       end

181       WAITING_FOR_DIVS: begin
             // if all of the dividers are done
183          if (&divsready) begin
                // new center of mass = calculated center of mass * (1/4) + old center of mass
                    * (3/4)
185             a_x <= (averagex[0] >> 2) + (a_x >> 1) + (a_x >> 2);
                b_x <= (averagex[1] >> 2) + (b_x >> 1) + (b_x >> 2);
187             c_x <= (averagex[2] >> 2) + (c_x >> 1) + (c_x >> 2);
                d_x <= (averagex[3] >> 2) + (d_x >> 1) + (d_x >> 2);
189             a_y <= (averagey[0] >> 2) + (a_y >> 1) + (a_y >> 2);
                b_y <= (averagey[1] >> 2) + (b_y >> 1) + (b_y >> 2);
191             c_y <= (averagey[2] >> 2) + (c_y >> 1) + (c_y >> 2);
                d_y <= (averagey[3] >> 2) + (d_y >> 1) + (d_y >> 2);
193
                // calculate distances squared
```

```
195             // sqrt does not begin here because of multiplier delay
                top <= (averagex[1] - averagex[0]) * (averagex[1] - averagex[0]) + (averagey
                    [1] - averagey[0]) * (averagey[1] - averagey[0]);
197             bottom <= (averagex[2] - averagex[3]) * (averagex[2] - averagex[3]) + (
                    averagey[2] - averagey[3]) * (averagey[2] - averagey[3]);
                left <= (averagex[3] - averagex[0]) * (averagex[3] - averagex[0]) + (averagey
                    [3] - averagey[0]) * (averagey[3] - averagey[0]);
199             right <= (averagex[2] - averagex[1]) * (averagex[2] - averagex[1]) + (averagey
                    [2] - averagey[1]) * (averagey[2] - averagey[1]);

201             state <= STARTSQRTS;

203         end // if (&divsready)
        end // case: WAITING_FOR_DIVS
205
        STARTSQRTS: begin
207         // start the square rooters
            sqrtstart <= 1;
209         state <= WAITING_FOR_SQRT;
        end
211
        WAITING_FOR_SQRT: begin
213         // if all sqrts are done;
            if (&sqrtdone) begin
215             if (topd < bottomd) m_x <= topd;
                else m_x <= bottomd;
217
                if (leftd < rightd) m_y <= leftd;
219             else m_y <= rightd;

221             corners_flag <= 1;
            end
223
            // if the vsync is over
225         if (~frame_flag) begin
                state <= COUNTING;
227
                // reset accumulators
229             sumx[0] <= 0;
                sumy[0] <= 0;
231             num[0] <= 0;
                sumx[1] <= 0;
233             sumy[1] <= 0;
                num[1] <= 0;
235             sumx[2] <= 0;
                sumy[2] <= 0;
237             num[2] <= 0;
                sumx[3] <= 0;
239             sumy[3] <= 0;
                num[3] <= 0;
241         end
        end // case: WAITING_FOR_SQRT
```

```
243
        endcase // case state
245    end // always @ (posedge clk)
endmodule // object_recognition
247

249 // takes integer square root iteratively
module sqrt #(parameter NBITS = 8,  // max 32
251               MBITS = (NBITS+1)/2)
    (input wire clk,start,
253     input wire [NBITS-1:0] data,
     output reg [MBITS-1:0] answer,
255     output wire            done);
    reg                    busy;
257    reg [4:0]              bit;
    // compute answer bit-by-bit, starting at MSB
259    wire [MBITS-1:0]       trial = answer | (1 << bit);
    always @(posedge clk) begin
261       if (busy) begin
          if (bit == 0) busy <= 0;
263          else bit <= bit - 1;
          if (trial*trial <= data) answer <= trial;
265       end
       else if (start) begin
267          busy <= 1;
          answer <= 0;
269          bit <= MBITS - 1;
       end
271    end
    assign done = ~busy;
273 endmodule // sqrt
```

A.4.1 test_object_recognition.v

```
1 `include "object_recognition.v";
  `include "divider.v";
3
  module test_object_recognition();
5    reg clk;
    reg [1:0] color;
7    reg [9:0] ix;
    reg [8:0] iy;
9    reg       int_flag;
    reg       frame_flag;
11
    wire [9:0] m_x;
13    wire [8:0] m_y;
    wire [9:0] a_x;
15    wire [8:0] a_y;
    wire [9:0] b_x;
17    wire [8:0] b_y;
```

```
    wire [9:0] c_x;
19  wire [8:0] c_y;
    wire [9:0] d_x;
21  wire [8:0] d_y;
    wire       corners_flag;
23
    integer    i;
25  reg        reset;

27  object_recognition obrec(.clk(clk), .color(color), .interesting_x(ix),
                             .interesting_y(iy), .interesting_flag(int_flag),
29                           .frame_flag(frame_flag), .m_x(m_x), .m_y(m_y),
                             .a_x(a_x), .a_y(a_y), .b_x(b_x), .b_y(b_y), .c_x(c_x),
31                           .c_y(c_y), .d_x(d_x), .d_y(d_y), .corners_flag(corners_flag), .
                                 reset(reset));

33  initial begin
       clk = 0;
35     frame_flag = 0;
       int_flag = 0;
37
       forever #10 clk = ~clk;
39  end

41  initial begin
       #20;
43
       reset = 1;
45
       #20
47
         reset = 0;
49     frame_flag = 0;

51     #20;

53     for (i = 0; i < 20; i = i + 1) begin
          int_flag = 1;
55        color = 0;
          ix = i;
57        iy = i;

59        #20;

61        color = 1;
          ix = i*2;
63        iy = i*2;

65        #20;

67        color = 2;
```

```
69            ix = i*3;
              iy = i*3;
71
              #20;
73
              color = 3;
75
              ix = i*4;
77            iy = i*4;

79            #20;
          end // for (i = 0; i < 20; i = i + 1)
81
          frame_flag = 1;
83
          #1000;
85
          $display("Average values:");
87        $display("a_x: %d", a_x);
          $display("a_y: %d", a_y);
89        $display("b_x: %d", b_x);
          $display("b_y: %d", b_y);
91        $display("c_x: %d", c_x);
          $display("c_y: %d", c_y);
93        $display("d_x: %d", d_x);
          $display("d_y: %d", d_y);
95
          $display("Min distances:");
97        $display("m_x: %d", m_x);
          $display("m_y: %d", m_y);
99    end // initial begin
endmodule // test_object_recognition
```

A.5 lpf.v

```
`default_nettype none
2 `include "params.v"

4 module dumb_lpf(
          input clock,
6         input reset,
          input frame_flag,
8         // memory_interface
          input done_lpf,
10        output reg lpf_flag,
          output lpf_wr,
12        output reg [`LOG_WIDTH-1:0] lpf_x,
          output reg [`LOG_HEIGHT-1:0] lpf_y,
14        output [`LOG_MEM-1:0] lpf_pixel_write,
          input [`LOG_MEM-1:0] lpf_pixel_read,
16        // projective_transform
```

```
        input request,
        output reg [`LOG_TRUNC-1:0] pixel,
        output [9:0] x_out,
        output [8:0] y_out,
        output pixel_flag,
        input testing
);

        reg advanced_pixel_flag;
        reg [`LOG_WIDTH-1:0] x;
        reg [`LOG_HEIGHT-1:0] y;
        reg pixel_flag_odd;

        // never writing
        assign lpf_wr = 1'b0;
        assign lpf_pixel_write = `LOG_MEM'd0;

        always @(*) begin
                // pulse lpf_flag only when x is even and a pixel is requested
                lpf_flag = request & ~lpf_x[0] & ~done_lpf;
                // pulse pixel flag when done_lpf is high and x[0] is even
                // or 1 cycle after request when lpf_x is odd
                advanced_pixel_flag = done_lpf | pixel_flag_odd;

                // x and y are the next set of coordinates
                if (reset || frame_flag) begin
                        x = 0;
                        y = 0;
                end
                else if (!advanced_pixel_flag) begin
                        x = lpf_x;
                        y = lpf_y;
                end
                else if (lpf_x == `IMAGE_WIDTH-1) begin
                        x = `LOG_WIDTH'd0;
                        y = lpf_y+1;
                end
                else begin
                        x = lpf_x+1;
                        y = lpf_y;
                end
        end

        always @(posedge clock) begin
                // update lpf_x and lpf_y
                lpf_x <= x;
                lpf_y <= y;
                pixel_flag_odd <= (request & x[0]);
        end

        wire pixel_flag_temp;

```

```
        // delay lpf_x, lpf_y | module is located in vga_write_new.v
70      delay #(.N(3), .LOG(10)) dx(.clock(clock), .reset(reset | frame_flag), .x(lpf_x), .y
            (x_out));
        delay #(.N(4), .LOG(9)) dy(.clock(clock), .reset(reset), .x(lpf_y), .y(y_out));
72      delay #(.N(3), .LOG(1)) df(.clock(clock), .reset(reset | frame_flag), .x(
            advanced_pixel_flag), .y(pixel_flag_temp));

74      assign pixel_flag = pixel_flag_temp & !frame_flag;

76      always @(*) begin
                if (testing)
78                      pixel = (x_out[0] == 1'b0) ? lpf_pixel_read[`LOG_MEM-1:`LOG_TRUNC] :
                            lpf_pixel_read[`LOG_TRUNC-1:0];
                else
80                      pixel = {{8{x_out[3] & y_out[3]}}, 5'b10000, 5'b10000};
        end
82  endmodule

84  module lpf(
        input clock,
86      input reset,
        input frame_flag,
88      // memory_interface
        input done_lpf,
90      output reg lpf_flag,
        output reg lpf_wr,
92      output reg [`LOG_WIDTH-1:0] lpf_x,
        output reg [`LOG_HEIGHT-1:0] lpf_y,
94      output reg [`LOG_MEM-1:0] lpf_pixel_write,
        input [`LOG_MEM-1:0] lpf_pixel_read,
96      // projective_transform
        input request,
98      output reg [`LOG_TRUNC-1:0] pixel,
        output reg [9:0] x_out,
100     output reg [8:0] y_out,
        output reg pixel_flag
102 );

104     parameter FILTER_LENGTH=41;
        parameter CENTER_LOC = 9'd20;
106     parameter FILTER_COEFF_WIDTH=10;
        reg [FILTER_LENGTH*FILTER_COEFF_WIDTH-1:0] col_coeffs;
108     reg [FILTER_LENGTH*FILTER_COEFF_WIDTH-1:0] row_coeffs;

110     reg [2:0] state;
        parameter COLS   = 3'd0;
112     parameter ROWS   = 3'd4;

114     reg [FILTER_LENGTH*`LOG_MEM-1:0] read_cols[0:3];

116     reg [(FILTER_LENGTH+8)*`LOG_TRUNC-1:0] next_sample_cols;
```

```
118         reg [`LOG_WIDTH-1:0] request_x;
            reg [`LOG_HEIGHT-1:0] request_y;
120         wire [`LOG_WIDTH-1:0] del_request_x;
            wire [`LOG_HEIGHT-1:0] del_request_y;
122         reg [`LOG_WIDTH-1:0] read_x;
            reg [`LOG_HEIGHT-1:0] read_y;
124         reg [`LOG_WIDTH-1:0] write_x;
            reg [`LOG_HEIGHT-1:0] write_y;
126
            wire del_done_lpf;
128         wire del_lpf_wr;
            delay #(.N(4)) dw(.clock(clock), .reset(reset), .x(lpf_wr), .y(del_lpf_wr));
130         delay #(.N(3)) dp(.clock(clock), .reset(reset), .x(done_lpf), .y(del_done_lpf));
            delay #(.N(4), .LOG(`LOG_WIDTH)) dx(.clock(clock), .reset(reset), .
132
            // fetching pixels from RAM
134         always @(posedge clock) begin
                if (!del_lpf_wr && del_done_lpf) begin
136                     next_sample_cols <= {next_sample_cols[(FILTER_LENGTH+8)*`LOG_TRUNC
                            -1:2*`LOG_TRUNC], lpf_pixel};
                        // TODO: update based on whether near the end
138                     read_x <= read_x;
                        read_y <= read_y+2;
140             end
                else begin
142                     next_sample_cols <= next_sample_cols;
                        read_x <= read_x;
144                     read_y <= read_y;
                end
146         end

