Interactive 3D Processing Framework

Adam Gleitman
Andrew Shum
Tim Balbakov

6.111 Fall 2011

1 Introduction

The 21st century has seen an abundance of advancements in the field of 3D human-computer interaction (HCI), whereby users are able to interact with 3D models either through virtual space or physical space. The advent of technologies such as the Nintendo 3DS^{TM} and Oblong g-speak^{TM} is a testament to current developments in stereoscopy and 3D motion capture / gesture-based interaction. Such developments have led to the use of interactive simulation software to train soldiers in hand-to-hand military combat or to train surgeons in performing complicated operations.

3D HCI technology can be dichotomized by the way virtual and physical spaces are used in the interaction. When the physical space is used for data input, users send commands to the machine using an input device for detecting the 3D position of the user action. When it is used for data output, the simulated 3D rendering is projected onto the physical space through an output device.

In this project, we develop an FPGA-based interactive 3D processing framework to explore both of these modes of interaction. In stereoscopy mode, our system renders a stereoscopic anaglyph image from 2D images captured using two NTSC cameras that are positioned a few inches apart to mimic a pair of human eyes. This 3D rendering is then viewable through an output device, namely red and cyan 3D glasses. In gestural interaction mode, we use the same camera setup as a motion capture input device. Our system captures simple gestures and interprets them to control the viewing position of a virtual rendering of a polyhedron.

2 Functionality

2.1 Overview

The 3D processing framework works in two modes: stereoscopy (or anaglyph) mode and gestural interaction mode.

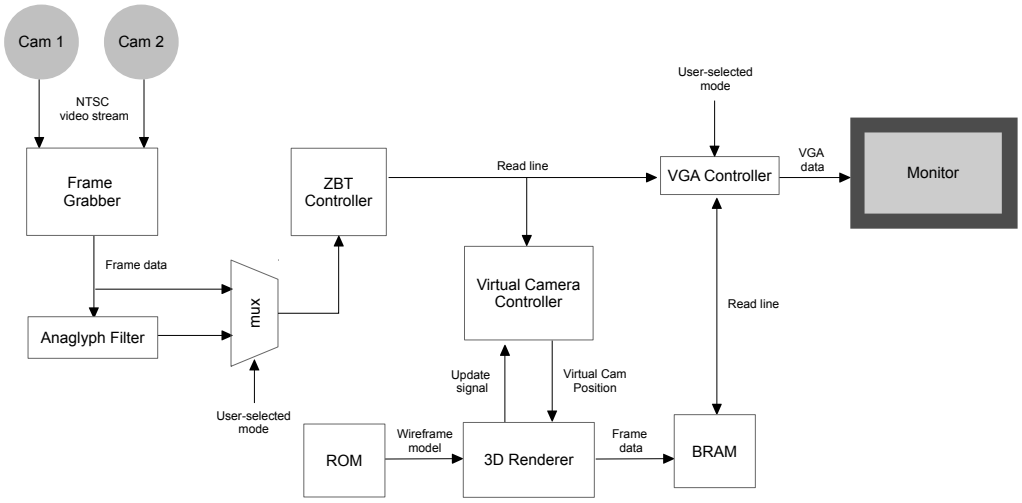

Figure 1: A high-level block diagram of the processing pipeline.

In the first mode, a stereoscopic image is generated that can be viewed with 3D glasses. The frame grabber outputs deinterlaced video frames to an anaglyph filter. This filter uses information from the left and right frames and generates an anaglyph image. This image is saved to a frame buffer by the ZBT controller. The VGA controller renders the resulting anaglyph to the monitor.

In the second mode, a specially marked object is used as a gestural input controller for a virtual camera viewing a 3D model. The frame grabber outputs a deinterlaced video frame directly to the ZBT controller, which is saved to a frame buffer. The virtual camera controller module reads data from the frame buffer to detect the position and orientation of a specially marked object. It determines the location of calibration dots on an object, and uses that information to calculate a virtual camera position. This virtual camera position determines the perspective from which the 3D renderer will render a pre-configured wire-frame stored in ROM. The resulting image is saved to a BRAM frame buffer, from which the VGA controller renders an image on the screen.

2.2 Modules

2.2.1 Frame Grabber – Tim

The frame grabber interfaces with the ADV7185 video capture chip and simultaneously records two video feeds to memory.

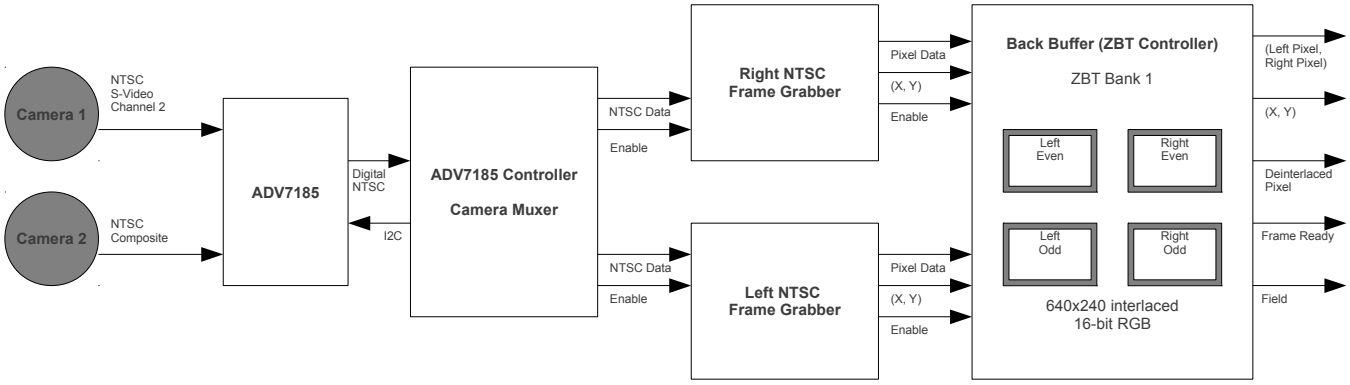

Figure 2: Block diagram for the Frame Grabber module.

NTSC Decoder Controller (ntsc_decoder.v) The NTSC decoder module generates control signals for the onboard video decoder chip. This module is based on the staff provided ADV7185 controller code, but is modified to capture two video feeds simultaneously. This is possible by processing image data from one camera at a time, and switching between cameras at the expense of frame rate. There are three analog video input lines that are physically accessible on the Labkit: composite, S-Video chrominance, and S-Video luminance. The composite video signal of the second camera is connected to the chrominance input of the S-Video connector. The `INPUT_SEL` register on the ADV7185 decoder chip selects which video stream is selected. The appropriate values for labkit video inputs AIN2 and AIN5 are available on the datasheet. A single bit source input to this module selects the input channel. The register is periodically updated by the FPGA over the i2c bus at a rate much faster than that of the camera frame period.

Camera Multiplexer (framegrabber.v) The controller captures two streams at a reduced frame rate by alternating the incoming video stream. The NTSC standard specifies a 30 frame per second (FPS) frame rate for the interlaced signal that is generated by the cameras. Since the video decoder is shared by two video feeds, it is not possible to capture video

at the full frame rate. The subsequent image processing modules do calculations on a whole frame, so the input video stream must be deinterlaced. De-interlacing (which is performed by address manipulation) halves the frame rate, and switching the input feed discards an input frame. The resulting frame rate is 6 FPS at the original 640x480 resolution.

Internally, this is accomplished by reading the FVH (field, vertical sync, horizontal sync) bus from the staff-provided NTSC decoder. On every second field change, the source input to the ADV7185 controller is inverted. Write enable is held low immediately after a source switch for one field (even or odd) to give the video decoder chip time to lock onto the signal (which takes two vertical sync transitions, a time period much longer than field transition).

NTSC to ZBT frame grabber (ntsc2zbt.v) The frame grabber takes interlaced NTSC frame data as input and generates the proper control and data signals for the ZBT. The frame grabber converts from the input color space to an internal 18-bit RGB palette using the staff-supplied YCrCb to RGB converter. The deinterlacing is accomplished by keeping track of the field signal from the NTSC decoder and matching the low order bit of the column count memory address to the even or odd field. Since color video requires at least 16 bits per pixel, only two pixels per memory word could be recorded. This required modifying the address and data signals generated by the staff provided module.

Since a functional memory controller was not synthesized, an anaglyph filter relying on persistence of vision was implemented in this module. Since only one frame of data was available at a time, left frames retained their red channel and right frames retained their green and blue channel. At a high enough switching frequency, persistence of vision would create a composite image. However, due to frame rate limitations the resulting image did not create a 3D illusion.

This module and the preceding NTSC decoder was tested tested in a testjig that outputs the contents of the front buffer to a VGA monitor, with the input source switched automatically by the camera multiplexer.

2.2.2 ZBT Memory Controller (memcontrol.v) – Tim

There are two one-port ZBT memory chips that allow for 36 bits to be written or read every clock cycle. Internally, two 18-bit RGB pixels are stored per line of memory in both buffers. Since NTSC memory access is time-critical, the `ntsc_to_zbt` module has access priority to the back buffer ZBT. The back buffer contains two video frames at any time: one frame for each of the cameras. The front buffer is stored on the second ZBT chip, and access to the chip is shared by the virtual camera controller and the VGA controller. Only one module actively accesses the front buffer at a time. Data is transferred between the memories by

the `flipbuffer` module.

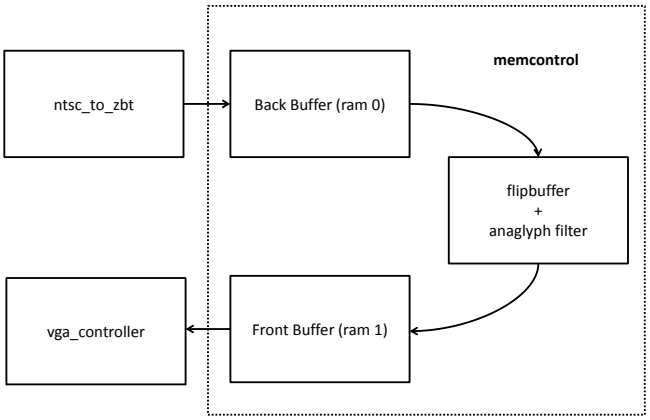

Figure 3: Block diagram for the overall memory architecture.

The `ntsc_to_zbt` module provides a raw memory address and a write enable signal, since it was based on the staff provided code. The VGA controller does not provide a write enable signal, and the memory controller deduces a read request by detecting a change on the address bus. Since the `flipbuffer` module needs to know when both the front and back buffers are free to be written to (it has the lowest access priority), all external read and write requests are delayed by one cycle. When an external read/write request is detected, the `flipbuffer` module holds state the next cycle and allows the memory access to flow through. If one of the buffers is accessed repeatedly without a two cycle window for the `flipbuffer` module to perform a read or write operation, buffer flipping will stall.

The `flipbuffer` module operates by reading data from the back buffer, applying an anaglyph filter if the system is in anaglyph mode, and writing the data to the front buffer. Since two data words are needed for the anaglyph filter (left and right frames), two memory accesses are needed to the back buffer for every word added to the front buffer. The anaglyph conversion has a latency of one clock cycle, and it takes two cycles for the SRAM to respond to a read request. Therefore a minimum of six cycles is required to copy two pixels (left and right) and output the composite two pixels to the front memory. The VGA controller and virtual camera controller never read data that is being updated, since an internal frame completion signal selects a complete frame for reading and writes to a different image frame.

This memory topology was a risky design decision: the `flipbuffer` module is a state machine that needs to hold data and carefully time its memory accesses with the other modules. This simulation ran well in Modelsim under straightforward memory access patterns. However, the memory controller was unable to arbitrate chaotic memory access when im-

plemented in hardware. A logic analyzer connected to the physical control and data lines on the memory controller showed erratic output from the ZBT memory: with a constant write enable signal and address bus, the data bus would fluctuate. This was likely caused by tri-state bus mismanagement or timing violations. In the test jig, data was successfully stored, copied, and output from the front buffer, but the memory integrity was too poor to display a camera feed.

A safer design topology was devised, but not implemented: instead of a module flipping data between two memories, data would be recorded to the memory in parallel. Each camera would record to its own memory, and the VGA controller would read both words during the same clock cycle and apply the anaglyph filter as data was streamed to the VGA display.

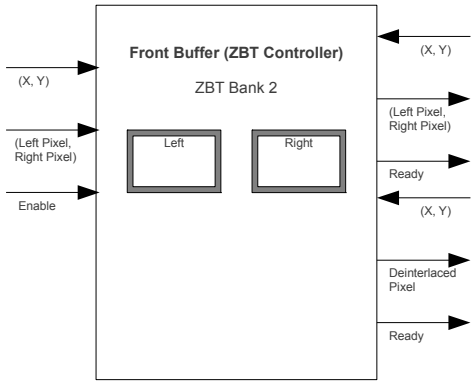

Figure 4: Block diagram for the front buffer ZBT controller.

2.2.3 Virtual Camera Controller – Adam

The virtual camera controller is responsible for controlling the position of a virtual camera viewing a wireframe object using camera data. The user controls the position of the virtual camera by moving a colored dot. This module tracks the location of the dot on the two cameras, calculates the dots location in three-dimensional space, and translates its position into a change in the virtual cameras position.

Fetch Controller The fetch controller is responsible for coordinating the actions of all the modules in the virtual camera controller. When it is not processing data, the fetch controller holds its request line high, telling the frame buffer that it may write data to it. When the data ready line goes high, the fetch controller brings the request line low, resets the center of mass calculators, and starts calling addresses so the center of mass calculators can receive

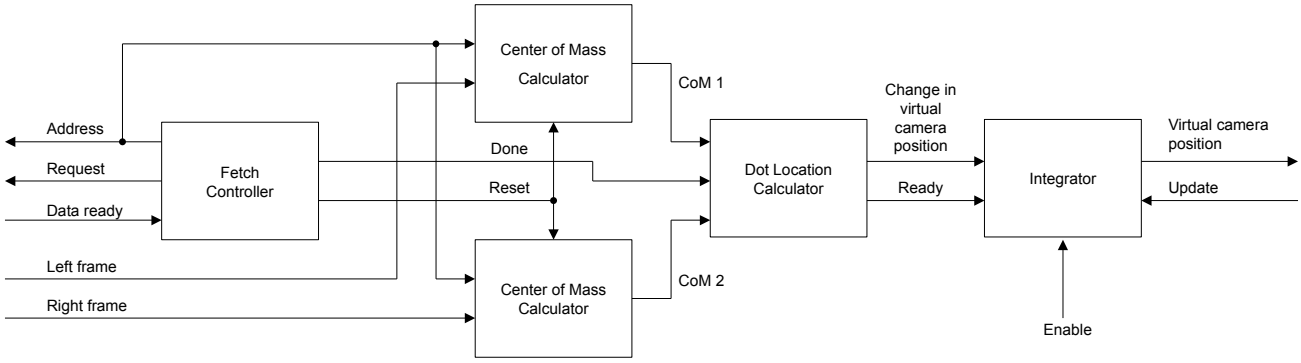

Figure 5: Block diagram for the Virtual Camera Controller module.

frame data. After going through every possible address on each frame, this module pulses the done line and brings its request line high again.

The fetch controller starts by calling address $(0, 0)$ on the left frame. Since the frames are stored in memory with two pixels per memory address, this call yields the pixels at $(0, 0)$ and $(1, 0)$ on the left image. Using a handshaking protocol with the memory controller, the fetch controller knows when the frame data becomes valid, after which the fetch controller continues by calling addresses $(2, 0)$, $(4, 0)$, and so on. After retrieving all pixels in the left frame, the fetch controller starts over by calling addresses to retrieve the pixels in the right frame.

Field Selector The field selector is just a demultiplexer. Since the memory controller does not permit accessing both the left and right frames at the same time, we only feed the center of mass calculator corresponding to whether we are reading the left frame or the right frame. The other frame is fed a pixel value of 0. However, this does not affect the calculations of the center of mass due to how the center of mass calculators are implemented.

Center of Mass Calculator The center of mass calculator modules are responsible for locating the centers of the dots on each of the camera frames. There are two of these modules, one for the left frame and one for the right frame. Each module keeps track of three values: S_x, S_y, and N, which represent the sum of the x-coordinates of the light pixels, the sum of the y-coordinates of the light pixels, and the number of light pixels, respectively. When reset, these values are set to zero and any additional information used to perform the calculations is discarded.

Each module outputs what it calculates to be the center of the dot on the frame it analyzed. The outputs are not necessarily valid until the modules have received all the pixel data from the frame buffers.

Each center of mass calculator is capable of filtering out noise to an extent. Since each center of mass calculator is fed two pixels at a time, we test if both pixel values are above a certain brightness threshold. The brightness here is calculated by adding up the 6-bit red, green, and blue values, making the brightness range from 0 (pure black) to 189 (pure white). If both pixels have a brightness of at least 150, then both pixels are determined to be light pixels, and S_x, S_y, and N are adjusted accordingly. Otherwise, both pixels are determined to be dark pixels. This algorithm was tested using a Python script and has proven to be reliable with moderate amounts of noise.

The center of mass of the light pixels is then approximately at $(\frac{S_x}{N}, \frac{S_y}{N})$. However, this module does not do any direct division; rather, it outputs the values of S_x, S_y, and N directly. This minimizes the number of dividers, which are very expensive in terms of hardware, needed to build this digital system.

Dot Location Calculator The dot location calculator is responsible for taking the centers of the dots on each camera frame and calculating the location of the dot in three-dimensional space. The location of the dot depends not only on the centers of mass given by the center of mass calculators, but several physical properties of the cameras, such as the distance between the two cameras and their viewing angle. We will now construct a mathematical model of this problem, solve it, and discuss how it is implemented in hardware.

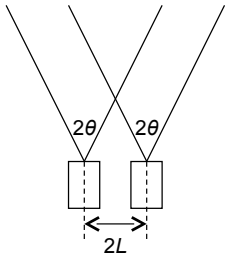

Figure 6: An image of the setup of the two cameras.

