

Stereoscopic Observation System (SOS)

6.111 Fall 2019

Jeana Choi, Leilani Trautman, Ryan Mansilla

Table of Contents

- 1 [Background](#)
 - 2 [Overview](#)
 - 3 [Project Goals](#)
 - 4 [System Block Diagram](#)
 - 5 [Subsystems](#)
 - 5.1 [Camera Setup/Display](#)
 - 5.2 [Object Detection](#)
 - 5.3 [Camera Calibration](#)
 - 5.4 [Distance Calculation](#)
 - 5.5 [Servo](#)
 - 5.6 [Display](#)
 - 6 [Testing & Debugging](#)
 - 7 [Challenges](#)
 - 8 [Design Decisions](#)
 - 9 [Reflections](#)
 - 10 [Conclusion](#)
 - 11 [Acknowledgements](#)
- [Appendix A - Verilog Code](#)
- [Appendix B - Non-Verilog Code](#)
- [Appendix C - CAD Files](#)

1 Background

You wish to gather information about your environment, particularly the location of a target and its distance away from you. Perhaps it is for animal tracking, securing your home, or miscellaneous projectiles. Believe it or not, this is possible with two cameras rotating on a servo!

To implement our system, we will have two calibrated cameras attached rigidly to a servo that sweeps the room. Both cameras will be sending image data to the FPGA. We then process this image data by converting from RGB to HSV, threshold based on Hue, and erode the resulting image to get rid of noise. This binary image is then passed to the object detector to detect the coordinates of a target of a known shape and color. If the target is detected, we will use projective geometry to locate its coordinates relative to the camera and calculate its distance from the cameras. Lastly, we will have a visual representation through the VGA display of the two camera feeds and the resulting top and side views of the environment.

2 Overview

We have two calibrated cameras rotating on a servo, scanning the environment. Once a target of a known color and shape comes into the field of view for both cameras, we can detect the center of the target. With this, we can project two 3D vectors from each 2D image plane to a point on a plane we have calibrated to. Then, we can find the shortest distance between these two skew lines and take its midpoint - this is our world coordinate of the target. Finally, we can use the coordinates of the target to calculate the distance.

To imitate a practical stereoscopic system, we wish to have a tracking mechanism by rigidly attaching a laser to the mounted cameras. Once we know the location and velocity of the ball, we can use the servo to rotate the cameras and follow the target's predicted trajectory. We display the camera data on a VGA display, where we show the centroid detection in real time and also the top and side views of our world environment.

3 Project Goals

Commitment:

- Centroid of green ball displayed on screen
- Servo spinning on one axis
- Calibration of camera (including serial images to laptop)

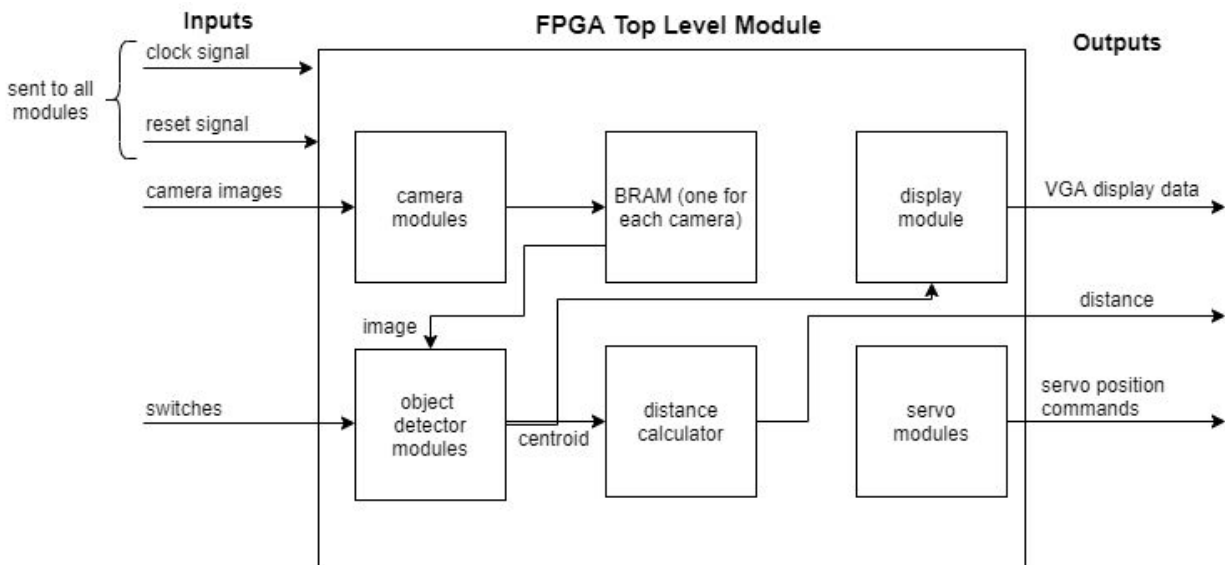
Goal:

- Distance to ball displayed on screen
 - Stationary ball centroid detection and distance calculation
 - Moving ball centroid detection and distance calculation
- Detect balls of different colors (not at the same time)
- Showing vector from each camera to ball on the VGA display

Stretch:

- Two balls detected with centroid displayed on screen
- Distance between two balls displayed on screen
- Following a single ball with 2-axes of motion from servo

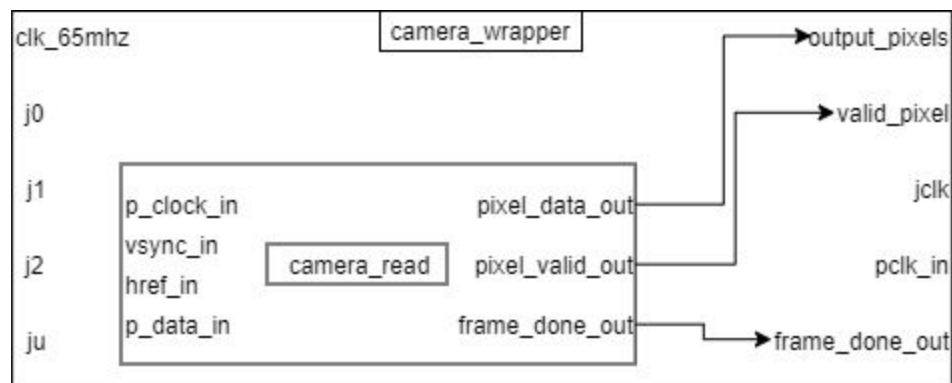
4 System Block Diagram



5 Subsystems

5.1 Camera Setup/Display - Leilani

- **Modules:** [camera_wrapper.sv](#), [top_level.sv](#), [camera_read.sv](#)



To begin, we started off with the camera code provided by Joe. We modified it to remove unnecessary remnants of the pong game code from Lab 3. We then wrapped up much of the code from the original `top_level` file that read in the camera data and put it into a

camera_wrapper module which we then instantiated twice - once for each of the cameras. The wrapper mainly deals with taking the inputs from the GPIO pins connected to the camera and passing them into top_level under the correct variable names. The camera_wrapper also instantiates an instance of the camera_read module that contains the state machine to read the pixels from the incoming frame. The main task with reading in the camera data is dealing with the camera's timing then moving the pixel data into a BRAM for each camera. This pixel data is then passed into the object_detection module for RGB to HSV conversion, hue thresholding, erosion and dilation.