148         // carry out mirroring here
            always @(posedge clock) begin
150
            end
152
            // decide whether to change states here
154         always @(posedge clock) begin
            end
156
            // decide whether to read, write, and request
158         always @(posedge clock) begin
                case (state)
160                     COLS: begin
                            // stall if memory_interface is busy
162                         if (lpf_flag == 1'b1 && done_lpf == 1'b0) begin
                                lpf_flag <= lpf_flag;
164                             lpf_wr <= lpf_wr;
                                lpf_x <= lpf_x;
166                             lpf_y <= lpf_y;
                                lpf_pixel_write <= lpf_pixel_write;
168                             write_y <= write_y;
```

```
                write_x <= write_x;
                request_y <= request_y;
                request_x <= request_x;
            end
            // write a pixel if ready
            else if ((read_y-write_y) >= CENTER_LOC) begin
                lpf_flag <= 1;
                lpf_wr <= 1;
                lpf_x <= write_x;
                lpf_y <= write_y;
                lpf_pixel_write <= {next_sample_cols[;
                write_y <= write_y+1;
                write_x <= write_x;
                request_y <= request_y;
                request_x <= request_x;
            end
            // request a pixel if the buffer isn't full
            else if ((request_y-read_y) < 8) begin
                lpf_flag <= 1;
                lpf_wr <= 0;
                lpf_x <= request_x;
                lpf_y <= request_y;
                lpf_pixel_write <= lpf_pixel_write;
                write_y <= write_y;
                write_x <= write_x;
                request_y <= request_y+2;
                request_x <= request_x;
            end
            // doing nothing
            else begin
                lpf_flag <= 1'b0;
                lpf_wr <= 1'b0;
                lpf_x <= lpf_x;
                lpf_y <= lpf_y;
                lpf_pixel_write <= lpf_pixel_write;
                write_y <= write_y;
                write_x <= write_x;
                request_y <= request_y;
                request_x <= request_x;
            end
        end
    ROWS: begin
            // stall if memory_interface is busy
            if (lpf_flag == 1'b1 && done_lpf == 1'b0) begin
                lpf_flag <= lpf_flag;
                lpf_wr <= lpf_wr;
                lpf_x <= lpf_x;
                lpf_y <= lpf_y;
                lpf_pixel_write <= lpf_pixel_write;
                write_y <= write_y;
                write_x <= write_x;
                request_y <= request_y;
```

```
                                        request_x <= request_x;
                                end
                                // write a pixel if ready
                                else if ((read_x-write_x) >= CENTER_LOC) begin
                                        lpf_flag <= 1;
                                        lpf_wr <= 1;
                                        lpf_x <= write_x;
                                        lpf_y <= write_y;
                                        lpf_pixel_write <= {next_sample_cols[;
                                        write_y <= write_y;
                                        write_x <= write_x+1;
                                        request_y <= request_y;
                                        request_x <= request_x;
                                end
                                // request a pixel if the buffer isn't full
                                else if ((request_x-read_x) < 8) begin
                                        lpf_flag <= 1;
                                        lpf_wr <= 0;
                                        lpf_x <= request_x;
                                        lpf_y <= request_y;
                                        lpf_pixel_write <= lpf_pixel_write;
                                        write_y <= write_y;
                                        write_x <= write_x;
                                        request_y <= request_y;
                                        request_x <= request_x+2;
                                end
                                // doing nothing
                                else begin
                                        lpf_flag <= 1'b0;
                                        lpf_wr <= 1'b0;
                                        lpf_x <= lpf_x;
                                        lpf_y <= lpf_y;
                                        lpf_pixel_write <= lpf_pixel_write;
                                        write_y <= write_y;
                                        write_x <= write_x;
                                        request_y <= request_y;
                                        request_x <= request_x;
                                end
                        end
                endcase
        end
endmodule
```

A.6 projective_transform.v

```
module projective_transform_srl(
                        input           clk, // System clock (global ->)
                        input           frame_flag, // New frame flag (ntsc_capture
                            ->)
                        input [17:0]    pixel, // Pixel data input (lpf ->)
                        input           pixel_flag, // New pixel recieved? (lpf ->)
```

```
                        input [9:0]       a_x, // coordinates of the corners
                        input [8:0]       a_y, //  |  (object_recognition ->)
                        input [9:0]       b_x, //  |
                        input [8:0]       b_y, //  |
                        input [9:0]       c_x, //  |
                        input [8:0]       c_y, //  |
                        input [9:0]       d_x, //  |
                        input [8:0]       d_y, //  |
                        input             corners_flag, // (object_recognition ->)

                        input done_pt,



                        input             ptflag, // Okay to send new data (
                            memory_interface ->)
                        output reg [17:0] pt_pixel_write, // Pixel data output (->
                            memory_interface)
                        output reg [9:0]  pt_x, // Pixel output data location
                        output reg [8:0]  pt_y, //  | (-> memory_interface)
                        output reg        pt_wr, // Want to write pixel flag (->
                            memory_interface)
                        output reg        request_pixel = 0 // request a pixel to
                            process (-> lpf)
                        );


    reg [1:0]                           state = 0;


    // signed numbers for computation with coordinates
    wire signed [10:0]                  sa_x;
    wire signed [10:0]                  sa_y;
    wire signed [10:0]                  sb_x;
    wire signed [10:0]                  sb_y;
    wire signed [10:0]                  sc_x;
    wire signed [10:0]                  sc_y;
    wire signed [10:0]                  sd_x;
    wire signed [10:0]                  sd_y;

    assign sa_x = {1'b0, a_x};
    assign sa_y = {2'b0, a_y};
    assign sb_x = {1'b0, b_x};
    assign sb_y = {2'b0, b_y};
    assign sc_x = {1'b0, c_x};
    assign sc_y = {2'b0, c_y};
    assign sd_x = {1'b0, d_x};
    assign sd_y = {2'b0, d_y};

    // iterator coordinates for the three iterator points
    // these all have 10 extra bits of resolution to simulate decimals
    //   (for example 1 is represented by 1 << 10)
```

```
54    reg [40:0]                              i_a_x;
      reg [40:0]                              i_a_y;
56    reg [40:0]                              i_b_x;
      reg [40:0]                              i_b_y;
58    reg [40:0]                              i_c_x;
      reg [40:0]                              i_c_y;
60
      // iterator incrementors
62    reg signed [41:0]                       delta_a_x;
      reg signed [41:0]                       delta_a_y;
64    reg signed [41:0]                       delta_b_x;
      reg signed [41:0]                       delta_b_y;
66    reg signed [41:0]                       delta_c_x;
      reg signed [41:0]                       delta_c_y;
68    reg signed [41:0]                       delta_c_x_next;
      reg signed [41:0]                       delta_c_y_next;
70
      // wires/registers for diving
72    wire                                    rfd_a;
      wire                                    rfd_b;
74    wire                                    rfd_c;
      wire                                    rfd_d;
76    wire                                    rfd_e;
      wire                                    rfd_f;
78
      reg signed [41:0]                       dividend_a;
80    reg signed [41:0]                       dividend_b;
      reg signed [41:0]                       dividend_c;
82    reg signed [41:0]                       dividend_d;
      reg signed [41:0]                       dividend_e;
84    reg signed [41:0]                       dividend_f;

86    reg signed [41:0]                       divisor_a;
      reg signed [41:0]                       divisor_b;
88    reg signed [41:0]                       divisor_c;
      reg signed [41:0]                       divisor_d;
90    reg signed [41:0]                       divisor_e;
      reg signed [41:0]                       divisor_f;
92
      wire signed [41:0]                      quotient_a;
94    wire signed [41:0]                      quotient_b;
      wire signed [41:0]                      quotient_c;
96    wire signed [41:0]                      quotient_d;
      wire signed [41:0]                      quotient_e;
98    wire signed [41:0]                      quotient_f;

100   reg                                     startdivs;

102   // coordinates iterators in the untransformed images
      reg [9:0]                               o_x;
104   reg [8:0]                               o_y;
```

```
106     // create some registers for dealing with possible delays
        // in memory_write
108     reg [17:0]                              pixel_save [0:15];
        reg [3:0]                               waiting_for_write = 0;
110
        parameter WAIT_FOR_CORNERS = 0;
112     parameter WAIT_FOR_DIVIDERS = 1;
        parameter WAIT_FOR_PIXEL = 2;
114
        // six dividers, for parallelization. these are used to calculate
116     // iteration "deltas"
        divider #(.WIDTH(42)) diva(.clk(clk), .ready(rfd_a), .dividend(dividend_a),
118                             .divider(divisor_a), .quotient(quotient_a), .sign(1'b1), .
                                    start(startdivs));

120     divider #(.WIDTH(42)) divb(.clk(clk), .ready(rfd_b), .dividend(dividend_b),
                                .divider(divisor_b), .quotient(quotient_b), .sign(1'b1), .
                                    start(startdivs));
122
        divider #(.WIDTH(42)) divc(.clk(clk), .ready(rfd_c), .dividend(dividend_c),
124                             .divider(divisor_c), .quotient(quotient_c), .sign(1'b1), .
                                    start(startdivs));

126     divider #(.WIDTH(42)) divd(.clk(clk), .ready(rfd_d), .dividend(dividend_d),
                                .divider(divisor_d), .quotient(quotient_d), .sign(1'b1), .
                                    start(startdivs));
128
        divider #(.WIDTH(42)) dive(.clk(clk), .ready(rfd_e), .dividend(dividend_e),
130                             .divider(divisor_e), .quotient(quotient_e), .sign(1'b1), .
                                    start(startdivs));

132     divider #(.WIDTH(42)) divf(.clk(clk), .ready(rfd_f), .dividend(dividend_f),
                                .divider(divisor_f), .quotient(quotient_f), .sign(1'b1), .
                                    start(startdivs));
134
        wire [17:0]                             buffered_pixel;
136

138     shift18 buffer(.clock(clk), .ce(pixel_flag), .dout(buffered_pixel), .length(
            waiting_for_write),
                        .din(pixel));
140
        always @(posedge clk) begin
142        case(state)
             WAIT_FOR_CORNERS: begin
144             o_x <= 0;
                o_y <= 0;
146
                if (corners_flag) begin
148
                   i_a_x <= a_x << 30;
150                i_a_y <= a_y << 30;
```

```
            i_b_x <= b_x << 30;
            i_b_y <= b_y << 30;
            i_c_x <= a_x << 30;
            i_c_y <= a_y << 30;

            //start dividers
            dividend_a <= (sd_x - sa_x) << 30;
            dividend_b <= (sd_y - sa_y) << 30;
            dividend_c <= (sc_x - sb_x) << 30;
            dividend_d <= (sc_y - sb_y) << 30;
            dividend_e <= (sb_x - sa_x) << 30;
            dividend_f <= (sb_y - sa_y) << 30;

            divisor_a <= 480;
            divisor_b <= 480;
            divisor_c <= 480;
            divisor_d <= 480;
            divisor_e <= 640;
            divisor_f <= 640;

            startdivs <= 1;

            // update state
            state <= WAIT_FOR_DIVIDERS;

        end // if (corners_flag)
    end // case: WAIT_FOR_CORNERS

    WAIT_FOR_DIVIDERS: begin
        startdivs <= 0;

        // if divider is done (divider delay = M + 4)
        // M = dividend width = 20 in this case
        if (rfd_a & rfd_b & rfd_c & rfd_d & rfd_e & rfd_f) begin
            request_pixel <= 1;

            delta_a_x <= quotient_a;
            delta_a_y <= quotient_b;
            delta_b_x <= quotient_c;
            delta_b_y <= quotient_d;
            delta_c_x <= quotient_e;
            delta_c_y <= quotient_f;