Suppose the cameras are separated by a distance $2L$ and that they each have viewing angle 2θ. On each camera frame we are given center of mass $(x_{\mathrm{L}}, y_{\mathrm{L}})$ and $(x_{\mathrm{R}}, y_{\mathrm{R}})$ in image coordinates. From this we wish to find the real position (x, y, d) of the dot. Our system does

not use d, but we will figure it out anyway to help us in calculations. Set up a coordinate system that places the cameras at $(\pm L, 0)$ facing in the direction of the positive d-axis. First we will solve for x and d.

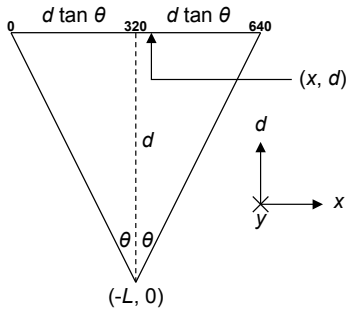

Figure 7: A top-down view of the area viewable by the left camera.

The bold numbers in the above frame correspond to image coordinates. From this we get the following equation:

$$x = -L - d\tan\theta + d\tan\theta \cdot \frac{x_{\mathrm{L}}}{320}$$

We may rewrite the above equation as follows:

$$x = -L + d\tan\theta\left(\frac{x_{\mathrm{L}}}{320} - 1\right)$$

Similarly, we may get the following equation from the right camera:

$$x = L + d\tan\theta\left(\frac{x_{\mathrm{R}}}{320} - 1\right)$$

Solving for x and d, we get the following:

$$d = \frac{640L}{(x_{\mathrm{L}} - x_{\mathrm{R}})\tan\theta}$$

$$x = L\left(\frac{x_{\mathrm{L}} + x_{\mathrm{R}} - 640}{x_{\mathrm{L}} - x_{\mathrm{R}}}\right)$$

To calculate y, consider a side view of the area that the cameras can see:

Since the width of the viewable area is $2d\tan\theta$ it must follow that the height of the viewable area is $\frac{3}{2}d\tan\theta$ because the camera has a 4:3 aspect ratio. This gives us the following equation:

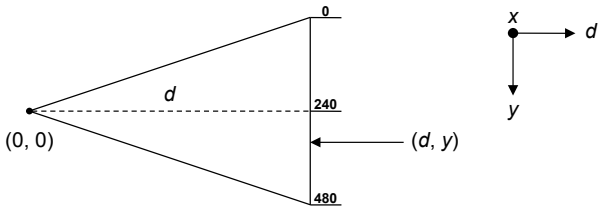

Figure 8: A side view of the area viewable by either camera.

$$y = \frac{3}{4} d \tan\theta \left(\frac{240 - y_{\mathrm{L}}}{240} \right)$$

Theoretically, $y_{\mathrm{L}} = y_{\mathrm{R}}$ if we assume that the cameras are at the same height. However, due to possible measurement errors this may not be the case. Therefore we will modify this formula slightly:

$$y = \frac{3}{4} d \tan\theta \left(\frac{240 - y_{\mathrm{C}}}{240} \right)$$

where $y_{\mathrm{C}} = \frac{1}{2}(y_{\mathrm{L}} + y_{\mathrm{R}})$. Since we already know d we may compute y:

$$y = \frac{2L(240 - y_{\mathrm{C}})}{x_{\mathrm{L}} - x_{\mathrm{R}}} = \frac{L(480 - y_{\mathrm{L}} - y_{\mathrm{R}})}{x_{\mathrm{L}} - x_{\mathrm{R}}}$$

However, we are not given the centers of mass directly; instead we are given S_x^{L}, S_y^{L}, and N^{L} from the left center of mass calculator and S_x^{R}, S_y^{R}, and N^{R} from the right center of mass calculator. Therefore we must make four substitutions to make our system compatible with its inputs: $x_{\mathrm{L}} = S_x^{\mathrm{L}}/N^{\mathrm{L}}$, $x_{\mathrm{R}} = S_x^{\mathrm{R}}/N^{\mathrm{R}}$, $y_{\mathrm{L}} = S_y^{\mathrm{L}}/N^{\mathrm{L}}$, and $y_{\mathrm{R}} = S_y^{\mathrm{R}}/N^{\mathrm{R}}$. After making these substitutions, assuming $L = 1$, and simplifying all complex fractions, we get:

$$x = \frac{640 N^{\mathrm{L}} N^{\mathrm{R}} - S_x^{\mathrm{L}} N^{\mathrm{R}} - N^{\mathrm{L}} S_x^{\mathrm{R}}}{N^{\mathrm{L}} S_x^{\mathrm{R}} - S_x^{\mathrm{L}} N^{\mathrm{R}}}$$

$$y = \frac{-480 N^{\mathrm{L}} N^{\mathrm{R}} + S_y^{\mathrm{L}} N^{\mathrm{R}} + N^{\mathrm{L}} S_y^{\mathrm{R}}}{N^{\mathrm{L}} S_x^{\mathrm{R}} - S_x^{\mathrm{L}} N^{\mathrm{R}}}$$

Therefore our system only requires at most two dividers, as opposed to six dividers had the task of computing x_{L}, x_{R}, y_{L}, and y_{R} been delegated to the center of mass calculators. Instead we perform more multiplications, which is better since multipliers have a much lower latency than dividers.

In order to perform these calculations, however, we require large multipliers and dividers. Note that the maximum possible values for S_x, S_y, and N from any center of mass calculator

are 98,150,400, 73,574,400, and 307,200, respectively. Therefore the S_x and S_y buses must be at least 27 bits wide, and the N buses must be at least 19 bits wide. Therefore our multipliers must be capable of multiplying 27-bit numbers by 19-bit numbers. We used CoreGen to create such a multiplier.

To provide enough of a safety factor in performing our divisions, we wanted to implement a 54-bit by 48-bit divider. Unfortunately, CoreGen was unable to synthesize such a large divider on the Virtex-II chip due to limitations on the maximum sizes of the dividends and divisors. Therefore our divider only divides 32-bit numbers by 32-bit numbers, the largest divider we could synthesize. Sadly, this does not allow our system to be as accurate as we would like, as larger blobs will result in incorrect calculations. (It should be noted, however, that we have done simulations with a 54-bit by 48-bit divider constructed for a more recent chip using a newer version of ISE, and we got accurate results.)

Then we translate the location of the object to a change in camera angle, which is specified in spherical coordinates (ρ, θ, ϕ). In our system ρ is held constant and is determined by the 3D renderer, so this module does not bother with it. Our divider extends the quotient to 12 bits beyond the decimal point, which gives us the objects position with resolution $\frac{L}{4096}$. Suppose our dividers generate the numbers X_{D} and Y_{D} to represent the objects position in space. We will then feed X_{D} and Y_{D} into the integrator. This means that the rate at which the camera changes is proportional to the linear displacement of the dot in the xy-plane.

Integrator The integrator is responsible for keeping track of the virtual cameras position. It takes as an input the rate at which the virtual camera moves and an enable line. When the enable line is low, the integrator does not change the position of the camera, which prevents this module from changing the position of the virtual camera when the input from the dot line calculator may not be valid or when we wish to keep the object still.

This module keeps track of the virtual cameras position in an internal register, with 41 bits for Θ and 41 bits for Φ, each possible value representing $1/2^{41}$ of a complete circle. For example, a value of 0 represents 0°, a value of 2^{40} represents 180°, a value of $3 \cdot 2^{39}$ represents 270°, and so on. It outputs the contents of this register only when the update line is pulsed high. This allows the 3D Renderer to read the virtual camera position without worrying about it changing while drawing a single frame.

This module receives X_{D} and Y_{D} values from the dot location calculator. When enabled, the integrator increments Θ by X_{D} and Φ by Y_{D}. Since the clock speed is 65 MHz (approximately 2^26 Hz), setting $X_{\mathrm{D}} = 32768$, which corresponds to $x = 8L$, results in the virtual camera revolving about the objects equator at a rate of approximately one revolution per second.

2.2.4 Anaglyph Filter (anaglyphFilter.v) – Andrew

Rendering an anaglyphic image involves applying separate color filters to two input images. The left image, which is viewed through a red filter, has its blue and green color components reduced, while the right image, which is viewed through a cyan filter, has its red component reduced. The composite image is then composed by combining the remaining color components into a single color image. When viewed through the red-cyan 3D glasses, the visual cortex of the brain fuses this into perception of a three dimensional image.

Our Anaglyph Filter module acts as a processing unit that takes a left and right pixel for a given (x, y)-coordinate and outputs a single RGB pixel, transformed according to the Optimized Anaglyphs method of [1].

This method works as follows. For a pixel on the left frame with RGB components $v_L = [r_L, g_L, b_L]$ and the corresponding pixel on the right frame with RGB components $v_R = [r_R, g_R, b_R]$, the resulting pixel that is rendered has RGB components

$$v_{anaglyph} = [0.7 \cdot g_L + 0.3 \cdot b_L, \quad g_R, \quad b_R].$$

Since g_L and b_L are each 6 bits wide, we implemented our Anaglyph Filter module in Verilog as two 64-row lookup tables and an adder. As a result, our module is lightweight and requires a single clock cycle per pixel of anaglyph image.

To verify the accuracy of this algorithm, we implemented a Python version using the Python Imaging Library (PIL) and applied it to left/right captures of an inanimate object. Our result is shown in Figure 9.

Figure 9: The results of performing a test of our Anaglyph Filter algorithm.

As mentioned in Section 2.2.1, the output of the Frame Grabber module is a stream of frame pixel data that is eventually passed into the ZBT controller and stored in the front buffer ZBT. In our originally proposed schematic, the user-selected mode dictates (through the mux shown in Figure 1) whether or not the stream of frame pixel data being stored is

anaglyph-filtered. Thus, as Figure 1 shows, the Anaglyph Filter is inserted before the ZBT Controller module.

However, after facing difficulties implementing the ZBT controller as shown in Figure 4, we drafted an alternate controller schematic that follows a different topology. In this schematic, left and right frames are each saved to a ZBT and are read out simultaneously. We could easily integrate this new design into our current implementation of the Anaglyph Filter simply by passing the output of the ZBT controller *through* the Anaglyph Filter and then into the VGA controller.

2.2.5 3D Renderer (threeDimRenderer.v) – Andrew

The 3D Renderer module is responsible for pulling the hard-coded polyhedron, specified by a set of (x, y, z) coordinates stored in memory, as well as the virtual camera position (θ, ϕ) from the Virtual Camera Controller, and performing all the necessary perspective transformation operations to output a 2D orthographic projection of the polyhedron as seen from the virtual camera. The module coordinates with the Virtual Camera Controller module by sending it request signals to specify when the 3D Renderer is done processing data and ready for the new virtual camera position. The 3D Renderer is comprised of the following three modules.

Angle Grabber (angleGrabber.v) The Angle Grabber is the interface between the 3D Renderer and the Virtual Camera Controller. Its responsibility is to request angles from the Virtual Camera Controller when the 3D Renderer is ready to perform transformation calculations on them, and likewise, initiate transformation calculations only when the angles are new. This was implemented as a 3-state state machine. In the initial state `S_WAITING`, the Angle Grabber continually sends request signals that prompt the Virtual Camera Controller to send back 16-bit θ and ϕ angles. Angle Grabber then takes the 8 most significant bits of each angle and checks if they are new using a wire called `isAngleNew`. To avoid glitchy user interaction, we perform transformation calculations only when `hcount`, `vcount` are outside the display frame. Thus, Angle Grabber also keeps track of when `vcount` exceeds 767 (and is under 800) in a wire called `isOutofFrame`.

When both `isAngleNew` and `isOutofFrame` are high, the state machine moves to the `S_READY` state. In this state, the Angle Grabber sends a ready signal to the Perspective Transformer module to indicate that a new angle is ready to be processed. The Angle Grabber then listens for an acknowledgement from the Perspective Transformer before transitioning into the `S_BUSY` state, at which point the ready signal is set low.

Perspective Transformer (perspectiveTransform.v) The responsibility of the Perspective Transformer is to take angles delivered from Angle Grabber and update the (x, y) coordinates of the points in the 2D wire-frame. In its initial state `S_WAITING`, the Perspective Transformer waits for a ready signal from Angle Grabber. Upon receiving the ready signal, the Perspective Transformer indicates acknowledgement by setting its angle request signal low. From here it proceeds into the `S_PROCESSING` state, during which it runs through all hard-coded (x, y, z) points in memory, applies a rotational transform to each, and stores the new (x, y) coordinates in a corresponding Point Sprite.

For a given point (x, y, z) and rotations θ and ϕ about the origin, the projected coordinates of the rotated point (x', y') are given by

$$x' = x \cos \phi - z \sin \phi$$
$$y' = y \cos \theta + z \cos \phi \sin \theta + x \sin \theta \sin \phi$$

These coordinates are then offset to correctly place the point on the screen. Performing this entire calculation requires a 7-stage pipeline with sine and cosine functions implemented as 256-case lookup tables. Upon completing this calculation for all hard-coded points, the Perspective Transformer sets its angle request signal to high again, signaling Angle Grabber to transition back to the `S_WAITING` state.

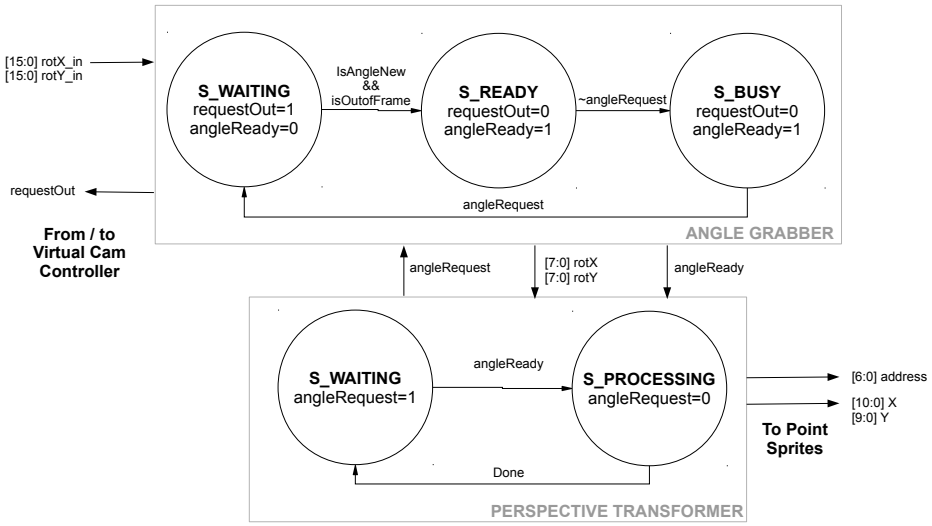

Figure 10: State diagram of Angle Grabber and Perspective Transformer as well as relevant connections.

Point Sprite (pointSprite.v) Each Point Sprite is responsible for generating 4 pixels on the screen at the (x, y) coordinate of the associated polyhedron point. The mechanics behind the pixel generation works similarly to that of the blob sprites in Lab 3, in that pixels turn on and off depending on the value of `hcount` and `vcount`. However, in our implementation, each Point Sprite has its own registers to keep track of its coordinates. The new coordinates are written as the Perspective Transformer is computing them when `hcount` and `vcount` are off-screen.

After experimenting with several hard-coded polyhedra, we realized the FPGA can handle approximately 100 points before the 2D projection starts appearing glitchy. The glitchiness appears because the propagation delay of the pixel generation logic for all the points exceeds one clock cycle. And so at this point, we would need to pipeline the pixel generating mechanism within each Point Sprite. Since we're using Point Sprites to represent both edges and vertices, we end up using 44 Point Sprites to represent a cube in our implementation. Thus, to scale our implementation for more complex polyhedra, it would be better to instead make each sprite a line, each containing a pipelined pixel generating mechanism.

Before synthesizing onto the FPGA labkit, a testjig was created and run on Modelsim to ensure that the control signals generated between Angle Grabber and Perspective Transformer were appropriate in response to incoming angle changes. Figure 11 and Figure 12 show results from running the testjig.

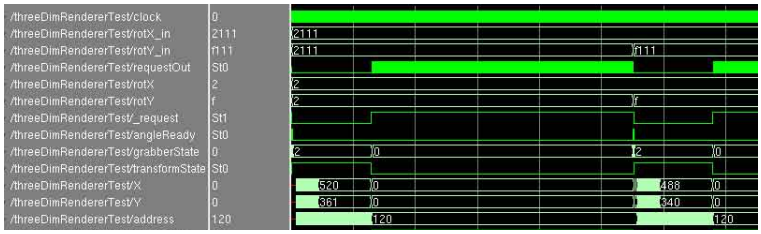

Figure 11: Simulation of 3D Renderer communication signals in response to rotational angle changes.

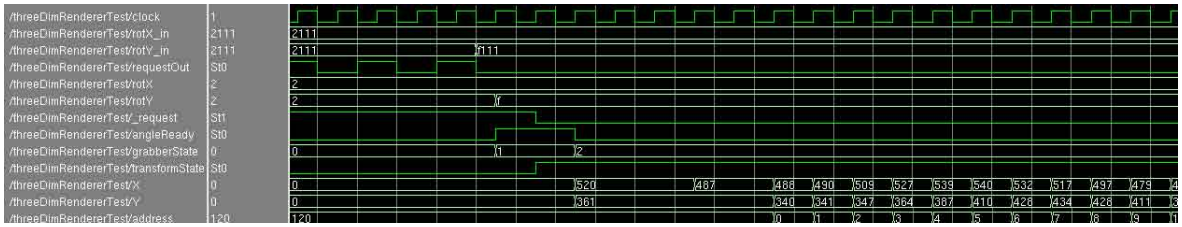

Figure 12: Close-up version of Figure 11 to reveal updates in projection coordinates.