5.2 Object Detection - Ryan

- **Modules:** [object_detection.sv](#), [rgb2hsv.sv](#), [hue_thresholding.sv](#), [erosion.sv](#), [dilation.sv](#), [localizer.sv](#)

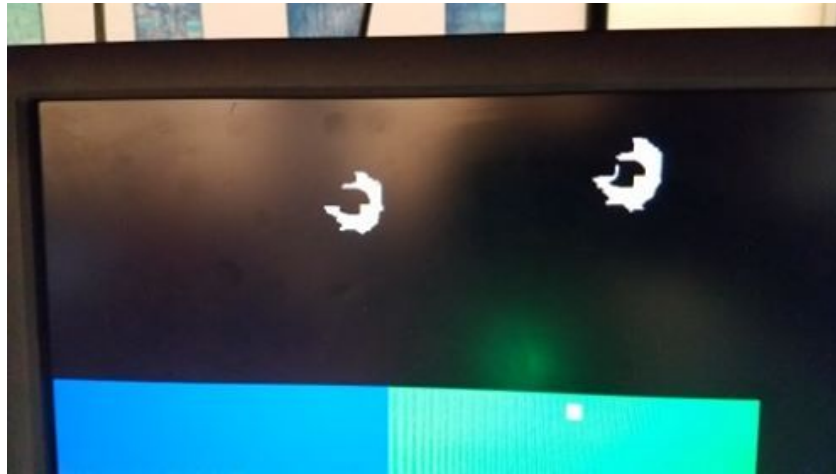
clk		
dilate		[15:0] centroid_x
erode		[15:0] centroid_y
[1:0] thresholds		[11:0] pixel_out
[23:0] pixel_in	object_detection	[24:0] x_acc
hcount		[24:0] y_acc
vcount		[24:0] bit-count
inBounds		

rgb2hsv.sv

To create data that can be easily used by the FPGA for calculating information (in this case, calculating the centroids of objects), we had to create modules that would process the RGB pixels that the cameras put out. The first step in processing involved converting the RGB pixels to HSV, which simplifies the process for binarizing the RGB pixels. By converting to HSV, you can easily select for certain colors based on the hue value (the "H" in HSV).

hue_thresholding.sv

We thresholded values for certain hue values to select for colors of our choosing (green and blue) with switches, so that objects of a single color can be detected while ignoring the background image. Unfortunately even if one were to choose good threshold values, a significant amount of noise remains. For this reason we employed a module that would erode the binarized values.



Hue thresholding for green light

erosion.sv

The erosion module takes in the output from the hue thresholding module and saves those values in a 320x1 register. Once enough of this register has been filled, the kernel (in the form of the AND of 7 contiguous values in the register) gives the output.

dilation.sv

A dilation module was then used to increase the number of pixels with a value of one, under the assumption that the erosion module would convert to many values to zero. However, upon testing, it was found that the dilation module did not serve its purpose well; in fact it usually increased the noise of the image, therefore we excluded it from our image processing pipelining. It only remains in the final iteration of the project as an option to view.

localizer.sv

Finally, to localize the centroid of the object, we have a module that takes in the output of the erosion module, which is the output of the kernel function. As the output of the erosion function continues to feed in this module, hcount and vcount, the coordinate of the pixels in the VGA display, are added to accumulators for the x- and y-axis, provided the erosion module's output was high. When hcount and vcount indicate that a whole frame of an image has been received by the module, it sends a signal to two divider IP modules to start division and resets the accumulators. The output of these dividers are the calculated x and y coordinates of the centroid.



Image as a result of erosion (see the noise). Top left corner of the yellow square represents the centroid's coordinates.

5.3 Camera Calibration - Jeana

➤ **Modules:** [serial_tx.sv](#), [top_level.sv](#)

The distance calculation is split into multiple steps, mainly into:

- **Sending and Saving Camera Snapshot**
- **Camera Calibration for Intrinsic Parameters**
- **Camera Calibration for Extrinsic Parameters**
- Distance Calculator Module
- Find projective lines from each camera to the object's center
- Calculate coordinates of where the lines come closest together

First, if we want to know the location of a camera or the size of an object in the real world, we need to retrieve the camera parameters, which are generally represented as a 3x4 camera matrix P . In order to estimate the intrinsic and extrinsic camera parameters, we need to have 3D world coordinates and their corresponding 2D image coordinates.

We can achieve these by introducing the pinhole camera model:

$$w \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \begin{bmatrix} R \\ t \end{bmatrix} K$$

Where:

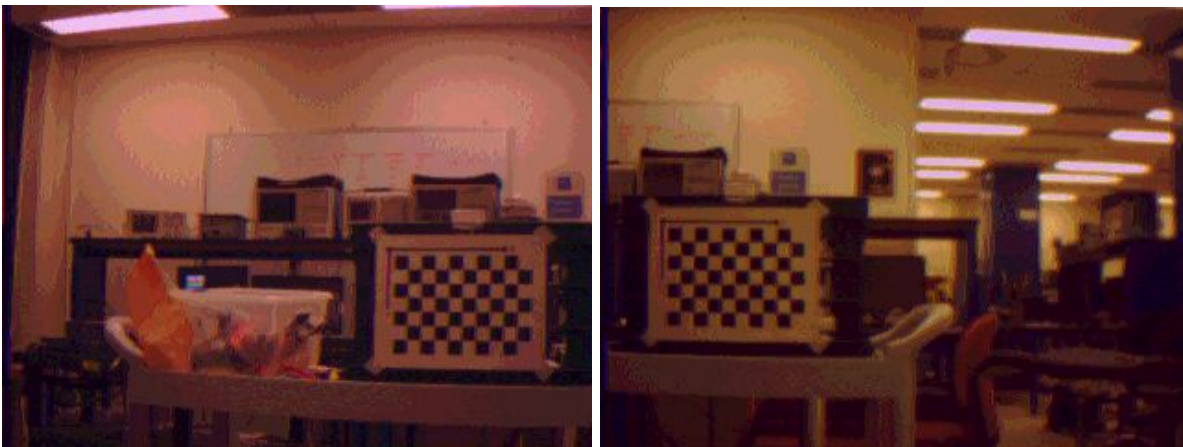
- (X, Y, Z) are the world coordinates of the point we are looking at
- (x,y) : 2D image coordinates of the 3D point in pixels
- w : scaling factor
- K : camera intrinsic matrix (explain that it gives us the focal length, optical center in pixels)
- R : rotation matrix representing 3D rotation of the camera
- t : translation of camera relative to the world coordinate system

Essentially, we can solve for (X, Y, Z) for each camera, which will give us two 3D vectors mapping from the camera's image plane to the checkerboard plane (the instance of the checkerboard that we used to tune our extrinsic parameters).

We calibrated each camera using Zhang's Calibration method, which is implemented in MatLab's Single Camera Calibration App. This application takes in images from the camera containing a fixed flat pattern, the most common being a checkerboard, in various orientations and calculates the intrinsic and extrinsic parameters.

Sending and Saving Camera Snapshot

In order to save the checkerboard calibration images, we save an entire frame of video then send the pixels over serial. We then have a Python script that converts the pixels to an image file.