            // update state
            state <= WAIT_FOR_PIXEL;
        end
    end

    // This is the state where the bulk of the module is accomplished.
    // This waits for LPF to send a new pixel value to projective_transform,
    // then echoes that value and new coords to the memory management module.
    // then it increments the iterators accordingly.
```

```
        WAIT_FOR_PIXEL: begin
204
            // a new pixel has arrived, process accordingly
206         if (pixel_flag || (|waiting_for_write)) begin
               if (ptflag) begin
208
                  pt_pixel_write <= buffered_pixel;
210
                  // if we just read a pixel out of the buffer, decrement the
212               // buffer count
                  if (~pixel_flag) begin
214                  waiting_for_write <= waiting_for_write - 1;
                  end
216
                  // if there are less than 5 pixels in the buffer (LPF delay)
218               // request a new pixel now.
                  if (waiting_for_write < 5) begin
220                  request_pixel <= 1;
                  end else begin
222                  request_pixel <= 0;
                  end
224

226               pt_x <= (i_c_x >> 30);
                  pt_y <= (i_c_y >> 30);
228               pt_wr <= 1;

230               // increment iterators
                  i_c_x <= i_c_x + delta_c_x;
232               i_c_y <= i_c_y + delta_c_y;
                  o_x <= o_x + 1;
234
                  // we are getting close to the end of this line. begin calculating the
236               // next lines deltas and distances.
                  if (o_x == 500) begin
238                  // start dividers
                     divisor_a <= 640;
240                  divisor_b <= 640;

242                  dividend_a <= ((i_b_x + delta_b_x) - (i_a_x + delta_a_x));
                     dividend_b <= ((i_b_y + delta_b_y) - (i_a_y + delta_a_y));
244
                     startdivs <= 1;
246
                  end else startdivs <= 0;
248
                  // the end of the line
250               if (o_x == 639 && o_y < 479) begin
                     // increment iterators
252                  o_y <= o_y + 1;
                     i_a_x <= i_a_x + delta_a_x;
254                  i_a_y <= i_a_y + delta_a_y;
```

```
                i_b_x <= i_b_x + delta_b_x;
256             i_b_y <= i_b_y + delta_b_y;

258             // reset I_C to the new location of I_A
                i_c_x <= i_a_x + delta_a_x;
260             i_c_y <= i_a_y + delta_a_y;

262             // update the deltas
                delta_c_x <= delta_c_x_next;
264             delta_c_y <= delta_c_y_next;

266             // reset o_x
                o_x <= 0;
268         end

270         // the end of the frame
            if ((o_x == 639 && o_y == 479)) begin
272             // reset the iterator variables
                o_x <= 0;
274             o_y <= 0;

276             // the other iterators will be reset when new corners arrive

278             // go back to waiting
                state <= WAIT_FOR_CORNERS;
280             pt_wr <= 0;
                request_pixel <= 0;
282
            end
284     end else begin // if (ptflag)
            if (pixel_flag) begin
286             waiting_for_write <= waiting_for_write + 1; // set a flag
            end
288
            request_pixel <= 0; // memory_interface is delayed, we do not
290         // want to deal with new pixels right now

292         pt_wr <= 0;
        end
294
    end else pt_wr <= 0; // if (pixel_flag || (waiting_for_write > 0))
296
    // if the divider is done
298 if (rfd_a & rfd_b) begin
        // save deltas
300     delta_c_x_next <= quotient_a;
        delta_c_y_next <= quotient_b;
302 end

304 // reset iterators when a new frame arrives
    if (frame_flag) begin
306     state <= WAIT_FOR_CORNERS;
```

```
                pt_wr <= 0;
308             o_x <= 0;
                o_y <= 0;
310         end


312
        end // case: WAIT_FOR_PIXEL
314
      endcase // case (state)
316   end // always @ (posedge clk)
endmodule // projective_transform
318


320 // The shift18 module provides a 18x16 shift register, to buffer pixel data
    // that arrives from LPF. The length input selects how long the shift register is,
322 // i.e., what flip flop creates the output.
    module shift18(input [17:0] din,
324                input [3:0] length,
                   output [17:0] dout,
326                input clock,
                   input ce);
328
   SRL16E s1(.CLK(clock), .CE(ce), .D(din[0]),
330          .A0(length[0]), .A1(length[1]),
             .A2(length[2]), .A3(length[3]),
332          .Q(dout[0]));
   SRL16E s2(.CLK(clock), .CE(ce), .D(din[1]),
334          .A0(length[0]), .A1(length[1]),
             .A2(length[2]), .A3(length[3]),
336          .Q(dout[1]));
   SRL16E s3(.CLK(clock), .CE(ce), .D(din[2]),
338          .A0(length[0]), .A1(length[1]),
             .A2(length[2]), .A3(length[3]),
340          .Q(dout[2]));
   SRL16E s4(.CLK(clock), .CE(ce), .D(din[3]),
342          .A0(length[0]), .A1(length[1]),
             .A2(length[2]), .A3(length[3]),
344          .Q(dout[3]));
   SRL16E s5(.CLK(clock), .CE(ce), .D(din[4]),
346          .A0(length[0]), .A1(length[1]),
             .A2(length[2]), .A3(length[3]),
348          .Q(dout[4]));
   SRL16E s6(.CLK(clock), .CE(ce), .D(din[5]),
350          .A0(length[0]), .A1(length[1]),
             .A2(length[2]), .A3(length[3]),
352          .Q(dout[5]));
   SRL16E s7(.CLK(clock), .CE(ce), .D(din[6]),
354          .A0(length[0]), .A1(length[1]),
             .A2(length[2]), .A3(length[3]),
356          .Q(dout[6]));
   SRL16E s8(.CLK(clock), .CE(ce), .D(din[7]),
358          .A0(length[0]), .A1(length[1]),
```

```
                .A2(length[2]), .A3(length[3]),
360             .Q(dout[7]));
    SRL16E s9(.CLK(clock), .CE(ce), .D(din[8]),
362             .A0(length[0]), .A1(length[1]),
                .A2(length[2]), .A3(length[3]),
364             .Q(dout[8]));
    SRL16E s10(.CLK(clock), .CE(ce), .D(din[9]),
366              .A0(length[0]), .A1(length[1]),
                 .A2(length[2]), .A3(length[3]),
368              .Q(dout[9]));
    SRL16E s11(.CLK(clock), .CE(ce), .D(din[10]),
370             .A0(length[0]), .A1(length[1]),
                .A2(length[2]), .A3(length[3]),
372             .Q(dout[10]));
    SRL16E s12(.CLK(clock), .CE(ce), .D(din[11]),
374             .A0(length[0]), .A1(length[1]),
                .A2(length[2]), .A3(length[3]),
376             .Q(dout[11]));
    SRL16E s13(.CLK(clock), .CE(ce), .D(din[12]),
378             .A0(length[0]), .A1(length[1]),
                .A2(length[2]), .A3(length[3]),
380             .Q(dout[12]));
    SRL16E s14(.CLK(clock), .CE(ce), .D(din[13]),
382             .A0(length[0]), .A1(length[1]),
                .A2(length[2]), .A3(length[3]),
384             .Q(dout[13]));
    SRL16E s15(.CLK(clock), .CE(ce), .D(din[14]),
386             .A0(length[0]), .A1(length[1]),
                .A2(length[2]), .A3(length[3]),
388             .Q(dout[14]));
    SRL16E s16(.CLK(clock), .CE(ce), .D(din[15]),
390             .A0(length[0]), .A1(length[1]),
                .A2(length[2]), .A3(length[3]),
392             .Q(dout[15]));
    SRL16E s17(.CLK(clock), .CE(ce), .D(din[16]),
394             .A0(length[0]), .A1(length[1]),
                .A2(length[2]), .A3(length[3]),
396             .Q(dout[16]));
    SRL16E s18(.CLK(clock), .CE(ce), .D(din[17]),
398             .A0(length[0]), .A1(length[1]),
                .A2(length[2]), .A3(length[3]),
400             .Q(dout[17]));

402 endmodule // shift18
```

A.6.1 projective_transform_testbench.v

```
`timescale 1ns / 100ps
2
module projective_transform_test();
4   // registers and wires for connection to the PT module
```

```
    reg clk;
    reg frame_flag;
    reg [17:0] pixel;
    reg        pixel_flag;
    reg [9:0]  a_x;
    reg [8:0]  a_y;
    reg [9:0]  b_x;
    reg [8:0]  b_y;
    reg [9:0]  c_x;
    reg [8:0]  c_y;
    reg [9:0]  d_x;
    reg [8:0]  d_y;
    reg        ptflag;
    reg        cf;

    wire [17:0] pixel_output;
    wire [9:0]  new_x;
    wire [8:0]  new_y;
    wire        wr;
    wire        request_pixel;

    // instantiate the projective_transform module
    projective_transform pt(.clk(clk), .frame_flag(frame_flag), .pixel(pixel),
                            .pixel_flag(pixel_flag), .a_x(a_x), .a_y(a_y), .b_x(b_x),
                            .b_y(b_y), .c_x(c_x), .c_y(c_y), .d_x(d_x), .d_y(d_y), .
                                corners_flag(cf),
                            .ptflag(ptflag), .pt_pixel_write(pixel_output),
                            .pt_x(new_x), .pt_y(new_y), .pt_wr(wr),
                            .request_pixel(request_pixel));

    integer     fin, fout,code;

    initial begin
       // open a file that contains a 640x480 image, stored linearly.
       // it really contains:
       //     [0]  [1]  [2]  [3] .... [639]
       //     [640][641][642][643] .... [1279]
       //       :
       //     [44576] .................[45055] (it has wrapped around after getting
       //                                       to 2^17, but it still linearly
       //                                       incrementing)
       fin = $fopen("sample_image.image","r");

       // create a file for output. this will contain the coordinates and pixel value
       // of each processed input pixel. this file can be read by MATLAB and displayed
       // to ensure that projective_transform is operating correctly.
       fout = $fopen("sample_output.image","w");

       if (fin == 0 || fout == 0) begin
          $display("Can't open file.");
          $stop;
       end
```

```
56
        // set register default values
58      clk = 0;
        frame_flag = 0;
60      pixel_flag = 0;
        ptflag = 1;
62
        #1000;
64
        // send some made up frame values
66      a_x = 0;
        a_y = 0;
68
        b_x = 300;
70      b_y = 100;

72      c_x = 250;
        c_y = 150;
74
        d_x = 50;
76      d_y = 200;

78      #20;
        // set a frame flag
80      frame_flag = 1;
        cf = 1;
82
    end // initial begin
84
    // generate a 50 Mhz clock
86  always #10 clk = ~clk;

88  always @(posedge clk) begin
        frame_flag <= 0;
90      cf <=0;

92      if (request_pixel) begin
            code = $fscanf(fin, "%d", pixel);
94          pixel_flag = 1;

96          if (code != 1) begin
                $fclose(fout);
98              $stop;
            end
100     end else begin
            pixel_flag = 0;
102     end

104     if (wr) begin
            $fdisplay(fout, "%d, %d, %d", new_x, new_y, pixel_output);
106     end
    end
```

```
108
endmodule // projective_transform_test
```

A.7 divider.v

```
1
// The divider module divides one number by another. It
3 // produces a signal named "ready" when the quotient output
// is ready, and takes a signal named "start" to indicate
5 // the the input dividend and divider is ready.
//     sign -- 0 for unsigned, 1 for twos complement
7
// It uses a simple restoring divide algorithm.
9 // http://en.wikipedia.org/wiki/Division_(digital)#Restoring_division
module divider #(parameter WIDTH = 8) (ready, start, quotient,
11                                     remainder, dividend,
                                       divider, sign, clk);
13
   input           clk;
15 input           sign;
   input           start;
17 input [WIDTH-1:0] dividend, divider;
   output [WIDTH-1:0] quotient, remainder;
19 output          ready;