In the test jigs, `_request` corresponds to the angle request coming from Perspective Transform. Also note that `rotX` and `rotY` are 4 bits wide instead of 8, since at the time

of testing sine and cosine were being granulated to 16 angles (22.5 degrees) instead of 256 (1.4 degrees). Figure 12 shows the projection coordinates being updated. To verify the projection, we printed out the coordinates corresponding to `rotX = 4'h2` and `rotY = 4'h2` (both 45 degrees) and plotted them using matplotlib. The result is shown in Figure 13.

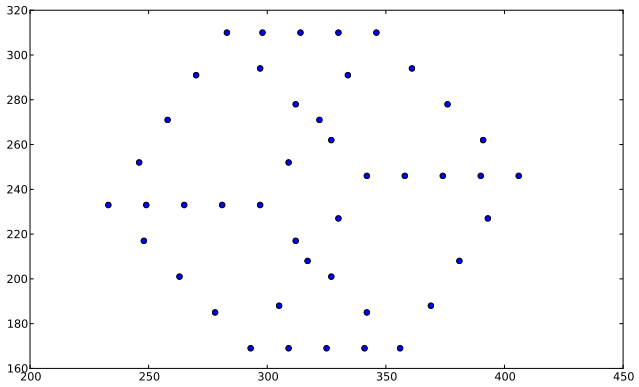

Figure 13: Verification of the updated coordinates shown in Figure 12.

2.2.6 VGA Controller (vgaController.v) – Andrew

The VGA controller module is responsible for providing a constant feed of either the anaglyph-filtered image we are capturing or a wire-frame rendering of a polyhedron as seen from a virtual camera position. Thus, depending on the user-selected mode, the VGA controller sends either anaglyph frames from the front buffer ZBT or displays the pixels associated with each Point Sprite of the hard-coded polyhedron.

VGA Signals (vgaSignal.v) In gestural interaction mode, the VGA controller routes `hcount` and `vcount` signals into the Point Sprites, from which each Point Sprite determines whether or not to display a pixel at a given `hcount`, `vcount` and sends its decision through the wire `pixelOn`. The VGA controller then takes a logical `OR` of every `pixelOn` to determine whether or not to display a white pixel at `hcount`, `vcount`.

ZBT to VGA (zbt2vga.v) In anaglyph mode, the VGA controller interfaces with the ZBT controller to read out frames. We note that ZBT has 2 cycles of read (and write) latency. We handle this by latching the data at an appropriate time using a modified version of the `vram_display` module provided for us.

3 Summary

To test our modules we created test benches for every critical modules and simulated their functionality in ModelSim. When more parts of the system became functional, we synthesized our modules on the labkit and manually tested the system.

The memory controller was tested in ModelSim, and it generated the proper control signals for accessing the SRAM under varying operating conditions. The module was tested with sequential writes from the frame grabber, with no requests to the module to test the functionality of the buffer flipper, and with simulated read requests from the VGA renderer. The same test bench was synthesized in hardware and a logic analyzer was used to verify both ZBT control lines. However, the back buffer ZBT data line did not behave as expected – a static address would receive varying data from the memory. This was likely caused by timing violations or mismanagement of the tri-state bus and prevented the proper functionality of the memory controller.

The 3D Renderer was tested in ModelSim and appropriate control signals were observed for communication with the Virtual Camera Controller as well as communication between Angle Grabber and Perspective Transformer. The outputted coordinates of the Perspective Transformer were plotted using matplotlib to verify the implementation of the perspective transformation. From here, we proceeded to synthesize the 3D Renderer to output 2D projections of a cube and cylinder rotated at angles specified by switches on the FPGA labkit. Once the projections were verified, we went on to test the interfacing between 3D Renderer and Virtual Camera Controller.

To do this, we created fake camera data, which was a square visible on each camera frame, similar to the blob module in Lab 3. We also attached two NES controllers to the labkit to allow us to manipulate the blobs to see how they affect the rotational velocity of a spinning cube. Our test, which can be found in main.v below, consists of the spinning cube overlayed with the fake camera frames turned sideways. This test turned out to be successful.

However, we could not get our system to communicate with the NES controllers, even though our NES controller module was tested beforehand. When we tried other input methods, such as the buttons on the labkit, the camera location did not properly update.

References

[1] Anaglyph Methods Comparison [Online]. Available from: `http://3dtv.at/Knowhow/AnaglyphComparison_en.aspx`. Accessed 2011 Nov 1.

VERILOG CODE

field_selector.v - Adam

```
module field_selector (input [35:0] data, input lr, output [35:0] left,
                       output [35:0] right);
      assign left = lr ? 36'd0 : data;
      assign right = lr ? data : 36'd0;
endmodule
```

fetch_controller.v - Adam

```
module fetch_controller(input clk, input global_reset, input all_data_ready,
                        input address_ready, output [9:0] x_address,
                        output [8:0] y_address, output request,
                        output right_frame, output done, output reset);
      reg [9:0] r_x_address = 0;
      reg [8:0] r_y_address = 0;
      reg r_request = 1;
      reg r_right_frame = 0;
      reg r_done = 1;
      reg r_reset = 0;

      reg waiting = 1;

      always @(posedge clk) begin
      if (global_reset) begin
            r_x_address <= 0;
            r_y_address <= 0;
            r_request <= 1;
            r_right_frame <= 0;
            r_done <= 1;
            r_reset <= 0;
            waiting <= 1;
      end else if (waiting) begin
            // Waiting for all data to be ready
            if (all_data_ready) begin
            waiting <= 0;
            r_request <= 0;
            r_right_frame <= 0;
            r_done <= 0;
            r_reset <= 1;
            end
      end else begin
            // Reading the data
            if (address_ready) begin
            // Advance to the next address, if we can
            if (r_x_address == 638) begin
                  if (r_y_address == 479) begin
                        if (r_right_frame) begin
                        r_y_address <= 0;
                        r_request <= 1;
                        r_right_frame <= 0;
                        r_done <= 1;
                        waiting <= 1;
                        end else begin
                        r_y_address <= 0;
```

```
                        r_right_frame <= 1;
                        end
                  end else begin
                        r_y_address <= r_y_address + 1;
                  end
                  r_x_address <= 0;
            end else begin
                  r_x_address <= r_x_address + 2;
            end
            r_reset <= 0;
            end
      end
      end

      assign x_address = r_x_address;
      assign y_address = r_y_address;
      assign request = r_request;
      assign right_frame = r_right_frame;
      assign done = r_done;
      assign reset = r_reset;
endmodule
```

center_of_mass_calculator.v - Adam

```
module center_of_mass_calculator(input clk, input global_reset, input reset,
                              input [9:0] x_address, input [8:0] y_address,
                              input [17:0] rgb_left, input [17:0] rgb_right,
                              input [7:0] threshold, output [26:0] x_wt_sum,
                              output[26:0] y_wt_sum, output [18:0] total);
      reg [26:0] r_x_sum = 0;
      reg [26:0] r_y_sum = 0;
      reg [18:0] r_total = 0;

      reg [18:0] r_last_address = 0;

      wire [7:0] gray_left;
      wire [7:0] gray_right;

      assign gray_left = {2'b0, rgb_left[17:12]} + {2'b0, rgb_left[11:6]} +
                        {2'b0, rgb_left[5:0]};
      assign gray_right = {2'b0, rgb_right[17:12]} + {2'b0, rgb_right[11:6]} +
                        {2'b0, rgb_right[5:0]};

      always @(posedge clk) begin
      if (global_reset || reset ||
            ({x_address, y_address} != r_last_address)) begin
            case ({global_reset || reset,
                  (gray_left >= threshold) && (gray_right >= threshold)})
            2'b00: begin // no reset, dark pixel
                  r_x_sum <= r_x_sum;
                  r_y_sum <= r_y_sum;
                  r_total <= r_total;
            end
            2'b01: begin // no reset, light pixel
                  r_x_sum <= r_x_sum + (x_address << 1) + 1;
                  r_y_sum <= r_y_sum + (y_address << 1);
                  r_total <= r_total + 2;
```

```
            end
            2'b10: begin // reset, dark pixel
                  r_x_sum <= 0;
                  r_y_sum <= 0;
                  r_total <= 0;
            end
            2'b11: begin // reset, light pixel
                  r_x_sum <= (x_address << 1) + 1;
                  r_y_sum <= y_address << 1;
                  r_total <= 2;
            end
            default: begin // don't do anything
                  r_x_sum <= r_x_sum;
                  r_y_sum <= r_y_sum;
                  r_total <= r_total;
            end
            endcase
      end
      r_last_address <= {x_address, y_address};
      end
      assign x_wt_sum = r_x_sum;
      assign y_wt_sum = r_y_sum;
      assign total = r_total;
endmodule
```

dot_location_calculator.v - Adam

```
module dot_location_calculator(input clk, input done, input [26:0] SLx,
                               input [26:0] SLy, input [18:0] SL,
                               input [26:0] SRx, input [26:0] SRy,
                               input [18:0] SR, output [31:0] dtheta,
                               output [31:0] dphi, output valid);
      reg [71:0] r_done_history = 72'd0;
      reg [31:0] r_dtheta = 32'd0;
      reg [31:0] r_dphi = 32'd0;
      reg r_valid = 0;

      wire [45:0] SLxSR;
      wire [45:0] SLySR;
      wire [45:0] SLSRx;
      wire [45:0] SLSRy;
      mul_27_by_19 foo1(clk, SLx, SR, SLxSR);
      mul_27_by_19 foo2(clk, SLy, SR, SLySR);
      mul_27_by_19 foo3(clk, SRx, SL, SLSRx);
      mul_27_by_19 foo4(clk, SRy, SL, SLSRy);

      wire [35:0] SLSR_div_4_delay_4;
      assign SLSR_div_4_delay_4[35:0] = SL[18:1] * SR[18:1];
      reg [35:0] SLSR_div_4_delay_3 = 36'd0;
      reg [35:0] SLSR_div_4_delay_2 = 36'd0;
      reg [35:0] SLSR_div_4_delay_1 = 36'd0;
      reg [35:0] SLSR_div_4_delay_0 = 36'd0;
      wire [37:0] SLSR;
      assign SLSR = {SLSR_div_4_delay_0, 2'd0};

      wire signed [46:0] w640SLSR;
      assign w640SLSR = {SLSR, 9'd0} + {2'd0, SLSR, 7'd0};
```

```
//   640 * SLSR = (512 * SLSR) + (   128 * SLSR   )
wire signed [46:0] w480SLSR;
assign w480SLSR = {SLSR, 9'd0} - {4'd0, SLSR, 5'd0};
//   480 * SLSR = (512 * SLSR) - (  32 * SLSR   )

wire signed [46:0] A;
assign A = {1'sd0, SLxSR} + {1'sd0, SLSRx};
wire signed [46:0] B;
assign B = {1'sd0, SLySR} + {1'sd0, SLSRy};

wire signed [47:0] denominator;
assign denominator = SLSRx - SLxSR;

wire signed [55:0] numerator_x;
assign numerator_x = w640SLSR - A;
wire signed [55:0] numerator_y;
assign numerator_y = B - w480SLSR;

wire x_dividend_ready;
wire x_divisor_ready;
wire x_quotient_ready;
wire signed [65:0] x_real;
/*div_54_by_48 xdiv(
.aclk(clk),
.s_axis_divisor_tvalid(r_done_history[5:0] == 6'b011111),
.s_axis_dividend_tvalid(r_done_history[5:0] == 6'b011111),
.s_axis_divisor_tready(x_divisor_ready),
.s_axis_dividend_tready(x_dividend_ready),
.m_axis_dout_tvalid(x_quotient_ready),
.s_axis_divisor_tdata(denominator),
.s_axis_dividend_tdata(numerator_x),
.m_axis_dout_tdata(x_real)
);*/
div_32_by_32 xdiv(
.rfd(x_quotient_ready),
.clk(clk),
.dividend(numerator_x[31:0]),
.quotient(x_real[43:12]),
.divisor(denominator[31:0]),
.fractional(x_real[11:0])
);

wire y_dividend_ready;
wire y_divisor_ready;
wire y_quotient_ready;
wire signed [65:0] y_real;
/*div_54_by_48 ydiv(
.aclk(clk),
.s_axis_divisor_tvalid(r_done_history == 6'b011111),
.s_axis_dividend_tvalid(r_done_history == 6'b011111),
.s_axis_divisor_tready(y_divisor_ready),
.s_axis_dividend_tready(y_dividend_ready),
.m_axis_dout_tvalid(y_quotient_ready),
.s_axis_divisor_tdata(denominator),
.s_axis_dividend_tdata(numerator_y),
.m_axis_dout_tdata(y_real)
```

```
      );*/
      div_32_by_32 ydiv(
      .rfd(y_quotient_ready),
      .clk(clk),
      .dividend(numerator_y[31:0]),
      .quotient(y_real[43:12]),
      .divisor(denominator[31:0]),
      .fractional(y_real[11:0])
      );

      always @(posedge clk) begin
      SLSR_div_4_delay_0 <= SLSR_div_4_delay_1;
      SLSR_div_4_delay_1 <= SLSR_div_4_delay_2;
      SLSR_div_4_delay_2 <= SLSR_div_4_delay_3;
      SLSR_div_4_delay_3 <= SLSR_div_4_delay_4;
      r_done_history <= {r_done_history[70:0], done};
      if (x_quotient_ready && y_quotient_ready) begin
            r_dtheta <= x_real[31:0];
            r_dphi <= y_real[31:0];
            r_valid <= 1;
      end
      end

      assign dtheta = r_dtheta;
      assign dphi = r_dphi;
      assign valid = &r_done_history;
endmodule
```

integrator.v - Adam

```
module integrator(input clk, input global_reset, input signed [31:0] dtheta,
                  input signed [31:0] dphi, input change_ready, input update,
                  output [40:0] theta, output [40:0] phi);
      reg signed [40:0] r_theta = 41'sd0;
      reg signed [40:0] r_phi = 41'sh08000000000;
      reg [40:0] r_theta_out = 41'd0;
      reg [40:0] r_phi_out = 41'h08000000000;
      reg last_update = 0;

      always @(posedge clk) begin
      if (change_ready) begin
            if (global_reset) begin
            r_theta <= 41'sd0;
            r_phi <= 41'sh08000000000;
            r_theta_out <= 41'd0;
            r_phi_out <= 41'h08000000000;
            last_update <= 0;
            end else begin
            r_theta <= r_theta + dtheta;
            r_phi <= r_phi + dphi;
            if (update && !last_update) begin
                  r_theta_out <= r_theta[40:0];
                  r_phi_out <= r_phi[40:0];
            end
            end
            last_update <= update;
      end
```

```
    end
    assign theta = r_theta_out;
    assign phi = r_phi_out;
endmodule
```

virtual_camera_controller.v - Adam

```
module virtual_camera_controller(input clk, input global_reset, input enable,
                            input data_ready, input address_ready,
                            input [35:0] frame_data, input update,
                            input [7:0] threshold, output right_frame,
                            output [9:0] x_address,
                            output [8:0] y_address, output request,
                            output [40:0] theta, output[40:0] phi);
    wire [35:0] left_frame_data;
    wire [35:0] right_frame_data;
    wire fetch_done;
    wire com_reset;
    wire [26:0] SLx;
    wire [26:0] SLy;
    wire [18:0] SL;
    wire [26:0] SRx;
    wire [26:0] SRy;
    wire [18:0] SR;
    wire [31:0] dtheta;
    wire [31:0] dphi;
    wire dot_loc_valid;

    // Field selector
    field_selector fsel(.data(frame_data), .lr(right_frame),
                    .left(left_frame_data), .right(right_frame_data));
    // Fetch controller
    fetch_controller fc(.clk(clk), .global_reset(global_reset),
                    .all_data_ready(data_ready),
                    .address_ready(address_ready), .x_address(x_address),
                    .y_address(y_address), .request(request),
                    .right_frame(right_frame), .done(fetch_done),
                    .reset(com_reset));
    // Left frame center of mass calculator
    center_of_mass_calculator com_l(.clk(clk), .global_reset(global_reset),
                                .reset(com_reset), .x_address(x_address),
                                .y_address(y_address),
                                .rgb_left(left_frame_data[35:18]),
                                .rgb_right(left_frame_data[17:0]),
                                .threshold(threshold), .x_wt_sum(SLx),
                                .y_wt_sum(SLy), .total(SL));
    // Right frame center of mass calculator
    center_of_mass_calculator com_r(.clk(clk), .global_reset(global_reset),
                                .reset(com_reset), .x_address(x_address),
                                .y_address(y_address),
                                .rgb_left(right_frame_data[35:18]),
                                .rgb_right(right_frame_data[17:0]),
                                .threshold(threshold), .x_wt_sum(SRx),
                                .y_wt_sum(SRy), .total(SR));
    // Dot location calculator
    dot_location_calculator dot_loc(.clk(clk), .done(fetch_done), .SLx(SLx),
                                .SLy(SLy), .SL(SL), .SRx(SRx), .SRy(SRy),
```

```
                                    .SR(SR), .dtheta(dtheta), .dphi(dphi),
                                    .valid(dot_loc_valid));
      // Integrator
      integrator intg(.clk(clk), .global_reset(global_reset), .dtheta(dtheta),
                  .dphi(dphi), .change_ready(dot_loc_valid && enable),
                  .update(update), .theta(theta), .phi(phi));
endmodule
```

box.v - Adam

```
module box(input [9:0] xlo, input [9:0] xhi, input [8:0] ylo,
            input [8:0] yhi, input [9:0] x, input [8:0] y, output out);
      assign out = (xlo <= x && x < xhi && ylo <= y && y < yhi);
endmodule
```