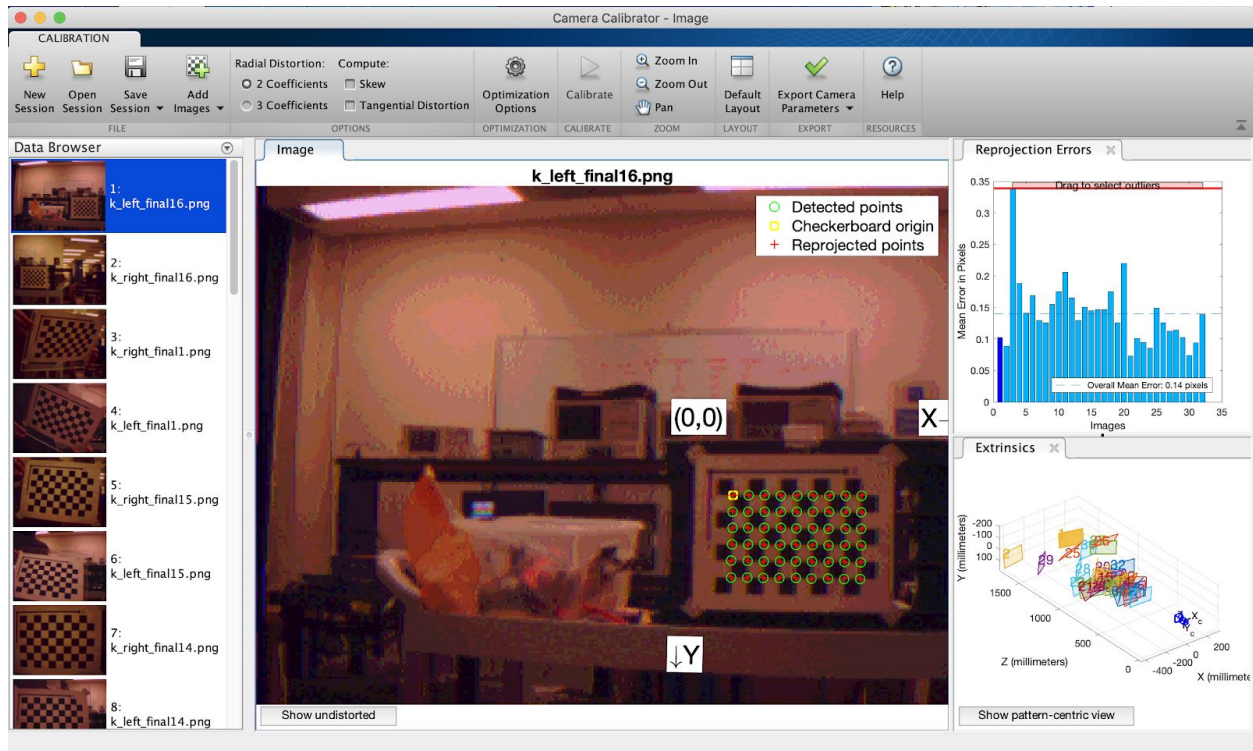


Shown above: a snapshot of the checkerboard taken simultaneously by both cameras. These two images were used to retrieve the extrinsic parameters R and t .

To create each image file, we have to take a couple steps:

First, we take a 'snapshot' on the camera, or essentially disable the wea pin when we flip a switch (sw[7]). Then, we have a state machine that reads from the BRAM pixel by pixel and

sends R, G, then B to our laptop through serial transmission. The receiving computer has a Python script that receives and reformats the data into a Numpy array, making sure that the values are bit shifted by 4 (since we are taking the 4 MSB of color). Lastly, we save this array into an image file using Pillow (a Python package).



Calibration results using MATLAB's Single Camera Calibrator App. The checkerboard pattern is detected in every image and compiles them to compute intrinsic and extrinsic parameters. This calibration sets the real world origin to the origin of the checkerboard.

OFF-FPGA Calculations

Intrinsic Parameters

The intrinsic camera matrix, most commonly denoted by K , tells us the underlying camera properties such as focal length and the optical center. We use the same K for both cameras since they are essentially identical. In order to retrieve a more accurate intrinsic camera matrix, we need to upload at least 10 images of the checkerboard in varying orientations. In our project, we used 16 images.

Extrinsic Parameters

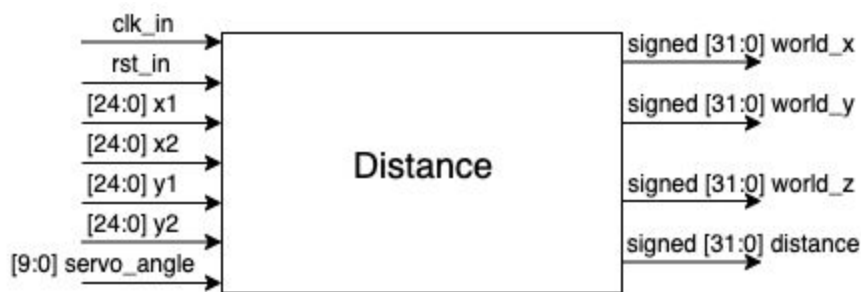
The extrinsic parameters are given as R and t , the rotation and translation matrices. These tell us how to map each camera's image plane to the instance of the checkerboard in the real world, which is where the real world origin is located.

Calculating Camera Matrix

We can rearrange the pinhole camera model from $w [x \ y \ 1] = [X \ Y \ Z \ 1] [R \ t]^T K$ to $[X \ Y \ Z \ 1] = W P^+ [x \ y \ 1]$, where $P = [R \ t]^T K$ and P^+ is its pseudoinverse. Since these are all precalculated, we can solve for P^+ and hardcode them onto the FPGA.

5.4 Distance Calculation - Jeana

➤ **Modules:** [distance.sv](#), [my_division.sv](#), [my_sqrt.sv](#)

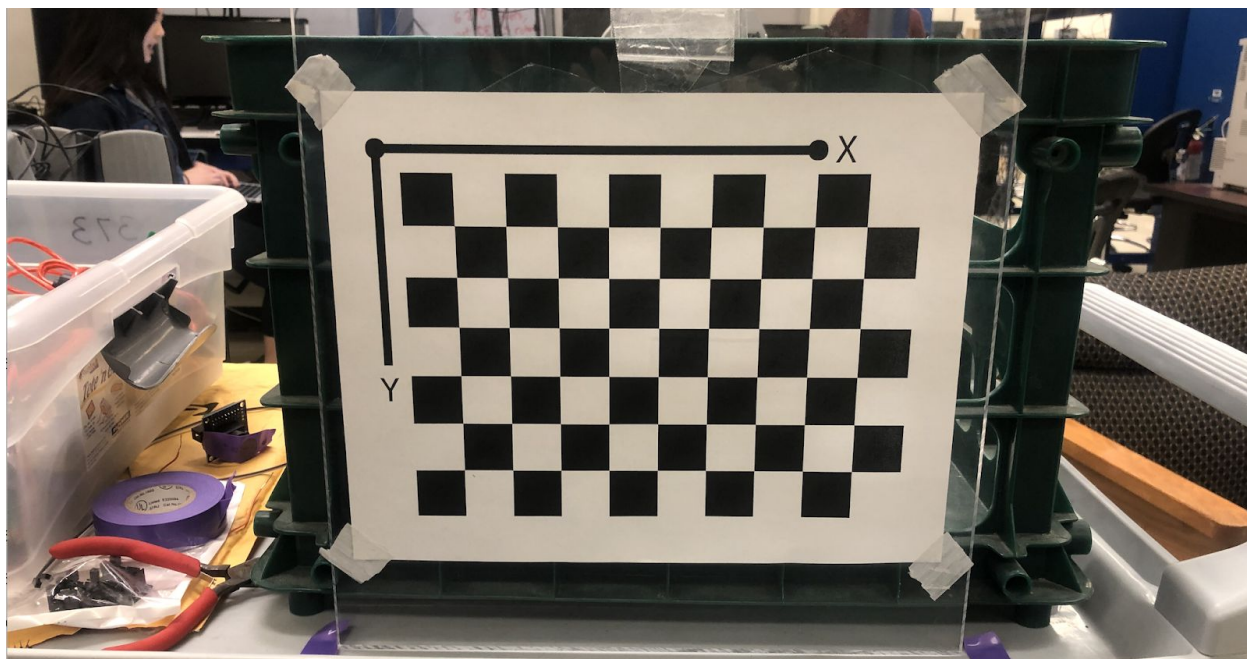


The distance calculation is split into multiple steps, mainly into:

- Sending and Saving Camera Snapshot
- Camera Calibration for Intrinsic Parameters
- Camera Calibration for Extrinsic Parameters
- **Distance Calculator Module**
 - **Find projective lines from each camera to the object's center**
 - **Calculate coordinates of where the lines come closest together**

We are looking to solve the two 3D world vectors, then find the midpoint of the line closest to them. One simplification we can make is that we know exactly what plane the checkerboard is in. Even if the checkerboard is out of the field of view, the calibration we made with a specific instance of the checkerboard determines the world origin. Although the checkerboard pattern itself is 2D, we set up the board so that it is perpendicular to the floor. As a result, the Z value in the checkerboard plane is a known constant value.