21 reg [WIDTH-1:0]     quotient, quotient_temp;
   reg [WIDTH*2-1:0]   dividend_copy, divider_copy, diff;
23 reg                 negative_output;

25 wire [WIDTH-1:0]    remainder = (!negative_output) ?
                       dividend_copy[31:0] :
27                     ~dividend_copy[31:0] + 1'b1;

29 reg [7:0]           bit;
   reg                 del_ready = 1;
31 wire                ready = (!bit) & ~del_ready;

33 wire [WIDTH-2:0]    zeros = 0;
   initial bit = 0;
35 initial negative_output = 0;

37 always @( posedge clk ) begin
      del_ready <= !bit;
39    if( start ) begin

41       bit = WIDTH;
         quotient = 0;
43       quotient_temp = 0;
         dividend_copy = (!sign || !dividend[WIDTH-1]) ?
45                       {1'b0,zeros,dividend} :
                         {1'b0,zeros,~dividend + 1'b1};
```

```
47          divider_copy = (!sign || !divider[WIDTH-1]) ?
                           {1'b0,divider,zeros} :
49                         {1'b0,~divider + 1'b1,zeros};

51          negative_output = sign &&
                              ((divider[WIDTH-1] && !dividend[WIDTH-1])
53                             ||(!divider[WIDTH-1] && dividend[WIDTH-1]));

55       end
         else if ( bit > 0 ) begin
57
            diff = dividend_copy - divider_copy;
59
            quotient_temp = quotient_temp << 1;
61
            if( !diff[WIDTH*2-1] ) begin
63
               dividend_copy = diff;
65             quotient_temp[0] = 1'd1;

67          end

69          quotient = (!negative_output) ?
                       quotient_temp :
71                     ~quotient_temp + 1'b1;

73          divider_copy = divider_copy >> 1;
            bit = bit - 1'b1;
75
         end
77    end
endmodule
```

A.8 clock_gen.v

```
1  `default_nettype none

3  module clock_gen(
                 input reset_button,
5                input clock_27mhz,
                 input clock_feedback_in,
7                output clock_feedback_out,
                 output clock_50mhz,
9                output clock_50mhz_90,
                 output clock_50mhz_270,
11               output clock_25mhz,
                 output ram0_clk,
13               output ram1_clk,
                 output locked_ram,
15               output locked_25mhz
           );
```

```
17
        wire reset_dcm;
19      debounce db_reset(.clock(clock_27mhz), .noisy(~reset_button), .clean(reset_dcm));

21      // generate 50 mhz clock
        wire clock_50mhz_unbuf, clock_50mhz_buf;
23      DCM vclk1(.CLKIN(clock_27mhz), .CLKFX(clock_50mhz_unbuf), .RST(reset_dcm));
        // synthesis attribute CLKFX_DIVIDE of vclk1 is 15
25      // synthesis attribute CLKFX_MULTIPLY of vclk1 is 28
        // synthesis attribute CLK_FEEDBACK of vclk1 is "NONE"
27      // synthesis attribute CLKIN_PERIOD of vclk1 is 37
        BUFG vclk2(.O(clock_50mhz_buf),.I(clock_50mhz_unbuf));
29
        ramclock rc(.ref_clock(clock_50mhz_buf), .fpga_clock(clock_50mhz), .fpga_clock_d2(
            clock_25mhz),
31                  .fpga_clock_90(clock_50mhz_90), .fpga_clock_270(clock_50mhz_270), .
                        ram0_clock(ram0_clk), .ram1_clock(ram1_clk),
                    .clock_feedback_in(clock_feedback_in), .clock_feedback_out(
                        clock_feedback_out),
33                  .locked(locked_ram));

35      assign locked_25mhz = locked_ram;
endmodule
37


39 // ramclock module
   ////////////////////////////////////////////////////////////////////////////////
41 //
   // 6.111 FPGA Labkit -- ZBT RAM clock generation
43 //
   //
45 // Created: April 27, 2004
   // Author: Nathan Ickes
47 //
   ////////////////////////////////////////////////////////////////////////////////
49 //
   // This module generates deskewed clocks for driving the ZBT SRAMs and FPGA
51 // 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
53 // rising clock edges reach the RAMs at exactly the same time as rising clock
   // edges reach the registers in the FPGA.
55 //
   // The RAM clock signals are driven by DDR output buffers, which further
57 // ensures that the clock-to-pad delay is the same for the RAM clocks as it is
   // for any other registered RAM signal.
59 //
   // When the FPGA is configured, the DCMs are enabled before the chip-level I/O
61 // drivers are released from tristate. It is therefore necessary to
   // artificially hold the DCMs in reset for a few cycles after configuration.
63 // 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
65 // 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, fpga_clock_d2, fpga_clock_90, fpga_clock_270,
    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 fpga_clock_d2;
   output fpga_clock_90;
        output fpga_clock_270;
   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, fpga_clk_d2, fpga_clk_90, fpga_clk_270, ram_clk, fb_clk, lock1,
       lock2, dcm_reset, ram_clock;

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

   // IBUFG ref_buf (.O(ref_clk), .I(ref_clock));
        assign ref_clk = ref_clock;

   BUFG int_buf (.O(fpga_clock), .I(fpga_clk));
   BUFG int_buf_d2 (.O(fpga_clock_d2), .I(fpga_clk_d2));
   BUFG int_buf_inv (.O(fpga_clock_90), .I(fpga_clk_90));
        BUFG int_buf_270 (.O(fpga_clock_270), .I(fpga_clk_270));

   DCM int_dcm (.CLKFB(fpga_clock),
                .CLKIN(ref_clk),
                .RST(dcm_reset),
                .CLK0(fpga_clk),
                .CLKDV(fpga_clk_d2),
                .CLK90(fpga_clk_90),
                .CLK180(fpga_clk_270),
                .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 "FIXED"
   // synthesis attribute PHASE_SHIFT of int_dcm is 0
   // synthesis attribute CLKDV_DIVIDE of int_dcm is 2

   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
```

A.9 vga_write.v

```
`include "params.v"
`default_nettype none

module stupid_vga_write
        (
                // STANDARD INPUTS
                input clock,
                input vclock,
                input reset,
                input frame_flag,
                // MEMORY_INTERFACE
                input [`LOG_MEM-1:0] vga_pixel,
                input done_vga,
                output reg vga_flag,
                // VGA
                output reg [7:0] vga_out_red,
                output reg [7:0] vga_out_green,
                output reg [7:0] vga_out_blue,
                output vga_out_sync_b,
                output reg vga_out_blank_b,
```

```
21              output vga_out_pixel_clock,
                output reg vga_out_hsync,
23              output reg vga_out_vsync,
                // ADDRESSING
25              output reg [`LOG_HCOUNT-1:0] clocked_hcount,
                output reg [`LOG_VCOUNT-1:0] clocked_vcount,
27
                input [9:0] a_x, b_x, c_x, d_x,
29              input [8:0] a_y, b_y, c_y, d_y,

31              output reg vga_will_request,

33              input enable_xhairs
        );
35
        // generate hcount, vcount, syncs, and blank
37      wire [`LOG_HCOUNT-1:0] hcount;
        wire [`LOG_VCOUNT-1:0] vcount;
39      wire hsync, vsync, blank;
        stupid_xvga xvga1(
41              .vclock(vclock), .reset(reset), .hcount(hcount),
                .vcount(vcount), .vsync(vsync), .hsync(hsync), .blank(blank));
43
        // account for delay in memory
45      wire [9:0] del_hcount;
        wire [9:0] del_vcount;
47      wire del_vsync, del_hsync, del_blank;
        parameter DELAY=2;
49      delay2 #(.LOG(10)) dhc(.clock(vclock), .reset(reset), .x(hcount), .y(del_hcount));
        delay2 #(.LOG(10)) dvc(.clock(vclock), .reset(reset), .x(vcount), .y(del_vcount));
51      delay2 #(.LOG(1)) dhs(.clock(vclock), .reset(reset), .x(hsync), .y(del_hsync));
        delay2 #(.LOG(1)) dvs(.clock(vclock), .reset(reset), .x(vsync), .y(del_vsync));
53      delay2 #(.LOG(1)) db(.clock(vclock), .reset(reset), .x(blank), .y(del_blank));

55      // assign color based on pixel fetched from memory
        wire [7:0] r;
57      wire [7:0] g;
        wire [7:0] b;
59      reg [35:0] pixel;
        ycrcb_lut ycc(
61              .clock(vclock),
                .ycrcb(del_hcount[0] ? pixel[35:18] : pixel[17:0]),
63              .r(r), .g(g), .b(b));

65      reg [1:0] count;
        always @(posedge clock) begin
67              // start count when both clock and vclock are rising (in sync)
                if (count != 2'd0) count <= count+1;
69              else if (!vclock && hcount[0]) count <= 1;
                else count <= 2'd0;
71
                // grab pixel right before it changes
```

```
73                  if (count == 2'd3) pixel <= vga_pixel;
                    else pixel <= pixel;
75
                    // assign address of next pixel
77                  // generate vga_flag 1 out of every 4 clock cycles
                    if (count == 2'd2) begin
79                          vga_flag <= 1;
                            clocked_hcount[9:0] <= hcount[9:0];
81                          clocked_vcount[9:0] <= vcount[9:0];
                    end
83                  else begin
                            vga_flag <= 0;
85                          clocked_hcount[9:0] <= clocked_hcount[9:0];
                            clocked_vcount[9:0] <= clocked_vcount[9:0];
87                  end

89                  vga_will_request <= (count == 2'd2);
            end
91
            // pipeline crosshair calculations
93          reg [3:0] pre_crosshairs;
            reg [3:0] crosshairs;
95          always @(posedge vclock) begin
                    pre_crosshairs[0] <= enable_xhairs & (del_hcount == a_x | del_vcount == a_y)
                        ;
97                  pre_crosshairs[1] <= enable_xhairs & (del_hcount == b_x | del_vcount == b_y)
                        ;
                    pre_crosshairs[2] <= enable_xhairs & (del_hcount == c_x | del_vcount == c_y)
                        ;
99                  pre_crosshairs[3] <= enable_xhairs & (del_hcount == d_x | del_vcount == d_y)
                        ;
                    crosshairs <= pre_crosshairs;
101         end

103         // delay blank, hsync, vsync to account for r,g,b 2 cycle delay
            wire out_hsync;
105         wire out_vsync;
            wire out_blank;
107         delay2 #(.LOG(1)) dhso(.clock(vclock), .reset(reset), .x(del_hsync), .y(out_hsync));
            delay2 #(.LOG(1)) dvso(.clock(vclock), .reset(reset), .x(del_vsync), .y(out_vsync));
109         delay2 #(.LOG(1)) dbo(.clock(vclock), .reset(reset), .x(del_blank), .y(out_blank));

111         // assign outputs to VGA chip
            always @(posedge vclock) begin
113                 if (out_blank) begin
                            vga_out_red[7:0] <= 8'd0;
115                         vga_out_green[7:0] <= 8'd0;
                            vga_out_blue[7:0] <= 8'd0;
117                 end else if (crosshairs[0]) begin
                            vga_out_red[7:0] <= 8'hFF;
119                         vga_out_green[7:0] <= 8'h00;
                            vga_out_blue[7:0] <= 8'hFF;
```

```
121                 end else if (crosshairs[1]) begin
                            vga_out_red[7:0] <= 8'hFF;
123                         vga_out_green[7:0] <= 8'hFF;
                            vga_out_blue[7:0] <= 8'h00;
125                 end else if (crosshairs[2]) begin
                            vga_out_red[7:0] <= 8'h00;
127                         vga_out_green[7:0] <= 8'hFF;
                            vga_out_blue[7:0] <= 8'hFF;
129                 end else if (crosshairs[3]) begin
                            vga_out_red[7:0] <= 8'hFF;
131                         vga_out_green[7:0] <= 8'h00;
                            vga_out_blue[7:0] <= 8'h00;
133                 end else begin
                            vga_out_red[7:0] <= r[7:0];
135                         vga_out_green[7:0] <= g[7:0];
                            vga_out_blue[7:0] <= b[7:0];
137                 end