display_16h.v - Adam

This code was slightly modified from the staff sample code. It was modified to accept a 65 MHz clock instead of a 27 MHz clock.

```
module display_16hex (reset, clock_65mhz, data,
       disp_blank, disp_clock, disp_rs, disp_ce_b,
       disp_reset_b, disp_data_out);

  input reset, clock_65mhz;   // clock and reset (active high reset)
  input [63:0] data;     // 16 hex nibbles to display

  output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
     disp_reset_b;

  reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

  ////////////////////////////////////////////////////////////////////////
/
  //
  // Display Clock
  //
  // Generate a 500kHz clock for driving the displays.
  //
  ////////////////////////////////////////////////////////////////////////
/

  reg [7:0] count;
  reg [7:0] reset_count;
  reg clock;
  wire dreset;

  always @(posedge clock_65mhz)
      begin
   if (reset)
     begin
      count = 0;
      clock = 0;
     end
   else if (count == 64)
     begin
      clock = ~clock;
      count = 8'd0;
     end
```

```
   else
     count = count+1;
      end

  always @(posedge clock_65mhz)
      if (reset)
      reset_count <= 100;
      else
      reset_count <= (reset_count==0) ? 0 : reset_count-1;

  assign dreset = (reset_count != 0);

  assign disp_clock = ~clock;

  ////////////////////////////////////////////////////////////////////////////
/
  //
  // Display State Machine
  //
  ////////////////////////////////////////////////////////////////////////////
/

  reg [7:0] state;       // FSM state
  reg [9:0] dot_index;         // index to current dot being clocked out
  reg [31:0] control;          // control register
  reg [3:0] char_index;    // index of current character
  reg [39:0] dots;       // dots for a single digit
  reg [3:0] nibble;      // hex nibble of current character

  assign disp_blank = 1'b0; // low <= not blanked

  always @(posedge clock)
      if (dreset)
      begin
     state <= 0;
     dot_index <= 0;
     control <= 32'h7F7F7F7F;
      end
      else
      casex (state)
    8'h00:
      begin
      // Reset displays
      disp_data_out <= 1'b0;
      disp_rs <= 1'b0; // dot register
      disp_ce_b <= 1'b1;
      disp_reset_b <= 1'b0;
      dot_index <= 0;
      state <= state+1;
      end

    8'h01:
      begin
      // End reset
      disp_reset_b <= 1'b1;
      state <= state+1;
      end
```

```
8'h02:
  begin
  // Initialize dot register (set all dots to zero)
  disp_ce_b <= 1'b0;
  disp_data_out <= 1'b0; // dot_index[0];
  if (dot_index == 639)
   state <= state+1;
  else
   dot_index <= dot_index+1;
  end

8'h03:
  begin
  // Latch dot data
  disp_ce_b <= 1'b1;
  dot_index <= 31;         // re-purpose to init ctrl reg
  disp_rs <= 1'b1; // Select the control register
  state <= state+1;
  end

8'h04:
  begin
  // Setup the control register
  disp_ce_b <= 1'b0;
  disp_data_out <= control[31];
  control <= {control[30:0], 1'b0};    // shift left
  if (dot_index == 0)
   state <= state+1;
  else
   dot_index <= dot_index-1;
  end

8'h05:
  begin
  // Latch the control register data / dot data
  disp_ce_b <= 1'b1;
  dot_index <= 39;         // init for single char
  char_index <= 15;        // start with MS char
  state <= state+1;
  disp_rs <= 1'b0;         // Select the dot register
  end

8'h06:
  begin
  // Load the user's dot data into the dot reg, char by char
  disp_ce_b <= 1'b0;
  disp_data_out <= dots[dot_index]; // dot data from msb
  if (dot_index == 0)
        if (char_index == 0)
        state <= 5;              // all done, latch data
   else
   begin
     char_index <= char_index - 1;    // goto next char
     dot_index <= 39;
   end
  else
```

```
        dot_index <= dot_index-1;    // else loop thru all dots
      end

      endcase

  always @ (data or char_index)
      case (char_index)
      4'h0:          nibble <= data[3:0];
      4'h1:          nibble <= data[7:4];
      4'h2:          nibble <= data[11:8];
      4'h3:          nibble <= data[15:12];
      4'h4:          nibble <= data[19:16];
      4'h5:          nibble <= data[23:20];
      4'h6:          nibble <= data[27:24];
      4'h7:          nibble <= data[31:28];
      4'h8:          nibble <= data[35:32];
      4'h9:          nibble <= data[39:36];
      4'hA:          nibble <= data[43:40];
      4'hB:          nibble <= data[47:44];
      4'hC:          nibble <= data[51:48];
      4'hD:          nibble <= data[55:52];
      4'hE:          nibble <= data[59:56];
      4'hF:          nibble <= data[63:60];
      endcase

  always @(nibble)
      case (nibble)
      4'h0: dots <= 40'b00111110_01010001_01001001_01000101_00111110;
      4'h1: dots <= 40'b00000000_01000010_01111111_01000000_00000000;
      4'h2: dots <= 40'b01100010_01010001_01001001_01001001_01000110;
      4'h3: dots <= 40'b00100010_01000001_01001001_01001001_00110110;
      4'h4: dots <= 40'b00011000_00010100_00010010_01111111_00010000;
      4'h5: dots <= 40'b00100111_01000101_01000101_01000101_00111001;
      4'h6: dots <= 40'b00111100_01001010_01001001_01001001_00110000;
      4'h7: dots <= 40'b00000001_01110001_00001001_00000101_00000011;
      4'h8: dots <= 40'b00110110_01001001_01001001_01001001_00110110;
      4'h9: dots <= 40'b00000110_01001001_01001001_00101001_00011110;
      4'hA: dots <= 40'b01111110_00001001_00001001_00001001_01111110;
      4'hB: dots <= 40'b01111111_01001001_01001001_01001001_00110110;
      4'hC: dots <= 40'b00111110_01000001_01000001_01000001_00100010;
      4'hD: dots <= 40'b01111111_01000001_01000001_01000001_00111110;
      4'hE: dots <= 40'b01111111_01001001_01001001_01001001_01000001;
      4'hF: dots <= 40'b01111111_00001001_00001001_00001001_00000001;
      endcase

endmodule
```

anaglyphFilter.v - Andrew

```
module anaglyphFilter(
   input clock,
   input [17:0] leftPixel,
   input [17:0] rightPixel,
   output [17:0] outPixel
   );

       reg [5:0] r1; // 0.7*g_L
```

```
reg [5:0] r2; // 0.3*b_L

assign outPixel = {r1 + r2, rightPixel[11:6], rightPixel[5:0]};

always @(posedge clock)
begin
     // Multiply g_L by 0.7
     case(leftPixel[11:6])
           6'b000000: r1 <= 6'b000000;
           6'b000001: r1 <= 6'b000001;
           6'b000010: r1 <= 6'b000001;
           6'b000011: r1 <= 6'b000010;
           6'b000100: r1 <= 6'b000011;
           6'b000101: r1 <= 6'b000100;
           6'b000110: r1 <= 6'b000100;
           6'b000111: r1 <= 6'b000101;
           6'b001000: r1 <= 6'b000110;
           6'b001001: r1 <= 6'b000110;
           6'b001010: r1 <= 6'b000111;
           6'b001011: r1 <= 6'b001000;
           6'b001100: r1 <= 6'b001000;
           6'b001101: r1 <= 6'b001001;
           6'b001110: r1 <= 6'b001010;
           6'b001111: r1 <= 6'b001011;
           6'b010000: r1 <= 6'b001011;
           6'b010001: r1 <= 6'b001100;
           6'b010010: r1 <= 6'b001101;
           6'b010011: r1 <= 6'b001101;
           6'b010100: r1 <= 6'b001110;
           6'b010101: r1 <= 6'b001111;
           6'b010110: r1 <= 6'b001111;
           6'b010111: r1 <= 6'b010000;
           6'b011000: r1 <= 6'b010001;
           6'b011001: r1 <= 6'b010010;
           6'b011010: r1 <= 6'b010010;
           6'b011011: r1 <= 6'b010011;
           6'b011100: r1 <= 6'b010100;
           6'b011101: r1 <= 6'b010100;
           6'b011110: r1 <= 6'b010101;
           6'b011111: r1 <= 6'b010110;
           6'b100000: r1 <= 6'b010110;
           6'b100001: r1 <= 6'b010111;
           6'b100010: r1 <= 6'b011000;
           6'b100011: r1 <= 6'b011001;
           6'b100100: r1 <= 6'b011001;
           6'b100101: r1 <= 6'b011010;
           6'b100110: r1 <= 6'b011011;
           6'b100111: r1 <= 6'b011011;
           6'b101000: r1 <= 6'b011100;
           6'b101001: r1 <= 6'b011101;
           6'b101010: r1 <= 6'b011101;
           6'b101011: r1 <= 6'b011110;
           6'b101100: r1 <= 6'b011111;
           6'b101101: r1 <= 6'b011111;
           6'b101110: r1 <= 6'b100000;
           6'b101111: r1 <= 6'b100001;
```

```
      6'b110000: r1 <= 6'b100010;
      6'b110001: r1 <= 6'b100010;
      6'b110010: r1 <= 6'b100011;
      6'b110011: r1 <= 6'b100100;
      6'b110100: r1 <= 6'b100100;
      6'b110101: r1 <= 6'b100101;
      6'b110110: r1 <= 6'b100110;
      6'b110111: r1 <= 6'b100111;
      6'b111000: r1 <= 6'b100111;
      6'b111001: r1 <= 6'b101000;
      6'b111010: r1 <= 6'b101001;
      6'b111011: r1 <= 6'b101001;
      6'b111100: r1 <= 6'b101010;
      6'b111101: r1 <= 6'b101011;
      6'b111110: r1 <= 6'b101011;
      6'b111111: r1 <= 6'b101100;
endcase

// Multiply b_L by 0.3
case(leftPixel[5:0])
      6'b000000: r2 <= 6'b000000;
      6'b000001: r2 <= 6'b000000;
      6'b000010: r2 <= 6'b000001;
      6'b000011: r2 <= 6'b000001;
      6'b000100: r2 <= 6'b000001;
      6'b000101: r2 <= 6'b000010;
      6'b000110: r2 <= 6'b000010;
      6'b000111: r2 <= 6'b000010;
      6'b001000: r2 <= 6'b000010;
      6'b001001: r2 <= 6'b000011;
      6'b001010: r2 <= 6'b000011;
      6'b001011: r2 <= 6'b000011;
      6'b001100: r2 <= 6'b000100;
      6'b001101: r2 <= 6'b000100;
      6'b001110: r2 <= 6'b000100;
      6'b001111: r2 <= 6'b000101;
      6'b010000: r2 <= 6'b000101;
      6'b010001: r2 <= 6'b000101;
      6'b010010: r2 <= 6'b000101;
      6'b010011: r2 <= 6'b000110;
      6'b010100: r2 <= 6'b000110;
      6'b010101: r2 <= 6'b000110;
      6'b010110: r2 <= 6'b000111;
      6'b010111: r2 <= 6'b000111;
      6'b011000: r2 <= 6'b000111;
      6'b011001: r2 <= 6'b001000;
      6'b011010: r2 <= 6'b001000;
      6'b011011: r2 <= 6'b001000;
      6'b011100: r2 <= 6'b001000;
      6'b011101: r2 <= 6'b001001;
      6'b011110: r2 <= 6'b001001;
      6'b011111: r2 <= 6'b001001;
      6'b100000: r2 <= 6'b001010;
      6'b100001: r2 <= 6'b001010;
      6'b100010: r2 <= 6'b001010;
      6'b100011: r2 <= 6'b001011;
      6'b100100: r2 <= 6'b001011;
```

```
                    6'b100101: r2 <= 6'b001011;
                    6'b100110: r2 <= 6'b001011;
                    6'b100111: r2 <= 6'b001100;
                    6'b101000: r2 <= 6'b001100;
                    6'b101001: r2 <= 6'b001100;
                    6'b101010: r2 <= 6'b001101;
                    6'b101011: r2 <= 6'b001101;
                    6'b101100: r2 <= 6'b001101;
                    6'b101101: r2 <= 6'b001110;
                    6'b101110: r2 <= 6'b001110;
                    6'b101111: r2 <= 6'b001110;
                    6'b110000: r2 <= 6'b001110;
                    6'b110001: r2 <= 6'b001111;
                    6'b110010: r2 <= 6'b001111;
                    6'b110011: r2 <= 6'b001111;
                    6'b110100: r2 <= 6'b010000;
                    6'b110101: r2 <= 6'b010000;
                    6'b110110: r2 <= 6'b010000;
                    6'b110111: r2 <= 6'b010001;
                    6'b111000: r2 <= 6'b010001;
                    6'b111001: r2 <= 6'b010001;
                    6'b111010: r2 <= 6'b010001;
                    6'b111011: r2 <= 6'b010010;
                    6'b111100: r2 <= 6'b010010;
                    6'b111101: r2 <= 6'b010010;
                    6'b111110: r2 <= 6'b010011;
                    6'b111111: r2 <= 6'b010011;
            endcase
      end
endmodule
```

angleGrabber.v - Andrew

```
module angleGrabberFSM(
   input clock,
   input [10:0] hcount,
   input [9:0] vcount,
   input requestIn, // request line from perspective transformer
   input [15:0] rotX_in, // inclination angle (0 to 360 deg) from vcc
   input [15:0] rotY_in, // azimuthal angle (0 to 360 deg) from vcc
   output reg angleReady, // ready line to perspective transformer
   output requestOut, // request line to vcc
   output reg [7:0] rotX, // quantize by ~ 1.4 deg
   output reg [7:0] rotY, // quantize by ~ 1.4 deg
   output [1:0] State
  );

   initial begin
       angleReady <= 1'b0;
       rotX <= 8'h77;
       rotY <= 8'h77;
   end

   // In S_READY, pulse a request for an angle from vcc on every clock
posedge
   reg _requestOut;
   pulseOut pulser1(.clock(clock), .reset(_requestOut), .out(requestOut));
```

```
   wire isAngleNew;
   assign isAngleNew = (rotX !== rotX_in[15:8] || rotY !== rotY_in[15:8]);

   wire isOutofFrame;
   assign isOutofFrame = (vcount > 767 && vcount < 800);

   // FSM States
   parameter S_WAITING = 2'd0; // Waiting for a new angle from VCC
   parameter S_READY = 2'd1; // Has a new angle, waiting for PT to
acknowledge
   parameter S_BUSY = 2'd2; // PT is busy processing a new angle

   reg [1:0] state = S_WAITING;

   assign State = state;

   always @(posedge clock) begin
       case (state)
             S_WAITING:
                   begin
                         // We begin the new transformations only when
                         // the angle is new and we're out of frame
                         if (isAngleNew && isOutofFrame) begin
                               rotX <= rotX_in[15:8];
                               rotY <= rotY_in[15:8];

                               _requestOut <= 1'b0;
                               angleReady <= 1'b1;
                               state <= S_READY;
                         end
                         else begin
                               _requestOut <= 1'b1; // Request angle from vcc
                               angleReady <= 1'b0;
                         end
                   end
             S_READY:
                   begin
                         if (~requestIn) begin
                               angleReady <= 1'b0;
                               state <= S_BUSY;
                         end
                         else begin
                               _requestOut <= 1'b0;
                               angleReady <= 1'b1;
                         end
                   end
             S_BUSY:
                   begin
                         if (requestIn) begin
                               _requestOut <= 1'b1;
                               state <= S_WAITING;
                         end
                         else begin
                               _requestOut <= 1'b0;
                               angleReady <= 1'b0;
                         end
                   end
```

```
        endcase
    end
endmodule
```

angleGrabber.v - Andrew (pulser module for making angle requests from VCC)

```
module pulseOut(
    input clock,
    input reset,
    output out
    );

    reg pulse;
    assign out = pulse;

    always @(negedge clock) begin
        if (reset) pulse <= ~pulse;
        else pulse <= 0;
    end
endmodule
```