We need to note of the coordinate axes we are using - currently, we are following the same coordinate system in our 2D images as the checkerboard setup. Once we have a set coordinate system in the real world, it is easier to debug later when we check for signs.



Calculating 3D World Vectors

We can calculate the world vectors projecting from the image plane to the checkerboard plane for each camera. This is our (X, Y, Z) for each camera when solving the camera model equation.

Solving for the 3D Coordinates of Target

After we solve the 3D world vectors, we can find the shortest distance between these two skew lines by using parametric equations.

As we increase the parametric variable t , we can solve for the t such that the points on both lines are closest to one another.

Then, we can average the coordinates of those two coordinates to get our midpoint, or the real world coordinates of our target.

$$u = [x_{\text{world1}} \ 0] - C_1$$

$$v = [x_{\text{world2}} \ 0] - C_2$$

$$t_{cpa} = - \frac{(C_1 - C_2) \cdot (u - v)}{|u - v|^2}$$

$$midpoint = \frac{(C1 + t_{cpa} * u + C2 + t_{cpa} * v)}{2}$$

Where u, v are unit vectors and t_{cpa} is the point in time where u and v are closest to each other. We take this point in time and find the midpoint of that shortest line, as shown above.

Solving for Distance from Origin

Once we solve for the real world coordinates, we can easily find the distance from the origin by calculating the magnitude between the two points $(0, 0, 0)$ and (X, Y, Z) , which is:

$$D = \sqrt{X^2 + Y^2 + Z^2}$$

On-FPGA Calculations

Since we wanted these distances to update in real time based on our current centroid inputs $(x1, y1)$ and $(x2, y2)$, we decided to pipeline all the computations. This ended up becoming 14 stages, since we needed to separate all the multiplications in order to avoid timing problems. We decided to use 32 bit fixed point numbers for the majority of the module, although we did use floating points for the occasional division or square root.

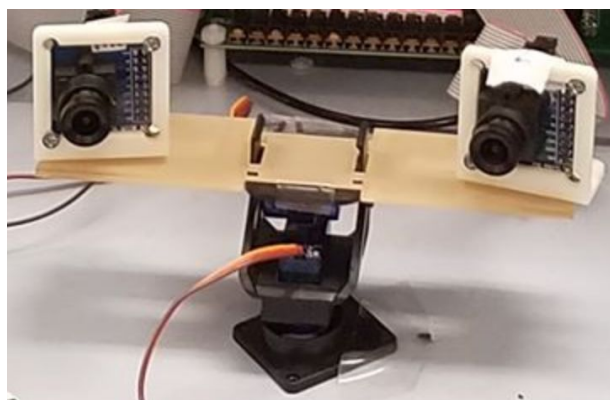
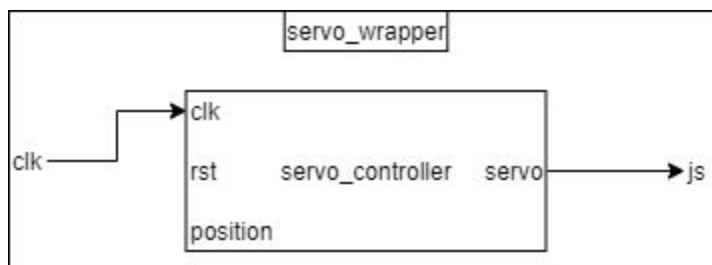
Fixed Point Numbers

I used 32 bit fixed point numbers for the majority of my variables (where the MSB corresponds to sign, the next 15 represent integers, and the 16 LSB represent the decimal portion of the number. Using fixed points is a lot more trivial to use for addition and subtraction than floating points, since we can just use the '+' and '-' operations. For multiplication, I assigned the product of two 32 fixed point numbers to a temporary 64 bit signed floating point. Then, I used an `always_comb` block to only keep the inner 32 bits of the product. For division, I used my `my_division` module, where I decided to convert a 32 bit fixed point to a floating point, performed division using the IP Catalog Modules, then converted the quotient back to a fixed point. I did the same for computing square roots, where I created a new module `my_sqrt.sv` to convert the fixed point to a floating point, then perform square root, then convert the result back to a fixed point.

5.5 Servo - Leilani

➤ Modules: `servo.sv`, `servo_wrapper.sv`, `fsm.sv`

The code for controlling the servo is organized similarly to the code for reading in camera data in that they both employ wrappers to clean up the code. The `servo_wrapper` is declared at top_level and within the wrapper, a 50MHz clock is made in order to communicate with the servo and the sweeping motion of the servo is set.



5.6 Display - Ryan

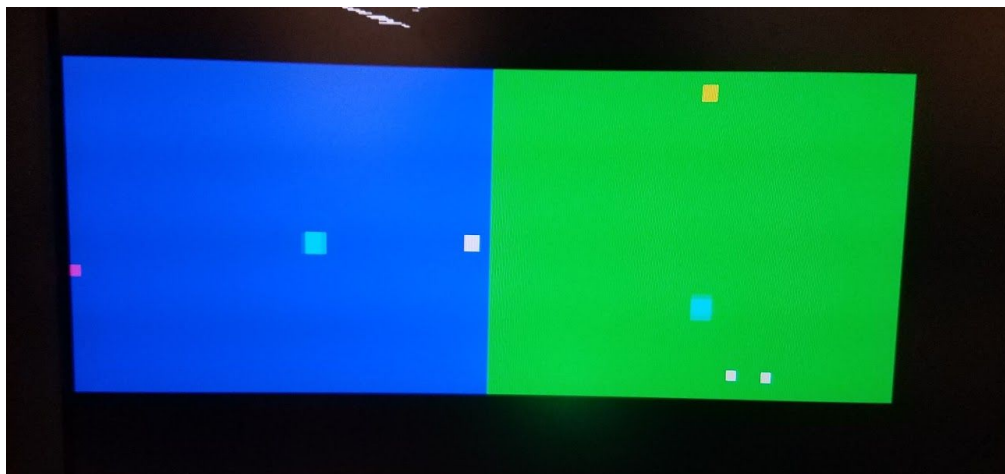
➤ Modules: `top_level.sv`



The data from our calculations are ultimately expressed in the VGA display for an easier understanding of the object's location and characteristics.

The `top_level` module instantiates the `display_select` module, determines what gets displayed and where, three times, once for the left camera, another time for the right camera, and the third time for the top and side views of the object's position relative to the camera and checkerboard origin. The object detection module is then used to give the eroded binarized image and the location of the centroid for the two cameras.