139                 vga_out_blank_b <= ~out_blank;
                    vga_out_hsync <= out_hsync;
141                 vga_out_vsync <= out_vsync;
            end
143         /*
            // generate video clock
145         wire vga_out_pixel_clk;
            wire lock_vga;
147         BUFG bufgv1(.O(vga_out_pixel_clock), .I(vga_out_pixel_clk));
            DCM vga_dcm (
149                 .CLKIN(vclock),
                    .CLK180(vga_out_pixel_clk),
151                 .LOCKED(lock_vga));
    // synthesis attribute DLL_FREQUENCY_MODE of vga_dcm is "LOW"
153 // synthesis attribute DUTY_CYCLE_CORRECTION of vga_dcm is "TRUE"
    // synthesis attribute STARTUP_WAIT of vga_dcm is "FALSE"
155 // synthesis attribute DFS_FREQUENCY_MODE of vga_dcm is "LOW"
    // synthesis attribute CLK_FEEDBACK of vga_dcm  is "NONE"
157 // synthesis attribute CLKOUT_PHASE_SHIFT of vga_dcm is "NONE"
    // synthesis attribute PHASE_SHIFT of vga_dcm is 0
159         */
            assign vga_out_pixel_clock = ~vclock;
161         assign vga_out_sync_b = 1'b1;
endmodule
163
////////////////////////////////////////////////////////////////////////////////
165 //
// xvga: Generate XVGA display signals (640 x 480 @ 60Hz)
167 //
////////////////////////////////////////////////////////////////////////////////
169 module stupid_xvga
        (
171                 input vclock,
                    input reset,
```

```
173                 input frame_flag,
                    output reg [`LOG_HCOUNT-1:0] hcount, // pixel number on current line
175                 output reg [`LOG_VCOUNT-1:0] vcount, // line number
                    output reg vsync, hsync, blank
177         );

179         // horizontal: 800 pixels total
            // display 640 pixels per line
181         reg hblank,vblank;
            wire hsyncon,hsyncoff,hreset,hblankon;
183         assign hblankon = (hcount == `VGA_HBLANKON);
            assign hsyncon = (hcount == `VGA_HSYNCON); // activated at end of front porch
185         assign hsyncoff = (hcount == `VGA_HSYNCOFF); // activated at end of sync interval
            assign hreset = (hcount == `VGA_HRESET); // activated at end of
187         // vertical: 524 lines total
            // display 480 lines
189         wire vsyncon,vsyncoff,vreset,vblankon;
            assign vblankon = hreset & (vcount == `VGA_VBLANKON);
191         assign vsyncon = hreset & (vcount == `VGA_VSYNCON);
            assign vsyncoff = hreset & (vcount == `VGA_VSYNCOFF);
193         assign vreset = hreset & (vcount == `VGA_VRESET);
            // sync and blanking
195         wire next_hblank,next_vblank;
            assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
197         assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;

199         always @(posedge vclock) begin
                    // TODO: revise this section
201                 // is it necessary?
                    if (reset) begin
203                         hcount <= 0;
                            hblank <= 0;
205                         hsync <= 1;

207                         vcount <= 0;
                            vblank <= 0;
209                         vsync <= 1;

211                         blank <= 0;
                    end
213                 else begin
                            hcount <= hreset ? 0 : hcount + 1;
215                         hblank <= next_hblank;
                            hsync <= hsyncon ? 0 : hsyncoff ? 1 : hsync; // active low
217
                            vcount <= hreset ? (vreset ? 0 : vcount + 1) : vcount;
219                         vblank <= next_vblank;
                            vsync <= vsyncon ? 0 : vsyncoff ? 1 : vsync; // active low
221
                            blank <= next_vblank | (next_hblank & ~hreset);
223                 end
            end
```

```
225 endmodule

227 module delay2 #(parameter LOG=1)
        (
229             input clock,
                input reset,
231             input [LOG-1:0] x,
                output reg [LOG-1:0] y
233     );

235     reg [LOG-1:0] d;

237     always @(posedge clock) begin
                if(reset) begin
239                     d <= 0;
                        y <= 0;
241             end
                else begin
243                     d <= x;
                        y <= d;
245             end
        end
247 endmodule

249 module delay #(parameter N=3, LOG=1)
        (
251             input clock,
                input reset,
253             input [LOG-1:0] x,
                output reg [LOG-1:0] y
255     );

257     reg [(N-1)*LOG-1:0] d;

259     always @(posedge clock) begin
                if (reset) begin
261                     d <= 0;
                        y <= 0;
263             end
                else begin
265                     d[LOG-1:0] <= x[LOG-1:0];
                        d[(N-1)*LOG-1:LOG] <= d[(N-2)*LOG-1:0];
267                     y[LOG-1:0] <= d[(N-1)*LOG-1:(N-2)*LOG];
                end
269     end
    endmodule
```

A.10 ycbcr2rgb.v

```
1 `default_nettype none
2
```

```
// converts YCrCb to RGB
module ycbcr2rgb
        (
                input clock,
                input reset,
                input [7:0] y,
                input [7:0] cb,
                input [7:0] cr,
                output reg [7:0] r,
                output reg [7:0] g,
                output reg [7:0] b
        );

        reg signed [8:0] y_fixed, cb_fixed, cr_fixed;

        // fix y, cb, cr values, to ensure that they're in valid ranges
        always @(*) begin
                if (y < 8'd16) y_fixed[8:0] = 9'sd16;
                else if (y > 8'd235) y_fixed[8:0] = 9'sd235;
                else y_fixed[8:0] = {1'b0, y};

                if (cb < 8'd16) cb_fixed[8:0] = 9'sd16;
                else if (cb > 8'd235) cb_fixed[8:0] = 9'sd235;
                else cb_fixed[8:0] = {1'b0, cb};

                if (cr < 8'd16) cr_fixed[8:0] = 9'sd16;
                else if (cr > 8'd235) cr_fixed[8:0] = 9'sd235;
                else cr_fixed[8:0] = {1'b0, cr};
        end

        // constants used in multiplication (*2^11)
        parameter RGB_Y = 14'sd2383; // 1.164

        parameter R_CR  = 14'sd3269; // 1.596
        parameter G_CR  = 14'sd1665; // 0.813

        parameter G_CB  = 14'sd803;  // 0.392
        parameter B_CB  = 14'sd4131;// 2.017

        // outputs of multiplications bitwidth=14+9+1 due to possible overflow
        reg signed [23:0] R_scaled;
        reg signed [23:0] G_scaled;
        reg signed [23:0] B_scaled;

        reg signed [9:0] R_signed;
        reg signed [9:0] G_signed;
        reg signed [9:0] B_signed;

        always @(*) begin
                // transformation
                R_scaled = RGB_Y*(y_fixed-9'sd16) + R_CR*(cr_fixed-9'sd128);
```

```
54                  G_scaled = RGB_Y*(y_fixed-9'sd16) - G_CR*(cr_fixed-9'sd128) - G_CB*(cb_fixed
                        -9'sd128);
                    B_scaled = RGB_Y*(y_fixed-9'sd16) + B_CB*(cb_fixed-9'sd128);
56
                    // scaling down
58                  R_signed = R_scaled >>> 11;
                    G_signed = G_scaled >>> 11;
60                  B_signed = B_scaled >>> 11;
            end
62
            always @(posedge clock) begin
64                  // saturation and assignment
                    if (reset || R_signed < 0) r <= 8'd0;
66                  else if (R_signed > 255) r <= 8'd255;
                    else r <= R_signed[7:0];
68
                    if (reset || G_signed < 0) g <= 8'd0;
70                  else if (G_signed > 255) g <= 8'd255;
                    else g <= G_signed[7:0];
72
                    if (reset || B_signed < 0) b <= 8'd0;
74                  else if (B_signed > 255) b <= 8'd255;
                    else b <= B_signed[7:0];
76          end
    endmodule
78
    // 2 cycle delay
80  module ycrcb_lut(
                    input clock,
82                  input [17:0] ycrcb,
                    output reg [7:0] r,
84                  output reg [7:0] g,
                    output reg [7:0] b
86          );

88          wire [7:0] y;
            wire [4:0] cr;
90          wire [4:0] cb;

92          assign y = ycrcb[17:10];
            assign cr = ycrcb[9:5];
94          assign cb = ycrcb[4:0];

96          reg signed [9:0] rgb_y;
            reg signed [9:0] r_cr;
98          reg signed [9:0] g_cr;
            reg signed [9:0] g_cb;
100         reg signed [9:0] b_cb;

102         reg signed [10:0] r_big;
            reg signed [10:0] g_big;
104         reg signed [10:0] b_big;
```

```
106         always @(posedge clock) begin
                r_big <= rgb_y + r_cr;
108             g_big <= rgb_y - g_cr - g_cb;
                b_big <= rgb_y + b_cb;
110
                if (r_big < 0) r <= 8'd0;
112             else if (r_big > 255) r <= 8'd255;
                else r <= r_big[7:0];
114
                if (g_big < 0) g <= 8'd0;
116             else if (g_big > 255) g <= 8'd255;
                else g <= g_big[7:0];
118
                if (b_big < 0) b <= 8'd0;
120             else if (b_big > 255) b <= 8'd255;
                else b <= b_big[7:0];
122         end

124         always @(*) begin
                case (y)
126                     8'd0: rgb_y = 10'sd0;
                        8'd1: rgb_y = 10'sd0;
128                     8'd2: rgb_y = 10'sd0;
                        8'd3: rgb_y = 10'sd0;
130                     8'd4: rgb_y = 10'sd0;
                        8'd5: rgb_y = 10'sd0;
132                     8'd6: rgb_y = 10'sd0;
                        8'd7: rgb_y = 10'sd0;
134                     8'd8: rgb_y = 10'sd0;
                        8'd9: rgb_y = 10'sd0;
136                     8'd10: rgb_y = 10'sd0;
                        8'd11: rgb_y = 10'sd0;
138                     8'd12: rgb_y = 10'sd0;
                        8'd13: rgb_y = 10'sd0;
140                     8'd14: rgb_y = 10'sd0;
                        8'd15: rgb_y = 10'sd0;
142                     8'd16: rgb_y = 10'sd0;
                        8'd17: rgb_y = 10'sd1;
144                     8'd18: rgb_y = 10'sd2;
                        8'd19: rgb_y = 10'sd3;
146                     8'd20: rgb_y = 10'sd5;
                        8'd21: rgb_y = 10'sd6;
148                     8'd22: rgb_y = 10'sd7;
                        8'd23: rgb_y = 10'sd8;
150                     8'd24: rgb_y = 10'sd9;
                        8'd25: rgb_y = 10'sd10;
152                     8'd26: rgb_y = 10'sd12;
                        8'd27: rgb_y = 10'sd13;
154                     8'd28: rgb_y = 10'sd14;
                        8'd29: rgb_y = 10'sd15;
156                     8'd30: rgb_y = 10'sd16;
```