trigLUT.v - Andrew

```
module trigLUT(
      input [7:0] angle,
      output reg [11:0] cos,
      output reg [11:0] sin
      );

    always @(*) begin
        case (angle)
              8'h00:    begin    cos = 12'd32;    sin = 12'd0;    end

              8'h01:    begin    cos = 12'd31;    sin = 12'd0;    end

              8'h02:    begin    cos = 12'd31;    sin = 12'd1;    end

              8'h03:    begin    cos = 12'd31;    sin = 12'd2;    end

              8'h04:    begin    cos = 12'd31;    sin = 12'd3;    end

              8'h05:    begin    cos = 12'd31;    sin = 12'd3;    end

              8'h06:    begin    cos = 12'd31;    sin = 12'd4;    end

              8'h07:    begin    cos = 12'd31;    sin = 12'd5;    end

              8'h08:    begin    cos = 12'd31;    sin = 12'd6;    end

              8'h09:    begin    cos = 12'd31;    sin = 12'd7;    end

              8'h0A:    begin    cos = 12'd31;    sin = 12'd7;    end

              8'h0B:    begin    cos = 12'd30;    sin = 12'd8;    end

              8'h0C:    begin    cos = 12'd30;    sin = 12'd9;    end

              8'h0D:    begin    cos = 12'd30;    sin = 12'd10;    end
```

```
8'h0E:    begin    cos = 12'd30;    sin = 12'd10;    end
8'h0F:    begin    cos = 12'd29;    sin = 12'd11;    end
8'h10:    begin    cos = 12'd29;    sin = 12'd12;    end
8'h11:    begin    cos = 12'd29;    sin = 12'd12;    end
8'h12:    begin    cos = 12'd28;    sin = 12'd13;    end
8'h13:    begin    cos = 12'd28;    sin = 12'd14;    end
8'h14:    begin    cos = 12'd28;    sin = 12'd15;    end
8'h15:    begin    cos = 12'd27;    sin = 12'd15;    end
8'h16:    begin    cos = 12'd27;    sin = 12'd16;    end
8'h17:    begin    cos = 12'd27;    sin = 12'd17;    end
8'h18:    begin    cos = 12'd26;    sin = 12'd17;    end
8'h19:    begin    cos = 12'd26;    sin = 12'd18;    end
8'h1A:    begin    cos = 12'd25;    sin = 12'd19;    end
8'h1B:    begin    cos = 12'd25;    sin = 12'd19;    end
8'h1C:    begin    cos = 12'd24;    sin = 12'd20;    end
8'h1D:    begin    cos = 12'd24;    sin = 12'd20;    end
8'h1E:    begin    cos = 12'd23;    sin = 12'd21;    end
8'h1F:    begin    cos = 12'd23;    sin = 12'd22;    end
8'h20:    begin    cos = 12'd22;    sin = 12'd22;    end
8'h21:    begin    cos = 12'd22;    sin = 12'd23;    end
8'h22:    begin    cos = 12'd21;    sin = 12'd23;    end
8'h23:    begin    cos = 12'd20;    sin = 12'd24;    end
8'h24:    begin    cos = 12'd20;    sin = 12'd24;    end
8'h25:    begin    cos = 12'd19;    sin = 12'd25;    end
8'h26:    begin    cos = 12'd19;    sin = 12'd25;    end
8'h27:    begin    cos = 12'd18;    sin = 12'd26;    end
8'h28:    begin    cos = 12'd17;    sin = 12'd26;    end
8'h29:    begin    cos = 12'd17;    sin = 12'd27;    end
```

```
8'h2A:    begin    cos = 12'd16;    sin = 12'd27;    end
8'h2B:    begin    cos = 12'd15;    sin = 12'd27;    end
8'h2C:    begin    cos = 12'd15;    sin = 12'd28;    end
8'h2D:    begin    cos = 12'd14;    sin = 12'd28;    end
8'h2E:    begin    cos = 12'd13;    sin = 12'd28;    end
8'h2F:    begin    cos = 12'd12;    sin = 12'd29;    end
8'h30:    begin    cos = 12'd12;    sin = 12'd29;    end
8'h31:    begin    cos = 12'd11;    sin = 12'd29;    end
8'h32:    begin    cos = 12'd10;    sin = 12'd30;    end
8'h33:    begin    cos = 12'd10;    sin = 12'd30;    end
8'h34:    begin    cos = 12'd9;    sin = 12'd30;    end
8'h35:    begin    cos = 12'd8;    sin = 12'd30;    end
8'h36:    begin    cos = 12'd7;    sin = 12'd31;    end
8'h37:    begin    cos = 12'd7;    sin = 12'd31;    end
8'h38:    begin    cos = 12'd6;    sin = 12'd31;    end
8'h39:    begin    cos = 12'd5;    sin = 12'd31;    end
8'h3A:    begin    cos = 12'd4;    sin = 12'd31;    end
8'h3B:    begin    cos = 12'd3;    sin = 12'd31;    end
8'h3C:    begin    cos = 12'd3;    sin = 12'd31;    end
8'h3D:    begin    cos = 12'd2;    sin = 12'd31;    end
8'h3E:    begin    cos = 12'd1;    sin = 12'd31;    end
8'h3F:    begin    cos = 12'd0;    sin = 12'd31;    end
8'h40:    begin    cos = 12'd0;    sin = 12'd32;    end
8'h41:    begin    cos = 12'd0;    sin = 12'd31;    end
8'h42:    begin    cos = -12'd1;    sin = 12'd31;    end
8'h43:    begin    cos = -12'd2;    sin = 12'd31;    end
8'h44:    begin    cos = -12'd3;    sin = 12'd31;    end
8'h45:    begin    cos = -12'd3;    sin = 12'd31;    end
8'h46:    begin    cos = -12'd4;    sin = 12'd31;    end
```

```
8'h47:    begin    cos = -12'd5;    sin = 12'd31;    end
8'h48:    begin    cos = -12'd6;    sin = 12'd31;    end
8'h49:    begin    cos = -12'd7;    sin = 12'd31;    end
8'h4A:    begin    cos = -12'd7;    sin = 12'd31;    end
8'h4B:    begin    cos = -12'd8;    sin = 12'd30;    end
8'h4C:    begin    cos = -12'd9;    sin = 12'd30;    end
8'h4D:    begin    cos = -12'd10;    sin = 12'd30;    end
8'h4E:    begin    cos = -12'd10;    sin = 12'd30;    end
8'h4F:    begin    cos = -12'd11;    sin = 12'd29;    end
8'h50:    begin    cos = -12'd12;    sin = 12'd29;    end
8'h51:    begin    cos = -12'd12;    sin = 12'd29;    end
8'h52:    begin    cos = -12'd13;    sin = 12'd28;    end
8'h53:    begin    cos = -12'd14;    sin = 12'd28;    end
8'h54:    begin    cos = -12'd15;    sin = 12'd28;    end
8'h55:    begin    cos = -12'd15;    sin = 12'd27;    end
8'h56:    begin    cos = -12'd16;    sin = 12'd27;    end
8'h57:    begin    cos = -12'd17;    sin = 12'd27;    end
8'h58:    begin    cos = -12'd17;    sin = 12'd26;    end
8'h59:    begin    cos = -12'd18;    sin = 12'd26;    end
8'h5A:    begin    cos = -12'd19;    sin = 12'd25;    end
8'h5B:    begin    cos = -12'd19;    sin = 12'd25;    end
8'h5C:    begin    cos = -12'd20;    sin = 12'd24;    end
8'h5D:    begin    cos = -12'd20;    sin = 12'd24;    end
8'h5E:    begin    cos = -12'd21;    sin = 12'd23;    end
8'h5F:    begin    cos = -12'd22;    sin = 12'd23;    end
8'h60:    begin    cos = -12'd22;    sin = 12'd22;    end
8'h61:    begin    cos = -12'd23;    sin = 12'd22;    end
8'h62:    begin    cos = -12'd23;    sin = 12'd21;    end
```

```
8'h63:    begin    cos = -12'd24;    sin = 12'd20;    end
8'h64:    begin    cos = -12'd24;    sin = 12'd20;    end
8'h65:    begin    cos = -12'd25;    sin = 12'd19;    end
8'h66:    begin    cos = -12'd25;    sin = 12'd19;    end
8'h67:    begin    cos = -12'd26;    sin = 12'd18;    end
8'h68:    begin    cos = -12'd26;    sin = 12'd17;    end
8'h69:    begin    cos = -12'd27;    sin = 12'd17;    end
8'h6A:    begin    cos = -12'd27;    sin = 12'd16;    end
8'h6B:    begin    cos = -12'd27;    sin = 12'd15;    end
8'h6C:    begin    cos = -12'd28;    sin = 12'd15;    end
8'h6D:    begin    cos = -12'd28;    sin = 12'd14;    end
8'h6E:    begin    cos = -12'd28;    sin = 12'd13;    end
8'h6F:    begin    cos = -12'd29;    sin = 12'd12;    end
8'h70:    begin    cos = -12'd29;    sin = 12'd12;    end
8'h71:    begin    cos = -12'd29;    sin = 12'd11;    end
8'h72:    begin    cos = -12'd30;    sin = 12'd10;    end
8'h73:    begin    cos = -12'd30;    sin = 12'd10;    end
8'h74:    begin    cos = -12'd30;    sin = 12'd9;    end
8'h75:    begin    cos = -12'd30;    sin = 12'd8;    end
8'h76:    begin    cos = -12'd31;    sin = 12'd7;    end
8'h77:    begin    cos = -12'd31;    sin = 12'd7;    end
8'h78:    begin    cos = -12'd31;    sin = 12'd6;    end
8'h79:    begin    cos = -12'd31;    sin = 12'd5;    end
8'h7A:    begin    cos = -12'd31;    sin = 12'd4;    end
8'h7B:    begin    cos = -12'd31;    sin = 12'd3;    end
8'h7C:    begin    cos = -12'd31;    sin = 12'd3;    end
8'h7D:    begin    cos = -12'd31;    sin = 12'd2;    end
8'h7E:    begin    cos = -12'd31;    sin = 12'd1;    end
8'h7F:    begin    cos = -12'd31;    sin = 12'd0;    end
```

```
8'h80:    begin    cos = -12'd32;    sin = 12'd0;    end
8'h81:    begin    cos = -12'd31;    sin = 12'd0;    end
8'h82:    begin    cos = -12'd31;    sin = -12'd1;    end
8'h83:    begin    cos = -12'd31;    sin = -12'd2;    end
8'h84:    begin    cos = -12'd31;    sin = -12'd3;    end
8'h85:    begin    cos = -12'd31;    sin = -12'd3;    end
8'h86:    begin    cos = -12'd31;    sin = -12'd4;    end
8'h87:    begin    cos = -12'd31;    sin = -12'd5;    end
8'h88:    begin    cos = -12'd31;    sin = -12'd6;    end
8'h89:    begin    cos = -12'd31;    sin = -12'd7;    end
8'h8A:    begin    cos = -12'd31;    sin = -12'd7;    end
8'h8B:    begin    cos = -12'd30;    sin = -12'd8;    end
8'h8C:    begin    cos = -12'd30;    sin = -12'd9;    end
8'h8D:    begin    cos = -12'd30;    sin = -12'd10;    end
8'h8E:    begin    cos = -12'd30;    sin = -12'd10;    end
8'h8F:    begin    cos = -12'd29;    sin = -12'd11;    end
8'h90:    begin    cos = -12'd29;    sin = -12'd12;    end
8'h91:    begin    cos = -12'd29;    sin = -12'd12;    end
8'h92:    begin    cos = -12'd28;    sin = -12'd13;    end
8'h93:    begin    cos = -12'd28;    sin = -12'd14;    end
8'h94:    begin    cos = -12'd28;    sin = -12'd15;    end
8'h95:    begin    cos = -12'd27;    sin = -12'd15;    end
8'h96:    begin    cos = -12'd27;    sin = -12'd16;    end
8'h97:    begin    cos = -12'd27;    sin = -12'd17;    end
8'h98:    begin    cos = -12'd26;    sin = -12'd17;    end
8'h99:    begin    cos = -12'd26;    sin = -12'd18;    end
8'h9A:    begin    cos = -12'd25;    sin = -12'd19;    end
8'h9B:    begin    cos = -12'd25;    sin = -12'd19;    end
```

```
8'h9C:    begin    cos = -12'd24;    sin = -12'd20;    end
8'h9D:    begin    cos = -12'd24;    sin = -12'd20;    end
8'h9E:    begin    cos = -12'd23;    sin = -12'd21;    end
8'h9F:    begin    cos = -12'd23;    sin = -12'd22;    end
8'hA0:    begin    cos = -12'd22;    sin = -12'd22;    end
8'hA1:    begin    cos = -12'd22;    sin = -12'd23;    end
8'hA2:    begin    cos = -12'd21;    sin = -12'd23;    end
8'hA3:    begin    cos = -12'd20;    sin = -12'd24;    end
8'hA4:    begin    cos = -12'd20;    sin = -12'd24;    end
8'hA5:    begin    cos = -12'd19;    sin = -12'd25;    end
8'hA6:    begin    cos = -12'd19;    sin = -12'd25;    end
8'hA7:    begin    cos = -12'd18;    sin = -12'd26;    end
8'hA8:    begin    cos = -12'd17;    sin = -12'd26;    end
8'hA9:    begin    cos = -12'd17;    sin = -12'd27;    end
8'hAA:    begin    cos = -12'd16;    sin = -12'd27;    end
8'hAB:    begin    cos = -12'd15;    sin = -12'd27;    end
8'hAC:    begin    cos = -12'd15;    sin = -12'd28;    end
8'hAD:    begin    cos = -12'd14;    sin = -12'd28;    end
8'hAE:    begin    cos = -12'd13;    sin = -12'd28;    end
8'hAF:    begin    cos = -12'd12;    sin = -12'd29;    end
8'hB0:    begin    cos = -12'd12;    sin = -12'd29;    end
8'hB1:    begin    cos = -12'd11;    sin = -12'd29;    end
8'hB2:    begin    cos = -12'd10;    sin = -12'd30;    end
8'hB3:    begin    cos = -12'd10;    sin = -12'd30;    end
8'hB4:    begin    cos = -12'd9;    sin = -12'd30;    end
8'hB5:    begin    cos = -12'd8;    sin = -12'd30;    end
8'hB6:    begin    cos = -12'd7;    sin = -12'd31;    end
8'hB7:    begin    cos = -12'd7;    sin = -12'd31;    end
8'hB8:    begin    cos = -12'd6;    sin = -12'd31;    end
```

```
8'hB9:    begin    cos = -12'd5;    sin = -12'd31;    end
8'hBA:    begin    cos = -12'd4;    sin = -12'd31;    end
8'hBB:    begin    cos = -12'd3;    sin = -12'd31;    end
8'hBC:    begin    cos = -12'd3;    sin = -12'd31;    end
8'hBD:    begin    cos = -12'd2;    sin = -12'd31;    end
8'hBE:    begin    cos = -12'd1;    sin = -12'd31;    end
8'hBF:    begin    cos = 12'd0;    sin = -12'd31;    end
8'hC0:    begin    cos = 12'd0;    sin = -12'd32;    end
8'hC1:    begin    cos = 12'd0;    sin = -12'd31;    end
8'hC2:    begin    cos = 12'd1;    sin = -12'd31;    end
8'hC3:    begin    cos = 12'd2;    sin = -12'd31;    end
8'hC4:    begin    cos = 12'd3;    sin = -12'd31;    end
8'hC5:    begin    cos = 12'd3;    sin = -12'd31;    end
8'hC6:    begin    cos = 12'd4;    sin = -12'd31;    end
8'hC7:    begin    cos = 12'd5;    sin = -12'd31;    end
8'hC8:    begin    cos = 12'd6;    sin = -12'd31;    end
8'hC9:    begin    cos = 12'd7;    sin = -12'd31;    end
8'hCA:    begin    cos = 12'd7;    sin = -12'd31;    end
8'hCB:    begin    cos = 12'd8;    sin = -12'd30;    end
8'hCC:    begin    cos = 12'd9;    sin = -12'd30;    end
8'hCD:    begin    cos = 12'd10;    sin = -12'd30;    end
8'hCE:    begin    cos = 12'd10;    sin = -12'd30;    end
8'hCF:    begin    cos = 12'd11;    sin = -12'd29;    end
8'hD0:    begin    cos = 12'd12;    sin = -12'd29;    end
8'hD1:    begin    cos = 12'd12;    sin = -12'd29;    end
8'hD2:    begin    cos = 12'd13;    sin = -12'd28;    end
8'hD3:    begin    cos = 12'd14;    sin = -12'd28;    end
8'hD4:    begin    cos = 12'd15;    sin = -12'd28;    end
```

```
8'hD5:    begin    cos = 12'd15;    sin = -12'd27;    end
8'hD6:    begin    cos = 12'd16;    sin = -12'd27;    end
8'hD7:    begin    cos = 12'd17;    sin = -12'd27;    end
8'hD8:    begin    cos = 12'd17;    sin = -12'd26;    end
8'hD9:    begin    cos = 12'd18;    sin = -12'd26;    end
8'hDA:    begin    cos = 12'd19;    sin = -12'd25;    end
8'hDB:    begin    cos = 12'd19;    sin = -12'd25;    end
8'hDC:    begin    cos = 12'd20;    sin = -12'd24;    end
8'hDD:    begin    cos = 12'd20;    sin = -12'd24;    end
8'hDE:    begin    cos = 12'd21;    sin = -12'd23;    end
8'hDF:    begin    cos = 12'd22;    sin = -12'd23;    end
8'hE0:    begin    cos = 12'd22;    sin = -12'd22;    end
8'hE1:    begin    cos = 12'd23;    sin = -12'd22;    end
8'hE2:    begin    cos = 12'd23;    sin = -12'd21;    end
8'hE3:    begin    cos = 12'd24;    sin = -12'd20;    end
8'hE4:    begin    cos = 12'd24;    sin = -12'd20;    end
8'hE5:    begin    cos = 12'd25;    sin = -12'd19;    end
8'hE6:    begin    cos = 12'd25;    sin = -12'd19;    end
8'hE7:    begin    cos = 12'd26;    sin = -12'd18;    end
8'hE8:    begin    cos = 12'd26;    sin = -12'd17;    end
8'hE9:    begin    cos = 12'd27;    sin = -12'd17;    end
8'hEA:    begin    cos = 12'd27;    sin = -12'd16;    end
8'hEB:    begin    cos = 12'd27;    sin = -12'd15;    end
8'hEC:    begin    cos = 12'd28;    sin = -12'd15;    end
8'hED:    begin    cos = 12'd28;    sin = -12'd14;    end
8'hEE:    begin    cos = 12'd28;    sin = -12'd13;    end
8'hEF:    begin    cos = 12'd29;    sin = -12'd12;    end
8'hF0:    begin    cos = 12'd29;    sin = -12'd12;    end
8'hF1:    begin    cos = 12'd29;    sin = -12'd11;    end
```

```
            8'hF2:    begin    cos = 12'd30;    sin = -12'd10;    end
            8'hF3:    begin    cos = 12'd30;    sin = -12'd10;    end
            8'hF4:    begin    cos = 12'd30;    sin = -12'd9;    end
            8'hF5:    begin    cos = 12'd30;    sin = -12'd8;    end
            8'hF6:    begin    cos = 12'd31;    sin = -12'd7;    end
            8'hF7:    begin    cos = 12'd31;    sin = -12'd7;    end
            8'hF8:    begin    cos = 12'd31;    sin = -12'd6;    end
            8'hF9:    begin    cos = 12'd31;    sin = -12'd5;    end
            8'hFA:    begin    cos = 12'd31;    sin = -12'd4;    end
            8'hFB:    begin    cos = 12'd31;    sin = -12'd3;    end
            8'hFC:    begin    cos = 12'd31;    sin = -12'd3;    end
            8'hFD:    begin    cos = 12'd31;    sin = -12'd2;    end
            8'hFE:    begin    cos = 12'd31;    sin = -12'd1;    end
            8'hFF:    begin    cos = 12'd31;    sin = 12'd0;    end

        endcase
    end
endmodule
```