The distance module is also instantiated, so that the distance of the object from the origin checkerboard can be displayed via the position of the blobs in the top and side view regions on the display. Essentially, the detected object is the sky blue square and it moves along the respective axes in both the side and top view (according to the object's relative position to the origin and cameras).




The left view represents the side view, where the white square represents the camera and the pink represents the origin. The right view represents the top view, where the yellow square represents the origin and the white squares represent the cameras.

6 Testing & Debugging

Camera Setup

First, we tested the camera display with one camera using Joe's code. We also uploaded the ESP8266 code from Joe with slight modifications to the camera RGB settings in order to make it easier to detect green objects, which was our initial goal. Next, we tested the camera display with two cameras bit by bit as we moved code into the `camera_wrapper`. Finally, we integrated the object detection modules with the two camera display. Previously



the object detection had been tested on a static ROM so the integration with the cameras allowed for refining the hue thresholding on our real-world setting with our LEDs and balls.

Object Detection

Before writing any Verilog code, it was important to understand how the existing VGA display code worked and how it could be used. Once it made sense, we started writing the object detection module and tested it using three different kinds of images:

1. Black and white image (to avoid binarization, work on erosion/dilation first)
2. Color image of M&Ms to filter out the green M&M and perform the entire object detection pipeline on a still image
3. Once hue thresholding was perfected, we tested the module on real-time camera feed and began work on the centroid detection portion.

Camera Calibration

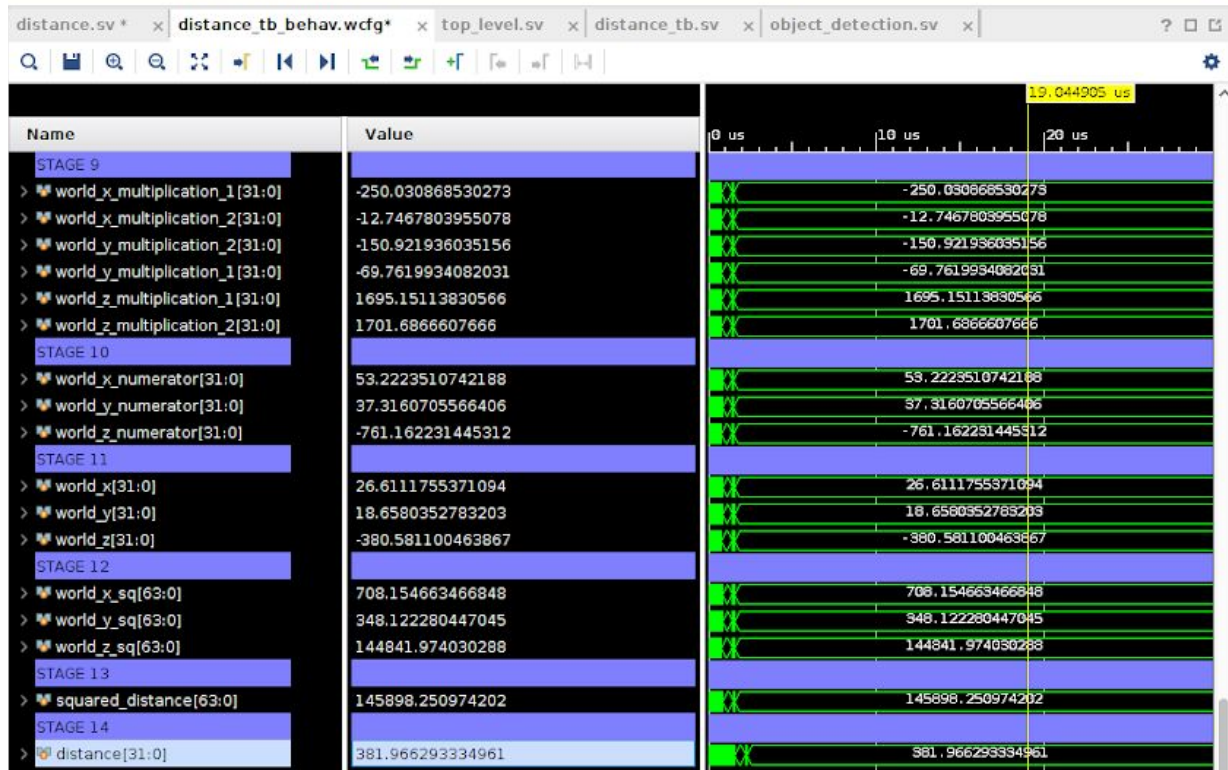
To get started on sending the pixels over serial transmission, I first used the code we had from Lab 2. Then, I tried sending only the 4 red bits padded with zeros over serial - with this, I debugged most of my bugs in my state machine. Then, I sent the RGB separately for a total of $320 \times 240 \times 3$ transmissions. I decided to use this method since serial can only receive 1 byte of data at a time, so I wanted to consistently send 4 bits of color for simplicity. Lastly, once we integrated the two cameras, I readjusted my state machine to be able to send images from both cameras, not just one.

Distance Calculation

Our first step for debugging and testing our distance calculation was to run all my equations on a new MATLAB script. This is separate from the Single Camera Calibration App, since I am using the program to check my matrix operations. The first test I did was to confirm that if I chose the pixels in both cameras corresponding to the checkerboard's origin and computed all the math, that the resulting distance should be near zero coordinates should be near (0,0,0) as well. Once I confirmed those calculations, I then measured a foot away from the real world origin and again, solved for distance with manual centroid inputs (since it was not completed yet at the time).

These two tests were what I used to debug my code when I converted everything from MATLAB to Verilog. Since I was pipelining 14 stages, I decided to create a testbench and painstakingly debug at each stage. I caught many errors this way, some of which being:

variable misassignments, incorrect signed bit shifts (\ggg not \gg), severe multiplication overflow (which I took care of by scaling down the numbers by 2^7). The distance module ended up being over 600 lines of code, and having a testbench + working MATLAB script definitely helped a lot. I compared each stage in MATLAB to in Verilog, which helped me spot the bug more quickly. Below is a screenshot of the testbench I was debugging from.




Display

First, we displayed only the fixed objects in side view and top view: both cameras and the origin. We also decided to have the top view background be contrasting from the side view background so that the difference was more clear. Then, we added one degree of freedom in the side view and top view.

7 Challenges

Serial transmission of pixels

One challenge we ran into was early into the project, where we realized that we had to transmit the pixels serially to our computer in order to calibrate the camera. We forgot to take that into account when we were making our goals and plans, and due to many misinterpretations of the function of both the serial receiver and transmitter, implementing this took much longer than expected. For example, the camera image would not generate correctly when we tried to send two bytes of data at once (RGB padded with zeros). For the



python receiver we were using, we found out that the receiver actually spends an arbitrary amount of time between each stop bit and next start bit, causing the receiver to not get all pixels that were needed to regenerate the image.

Integration

Integration was a challenge because we had different, complex modules that we needed to work together. We worked together by having different branches on Github and merging them to make sure we could identify what changes needed to be made. There was also sometimes a problem with integrating different modules because of problems with IP modules. We sometimes had to regenerate the IP modules manually in Vivado when merging. In addition, to see what was wrong with the integration, after resolving the merge conflicts we simply had to generate the bitstream and see if the system had the desired functionality. The more and more components we had made the bitstream take longer and longer which caused long debugging sessions.

Object detection

Binarization: To achieve binarization, we use hue thresholding to assign a range of colors to one and others to zero. We had to be precise when choosing hue boundaries to make sure the right part of the camera image (e.g. , the ball), is displayed. Unfortunately, even after hue thresholding, other objects of seemingly dissimilar color were detected by the camera as well. To address this problem, we stuck with green and blue as the colors our system could detect.