```
                8'd31: rgb_y = 10'sd17;
158             8'd32: rgb_y = 10'sd19;
                8'd33: rgb_y = 10'sd20;
160             8'd34: rgb_y = 10'sd21;
                8'd35: rgb_y = 10'sd22;
162             8'd36: rgb_y = 10'sd23;
                8'd37: rgb_y = 10'sd24;
164             8'd38: rgb_y = 10'sd26;
                8'd39: rgb_y = 10'sd27;
166             8'd40: rgb_y = 10'sd28;
                8'd41: rgb_y = 10'sd29;
168             8'd42: rgb_y = 10'sd30;
                8'd43: rgb_y = 10'sd31;
170             8'd44: rgb_y = 10'sd33;
                8'd45: rgb_y = 10'sd34;
172             8'd46: rgb_y = 10'sd35;
                8'd47: rgb_y = 10'sd36;
174             8'd48: rgb_y = 10'sd37;
                8'd49: rgb_y = 10'sd38;
176             8'd50: rgb_y = 10'sd40;
                8'd51: rgb_y = 10'sd41;
178             8'd52: rgb_y = 10'sd42;
                8'd53: rgb_y = 10'sd43;
180             8'd54: rgb_y = 10'sd44;
                8'd55: rgb_y = 10'sd45;
182             8'd56: rgb_y = 10'sd47;
                8'd57: rgb_y = 10'sd48;
184             8'd58: rgb_y = 10'sd49;
                8'd59: rgb_y = 10'sd50;
186             8'd60: rgb_y = 10'sd51;
                8'd61: rgb_y = 10'sd52;
188             8'd62: rgb_y = 10'sd54;
                8'd63: rgb_y = 10'sd55;
190             8'd64: rgb_y = 10'sd56;
                8'd65: rgb_y = 10'sd57;
192             8'd66: rgb_y = 10'sd58;
                8'd67: rgb_y = 10'sd59;
194             8'd68: rgb_y = 10'sd61;
                8'd69: rgb_y = 10'sd62;
196             8'd70: rgb_y = 10'sd63;
                8'd71: rgb_y = 10'sd64;
198             8'd72: rgb_y = 10'sd65;
                8'd73: rgb_y = 10'sd66;
200             8'd74: rgb_y = 10'sd68;
                8'd75: rgb_y = 10'sd69;
202             8'd76: rgb_y = 10'sd70;
                8'd77: rgb_y = 10'sd71;
204             8'd78: rgb_y = 10'sd72;
                8'd79: rgb_y = 10'sd73;
206             8'd80: rgb_y = 10'sd74;
                8'd81: rgb_y = 10'sd76;
208             8'd82: rgb_y = 10'sd77;
```

```
                8'd83: rgb_y = 10'sd78;
210             8'd84: rgb_y = 10'sd79;
                8'd85: rgb_y = 10'sd80;
212             8'd86: rgb_y = 10'sd81;
                8'd87: rgb_y = 10'sd83;
214             8'd88: rgb_y = 10'sd84;
                8'd89: rgb_y = 10'sd85;
216             8'd90: rgb_y = 10'sd86;
                8'd91: rgb_y = 10'sd87;
218             8'd92: rgb_y = 10'sd88;
                8'd93: rgb_y = 10'sd90;
220             8'd94: rgb_y = 10'sd91;
                8'd95: rgb_y = 10'sd92;
222             8'd96: rgb_y = 10'sd93;
                8'd97: rgb_y = 10'sd94;
224             8'd98: rgb_y = 10'sd95;
                8'd99: rgb_y = 10'sd97;
226             8'd100: rgb_y = 10'sd98;
                8'd101: rgb_y = 10'sd99;
228             8'd102: rgb_y = 10'sd100;
                8'd103: rgb_y = 10'sd101;
230             8'd104: rgb_y = 10'sd102;
                8'd105: rgb_y = 10'sd104;
232             8'd106: rgb_y = 10'sd105;
                8'd107: rgb_y = 10'sd106;
234             8'd108: rgb_y = 10'sd107;
                8'd109: rgb_y = 10'sd108;
236             8'd110: rgb_y = 10'sd109;
                8'd111: rgb_y = 10'sd111;
238             8'd112: rgb_y = 10'sd112;
                8'd113: rgb_y = 10'sd113;
240             8'd114: rgb_y = 10'sd114;
                8'd115: rgb_y = 10'sd115;
242             8'd116: rgb_y = 10'sd116;
                8'd117: rgb_y = 10'sd118;
244             8'd118: rgb_y = 10'sd119;
                8'd119: rgb_y = 10'sd120;
246             8'd120: rgb_y = 10'sd121;
                8'd121: rgb_y = 10'sd122;
248             8'd122: rgb_y = 10'sd123;
                8'd123: rgb_y = 10'sd125;
250             8'd124: rgb_y = 10'sd126;
                8'd125: rgb_y = 10'sd127;
252             8'd126: rgb_y = 10'sd128;
                8'd127: rgb_y = 10'sd129;
254             8'd128: rgb_y = 10'sd130;
                8'd129: rgb_y = 10'sd132;
256             8'd130: rgb_y = 10'sd133;
                8'd131: rgb_y = 10'sd134;
258             8'd132: rgb_y = 10'sd135;
                8'd133: rgb_y = 10'sd136;
260             8'd134: rgb_y = 10'sd137;
```

```
                8'd135: rgb_y = 10'sd139;
                8'd136: rgb_y = 10'sd140;
                8'd137: rgb_y = 10'sd141;
                8'd138: rgb_y = 10'sd142;
                8'd139: rgb_y = 10'sd143;
                8'd140: rgb_y = 10'sd144;
                8'd141: rgb_y = 10'sd146;
                8'd142: rgb_y = 10'sd147;
                8'd143: rgb_y = 10'sd148;
                8'd144: rgb_y = 10'sd149;
                8'd145: rgb_y = 10'sd150;
                8'd146: rgb_y = 10'sd151;
                8'd147: rgb_y = 10'sd152;
                8'd148: rgb_y = 10'sd154;
                8'd149: rgb_y = 10'sd155;
                8'd150: rgb_y = 10'sd156;
                8'd151: rgb_y = 10'sd157;
                8'd152: rgb_y = 10'sd158;
                8'd153: rgb_y = 10'sd159;
                8'd154: rgb_y = 10'sd161;
                8'd155: rgb_y = 10'sd162;
                8'd156: rgb_y = 10'sd163;
                8'd157: rgb_y = 10'sd164;
                8'd158: rgb_y = 10'sd165;
                8'd159: rgb_y = 10'sd166;
                8'd160: rgb_y = 10'sd168;
                8'd161: rgb_y = 10'sd169;
                8'd162: rgb_y = 10'sd170;
                8'd163: rgb_y = 10'sd171;
                8'd164: rgb_y = 10'sd172;
                8'd165: rgb_y = 10'sd173;
                8'd166: rgb_y = 10'sd175;
                8'd167: rgb_y = 10'sd176;
                8'd168: rgb_y = 10'sd177;
                8'd169: rgb_y = 10'sd178;
                8'd170: rgb_y = 10'sd179;
                8'd171: rgb_y = 10'sd180;
                8'd172: rgb_y = 10'sd182;
                8'd173: rgb_y = 10'sd183;
                8'd174: rgb_y = 10'sd184;
                8'd175: rgb_y = 10'sd185;
                8'd176: rgb_y = 10'sd186;
                8'd177: rgb_y = 10'sd187;
                8'd178: rgb_y = 10'sd189;
                8'd179: rgb_y = 10'sd190;
                8'd180: rgb_y = 10'sd191;
                8'd181: rgb_y = 10'sd192;
                8'd182: rgb_y = 10'sd193;
                8'd183: rgb_y = 10'sd194;
                8'd184: rgb_y = 10'sd196;
                8'd185: rgb_y = 10'sd197;
                8'd186: rgb_y = 10'sd198;
```

```
                8'd187: rgb_y = 10'sd199;
314             8'd188: rgb_y = 10'sd200;
                8'd189: rgb_y = 10'sd201;
316             8'd190: rgb_y = 10'sd203;
                8'd191: rgb_y = 10'sd204;
318             8'd192: rgb_y = 10'sd205;
                8'd193: rgb_y = 10'sd206;
320             8'd194: rgb_y = 10'sd207;
                8'd195: rgb_y = 10'sd208;
322             8'd196: rgb_y = 10'sd210;
                8'd197: rgb_y = 10'sd211;
324             8'd198: rgb_y = 10'sd212;
                8'd199: rgb_y = 10'sd213;
326             8'd200: rgb_y = 10'sd214;
                8'd201: rgb_y = 10'sd215;
328             8'd202: rgb_y = 10'sd217;
                8'd203: rgb_y = 10'sd218;
330             8'd204: rgb_y = 10'sd219;
                8'd205: rgb_y = 10'sd220;
332             8'd206: rgb_y = 10'sd221;
                8'd207: rgb_y = 10'sd222;
334             8'd208: rgb_y = 10'sd223;
                8'd209: rgb_y = 10'sd225;
336             8'd210: rgb_y = 10'sd226;
                8'd211: rgb_y = 10'sd227;
338             8'd212: rgb_y = 10'sd228;
                8'd213: rgb_y = 10'sd229;
340             8'd214: rgb_y = 10'sd230;
                8'd215: rgb_y = 10'sd232;
342             8'd216: rgb_y = 10'sd233;
                8'd217: rgb_y = 10'sd234;
344             8'd218: rgb_y = 10'sd235;
                8'd219: rgb_y = 10'sd236;
346             8'd220: rgb_y = 10'sd237;
                8'd221: rgb_y = 10'sd239;
348             8'd222: rgb_y = 10'sd240;
                8'd223: rgb_y = 10'sd241;
350             8'd224: rgb_y = 10'sd242;
                8'd225: rgb_y = 10'sd243;
352             8'd226: rgb_y = 10'sd244;
                8'd227: rgb_y = 10'sd246;
354             8'd228: rgb_y = 10'sd247;
                8'd229: rgb_y = 10'sd248;
356             8'd230: rgb_y = 10'sd249;
                8'd231: rgb_y = 10'sd250;
358             8'd232: rgb_y = 10'sd251;
                8'd233: rgb_y = 10'sd253;
360             8'd234: rgb_y = 10'sd254;
                8'd235: rgb_y = 10'sd255;
362             8'd236: rgb_y = 10'sd255;
                8'd237: rgb_y = 10'sd255;
364             8'd238: rgb_y = 10'sd255;
```

```
                8'd239: rgb_y = 10'sd255;
366             8'd240: rgb_y = 10'sd255;
                8'd241: rgb_y = 10'sd255;
368             8'd242: rgb_y = 10'sd255;
                8'd243: rgb_y = 10'sd255;
370             8'd244: rgb_y = 10'sd255;
                8'd245: rgb_y = 10'sd255;
372             8'd246: rgb_y = 10'sd255;
                8'd247: rgb_y = 10'sd255;
374             8'd248: rgb_y = 10'sd255;
                8'd249: rgb_y = 10'sd255;
376             8'd250: rgb_y = 10'sd255;
                8'd251: rgb_y = 10'sd255;
378             8'd252: rgb_y = 10'sd255;
                8'd253: rgb_y = 10'sd255;
380             8'd254: rgb_y = 10'sd255;
                8'd255: rgb_y = 10'sd255;
382             default: rgb_y = 10'sd0;
            endcase
384
            case (cr)
386             5'd0: r_cr = -10'sd179;
                5'd1: r_cr = -10'sd179;
388             5'd2: r_cr = -10'sd179;
                5'd3: r_cr = -10'sd166;
390             5'd4: r_cr = -10'sd153;
                5'd5: r_cr = -10'sd140;
392             5'd6: r_cr = -10'sd128;
                5'd7: r_cr = -10'sd115;
394             5'd8: r_cr = -10'sd102;
                5'd9: r_cr = -10'sd89;
396             5'd10: r_cr = -10'sd77;
                5'd11: r_cr = -10'sd64;
398             5'd12: r_cr = -10'sd51;
                5'd13: r_cr = -10'sd38;
400             5'd14: r_cr = -10'sd26;
                5'd15: r_cr = -10'sd13;
402             5'd16: r_cr = 10'sd0;
                5'd17: r_cr = 10'sd13;
404             5'd18: r_cr = 10'sd26;
                5'd19: r_cr = 10'sd38;
406             5'd20: r_cr = 10'sd51;
                5'd21: r_cr = 10'sd64;
408             5'd22: r_cr = 10'sd77;
                5'd23: r_cr = 10'sd89;
410             5'd24: r_cr = 10'sd102;
                5'd25: r_cr = 10'sd115;
412             5'd26: r_cr = 10'sd128;
                5'd27: r_cr = 10'sd140;
414             5'd28: r_cr = 10'sd153;
                5'd29: r_cr = 10'sd166;
416             5'd30: r_cr = 10'sd171;
```