polyhedron2.v - Andrew

```
module polyhedron2(
        input [6:0] address,
        output reg [35: 0] dataOut
       );
    always @(*) begin
        case (address)
              12'd0:
              begin
              dataOut <= {-12'd50, -12'd50, -12'd50};
              end
              12'd1:
              begin
              dataOut <= {-12'd50, -12'd50, -12'd25};
              end
              12'd2:
              begin
              dataOut <= {-12'd50, -12'd50, 12'd0};
              end
              12'd3:
              begin
              dataOut <= {-12'd50, -12'd50, 12'd25};
              end
              12'd4:
```

```
begin
dataOut <= {-12'd50, -12'd50, 12'd50};
end
12'd5:
begin
dataOut <= {-12'd50, -12'd25, -12'd50};
end
12'd6:
begin
dataOut <= {-12'd50, -12'd25, 12'd50};
end
12'd7:
begin
dataOut <= {-12'd50, 12'd0, -12'd50};
end
12'd8:
begin
dataOut <= {-12'd50, 12'd0, 12'd50};
end
12'd9:
begin
dataOut <= {-12'd50, 12'd25, -12'd50};
end
12'd10:
begin
dataOut <= {-12'd50, 12'd25, 12'd50};
end
12'd11:
begin
dataOut <= {-12'd50, 12'd50, -12'd50};
end
12'd12:
begin
dataOut <= {-12'd50, 12'd50, -12'd25};
end
12'd13:
begin
dataOut <= {-12'd50, 12'd50, 12'd0};
end
12'd14:
begin
dataOut <= {-12'd50, 12'd50, 12'd25};
end
12'd15:
begin
dataOut <= {-12'd50, 12'd50, 12'd50};
end
12'd16:
begin
dataOut <= {-12'd25, -12'd50, -12'd50};
end
12'd17:
begin
dataOut <= {-12'd25, -12'd50, 12'd50};
end
12'd18:
begin
```

```
dataOut <= {-12'd25, 12'd50, -12'd50};
end
12'd19:
begin
dataOut <= {-12'd25, 12'd50, 12'd50};
end
12'd20:
begin
dataOut <= {12'd0, -12'd50, -12'd50};
end
12'd21:
begin
dataOut <= {12'd0, -12'd50, 12'd50};
end
12'd22:
begin
dataOut <= {12'd0, 12'd50, -12'd50};
end
12'd23:
begin
dataOut <= {12'd0, 12'd50, 12'd50};
end
12'd24:
begin
dataOut <= {12'd25, -12'd50, -12'd50};
end
12'd25:
begin
dataOut <= {12'd25, -12'd50, 12'd50};
end
12'd26:
begin
dataOut <= {12'd25, 12'd50, -12'd50};
end
12'd27:
begin
dataOut <= {12'd25, 12'd50, 12'd50};
end
12'd28:
begin
dataOut <= {12'd50, -12'd50, -12'd50};
end
12'd29:
begin
dataOut <= {12'd50, -12'd50, -12'd25};
end
12'd30:
begin
dataOut <= {12'd50, -12'd50, 12'd0};
end
12'd31:
begin
dataOut <= {12'd50, -12'd50, 12'd25};
end
12'd32:
begin
dataOut <= {12'd50, -12'd50, 12'd50};
```

```
            end
            12'd33:
            begin
            dataOut <= {12'd50, -12'd25, -12'd50};
            end
            12'd34:
            begin
            dataOut <= {12'd50, -12'd25, 12'd50};
            end
            12'd35:
            begin
            dataOut <= {12'd50, 12'd0, -12'd50};
            end
            12'd36:
            begin
            dataOut <= {12'd50, 12'd0, 12'd50};
            end
            12'd37:
            begin
            dataOut <= {12'd50, 12'd25, -12'd50};
            end
            12'd38:
            begin
            dataOut <= {12'd50, 12'd25, 12'd50};
            end
            12'd39:
            begin
            dataOut <= {12'd50, 12'd50, -12'd50};
            end
            12'd40:
            begin
            dataOut <= {12'd50, 12'd50, -12'd25};
            end
            12'd41:
            begin
            dataOut <= {12'd50, 12'd50, 12'd0};
            end
            12'd42:
            begin
            dataOut <= {12'd50, 12'd50, 12'd25};
            end
            12'd43:
            begin
            dataOut <= {12'd50, 12'd50, 12'd50};
            end
        endcase
    end

endmodule
```

perspectiveTransform.v - Andrew

```
module perspectiveTransform(
        input clk,
        input [7:0] rotX,
        input [7:0] rotY,
        input angleReady,
        output reg angleRequest,
```

```
    output State,
    output [10:0] X,
    output [9:0] Y,
    output reg [6:0] addressOut
   );

wire [11:0] cosX;
wire [11:0] cosY;
wire [11:0] sinX;
wire [11:0] sinY;

reg [6:0] address = 0;
wire [35:0] vertex;
trigLUT trig1(.angle(rotX), .cos(cosX), .sin(sinX));
trigLUT trig2(.angle(rotY), .cos(cosY), .sin(sinY));
polyhedron2 poly1(.address(address), .dataOut(vertex));

reg [11:0] ax;
reg [11:0] ay;
reg [11:0] az;

reg [11:0] r1;
reg [11:0] r2;
reg [11:0] r3;
reg [11:0] r4;
reg [11:0] r5;

reg [6:0] r1d;
reg [6:0] r2d;
reg [6:0] r3d;
reg [6:0] r4d;
reg [6:0] r5d;

reg [11:0] r6;
reg [11:0] r7;

reg [6:0] r6d;
reg [6:0] r7d;

reg [35:0] r8;
reg [35:0] r9;

reg [11:0] r10;
reg [11:0] r11;
reg [11:0] r12;
reg [11:0] r13;

reg [10:0] x;
reg [9:0] y;

assign X = x;
assign Y = y;

// FSM States
parameter S_WAITING = 1'b0; // Waiting for a new angle from angleGrabber
parameter S_PROCESSING = 1'b1; // Processing the new angle
reg state = S_WAITING;
```

```
   always @(posedge clk) begin
       case (state)
             S_PROCESSING:
                   begin
                         angleRequest <= 1'b0;

                         // Stage 1
                         ax <= vertex[35:24];
                         ay <= vertex[23:12];
                         az <= vertex[11:0];

                         // Stage 2
                         r1 <= (ax*cosY);
                         r2 <= (az*sinY);
                         r3 <= (ay*cosX);
                         r4 <= (az*cosY);
                         r5 <= (ax*sinX);

                         // Stage 3
                         r1d <= r1/32;
                         r2d <= r2/32;
                         r3d <= r3/32;
                         r4d <= r4/32;
                         r5d <= r5/32;

                         // Stage 4
                         r8 <= {(r1d[6] == 1'b1) ? {5'b11111, r1d} : {5'b0,
r1d} , (r2d[6] == 1'b1) ? {5'b11111, r2d} : {5'b0, r2d}, (r3d[6] == 1'b1) ?
{5'b11111, r3d} : {5'b0, r3d}};
                         r6 <= ((r4d[6] == 1'b1) ? {5'b11111, r4d} : {5'b0,
r4d})*sinX;
                         r7 <= ((r5d[6] == 1'b1) ? {5'b11111, r5d} : {5'b0,
r5d})*sinY;

                         // Stage 5
                         r6d <= r6/32;
                         r7d <= r7/32;
                         r9 <= r8;

                         // Stage 6
                         r10 <= 12'd384 - r9[35:24];
                         r11 <= 12'd512 + r9[11:0];
                         r12 <= ((r6d[6] == 1'b1) ? {5'b11111, r6d} : {5'b0,
r6d}) + ((r7d[6] == 1'b1) ? {5'b11111, r7d} : {5'b0, r7d});
                         r13 <= r9[23:12];

                         // Stage 7
                         y <= r10 + r13;
                         x <= r11 + r12;

                         address <= address + 1;

                         if (address === 7'd127) begin
                               state <= S_WAITING;
                               address <= 0;
                               angleRequest <= 1'b1;
```

```
                              y <= 0;
                              x <= 0;
                        end

                        else begin
                              if (address > 5) begin
                                    addressOut <= address - 6;
                              end
                        end
                  end

            S_WAITING:
                  begin
                        angleRequest <= 1'b1;

                        if (angleReady) begin
                              angleRequest <= 1'b0;
                              state <= S_PROCESSING;
                        end
                  end
        endcase
   end

endmodule
```

pointSprite.v - Andrew

```
module pointSprite
  #(parameter WIDTH = 2,
            HEIGHT = 2,
                        NUM = 0
                        )
  (input clock,
    input [10:0] X,hcount,
      input [9:0] Y,vcount,
    input [6:0] address,
      output pixelOn,
    output reg [10:0] x_in,
    output reg [9:0] y_in);

   always @(posedge clock) begin
       if (address === NUM) begin
             x_in <= X;
             y_in <= Y;
       end
   end

  assign pixelOn = ((hcount >= x_in && hcount < (x_in+WIDTH)) && (vcount >=
y_in && vcount < (y_in+HEIGHT))) ? 1 : 0;

endmodule
```

threeDimRenderer.v - Andrew

```
module threeDimRenderer(
      input clock,
    input [15:0] rotX_in,
```

```
     input [15:0] rotY_in,
     input [10:0] hcount,
     input [9:0] vcount,
     output requestOut,
     output [7:0] rotX,
     output [7:0] rotY,
     output _request,
     output angleReady,
     output [10:0] X,
     output [9:0] Y,
     output [43:0] pixels,
     output [6:0] addressOut
       );

    angleGrabberFSM anglegrabber1(
        .clock(clock),
        .hcount(hcount),
        .vcount(vcount),
        .requestIn(_request),
        .rotX_in(rotX_in),
        .rotY_in(rotY_in),
        .angleReady(angleReady),
        .requestOut(requestOut),
        .rotX(rotX),
        .rotY(rotY)
        );

    perspectiveTransform pt1(
        .clk(clock),
        .rotX(rotX),
        .rotY(rotY),
        .angleReady(angleReady),
        .angleRequest(_request),
        .X(X),
        .Y(Y),
        .addressOut(addressOut)
        );

    assign pixel = |pixels;


pointSprite #(.NUM(0))

vs0(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addre
ssOut), .pixelOn(pixels[0]));

pointSprite #(.NUM(1))

vs1(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addre
ssOut), .pixelOn(pixels[1]));

pointSprite #(.NUM(2))

vs2(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addre
ssOut), .pixelOn(pixels[2]));

pointSprite #(.NUM(3))
```

```
vs3(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addre
ssOut), .pixelOn(pixels[3]));

pointSprite #(.NUM(4))

vs4(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addre
ssOut), .pixelOn(pixels[4]));

pointSprite #(.NUM(5))

vs5(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addre
ssOut), .pixelOn(pixels[5]));

pointSprite #(.NUM(6))

vs6(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addre
ssOut), .pixelOn(pixels[6]));

pointSprite #(.NUM(7))

vs7(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addre
ssOut), .pixelOn(pixels[7]));

pointSprite #(.NUM(8))

vs8(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addre
ssOut), .pixelOn(pixels[8]));

pointSprite #(.NUM(9))

vs9(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addre
ssOut), .pixelOn(pixels[9]));

pointSprite #(.NUM(10))

vs10(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[10]));

pointSprite #(.NUM(11))

vs11(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[11]));

pointSprite #(.NUM(12))

vs12(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[12]));

pointSprite #(.NUM(13))

vs13(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[13]));

pointSprite #(.NUM(14))

vs14(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
```

```
essOut), .pixelOn(pixels[14]));

pointSprite #(.NUM(15))

vs15(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[15]));

pointSprite #(.NUM(16))

vs16(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[16]));

pointSprite #(.NUM(17))

vs17(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[17]));

pointSprite #(.NUM(18))

vs18(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[18]));

pointSprite #(.NUM(19))

vs19(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[19]));

pointSprite #(.NUM(20))

vs20(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[20]));

pointSprite #(.NUM(21))

vs21(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[21]));

pointSprite #(.NUM(22))

vs22(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[22]));

pointSprite #(.NUM(23))

vs23(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[23]));

pointSprite #(.NUM(24))

vs24(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[24]));

pointSprite #(.NUM(25))

vs25(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[25]));
```

```
pointSprite #(.NUM(26))

vs26(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[26]));

pointSprite #(.NUM(27))

vs27(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[27]));

pointSprite #(.NUM(28))

vs28(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[28]));

pointSprite #(.NUM(29))

vs29(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[29]));

pointSprite #(.NUM(30))

vs30(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[30]));

pointSprite #(.NUM(31))

vs31(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[31]));

pointSprite #(.NUM(32))

vs32(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[32]));

pointSprite #(.NUM(33))

vs33(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[33]));

pointSprite #(.NUM(34))

vs34(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[34]));

pointSprite #(.NUM(35))

vs35(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[35]));

pointSprite #(.NUM(36))

vs36(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[36]));

pointSprite #(.NUM(37))
```

```
vs37(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[37]));

pointSprite #(.NUM(38))

vs38(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[38]));

pointSprite #(.NUM(39))

vs39(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[39]));

pointSprite #(.NUM(40))

vs40(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[40]));

pointSprite #(.NUM(41))

vs41(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[41]));

pointSprite #(.NUM(42))

vs42(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[42]));

pointSprite #(.NUM(43))

vs43(.clock(clock), .X(X),.Y(Y),.hcount(hcount),.vcount(vcount),.address(addr
essOut), .pixelOn(pixels[43]));

endmodule
```

vgaSignal.v - Andrew (gestural interaction mode)

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

   initial begin
       vcount <= 0;
       hcount <= 0;
       vsync <= 0;
       hsync <= 0;
       blank <= 0;
   end

  // horizontal: 1344 pixels total
  // display 1024 pixels per line
  reg hblank,vblank;
  wire hsyncon,hsyncoff,hreset,hblankon;
  assign hblankon = (hcount == 1023);
  assign hsyncon = (hcount == 1047);
  assign hsyncoff = (hcount == 1183);
  assign hreset = (hcount == 1343);
```

```
  // vertical: 806 lines total
  // display 768 lines
  wire vsyncon,vsyncoff,vreset,vblankon;
  assign vblankon = hreset & (vcount == 767);
  assign vsyncon = hreset & (vcount == 776);
  assign vsyncoff = hreset & (vcount == 782);
  assign vreset = hreset & (vcount == 805);

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

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

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

endmodule
```

vgaSignal.v - Andrew (anaglyph mode)

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

  // horizontal: 1344 pixels total
  // display 640 pixels per line
  reg hblank,vblank;
  wire hsyncon,hsyncoff,hreset,hblankon;
  assign hblankon = (hcount == 639);
  assign hsyncon = (hcount == 652);
  assign hsyncoff = (hcount == 1183);
  assign hreset = (hcount == 1343);

  // vertical: 806 lines total
  // display 480 lines
  wire vsyncon,vsyncoff,vreset,vblankon;
  assign vblankon = hreset & (vcount == 479);
  assign vsyncon = hreset & (vcount == 492);
  assign vsyncoff = hreset & (vcount == 782);
  assign vreset = hreset & (vcount == 805);

      initial begin
            hcount      <= 0;
            vcount      <= 0;
            hsync       <= 0;
            vsync       <= 0;
            blank       <= 0;
      end
```

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

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

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

zbt2vga.v - Andrew

```
module zbt2vga(reset,clk,hcount,vcount,vr_pixel,
               vram_addr,vram_read_data);

  input reset, clk;
  input     [10:0]      hcount;
  input     [9:0]       vcount;
  output    [17:0]      vr_pixel;
  output    [18:0]      vram_addr;
  input     [35:0]      vram_read_data;

  wire [18:0]      vram_addr = {vcount[8:0], hcount[9:0]};

  wire [1:0]       hc4 = hcount[1:0];
  reg       [17:0]       vr_pixel;
  reg       [35:0]       vr_data_latched;
  reg       [35:0]       last_vr_data;

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

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

  always @(*)
    case (hc4)
      2'd3: vr_pixel = last_vr_data[17:0];
      2'd1: vr_pixel = last_vr_data[17+18:0+18];
    endcase
endmodule
```

vgaController.v - Andrew and Adam

```
module vgaController(
       input vclock, // 65 MHz
       input  [115:0] pixels,
      input [9:0] l_xlo,
```

```
    input [9:0] l_xhi,
    input [8:0] l_ylo,
    input [8:0] l_yhi,
    input [9:0] r_xlo,
    input [9:0] r_xhi,
    input [8:0] r_ylo,
    input [8:0] r_yhi,
    input [7:0] switch,
     output [7:0] vga_out_red,
     output [7:0] vga_out_green,
     output [7:0] vga_out_blue,
     output [10:0] hcount,
     output [9:0] vcount,
     output vga_out_sync_b,
     output vga_out_blank_b,
     output vga_out_pixel_clock,
     output vga_out_hsync,
     output vga_out_vsync
    );

 // generate basic VGA video signals
wire hsync,vsync,blank;

vgaSignal vgasignal1(.vclock(vclock),.hcount(hcount),.vcount(vcount),
          .hsync(hsync),.vsync(vsync),.blank(blank));

 reg b,hs,vs;
 reg pixel;
 always @(posedge vclock) begin
     hs     <= hsync;
     vs     <= vsync;
     b     <= blank;
     pixel    <= (|pixels == 1'b1) ? 1: 0;
 end

wire [9:0] x_actual;
assign x_actual = 639 - vcount;
wire [9:0] y_actual;
assign y_actual = hcount[9:0];

wire left_inside;
assign left_inside = hcount < 480 && vcount < 640;
wire right_inside;
assign right_inside = hcount >= 512 && hcount < 992 && vcount < 640;

wire left_on;
wire right_on;

box box_l(l_xlo, l_xhi, l_ylo, l_yhi, x_actual, y_actual, left_on);
box box_r(r_xlo, r_xhi, r_ylo, r_yhi, x_actual, y_actual, right_on);

wire background;
assign background = left_inside || right_inside;
wire blob;
assign blob = (left_inside && left_on) || (right_inside && right_on);

// VGA Output.  In order to meet the setup and hold times of the
```

```
  // AD7125, we send it ~clock_65mhz.
  assign vga_out_red = {switch[5:4], switch[5:4],
                        switch[5:4], switch[5:4]} & {8{pixel}};
  assign vga_out_green = {switch[3:2], switch[3:2],
                        switch[3:2], switch[3:2]} & {8{blob}};
  assign vga_out_blue = {switch[1:0], switch[1:0],
                        switch[1:0], switch[1:0]} & {8{background}};
  assign vga_out_sync_b = 1'b1;     // not used
  assign vga_out_blank_b = ~b;
  assign vga_out_pixel_clock = ~vclock;
  assign vga_out_hsync = hs;
  assign vga_out_vsync = vs;
endmodule
```