Erosion: Initially we thought of using a 3x3 kernel to erode the image's noise. However, after some contemplation, we decided that delay required to use such a kernel would not be worth the decline in potential quality. We therefore decided on using a one dimensional kernel. We began with a 3x1 kernel and increased the length to see if noise could be reduced even further. We found that a 7x1 kernel was effective for our purposes.

Dilation: It was found that the switch system created for debugging was a problem. Dilation had already been achieved, but confusion regarding the switches prevented us from knowing so until much later. Some flickering noise showed up as a result of the module's output - this was a problem so we decided that we did not need it. Also, we were only trying to find the centroid, so dilation is not that relevant.

Centroid: Calculating the centroid proved to be very difficult. For the first iterations of the module, the square that represented the module on the VGA display stayed at 0,0 for some reason. We were not sure if the problem had to do with overloading or a lack of appropriate calculations. Eventually, we found the issue was with the lack of use of hcount and vcount in the module, as well as faulty pipelining in the top level module. In addition,

we struggled with the divider modules that were provided, for we were unfamiliar with the signals that they put out.

Distance Calculation

Figuring out the math behind the distance calculation was definitely challenging. There were many points of confusion, but it first took me a while to fully understand the pinhole camera model and how we can utilize projective geometry to pinpoint where the object was (without using the Fundamental matrix, which is used in most of the epipolar geometry resources I found). After that, I was stuck for a while since I could not think of a way to solve for the closest point between the two 3D lines or even understand how to compute these 3D lines. I also thought that the origin would have to be between the two cameras for the longest time. It turned out that the MATLAB app actually set the checkerboard as the real world origin - however, there was not any documentation on this so it took many tests to figure that out.

Another big challenge was converting all these calculations from MATLAB (from the script I wrote, located at the very bottom of this report) to Verilog, since I had to deal with both very small and large numbers. I first decided to use floating points, but then got frustrated by the tediousness and also expensive IP modules. After a couple stages, I decided to start over and use fixed points instead, which worked out in the end. However, it was very painstaking to look through every stage and try to debug what could have gone wrong.


8 Design Decisions

➤ Two cameras vs. ultrasonic sensors

Upon reflection and consultation with the staff, we decided to use dual cameras rather than an array of ultrasonic sensors because we thought that the ultrasonic sensors would be too noisy and not precise enough. Additionally, we thought it would be interesting to implement image processing concepts such as HSV thresholding, erosion, and dilation.

➤ Known object of known shape and color (green ball)

Originally, our goal was to use colored golf balls as our objects of interest to detect the centroid and calculate the distance. We planned to use golf balls because they are spherical and of a known radius. However, as we refined our distance calculation technique, we



realized that we didn't need to account for the radius of the object and simply knowing the centroid was sufficient so any object that could be easily detected was sufficient. As we tested our HSV thresholding module, we found that if we thresholded based solely on hue, we could not detect the golf balls. Thus, we decided to switch the object of interest to colored LEDs which would be easier to pick up with our thresholding and were especially useful because using LEDs helped decrease the difference in testing based on time of day because the value was less important.

➤ **Using servo to expand field of view**

We decided to mount our stereoscopic cameras on a dual-axis servo system. Our original idea was to be able to use the servo to increase the field of view of the system so that if the object of interest appeared in one of the cameras' field of view the servo would rotate the system such that the object of interest would appear in both the cameras' fields of view. We also wanted to use the servo for our stretch goal of tracking the object of interest and following it.

➤ **Object detection**

The most significant design decision regarding the image processing portion was the creation of an intermediate top level module (`object_detection`) that would bundle all the image processing modules into one package. This module allowed us to easily input the data from the image processing modules and use them in other modules. Additionally, the intermediate top_level made pipelining its submodules quite easy. Having a top level to encompass all the submodules.

Each submodule in `object_detection.sv` included features with specific aims as well. The hue thresholding module employed the use of switches that determine what thresholds we use. Based on those switches, we compare the value of our pixel in HSV to the threshold we have. Based on the thresholds, outputs a 1 or 0 (1 if in the threshold, 0 otherwise). For the erosion and dilation modules, we decided to take advantage of the fact that the camera feeds in only one pixel at a time to the FPGA by only having really small registers that save the bare minimum need to conduct the process needed. We also chose to work with one dimensional kernels to minimize delay and reduce the memory needed. The centroid detection module was created with many outputs to make extracting data much easier at the top level.

➤ Fixing 'jitter-ness' of centroid detection

Since we are updating the centroid in real time, it was not a very stable value and that led to a worse visual representation and also a less accurate distance. We solved this problem by averaging every 16 values of the x and y coordinates of the centroid before sending them as inputs to the display and the distance calculation.

➤ Calibrating Closer to Cameras

In the interest of time and less high quality cameras, we decided to move our checkerboard from ~6 feet away to ~4 feet away. We decided to make this design decision because having a far calibration means that when we are projecting our lines to that plane, there is a much greater chance of having large errors. Also, this method also gave more accurate intrinsic matrices for fewer pictures.

➤ Simplifying our Real World Domain

We decided to simplify our real world coordinates by keeping the real world origin at where the checkerboard's origin was. We also solved extrinsic and intrinsic parameters for when the checkerboard was perpendicular to the ground such that the z stayed constant for the checkerboard plane. By simplifying this, we now have a fixed plane and can project points onto that plane. Since we have our 2D to 3D back projection, we can solve the 3D vectors that will help us find the distance of the target.


9 Reflections

Two Camera Integration

The main task for integrating the camera into the other parts of this project was to take the code given to us by Joe and to make it work for two cameras. Joe's starter code helped this process go smoothly. From there, integrating the camera data from the BRAM to feed into the other modules was not too bad. It was perhaps a little cumbersome to have to declare most modules twice in order to feed in the camera feeds separately but the modularity of other modules, such as `display_select`, made this process easier.

Servo

Writing the servo code for the pulse width modulation was slow going at first especially because the details on our specific servos were not easily accessible and we could not find a data sheet correlating the pulse widths to a change in angle for our servo. We eventually



used a module that translated a position (not necessarily an angle) into a pulse width for our servo and we integrated this module by writing a `servo_wrapper` module that would send commands to sweep the servo from left to right, which was one of our commitment goals. In general, we met our technical goal of getting the servo to move left and right but it would be interesting to use the servo to complete the tracking goal. We wrote an FSM for the servo that was meant to stop the servo once it had the object of interest in view of both cameras but we ended up not using it in our final edition of the project.

Object detection

Most of the image processing was relatively simple to achieve, such as the hue thresholding, erosion, and dilation modules. However, as mentioned earlier, we hit many snares in writing a module that could calculate the centroid of an object, such as pipelining the correct values and the use of `hcount` and `vcount`.

Display

Building a display for the user to take advantage of was not too difficult. Thanks to the many hours of testing done for the object detection modules, we were very well acquainted with how the VGA display functions. The neat organization of the cameras at the top level made integrating the cameras to the display a very smooth process as well..

Camera Calibration

The calibration took longer than expected. The first problem we ran into was actually transferring the images from the cameras to my laptop, since I needed to upload them into the MATLAB script to retrieve the calibration parameters. I believe that was more complex than initially thought, and my solution was a FSM with 10 states where we sent R, G, and B separately and padded with four zeros. Another difficult aspect about the calibration was to fully understand the math behind these parameters. It took me a while to connect that the Rotation and Translation matrices are only mapping towards one specific instance of the checkerboard, which would set our real world origin. Once I got through that obstacle, the rest of the calibration was smooth sailing.