```
                5'd31: r_cr = 10'sd171;
                default: r_cr = 10'sd0;
        endcase

        case (cr)
                5'd0: g_cr = -10'sd91;
                5'd1: g_cr = -10'sd91;
                5'd2: g_cr = -10'sd91;
                5'd3: g_cr = -10'sd85;
                5'd4: g_cr = -10'sd78;
                5'd5: g_cr = -10'sd72;
                5'd6: g_cr = -10'sd65;
                5'd7: g_cr = -10'sd59;
                5'd8: g_cr = -10'sd52;
                5'd9: g_cr = -10'sd46;
                5'd10: g_cr = -10'sd39;
                5'd11: g_cr = -10'sd33;
                5'd12: g_cr = -10'sd26;
                5'd13: g_cr = -10'sd20;
                5'd14: g_cr = -10'sd13;
                5'd15: g_cr = -10'sd7;
                5'd16: g_cr = 10'sd0;
                5'd17: g_cr = 10'sd7;
                5'd18: g_cr = 10'sd13;
                5'd19: g_cr = 10'sd20;
                5'd20: g_cr = 10'sd26;
                5'd21: g_cr = 10'sd33;
                5'd22: g_cr = 10'sd39;
                5'd23: g_cr = 10'sd46;
                5'd24: g_cr = 10'sd52;
                5'd25: g_cr = 10'sd59;
                5'd26: g_cr = 10'sd65;
                5'd27: g_cr = 10'sd72;
                5'd28: g_cr = 10'sd78;
                5'd29: g_cr = 10'sd85;
                5'd30: g_cr = 10'sd87;
                5'd31: g_cr = 10'sd87;
                default: g_cr = 10'sd0;
        endcase

        case (cb)
                5'd0: g_cb = -10'sd44;
                5'd1: g_cb = -10'sd44;
                5'd2: g_cb = -10'sd44;
                5'd3: g_cb = -10'sd41;
                5'd4: g_cb = -10'sd38;
                5'd5: g_cb = -10'sd34;
                5'd6: g_cb = -10'sd31;
                5'd7: g_cb = -10'sd28;
                5'd8: g_cb = -10'sd25;
                5'd9: g_cb = -10'sd22;
                5'd10: g_cb = -10'sd19;
```

```
                5'd11: g_cb = -10'sd16;
470             5'd12: g_cb = -10'sd13;
                5'd13: g_cb = -10'sd9;
472             5'd14: g_cb = -10'sd6;
                5'd15: g_cb = -10'sd3;
474             5'd16: g_cb = 10'sd0;
                5'd17: g_cb = 10'sd3;
476             5'd18: g_cb = 10'sd6;
                5'd19: g_cb = 10'sd9;
478             5'd20: g_cb = 10'sd13;
                5'd21: g_cb = 10'sd16;
480             5'd22: g_cb = 10'sd19;
                5'd23: g_cb = 10'sd22;
482             5'd24: g_cb = 10'sd25;
                5'd25: g_cb = 10'sd28;
484             5'd26: g_cb = 10'sd31;
                5'd27: g_cb = 10'sd34;
486             5'd28: g_cb = 10'sd38;
                5'd29: g_cb = 10'sd41;
488             5'd30: g_cb = 10'sd42;
                5'd31: g_cb = 10'sd42;
490             default: g_cb = 10'sd0;
        endcase
492
        case (cb)
494             5'd0: b_cb = -10'sd226;
                5'd1: b_cb = -10'sd226;
496             5'd2: b_cb = -10'sd226;
                5'd3: b_cb = -10'sd210;
498             5'd4: b_cb = -10'sd194;
                5'd5: b_cb = -10'sd177;
500             5'd6: b_cb = -10'sd161;
                5'd7: b_cb = -10'sd145;
502             5'd8: b_cb = -10'sd129;
                5'd9: b_cb = -10'sd113;
504             5'd10: b_cb = -10'sd97;
                5'd11: b_cb = -10'sd81;
506             5'd12: b_cb = -10'sd65;
                5'd13: b_cb = -10'sd48;
508             5'd14: b_cb = -10'sd32;
                5'd15: b_cb = -10'sd16;
510             5'd16: b_cb = 10'sd0;
                5'd17: b_cb = 10'sd16;
512             5'd18: b_cb = 10'sd32;
                5'd19: b_cb = 10'sd48;
514             5'd20: b_cb = 10'sd65;
                5'd21: b_cb = 10'sd81;
516             5'd22: b_cb = 10'sd97;
                5'd23: b_cb = 10'sd113;
518             5'd24: b_cb = 10'sd129;
                5'd25: b_cb = 10'sd145;
520             5'd26: b_cb = 10'sd161;
```

```
                        5'd27: b_cb = 10'sd177;
                        5'd28: b_cb = 10'sd194;
                        5'd29: b_cb = 10'sd210;
                        5'd30: b_cb = 10'sd216;
                        5'd31: b_cb = 10'sd216;
                        default: b_cb = 10'sd0;
                endcase
        end
endmodule
```

A.11 parameter_set.v

```
// parameter_set
// used for setting the parameters used by NTSC
// in order to detect "interesting" pixels
module parameter_set(
        input clock, reset,
        input [4:0] switch,
        output [63:0] hex_output,

        output reg [9:0] GREEN_LUM_MAX,
        output reg [9:0] GREEN_LUM_MIN,
        output reg [9:0] GREEN_CR_MAX,
        output reg [9:0] GREEN_CR_MIN,
        output reg [9:0] GREEN_CB_MAX,
        output reg [9:0] GREEN_CB_MIN,

        output reg [9:0] ORANGE_LUM_MAX,
        output reg [9:0] ORANGE_LUM_MIN,
        output reg [9:0] ORANGE_CR_MAX,
        output reg [9:0] ORANGE_CR_MIN,
        output reg [9:0] ORANGE_CB_MAX,
        output reg [9:0] ORANGE_CB_MIN,

        output reg [9:0] PINK_LUM_MAX,
        output reg [9:0] PINK_LUM_MIN,
        output reg [9:0] PINK_CR_MAX,
        output reg [9:0] PINK_CR_MIN,
        output reg [9:0] PINK_CB_MAX,
        output reg [9:0] PINK_CB_MIN,

        output reg [9:0] BLUE_LUM_MAX,
        output reg [9:0] BLUE_LUM_MIN,
        output reg [9:0] BLUE_CR_MAX,
        output reg [9:0] BLUE_CR_MIN,
        output reg [9:0] BLUE_CB_MAX,
        output reg [9:0] BLUE_CB_MIN
);

        wire SELECT_LUM_MAX;
        wire SELECT_LUM_MIN;
```

```
        wire SELECT_CR_MAX;
41      wire SELECT_CR_MIN;
        wire SELECT_CB_MAX;
43      wire SELECT_CB_MIN;

45      wire [2:0] SELECT;
        reg [9:0] selected_parameter;
47      wire [1:0] selected_color;

49      reg [9:0] LUM_MAX;
        reg [9:0] LUM_MIN;
51      reg [9:0] CR_MAX;
        reg [9:0] CR_MIN;
53      reg [9:0] CB_MAX;
        reg [9:0] CB_MIN;
55
        wire b_clock;
57      counter #(.PERIOD(2000000)) bclk(.clock(clock), .reset(reset),
                .enable(b_clock));
59
        debounce dbc1(.clock(clock), .reset(reset), .noisy(switch[4]), .clean(SELECT[2]));
61      debounce dbc2(.clock(clock), .reset(reset), .noisy(switch[3]), .clean(SELECT[1]));
        debounce dbc3(.clock(clock), .reset(reset), .noisy(switch[2]), .clean(SELECT[0]));
63      debounce dbc4(.clock(clock), .reset(reset), .noisy(switch[1]), .clean(selected_color
            [1]));
        debounce dbc5(.clock(clock), .reset(reset), .noisy(switch[0]), .clean(selected_color
            [0]));
65
        assign hex_output = {2'd0, selected_color, 4'd0, 5'd0, SELECT, 6'd0,
            selected_parameter, 32'h0},
67
        always @(*) begin
69              case (selected_color)
                2'd0: begin
71                      LUM_MAX = GREEN_LUM_MAX;
                        LUM_MIN = GREEN_LUM_MIN;
73                      CR_MAX = GREEN_CR_MAX;
                        CR_MIN = GREEN_CR_MIN;
75                      CB_MAX = GREEN_CB_MAX;
                        CB_MIN = GREEN_CB_MIN;
77              end
                2'd1: begin
79                      LUM_MAX = ORANGE_LUM_MAX;
                        LUM_MIN = ORANGE_LUM_MIN;
81                      CR_MAX = ORANGE_CR_MAX;
                        CR_MIN = ORANGE_CR_MIN;
83                      CB_MAX = ORANGE_CB_MAX;
                        CB_MIN = ORANGE_CB_MIN;
85              end
                2'd2: begin
87                      LUM_MAX = PINK_LUM_MAX;
                        LUM_MIN = PINK_LUM_MIN;
```

```
89                      CR_MAX = PINK_CR_MAX;
                        CR_MIN = PINK_CR_MIN;
91                      CB_MAX = PINK_CB_MAX;
                        CB_MIN = PINK_CB_MIN;
93              end
                2'd3: begin
95                      LUM_MAX = BLUE_LUM_MAX;
                        LUM_MIN = BLUE_LUM_MIN;
97                      CR_MAX = BLUE_CR_MAX;
                        CR_MIN = BLUE_CR_MIN;
99                      CB_MAX = BLUE_CB_MAX;
                        CB_MIN = BLUE_CB_MIN;
101             end
                endcase
103
                case (SELECT)
105             3'd7: selected_parameter = LUM_MAX;
                3'd6: selected_parameter = LUM_MIN;
107             3'd5: selected_parameter = CR_MAX;
                3'd4: selected_parameter = CR_MIN;
109             3'd3: selected_parameter = CB_MAX;
                3'd2: selected_parameter = CB_MIN;
111             default: selected_parameter = 10'd0;
                endcase
113     end

115     always @(posedge b_clock) begin
                if (reset) begin
117                     GREEN_LUM_MAX <= `GREEN_LUM_MAX;
                        GREEN_LUM_MIN <= `GREEN_LUM_MIN;
119                     GREEN_CR_MAX <= `GREEN_CR_MAX;
                        GREEN_CR_MIN <= `GREEN_CR_MIN;
121                     GREEN_CB_MAX <= `GREEN_CB_MAX;
                        GREEN_CB_MIN <= `GREEN_CB_MIN;
123
                        ORANGE_LUM_MAX <= `ORANGE_LUM_MAX;
125                     ORANGE_LUM_MIN <= `ORANGE_LUM_MIN;
                        ORANGE_CR_MAX <= `ORANGE_CR_MAX;
127                     ORANGE_CR_MIN <= `ORANGE_CR_MIN;
                        ORANGE_CB_MAX <= `ORANGE_CB_MAX;
129                     ORANGE_CB_MIN <= `ORANGE_CB_MIN;

131                     PINK_LUM_MAX <= `PINK_LUM_MAX;
                        PINK_LUM_MIN <= `PINK_LUM_MIN;
133                     PINK_CR_MAX <= `PINK_CR_MAX;
                        PINK_CR_MIN <= `PINK_CR_MIN;
135                     PINK_CB_MAX <= `PINK_CB_MAX;
                        PINK_CB_MIN <= `PINK_CB_MIN;
137
                        BLUE_LUM_MAX <= `BLUE_LUM_MAX;
139                     BLUE_LUM_MIN <= `BLUE_LUM_MIN;
                        BLUE_CR_MAX <= `BLUE_CR_MAX;
```