nes_controller.v - Adam

This was used for testing the virtual camera controller and 3D renderer with fake camera data.

```
module nes_controller(output nes_clk, output nes_latch, input nes_data,
      input start_read, input clk, output [7:0] buttons);
  // 7: A
  // 6: B
  // 5: Select
  // 4: Start
  // 3: Up
  // 2: Down
  // 1: Left
  // 0: Right
  reg [7:0] temp_buttons;
  reg [7:0] r_buttons;
  reg [4:0] state;

  reg [8:0] clock_ticks;

  reg r_clk, r_latch;

  initial begin
      temp_buttons = 8'd0;
      state = 5'd0;
  end

  always @(posedge clk) begin
      if (clock_ticks == 9'd389) begin
      if (start_read && state == 5'd0) begin
            // Begin polling the controller
            state <= 5'd18;
            r_latch <= 1'b1;
            temp_buttons <= 8'b00000000;
      end else begin
            case (state)
            5'd18: begin
                  state <= 5'd17;
            end
            5'd17, 5'd15, 5'd13, 5'd11,
            5'd9, 5'd7, 5'd5, 5'd3: begin
                  // Read a button
                  state <= (state - 5'd1);
                  r_clk <= 1'b0;
```

```
                r_latch <= 1'b0;
                temp_buttons[7] <= temp_buttons[6];
                temp_buttons[6] <= temp_buttons[5];
                temp_buttons[5] <= temp_buttons[4];
                temp_buttons[4] <= temp_buttons[3];
                temp_buttons[3] <= temp_buttons[2];
                temp_buttons[2] <= temp_buttons[1];
                temp_buttons[1] <= temp_buttons[0];
                temp_buttons[0] <= nes_data;
            end
            5'd16, 5'd14, 5'd12, 5'd10,
            5'd8, 5'd6, 5'd4, 5'd2: begin
                // Bring nes_clk up to load the next button
                state <= (state - 5'd1);
                r_clk <= 1'b1;
            end
            5'd1: begin
                // Finished polling, push the new changes
                state <= 5'd0;
                r_clk <= 1'b0;
                r_buttons <= ~temp_buttons;
            end
            default: begin
            end
            endcase
        end
        clock_ticks <= 9'd0;
        end else begin
        clock_ticks <= clock_ticks + 9'd1;
        end
    end

    assign nes_clk = r_clk;
    assign nes_latch = r_latch;
    assign buttons = r_buttons;
endmodule
```

main.v - Andrew and Adam
Adapted from labkit.v.

```
module Main (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,

            disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
            disp_reset_b, disp_data_in,

            button0, button1, button2, button3, button_enter, button_right,
            button_left, button_down, button_up,

            switch,

            led,

            user1, user2, user3, user4,

            daughtercard,

            systemace_data, systemace_address, systemace_ce_b,
            systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

            analyzer1_data, analyzer1_clock,
            analyzer2_data, analyzer2_clock,
            analyzer3_data, analyzer3_clock,
            analyzer4_data, analyzer4_clock);

output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
input  ac97_bit_clock, ac97_sdata_in;

output [7:0] vga_out_red, vga_out_green, vga_out_blue;
output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
   vga_out_hsync, vga_out_vsync;

output [9:0] tv_out_ycrcb;
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,
   tv_out_subcar_reset;

input  [19:0] tv_in_ycrcb;
input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
   tv_in_hff, tv_in_aff;
output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
   tv_in_reset_b, tv_in_clock;
inout  tv_in_i2c_data;
```

```
  inout  [35:0] ram0_data;
  output [18:0] ram0_address;
  output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
  output [3:0] ram0_bwe_b;

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

  input  clock_feedback_in;
  output clock_feedback_out;

  inout  [15:0] flash_data;
  output [23:0] flash_address;
  output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
  input  flash_sts;

  output rs232_txd, rs232_rts;
  input  rs232_rxd, rs232_cts;

  input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

  input  clock_27mhz, clock1, clock2;

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

  input  button0, button1, button2, button3, button_enter, button_right,
     button_left, button_down, button_up;
  input  [7:0] switch;
  output [7:0] led;

  inout [31:0] user1, user2, user3, user4;

  inout [43:0] daughtercard;

  inout  [15:0] systemace_data;
  output [6:0]  systemace_address;
  output systemace_ce_b, systemace_we_b, systemace_oe_b;
  input  systemace_irq, systemace_mpbrdy;

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

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

  // Audio Input and Output
  assign beep= 1'b0;
```

```
assign audio_reset_b = 1'b0;
assign ac97_synch = 1'b0;
assign ac97_sdata_out = 1'b0;
// ac97_sdata_in is an input

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

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

// SRAMs
assign ram0_data = 36'hZ;
assign ram0_address = 19'h0;
assign ram0_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;
// clock_feedback_in is an input

// Flash ROM
assign flash_data = 16'hZ;
assign flash_address = 24'h0;
assign flash_ce_b = 1'b1;
assign flash_oe_b = 1'b1;
assign flash_we_b = 1'b1;
assign flash_reset_b = 1'b0;
```

```
  assign flash_byte_b = 1'b1;
  // flash_sts is an input

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

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

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

  // Buttons, Switches, and Individual LEDs
  // assign led = 8'hFF;

  // User I/Os
  assign user1 = 32'hZ;
  assign user2 = 32'hZ;
  assign user3 = 32'hZ;
  assign user4[31:19] = 13'hZ;
  assign user4[15:3] = 13'hZ;

  // Daughtercard Connectors
  assign daughtercard = 44'hZ;

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

  // Logic Analyzer
  assign analyzer1_data = 16'h0;
  assign analyzer1_clock = 1'b1;
  assign analyzer2_data = 16'h0;
  assign analyzer2_clock = 1'b1;
  assign analyzer3_data = 16'h0;
  assign analyzer3_clock = 1'b1;
  assign analyzer4_data = 16'h0;
  assign analyzer4_clock = 1'b1;

   ////////////////////////////////////////////////////////////////////////
//
  //
  // Main
  //
  ////////////////////////////////////////////////////////////////////////
```

```
/

   // use FPGA's digital clock manager to produce a
  // 65MHz clock (actually 64.8MHz)
  wire clock_65mhz_unbuf,clock_65mhz;
  DCM vclk1(.CLKIN(clock_27mhz),.CLKFX(clock_65mhz_unbuf));
  // synthesis attribute CLKFX_DIVIDE of vclk1 is 10
  // synthesis attribute CLKFX_MULTIPLY of vclk1 is 24
  // synthesis attribute CLK_FEEDBACK of vclk1 is NONE
  // synthesis attribute CLKIN_PERIOD of vclk1 is 37
  BUFG vclk2(.O(clock_65mhz),.I(clock_65mhz_unbuf));

    wire [15:0] rotX_in;
    wire [15:0] rotY_in;
    wire [10:0] hcount;
    wire [9:0] vcount;
    wire angleRequest;
    wire pixel;
    wire [115:0] pixels;

    wire clock;
    assign clock = clock_65mhz;

      // Start of Adam's code

    reg [31:0] count_ticks = 32'd0;

      wire data_ready;
      wire address_ready;
      wire [35:0] frame_data;
      wire right_frame;
      wire [9:0] x_address;
      wire [8:0] y_address;
      wire frame_request;
      wire [40:0] theta;
      wire [40:0] phi;

      wire [35:0] left_data;
      wire [35:0] right_data;

      assign data_ready = ~|count_ticks[25:10];
      assign address_ready = 1;

      reg [9:0] left_xc = 480;
      reg [8:0] left_yc = 240;
      reg [7:0] left_halfsize = 8;

      reg [9:0] right_xc = 160;
      reg [8:0] right_yc = 240;
      reg [7:0] right_halfsize = 8;

      wire [9:0] l_xlo;
      assign l_xlo = left_xc - left_halfsize;
      wire [9:0] l_xhi;
      assign l_xhi = left_xc + left_halfsize;
      wire [8:0] l_ylo;
      assign l_ylo = left_yc - left_halfsize;
```

```
  wire [8:0] l_yhi;
  assign l_yhi = left_yc + left_halfsize;

  wire [9:0] r_xlo;
  assign r_xlo = right_xc - right_halfsize;
  wire [9:0] r_xhi;
  assign r_xhi = right_xc + right_halfsize;
  wire [8:0] r_ylo;
  assign r_ylo = right_yc - right_halfsize;
  wire [8:0] r_yhi;
  assign r_yhi = right_yc + right_halfsize;

  // Left frame sees 16x16 block centered at (480, 240)
  wire left_on;
  box box_l_1(l_xlo, l_xhi, l_ylo, l_yhi, x_address, y_address, left_on);
  assign left_data = left_on ? 36'hFFFFFFFFF : 36'd0;
  // Right frame sees 16x16 block centered at (160, 240)
  wire right_on;
  box box_r_1(r_xlo, r_xhi, r_ylo, r_yhi, x_address, y_address, right_on);
  assign right_data = right_on ? 36'hFFFFFFFFF : 36'd0;

  assign frame_data = right_frame ? right_data : left_data;

  virtual_camera_controller vcc(.clk(clock_65mhz),
                           .global_reset(reset),
                           .enable(switch[7]),
                           .data_ready(data_ready),
                           .address_ready(address_ready),
                           .frame_data(frame_data),
                           .update(angleRequest),
                           .threshold(8'd150),
                           .right_frame(right_frame),
                           .x_address(x_address),
                           .y_address(y_address),
                           .request(frame_request),
                           .theta(theta),
                           .phi(phi));

  wire [63:0] data_display;

  display_16hex d16h(reset, clock_65mhz, {theta[40:9], phi[40:9]},
                     disp_blank, disp_clock, disp_rs, disp_ce_b,
                     disp_reset_b, disp_data_out);

  // Rest of Andrew's code

assign rotX_in = theta[40:25];
assign rotY_in = phi[40:25];

threeDimRenderer threeDim1(
   .clock(clock),
   .rotX_in(rotX_in),
   .rotY_in(rotY_in),
   .hcount(hcount),
   .vcount(vcount),
   .requestOut(angleRequest),
   .pixels(pixels)
```

```
);

vgaController vgaControl1(
   .vclock(clock),
   .pixels(pixels),
  .l_xlo(l_xlo),
  .l_xhi(l_xhi),
  .l_ylo(l_ylo),
  .l_yhi(l_yhi),
  .r_xlo(r_xlo),
  .r_xhi(r_xhi),
  .r_ylo(r_ylo),
  .r_yhi(r_yhi),
  .switch(switch),
   .vga_out_red(vga_out_red),
   .vga_out_green(vga_out_green),
   .vga_out_blue(vga_out_blue),
   .hcount(hcount),
   .vcount(vcount),
   .vga_out_sync_b(vga_out_sync_b),
   .vga_out_blank_b(vga_out_blank_b),
   .vga_out_pixel_clock(vga_out_pixel_clock),
   .vga_out_hsync(vga_out_hsync),
   .vga_out_vsync(vga_out_vsync)
  );

  wire [7:0] left_buttons;
  nes_controller left_edit(.nes_clk(user4[0]),
                           .nes_latch(user4[1]),
                           .nes_data(user4[2]),
                           .start_read(vsync),
                           .clk(clock_65mhz),
                           .buttons(left_buttons));

  wire [7:0] right_buttons;
  nes_controller right_edit(.nes_clk(user4[16]),
                           .nes_latch(user4[17]),
                           .nes_data(user4[18]),
                           .start_read(vsync),
                           .clk(clock_65mhz),
                           .buttons(right_buttons));

  assign led = ~{left_buttons[3:0], right_buttons[3:0]};

  always @(posedge clock_65mhz) begin
  count_ticks <= count_ticks + 1;
  if (vcount == 768 && hcount == 0) begin
        case (left_buttons[1:0])
        2'b01: begin // move right
              if (left_xc + left_halfsize < 640) begin
                    left_xc <= left_xc + 1;
              end
        end
        2'b10: begin // move left
              if (left_xc > left_halfsize) begin
                    left_xc <= left_xc - 1;
              end
```

```
            end
            endcase
            case (left_buttons[3:2])
            2'b01: begin // move down
                  if (left_yc + left_halfsize < 480) begin
                        left_yc <= left_yc + 1;
                  end
            end
            2'b10: begin // move up
                  if (left_yc > left_halfsize) begin
                        left_yc <= left_yc - 1;
                  end
            end
            endcase

            case (right_buttons[1:0])
            2'b01: begin // move right
                  if (right_xc + right_halfsize < 640) begin
                        right_xc <= right_xc + 1;
                  end
            end
            2'b10: begin // move left
                  if (right_xc > right_halfsize) begin
                        right_xc <= right_xc - 1;
                  end
            end
            endcase
            case (right_buttons[3:2])
            2'b01: begin // move down
                  if (right_yc + right_halfsize < 480) begin
                        right_yc <= right_yc + 1;
                  end
            end
            2'b10: begin // move up
                  if (right_yc > right_halfsize) begin
                        right_yc <= right_yc - 1;
                  end
            end
            endcase
      end
      end

endmodule
```

field_selector_tf.v (test bench) - Adam
Run for 3.2 ms to see the whole test bench.

```
module fetch_controller_tf;

   // Inputs
   reg clk;
   reg global_reset;
   reg all_data_ready;
   reg address_ready;

   // Outputs
```

```
    wire [9:0] x_address;
    wire [8:0] y_address;
    wire request;
       wire right_frame;
    wire done;
    wire reset;

    // Instantiate the Unit Under Test (UUT)
    fetch_controller uut (
        .clk(clk),
        .global_reset(global_reset),
        .all_data_ready(all_data_ready),
        .address_ready(address_ready),
        .x_address(x_address),
        .y_address(y_address),
        .request(request),
       .right_frame(right_frame),
        .done(done),
        .reset(reset)
    );

    initial begin
        clk = 1;
        forever #5 clk = ~clk;
    end

    initial begin
        #105;
        address_ready = 1;
        forever #50 address_ready = ~address_ready;
    end

    initial begin
        // Initialize Inputs
        global_reset = 0;
        all_data_ready = 0;
        address_ready = 0;

        // Wait 100 ns for global reset to finish
        #100;
        #5;
        all_data_ready = 1;
        #6100000;
        all_data_ready = 0;
        #100000;
        all_data_ready = 1;

        // Add stimulus here

    end

endmodule
```

field_selector_tf.v (test bench) - Adam

```
module field_selector_tf;

    // Inputs
```

```
    reg [35:0] data;
    reg lr;

    // Outputs
    wire [35:0] left;
    wire [35:0] right;

    // Instantiate the Unit Under Test (UUT)
    field_selector uut (
        .data(data),
        .lr(lr),
        .left(left),
        .right(right)
    );

    initial begin
        // Initialize Inputs
        data = 0;
        lr = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here
       data = 1;
       repeat (72) begin
             lr = 0;
             #10;
             lr = 1;
             #10;
             data = {data[34:0], data[35]};
       end
    end

endmodule
```

dot_location_calculator_tf.v (test bench) - Adam
Note: This was made in a project assuming the chip was an Artix 7, not a Virtex-II. Also, the divider is 54-bit by 48-bit, and was made with CoreGen's Divider Generator 4.0. This is available in the free web pack version of ISE 13, but it is not available in the lab!
Run for 2000 ns.
At 1000 ns, you should have dtheta ≈ -28942, dphi ≈ 25297.
At 2000 ns, you should have dtheta ≈ -10, dphi ≈ -15.

```
module dot_location_calculator_tf;

    // Inputs
    reg clk;
    reg done;
    reg [26:0] SLx;
    reg [26:0] SLy;
    reg [18:0] SL;
    reg [26:0] SRx;
    reg [26:0] SRy;
    reg [18:0] SR;
```

```
// Outputs
wire [31:0] dtheta;
wire [31:0] dphi;
wire valid;

// Instantiate the Unit Under Test (UUT)
dot_location_calculator uut (
    .clk(clk),
    .done(done),
    .SLx(SLx),
    .SLy(SLy),
    .SL(SL),
    .SRx(SRx),
    .SRy(SRy),
    .SR(SR),
    .dtheta(dtheta),
    .dphi(dphi),
    .valid(valid)
);

   initial begin
    clk = 1;
    forever #5 clk = ~clk;
end

initial begin
    // Initialize Inputs
    done = 0;
    SLx = 0;
    SLy = 0;
    SL = 0;
    SRx = 0;
    SRy = 0;
    SR = 0;

    // Wait 100 ns for global reset to finish
    #105;

    // Add stimulus here
   done = 1;
   SLx = 8014;
    SLy = 4384;
    SL = 44;
    SRx = 6014;
    SRy = 4384;
    SR = 44;
   #800;
   done = 0;
   #200;
   done = 1;
   SLx = 10573891;
   SLy = 4952382;
   SL = 20574;
   SRx = 2573891;
   SRy = 4952382;
   SR = 20574;
```

```
      #1000;
   end

endmodule
```

integrator_tf.v (test bench) - Adam
Run for 31 µs. Theta and phi should change by around 120 every 20 ns.

```
module integrator_tf;

   // Inputs
   reg clk;
   reg global_reset;
   reg [31:0] dtheta;
   reg [31:0] dphi;
      reg change_ready;
   reg update;

   // Outputs
   wire [15:0] theta;
   wire [15:0] phi;

   // Instantiate the Unit Under Test (UUT)
   integrator uut (
       .clk(clk),
       .global_reset(global_reset),
       .dtheta(dtheta),
       .dphi(dphi),
      .change_ready(change_ready),
       .update(update),
       .theta(theta),
       .phi(phi)
   );

      initial begin
       clk = 1;
       forever #5 clk = ~clk;
   end

      initial begin
      update = 0;
      #115;
      forever #10 update = ~update;
      end

   initial begin
       // Initialize Inputs
       global_reset = 0;
      change_ready = 1;
       dtheta = 0;
       dphi = 0;

       // Wait 100 ns for global reset to finish
       #105;

       // Add stimulus here
```

```
      dtheta = 2000000000;
      dphi = 2000000000;
      #15000;
      dtheta = -2000000000;
      dphi = -2000000000;
      #15200;
      global_reset = 1;
      #50;
      global_reset = 0;
   end

endmodule
```