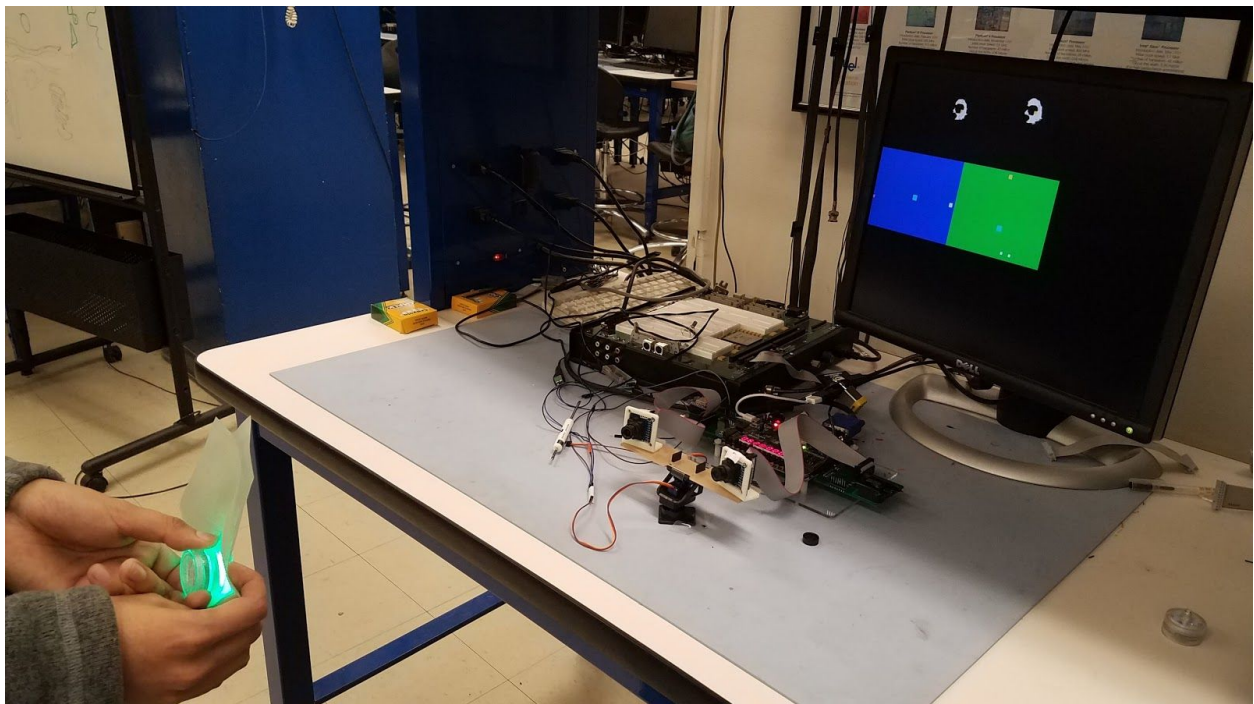
Distance Calculation

The distance calculation was definitely the complexity factor in this project. It took a while to figure out how exactly the projections worked when mapping from 2D to 3D and vice versa. I eventually figured out with Victor's help that we are essentially mapping from the detected centroid in the 2D image to the plane of the checkerboard instance defined by our extrinsic parameters. With these two points, we can form two 3D vectors and therefore find approximately where the target lies. One strange aspect was the scaling -- it was off by a large factor. When we go back to this project in January 2020, I plan to investigate this error further. I suspect, however, that it is a multiplication overflow problem since most of my errors in the distance module have been related to overflow.

10 Conclusion

Overall, we reached both our commitment and expected goals. We managed to detect targets of multiple colors through the two cameras on the servo, display the target's distance on the hex display, and display the object's relative distance in the environment through the side and top views.

Some aspects we wish to expand on in the future are to improve thresholding within object detection, integrate the servo FSM, add a tracking mechanism, and incorporate the two axes of the servo. Our team plans on continuing this project over IAP 2020 by taking 6.S186: FPGA Digital Design Competition.



11 Acknowledgements

We would like to acknowledge the following for their mentorship in this project:

- Victor Tom for his assistance in the camera calibration and projective geometry behind the distance calculation

- Diana Wofk for her extensive help on debugging our project and always supporting us!
- Joe Steinmeyer for his extremely timely responses to any question we had
- Gim Hom for his assistance in the final stretch of our project
- Mike Wang for his mentorship for our project in our final stretch

Appendix A - Verilog Code

Leilani:

Camera_wrapper.sv

```
module camera_wrapper(  
    input clk_65mhz,  
    input j0,  
    input j1,  
    input j2,  
    input [7:0] ju,  
    output logic [15:0] output_pixels,  
    output logic valid_pixel,  
    output logic jclk,  
    output logic pclk_in,  
    output logic frame_done_out  
);  
  
    logic xclk;  
    logic[1:0] xclk_count;  
    logic pclk_buff; //, pclk_in;  
    logic vsync_buff, vsync_in;  
    logic href_buff, href_in;  
    logic[7:0] pixel_buff, pixel_in;  
  
    assign xclk = (xclk_count >2'b01);  
    assign jclk = xclk;
```



```
always_ff @(posedge clk_65mhz) begin
    xclk_count <= xclk_count + 2'b01;
    pclk_buff <= j0;
    pclk_in <= pclk_buff;
    vsync_buff <= j1;
    vsync_in <= vsync_buff;
    href_buff <= j2;
    href_in <= href_buff;
    pixel_buff <= ju;
    pixel_in <= pixel_buff;
end

camera_read my_camera(.p_clock_in(pclk_in),
    .vsync_in(vsync_in),
    .href_in(href_in),
    .p_data_in(pixel_in),
    .pixel_data_out(output_pixels),
    .pixel_valid_out(valid_pixel),
    .frame_done_out(frame_done_out));

endmodule //camera_wrapper
```