```
141                 BLUE_CR_MIN <= `BLUE_CR_MIN;
                    BLUE_CB_MAX <= `BLUE_CB_MAX;
143                 BLUE_CB_MIN <= `BLUE_CB_MIN;
                end
145             else if(~button_up) begin
                    case (selected_color)
147                 2'd0:
                            case (SELECT)
149                         3'd7: GREEN_LUM_MAX <= GREEN_LUM_MAX+4'b1000;
                            3'd6: GREEN_LUM_MIN <= GREEN_LUM_MIN+4'b1000;
151                         3'd5: GREEN_CR_MAX <= GREEN_CR_MAX+4'b1000;
                            3'd4: GREEN_CR_MIN <= GREEN_CR_MIN+4'b1000;
153                         3'd3: GREEN_CB_MAX <= GREEN_CB_MAX+4'b1000;
                            3'd2: GREEN_CB_MIN <= GREEN_CB_MIN+4'b1000;
155                         endcase
                    2'd1:
157                         case (SELECT)
                            3'd7: ORANGE_LUM_MAX <= ORANGE_LUM_MAX+4'b1000;
159                         3'd6: ORANGE_LUM_MIN <= ORANGE_LUM_MIN+4'b1000;
                            3'd5: ORANGE_CR_MAX <= ORANGE_CR_MAX+4'b1000;
161                         3'd4: ORANGE_CR_MIN <= ORANGE_CR_MIN+4'b1000;
                            3'd3: ORANGE_CB_MAX <= ORANGE_CB_MAX+4'b1000;
163                         3'd2: ORANGE_CB_MIN <= ORANGE_CB_MIN+4'b1000;
                            endcase
165                 2'd2:
                            case (SELECT)
167                         3'd7: PINK_LUM_MAX <= PINK_LUM_MAX+4'b1000;
                            3'd6: PINK_LUM_MIN <= PINK_LUM_MIN+4'b1000;
169                         3'd5: PINK_CR_MAX <= PINK_CR_MAX+4'b1000;
                            3'd4: PINK_CR_MIN <= PINK_CR_MIN+4'b1000;
171                         3'd3: PINK_CB_MAX <= PINK_CB_MAX+4'b1000;
                            3'd2: PINK_CB_MIN <= PINK_CB_MIN+4'b1000;
173                         endcase
                    2'd3:
175                         case (SELECT)
                            3'd7: BLUE_LUM_MAX <= BLUE_LUM_MAX+4'b1000;
177                         3'd6: BLUE_LUM_MIN <= BLUE_LUM_MIN+4'b1000;
                            3'd5: BLUE_CR_MAX <= BLUE_CR_MAX+4'b1000;
179                         3'd4: BLUE_CR_MIN <= BLUE_CR_MIN+4'b1000;
                            3'd3: BLUE_CB_MAX <= BLUE_CB_MAX+4'b1000;
181                         3'd2: BLUE_CB_MIN <= BLUE_CB_MIN+4'b1000;
                            endcase
183                 endcase
                end
185             else if(~button_down) begin
                    case (selected_color)
187                 2'd0:
                            case (SELECT)
189                         3'd7: GREEN_LUM_MAX <= GREEN_LUM_MAX-4'b1000;
                            3'd6: GREEN_LUM_MIN <= GREEN_LUM_MIN-4'b1000;
191                         3'd5: GREEN_CR_MAX <= GREEN_CR_MAX-4'b1000;
                            3'd4: GREEN_CR_MIN <= GREEN_CR_MIN-4'b1000;
```

```
193                                 3'd3: GREEN_CB_MAX <= GREEN_CB_MAX-4'b1000;
                                    3'd2: GREEN_CB_MIN <= GREEN_CB_MIN-4'b1000;
195                                 endcase
                            2'd1:
197                                 case (SELECT)
                                    3'd7: ORANGE_LUM_MAX <= ORANGE_LUM_MAX-4'b1000;
199                                 3'd6: ORANGE_LUM_MIN <= ORANGE_LUM_MIN-4'b1000;
                                    3'd5: ORANGE_CR_MAX <= ORANGE_CR_MAX-4'b1000;
201                                 3'd4: ORANGE_CR_MIN <= ORANGE_CR_MIN-4'b1000;
                                    3'd3: ORANGE_CB_MAX <= ORANGE_CB_MAX-4'b1000;
203                                 3'd2: ORANGE_CB_MIN <= ORANGE_CB_MIN-4'b1000;
                                    endcase
205                         2'd2:
                                    case (SELECT)
207                                 3'd7: PINK_LUM_MAX <= PINK_LUM_MAX-4'b1000;
                                    3'd6: PINK_LUM_MIN <= PINK_LUM_MIN-4'b1000;
209                                 3'd5: PINK_CR_MAX <= PINK_CR_MAX-4'b1000;
                                    3'd4: PINK_CR_MIN <= PINK_CR_MIN-4'b1000;
211                                 3'd3: PINK_CB_MAX <= PINK_CB_MAX-4'b1000;
                                    3'd2: PINK_CB_MIN <= PINK_CB_MIN-4'b1000;
213                                 endcase
                            2'd3:
215                                 case (SELECT)
                                    3'd7: BLUE_LUM_MAX <= BLUE_LUM_MAX-4'b1000;
217                                 3'd6: BLUE_LUM_MIN <= BLUE_LUM_MIN-4'b1000;
                                    3'd5: BLUE_CR_MAX <= BLUE_CR_MAX-4'b1000;
219                                 3'd4: BLUE_CR_MIN <= BLUE_CR_MIN-4'b1000;
                                    3'd3: BLUE_CB_MAX <= BLUE_CB_MAX-4'b1000;
221                                 3'd2: BLUE_CB_MIN <= BLUE_CB_MIN-4'b1000;
                                    endcase
223                         endcase
                    end
225         end
    endmodule
```

A.12 params.v

```
    // clock frequencies
2   `define FPGA_CLOCK 26'd60000000
    `define  VGA_CLOCK 26'd25175000
4   `define NTSC_CLOCK 25'd12587500

6   // bitwidth of clock frequencies
    `define LOG_FPGA_CLOCK 26
8   `define LOG_VGA_CLOCK  26
    `define LOG_NTSC_CLOCK 25
10
    // image sizes
12  `define TOTAL_PIXELS     19'd307200
    `define IMAGE_WIDTH      10'd640
14  `define IMAGE_WIDTH_D2    9'd320
```

```
`define IMAGE_HEIGHT      9'd480

// bitwidth of image sizes
`define LOG_TOTAL_PIXELS 19
`define LOG_WIDTH        10
`define LOG_WIDTH_D2      9
`define LOG_HEIGHT                9

// memory sizes
`define IMAGE_LENGTH 153600
`define MEM_ADDR     524288

// memory bitwidths
`define LOG_IMAGE_ADDR  19
`define LOG_MEM                  36
`define LOG_ADDR                 19

// pixel bitwidths
`define LOG_TRUNC        18
`define LOG_FULL         24

// VGA (640x480@60Hz)
`define VGA_HBLANKON     10'd639
`define VGA_HSYNCON      10'd655
`define VGA_HSYNCOFF     10'd751
`define VGA_HRESET       10'd799
`define VGA_VBLANKON     10'd479
`define VGA_VSYNCON      10'd490
`define VGA_VSYNCOFF     10'd492
`define VGA_VRESET       10'd523

// VGA  bitwidths
`define LOG_HCOUNT       10
`define LOG_VCOUNT       10

// hue recognition parameters
`define GREEN_LUM_MAX 10'h338
`define GREEN_LUM_MIN 10'h228
`define GREEN_CR_MAX 10'h250
`define GREEN_CR_MIN 10'h0B8
`define GREEN_CB_MAX 10'h1A8
`define GREEN_CB_MIN 10'h008

`define ORANGE_LUM_MAX 10'h3FF
`define ORANGE_LUM_MIN 10'h200
`define ORANGE_CR_MAX 10'h3FF
`define ORANGE_CR_MIN 10'h250
`define ORANGE_CB_MAX 10'h190
`define ORANGE_CB_MIN 10'h000

`define PINK_LUM_MAX 10'h3FF
`define PINK_LUM_MIN 10'h190
```

```
`define PINK_CR_MAX 10'h3FF
`define PINK_CR_MIN 10'h290
`define PINK_CB_MAX 10'h3FF
`define PINK_CB_MIN 10'h1E0

`define BLUE_LUM_MAX 10'h3EF
`define BLUE_LUM_MIN 10'h1D0
`define BLUE_CR_MAX 10'h1E8
`define BLUE_CR_MIN 10'h000
`define BLUE_CB_MAX 10'h3EF
`define BLUE_CB_MIN 10'h1E0
```

A.13 zbt_test_pattern.v

```
`include "params.v"

module address_calculator(
        input [8:0] x,
        input [9:0] y,
        input [1:0] loc,
        output [18:0] addr);

        wire [18:0] of1, of2;

        loc_lut ll(.loc(loc), .addr_off(of1));
        y_lut yl(.y(y), .addr_off(of2));

        assign addr = of1 + of2 + x;
endmodule

module zbt_test_pattern(
                        input clock,
                        input reset,
                        input start,
                        output reg [`LOG_ADDR-1:0] mem0_addr,
                        output reg [`LOG_ADDR-1:0] mem1_addr,
                        output reg [`LOG_MEM-1:0] mem0_write,
                        output reg [`LOG_MEM-1:0] mem1_write,
                        output reg mem0_wr,
                        output reg mem1_wr);

   reg [1:0]                        state = 0;
   reg [8:0]                        y;
   reg [9:0]                        x;
        wire [18:0] addr;
        reg [1:0] loc;

        address_calculator test_ac(
                .x(x), .y(y),
                .loc(loc), .addr(addr));

```

```
    always @(posedge clock) begin
39      case (state)
          0: begin
41           mem0_wr <= 0;
             mem1_wr <= 0;
43
             if (start) begin
45             state <= 1;
               x <= 0;
47             y <= 0;
                        loc <= 0;
49           end
           end
51
           1: begin
53           mem0_addr <= addr;
             mem1_addr <= addr;
55           mem0_wr <= 1;
             mem1_wr <= 1;
57           if ((x[4] & ~y[4]) || (~x[4] & y[4])) begin
               mem0_write <= 36'b111111111100011000111111111100011000;
59             mem1_write <= 36'b111111111000010000111111111000010000;
             end else begin
61             mem0_write <= 36'b000000001000010000000000001000010000;
               mem1_write <= 36'b000000001000010000000000001000010000;
63           end

65           if (y == 479 && x == 639) begin
               state <= 2;
67                       loc <= 1;
               mem0_wr <= 0;
69             mem1_wr <= 0;
               x <= 0;
71             y <= 0;
             end else if (x == 639) begin
73             x <= 0;
               y <= y + 1;
75           end else x <= x + 1;
           end // case: 1
77
           2: begin
79           mem0_addr <= addr;
             mem1_addr <= addr;
81           mem0_wr <= 1;
             mem1_wr <= 1;
83           if ((x[5] & ~y[5]) || (~x[5] & y[5])) begin
               mem0_write <= 36'b111111111100011000111111111100011000;
85             mem1_write <= 36'b111111111000010000111111111000010000;
             end else begin
87             mem0_write <= 36'b000000001000010000000000001000010000;
               mem1_write <= 36'b000000001000010000000000001000010000;
89           end
```

```
91              if (y == 479 && x == 639) begin
                    state <= 0;
93                  mem0_wr <= 0;
                    mem1_wr <= 0;
95                  x <= 0;
                    y <= 0;
97              end else if (x == 639) begin
                    x <= 0;
99                  y <= y + 1;
                end else x <= x + 1;
101           end // case: 2

103           default: state <= 0;
          endcase // case state
105     end // always @ (posedge clock)
endmodule // zbt_test_pattern
```