virtual_camera_controller.tf (test bench) - Adam
Run for 3.6 ms.

```
module virtual_camera_controller_tf;

   // Inputs
   reg clk;
   reg global_reset;
   reg enable;
   reg data_ready;
   reg address_ready;
   wire [35:0] frame_data;
   reg update;
   reg [7:0] threshold;

   // Outputs
   wire right_frame;
   wire [9:0] x_address;
   wire [8:0] y_address;
   wire request;
   wire [40:0] theta;
   wire [40:0] phi;

  // Other stuff
  wire [35:0] left_data;
  wire [35:0] right_data;
  // Left frame sees 16x16 block centered at (320, 240)
  assign left_data = (x_address >= 312 && x_address < 328 &&
                      y_address >= 232 && y_address < 248) ?
36'hFFFFFFFFF : 36'd0;
  // Right frame sees 16x16 block centered at (160, 240)
  assign right_data = (x_address >= 152 && x_address < 168 &&
                       y_address >= 232 && y_address < 248) ?
36'hFFFFFFFFF : 36'd0;
  assign frame_data = right_frame ? right_data : left_data;

   // Instantiate the Unit Under Test (UUT)
   virtual_camera_controller uut (
       .clk(clk),
       .global_reset(global_reset),
       .enable(enable),
       .data_ready(data_ready),
       .address_ready(address_ready),
       .frame_data(frame_data),
```

```
        .update(update),
        .threshold(threshold),
        .right_frame(right_frame),
        .x_address(x_address),
        .y_address(y_address),
        .request(request),
        .theta(theta),
        .phi(phi)
    );

  initial begin
      clk = 1;
      forever #5 clk = ~clk;
  end

  initial begin
      #105;
      update = 0;
      forever #20 update = ~update;
  end

   initial begin
       // Initialize Inputs
       global_reset = 0;
       enable = 1;
       data_ready = 0;
       address_ready = 1;
       update = 0;
       threshold = 150;

       // Wait 100 ns for global reset to finish
       #105;

       // Add stimulus here
      data_ready = 1;
      #100;
      data_ready = 0;
   end

endmodule
```

memcontrol.v - Tim

```
module memcontrol(
   input clk,                      // system clock
   input reset,
   input mode,                     // system mode: 0 anaglyph,
                            // 1 virtual camera control

   input [18:0] ntsc_addr, // input from ntsc2zbt
   input [35:0] ntsc_data,
   input ntsc_we,

   output frame_edge,
   output vcc_ready_all,      // virtual camera control ready
   output [35:0] vcc_data, // 2 x 18 bit (6 bits / channel) pixels
```

```
   input [18:0] vcc_xy,    // 10 bits x, 9 bits y
   input vcc_frame,        // 0 left 1 right

   output [35:0] vga_data,
   input [18:0] vga_xy,

   output    ram1_we_b,         // RAM 1: physical line to ram we_b
   output [18:0] ram1_address,      // physical line to ram address
   inout [35:0]  ram1_data,   // physical line to ram data
   output    ram2_we_b,         // RAM 2: physical line to ram we_b
      output [18:0] ram2_address,   // physical line to ram address
   inout [35:0]  ram2_data        // physical line to ram data
   );

      // Memory arbitration (access to front before by VGA or VCC)
      reg [18:0] vga_xy_old;
      reg [18:0] vcc_xy_old;
      reg vcc_we;
      reg vga_we;
      reg [35:0] back_data;
      reg vga_frame;
      reg ntsc_we_delay;
      reg [18:0] ntsc_addr_delay;
      reg [35:0] ntsc_data_delay;
      reg [2:0] vga_readreq_delay;

      // if there's an address transition, and address is valid
      wire vcc_req = (vcc_xy != vcc_xy_old);
      wire vga_req = (vga_xy != vga_xy_old) && (vga_xy[9:1] < 10'd639) &&
(vga_xy[18:10] <= 10'd480);

      // use the old xy values since WE is delayed by 1 cycle as well
      wire [18:0] vcc_addr = {vga_frame, vcc_xy_old[8:0], vcc_xy_old[18:10]};
      wire [18:0] vga_addr = {vga_frame, vga_xy_old[8:0], vga_xy_old[18:10]};

      // ZBT free wires
      wire back_zbt_free = ~ntsc_we;
      wire front_zbt_free = mode ? ~vcc_req : ~vga_req;

      // Outputs
      wire flip_complete;
      wire [35:0] front_data;
      wire back_we;
      wire front_we;
      wire [18:0] flip_ram1_addr;
      wire [18:0] flip_ram2_addr;
      wire [35:0] flip_ram1_data;
      wire [35:0] flip_ram2_data;


      // Buffer flipping
      flipbuffer  flipper (
            .clk(clk),
            .reset(reset),
            .back_zbt_free(back_zbt_free),
            .front_zbt_free(front_zbt_free),
```

```
      .back_addr(flip_ram1_addr),
      .front_addr(flip_ram2_addr),
      .back_data(flip_ram1_data),
      .flip_complete(flip_complete),
      .frame(~vga_frame),
      .front_data(flip_ram2_data),
      .back_we(back_we),
      .front_we(front_we),
      .mode(mode)
);

wire [18:0] b_addr;
wire [18:0] f_addr;
wire [35:0] f_write_data;
wire [35:0] f_read_data;
wire [35:0] b_write_data;
wire [35:0] b_read_data;
wire f_we, b_we;

// ZBT controllers
zbt_6111 front_zbt(clk, f_we, f_addr, f_write_data, f_read_data,
        ram2_we_b, ram2_address, ram2_data);

zbt_6111 back_zbt(clk, b_we, b_addr, b_write_data, b_read_data,
        ram1_we_b, ram1_address, ram1_data);

// RAM input multiplexing
assign b_we = ntsc_we_delay;
assign f_we = front_we;
assign b_addr = ntsc_we_delay ? ntsc_addr_delay : flip_ram1_addr;
assign f_addr = front_we ? flip_ram2_addr : vga_addr;
assign b_write_data = ntsc_data_delay;
assign flip_ram1_data = b_read_data;
assign f_write_data = flip_ram2_data;
assign vga_data = vga_readreq_delay[2] ? f_read_data : 36'd0;
assign vcc_data = {36{1'bZ}};
assign frame_edge = flip_complete;

always @(posedge clk) begin
      if(reset == 1'b1) begin
            vcc_xy_old <= 19'd0;
            vga_xy_old <= 19'd0;
            vcc_we <= 1'b0;
            vga_we <= 1'b0;
            vga_frame <= 1'b0;
            ntsc_we_delay <= 1'b0;
      end else begin

            // delay the xy_old change detection
            vga_xy_old <= vga_xy;
            vcc_xy_old <= vcc_xy;


            // set the we register
            if(mode == 1'b0) begin
```

```
                // delay the write memory address (since NTSC we is also
delayed)
                        ntsc_addr_delay <= ntsc_addr;

                // delay the WE (to give flip buffer a chance)
                        ntsc_we_delay <= ntsc_we;

                // delay the NTSC data
                        ntsc_data_delay <= ntsc_data;

                // Delay output enable of vga read request
                        vga_readreq_delay <= {vga_readreq_delay[1],
vga_readreq_delay[0], ~vga_we} ;

                        if(vga_req) begin
                              vga_we <= 1'b0;

                        end else begin
                              vga_we <= 1'b1;
                        end
                   end else begin
                        if(vcc_req) begin
                              vcc_we <= 1'b0;
                        end else begin
                              vcc_we <= 1'b1;
                        end
                   end

                   // in anaglyph mode, switch the part of the front buffer
                   // that is used to output data to the screen
                   // (so there's no tearing while the screen is updated)
                   if(mode == 1'b0) begin
                        if(flip_complete == 1'b1) begin
                              vga_frame <= ~vga_frame;
                        end
                   end
             // end ~reset
             end
      // end always
      end

endmodule
```

flipbuffer.v - Tim

```
module flipbuffer(
   input clk,
      input reset,
      input back_zbt_free,
      input front_zbt_free,
   output reg [18:0] back_addr,
   output reg [18:0] front_addr,
   input [35:0] back_data,
      output reg flip_complete,
      // VGA reads from the frame not being written to
```

```
   input frame, // 0 == left, 1 == right (for writing to front)
output reg [35:0] front_data,
   output reg back_we,
   output reg front_we,
input mode
);

   // keeping track of what we've copied
   reg [9:0] x;
   reg [8:0] y;

   // true if already waiting for read response
   reg left_reqd;
   reg right_reqd;

   // input to delay line
   reg delay_l_pulse;
   reg delay_r_pulse;

   // true when valid data is latched into registers
   reg left_ready;
   reg right_ready;

   // both left and right pixels have been loaded
   wire data_ready = left_ready && right_ready;


         // true when the pulse from delay_l/r_pulse
         // propagates through delay line
         wire data_ready_l;
         wire data_ready_r;

         delayN d1(clk, delay_l_pulse, data_ready_l);
         delayN d2(clk, delay_r_pulse, data_ready_r);


         // contains data from the zbt reads
         reg [35:0] left_data;
         reg [35:0] right_data;

         // contains data to input to the anaglyph filters
         reg [17:0] left_px1;
         reg [17:0] left_px2;
         reg [17:0] right_px1;
         reg [17:0] right_px2;

         // output from the anaglyph filters
         wire [17:0] w_aglyph_px1;
         wire [17:0] w_aglyph_px2;


         // anaglyph filter instances
         anaglyph aglyph1 (clk, left_px1, right_px1, w_aglyph_px1);
         anaglyph aglyph2 (clk, left_px2, right_px2, w_aglyph_px2);

         // true when the anaglyph contains valid data
```

```
        reg pixel_ready;

    always @ (posedge clk) begin
        if(reset == 1'b1) begin
            x <= 10'd0;
            y <= 9'd0;
            left_reqd <= 1'b0;
            right_reqd <= 1'b0;
            delay_l_pulse <= 1'b0;
            delay_r_pulse <= 1'b0;
            pixel_ready <= 1'b0;
            left_ready <= 1'b0;
            right_ready <= 1'b0;
            left_px1 <= 18'd0;
            right_px1 <= 18'd0;
            left_px2 <= 18'd0;
            right_px2 <= 18'd0;
            left_data <= 36'd0;
            right_data <= 36'd0;
            flip_complete <= 1'b0;
            front_we <= 0;
        end else begin
            // only request data if the back buffer is free next cycle
            if(left_reqd == 1'b0) begin
                if(back_zbt_free == 1'b1) begin

                    delay_l_pulse <= 1'b1;
                    left_reqd <= 1'b1;
                    back_we <= 1'b0;
                    back_addr <= {1'b0, y, x[9:1]};
                end else begin
                    back_we <= 1'b1;
                    delay_l_pulse <= 1'b0;
                end
            end else if (right_reqd == 1'b0) begin

                delay_l_pulse <= 1'b0;

                if(back_zbt_free == 1'b1) begin

                    delay_r_pulse <= 1'b1;
                    right_reqd <= 1'b1;
                    back_we <= 1'b0;
                    back_addr <= {1'b1, y, x[9:1]};
                end else begin
                    back_we <= 1'b1;
                    delay_r_pulse <= 1'b0;
                end
            end else if(left_reqd == 1'b1 && right_reqd == 1'b1) begin

                    back_we <= 1'b1;
                    delay_l_pulse <= 1'b0;
                    delay_r_pulse <= 1'b0;

                    // reset the requested registers if memory was
access sucessfully
```

```
                        if(data_ready == 1'b1) begin
                              left_reqd <= 1'b0;
                              right_reqd <= 1'b0;
                        end
                  end

                        // load the data into the anaglyph filter
                        if(data_ready == 1'b1) begin
                              left_px1 <= left_data[17:0];
                              left_px2 <= left_data[35:18];
                              right_px1 <= right_data[17:0];
                              right_px2 <= right_data[35:18];
                              left_ready <= 1'b0;
                              right_ready <= 1'b0;
                              pixel_ready <= 1'b1;
                        end

                        // LATCH DATA FROM ZBT

                        // if the ready pulse came through, latch the result
                        if(data_ready_l == 1'b1) begin
                              left_ready <= 1'b1;
                              left_data <= back_data;
                        end else if(data_ready_r == 1'b1) begin
                              right_ready <= 1'b1;
                              right_data <= back_data;
                        end

                        if(pixel_ready == 1'b1 && front_zbt_free == 1'b1)
begin
                              front_we <= 1'b1;
                              front_data <= left_data;// ideal version:
{left_data[17:0], right_data[17:0]};
                              front_addr <= {1'b0, y, x[9:1]};
                              pixel_ready <= 1'b0;

                              // iterate through the image
                              x <= (x == 638) ? 0 : (x + 2);
                              if(y == 479 && x == 638) begin
                                    y <= 0;
                              end else if (x == 638) begin
                                    y <= y + 1;
                              end

                              // if we reached the end, pulse a completion
                              if(x == 638 && y == 479) begin
                                    flip_complete <= 1'b1;
                              end
                        end

                        if (flip_complete == 1'b1) begin
                              flip_complete <= 1'b0;
                        end

                        if(front_we == 1'b1) begin
                              front_we <= 1'b0;
```

```
                        end

                // // end if reset
                end

        // end always
        end
```

ntsc_to_zbt.v - Tim

```
module ntsc_to_zbt(clk, vclk, fvh, dv, din, ntsc_addr, ntsc_data, ntsc_we,
source);
        input clk; // system clock
        input vclk; // video clock from camera
        input [2:0] fvh;
        input dv;
        input [17:0] din;
        output [18:0] ntsc_addr;
        output [35:0] ntsc_data;
        input source;
        output ntsc_we; // write enable for NTSC data
        parameter COL_START = 10'd0;
        parameter ROW_START = 10'd0;

        // here put the luminance data from the ntsc decoder into the ram
        // this is for 1024 x 768 XGA display
        reg [9:0] col = 0;
        reg [9:0] row = 0;
        reg [17:0] vdata = 0;
        reg vwe;
        reg old_dv;
        reg old_frame;
        reg even_odd;
        wire frame = fvh[2];
        wire frame_edge = frame & ~old_frame;

        always @ (posedge vclk) //LLC1 is reference
                begin
                        old_dv <= dv;
                        vwe <= dv && !fvh[2] & ~old_dv; // if frame valid, write it
                        old_frame <= frame;
                        even_odd = frame_edge ? ~even_odd : even_odd;

                        if (!fvh[2]) begin
                                col <= fvh[0] ? (COL_START) :
                                                        (!fvh[2] && !fvh[1] && dv &&
(col < 640)) ? col + 1 : col;
                                row <= fvh[1] ? (ROW_START) :
                                                        (!fvh[2] && fvh[0] && (row <
480)) ? row + 1 : row;
                                vdata <= (dv && !fvh[2]) ? din : vdata;
                        end
                end

                // synchronize with system clock
                reg [9:0] x[1:0],y[1:0];
```

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

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

            // edge detection
            reg old_we;
            wire we_edge = we[1] & ~old_we;

            always @(posedge clk) old_we <= we[1];

            // shift two pixels into one word
            reg [35:0] mydata;

            always @(posedge clk)
                  if (we_edge)
                        mydata <= { mydata[17:0], data[1] };

            // data format, high bit is video frame
            wire [18:0] myaddr = {source, y[1][7:0], eo[1], x[1][9:1]};//{x[1]
[9:1], 1'b0, y[1][8:1], eo[1]};

            // update the output address and data only when word is ready
            reg [18:0] ntsc_addr;
            reg [35:0] ntsc_data;

            // write enable only on every second pixel
            wire ntsc_we = we_edge & (x[1][0]==1'b1);

            always @(posedge clk)
                  if ( ntsc_we ) begin
                        ntsc_addr <= myaddr;
                        ntsc_data <= mydata;
                  end

endmodule // ntsc_to_zbt
```

camera_multiplexer (in top level module) - Tim

```
      // Persistence of vision anaglyph conversion
      ycrcb2rgb ycrcb2rgb(red, grn, blu,
                        tv_in_line_clock1, reset, ycrcb[29:20], ycrcb[19:10],
ycrcb[9:0]);

      wire [7:0] r_aglph;
      wire [7:0] g_aglph;
      wire [7:0] b_aglph;

      assign r_aglph = side ? red : 8'd0;
      assign g_aglph = side? 8'd0 : grn;
```

```
	assign b_aglph = side ? 8'd0 : blu;

	// code to write NTSC data to video memory
	wire [18:0] ntsc_addr;
	wire [35:0] ntsc_data;
	wire        ntsc_we;
	ntsc_to_zbt n2z (clk, tv_in_line_clock1, fvh, dv, {r_aglph[7:2],
g_aglph[7:2], b_aglph[7:2]},
			ntsc_addr, ntsc_data, ntsc_we, zbt_page_write);

	// Frame switching
	wire field = fvh[2];
	reg old_field;
	wire field_edge;
	assign field_edge = (field != old_field);
	reg[1:0] frame;
	reg vga_frame_complete;
	reg side;

	always @(posedge clk) begin
		old_field <= field;

			if(field != old_field) begin
				if(frame == 2'b11 && vga_frame_complete == 1'b1) begin
					side <= ~side;
					frame <= 2'b00;
					vga_frame_complete <= 1'b0;
				end else begin
					frame <= frame + 1;
				end
			end else begin

				if(vga_out_vsync == 1'b1) begin
					vga_frame_complete <= 1'b1;
				end
			end
	end
```