Top_level.sv

```
`timescale 1ns / 1ps
```

```
module top_level(
```

```
    input clk_100mhz,
```

```
    input[15:0] sw,
```

```
input btnc, btneu, btnl, btnr, btnd,
input [7:0] ja,
input [2:0] jb,
input [7:0] jc,
input [2:0] jd,
output jbclk,
output jdclk,
output logic jdfour,
output[3:0] vga_r,
output[3:0] vga_b,
output[3:0] vga_g,
output vga_hs,
output vga_vs,
output led16_b, led16_g, led16_r,
output led17_b, led17_g, led17_r,
output[15:0] led,
output ca, cb, cc, cd, ce, cf, cg, dp, // segments a-g, dp
output[7:0] an // Display location 0-7
);

logic clk_65mhz;

// create 65mhz system clock, happens to match 1024 x 768 XVGA timing

clk_wiz_lab3 clkdivider(.clk_in1(clk_100mhz), .clk_out1(clk_65mhz));

logic [31:0] selector;
logic [24:0] center_x;
logic [24:0] center_x2;
logic [24:0] center_x3;
logic [24:0] center_y;
logic [24:0] center_y2;
```

```
logic [24:0] center_y3;
logic [24:0] x_acc;
logic [24:0] x_acc2;
logic [24:0] x_acc3;
logic [24:0] y_acc;
logic [24:0] y_acc2;
logic [24:0] y_acc3;
logic [24:0] bit_count;
logic [24:0] bit_count2;
logic [24:0] bit_count3;

/////Display Initialization/////
wire [10:0] hcount; // pixel on current line
wire [9:0] vcount; // line number
wire hsync, vsync, blank;
wire [11:0] pixel;
wire [11:0] pixel2;
wire [11:0] pixel3;
reg [11:0] rgb;
xvga xvga1(.vclock_in(clk_65mhz),.hcount_out(hcount),.vcount_out(vcount),
    .hsync_out(hsync),.vsync_out(vsync),.blank_out(blank));

// btnc button is user reset
wire reset;
debounce db1(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnc),.clean_out(reset));

////////// DISTANCE //////////
logic signed [31:0] distance;
logic signed [31:0] world_x;
logic signed [31:0] world_y;
```

```
logic signed [31:0] world_z;
logic [4:0] jitter_counter = 5'd0;
logic [28:0] jitter_accumulator_x1 = 29'd0;
logic [28:0] jitter_accumulator_y1 = 29'd0;
logic [28:0] jitter_accumulator_x2 = 29'd0;
logic [28:0] jitter_accumulator_y2 = 29'd0;

logic [24:0] nice_centroid_x1;
logic [24:0] nice_centroid_y1;
logic [24:0] nice_centroid_x2;
logic [24:0] nice_centroid_y2;

always_ff@(posedge clk_65mhz)begin
    if(jitter_counter == 15)begin
        nice_centroid_x1 <= jitter_accumulator_x1 >> 4;
        nice_centroid_y1 <= jitter_accumulator_y1 >> 4;

        nice_centroid_x2 <= jitter_accumulator_x2 >> 4;
        nice_centroid_y2 <= jitter_accumulator_y2 >> 4;

        jitter_counter <= 0;

        jitter_accumulator_x1 <= 0;
        jitter_accumulator_y1 <= 0;
        jitter_accumulator_x2 <= 0;
        jitter_accumulator_y2 <= 0;
    end else begin

        jitter_counter <= jitter_counter + 1;

        jitter_accumulator_x1 <= jitter_accumulator_x1 + center_x;
```

```
jitter_accumulator_y1 <= jitter_accumulator_y1 + center_y;

jitter_accumulator_x2 <= jitter_accumulator_x2 + center_x2;
jitter_accumulator_y2 <= jitter_accumulator_y2 + center_y2;

end
end

// logic [15:0] filtered_distance;
// always_ff@(posedge clk_65mhz)begin
//   if(distance[31:16] < 16'hB00)begin
//     filtered_distance <= distance[31:16];
//   end
// end

distance my_distance(
    // inputs
    .clk_in(clk_65mhz),
    .rst_in(reset),
//   .start(trigger),
    .x1(nice_centroid_x1),
    .y1(nice_centroid_y1),
    .x2(nice_centroid_x2),
    .y2(nice_centroid_y2),
//   .servo_angle(servo_angle),
    // outputs
    .distance(distance),
    .world_x(world_x),
    .world_y(world_y),
    .world_z(world_z)
);
```

```

logic [31:0] selected;

// always_ff@(posedge clk_65mhz)begin
//   if(sw[2])begin
//     selected <= {world_x[31:0]};
//   end else if(sw[1])begin
//     selected <= {world_y[31:0]};
//   end else if(sw[0])begin
//     selected <= {world_z[31:0]};
//   end else begin
//     selected <= {distance[31:16], 16'b0};
//   end
// end

assign selected = {distance[31:16], 16'b0}; // change this later
// assign selected = {nice_centroid_x1[7:0], nice_centroid_y1[7:0], nice_centroid_x2[7:0],
// nice_centroid_y2[7:0]};
// logic [31:0] sel_set;
// logic [26:0] counter_boi = 27'b0;
// always_ff @(posedge clk_65mhz) begin
//   if (counter_boi == 27'd100000000) begin
//     sel_set <= selected;
//     counter_boi <= 27'b0;
//   end else counter_boi <= counter_boi + 27'd1;

// end

seven_seg_controller my_controller(
    .clk_in(clk_65mhz),.rst_in(reset),
    .val_in(selected),
    .cat_out({cg, cf, ce, cd, cc, cb, ca}),
    .an_out(an)
);

```

```
////////////////////////////////////CAMERA_1////////////////////////////////////
```

```
logic [11:0] cam1;
```

```
logic pclk_in;
```

```
logic [11:0] frame_buff_out;
```

```
logic [15:0] output_pixels;
```

```
logic [15:0] old_output_pixels;
```

```
logic [12:0] processed_pixels;
```

```
logic valid_pixel;
```

```
logic frame_done_out;
```

```
logic she_valid;
```

```
assign she_valid = valid_pixel & ~sw[7];
```

```
logic [16:0] pixel_addr_in;
```

```
logic [16:0] pixel_addr_out;
```

```
blk_mem_gen_0 jojos_bram(.addra(pixel_addr_in), //take a pic based on switch and
```

```
    .clka(pclk_in),
```

```
    .dina(processed_pixels),
```

```
    .wea(she_valid),
```

```
    .addrb(pixel_addr_out),
```

```
    .clkb(clk_65mhz),
```

```
    .doutb(frame_buff_out));
```

```
always_ff @(posedge pclk_in)begin
```

```
    if (frame_done_out)begin
```

```
        pixel_addr_in <= 17'b0;
```

```
    end else if (valid_pixel)begin
```

```

        pixel_addr_in <= pixel_addr_in +1;
    end
end

always_ff @(posedge clk_65mhz) begin
    old_output_pixels <= output_pixels;
    processed_pixels = {output_pixels[15:12],output_pixels[10:7],output_pixels[4:1]};
end

assign pixel_addr_out = hcount+vcount*32'd320;
// assign cam1 = ((hcount<320) && (vcount<240))?frame_buff_out:12'h000;
assign cam1 = frame_buff_out;

camera_wrapper my_wrap(
    .clk_65mhz(clk_65mhz),
    .j0(jd[0]), .j1(jd[1]), .j2(jd[2]), //WAS JB
    .ju(jc), //WAS JA
    .output_pixels(output_pixels),
    .valid_pixel(valid_pixel),
    .jclk(jdclk),
    .pclk_in(pclk_in),
    .frame_done_out(frame_done_out));

////////////////////////////////////CAMERA_2////////////////////////////////////

logic [11:0] cam2;
logic pclk_in2;
logic [11:0] frame_buff_out2;
logic [15:0] output_pixels2;
logic [15:0] old_output_pixels2;

```



```
logic [12:0] processed_pixels2;
logic valid_pixel2;
logic frame_done_out2;

logic she_valid2;
assign she_valid2 = valid_pixel2 & ~sw[7];

logic [16:0] pixel_addr_in2;
logic [16:0] pixel_addr_out2;

blk_mem_gen_1 leileis_bram(.addra(pixel_addr_in2), //take a pic based on switch and
    .clka(pclk_in2),
    .dina(processed_pixels2),
    .wea(she_valid2),
    .addrb(pixel_addr_out2),
    .clkb(clk_65mhz),
    .doutb(frame_buff_out2));

always_ff @(posedge pclk_in2)begin
    if (frame_done_out2)begin
        pixel_addr_in2 <= 17'b0;
    end else if (valid_pixel2)begin
        pixel_addr_in2 <= pixel_addr_in2 +1;
    end
end

always_ff @(posedge clk_65mhz) begin
    old_output_pixels2 <= output_pixels2;
    processed_pixels2 = {output_pixels2[15:12],output_pixels2[10:7],output_pixels2[4:1]};
end
```

```
assign pixel_addr_out2 = hcount+vcount*32'd320;
assign cam2 = frame_buff_out2;

camera_wrapper my_wrap2(
    .clk_65mhz(clk_65mhz),
    .j0(jb[0]), .j1(jb[1]), .j2(jb[2]),
    .ju(ja),
    .output_pixels(output_pixels2),
    .valid_pixel(valid_pixel2),
    .jclk(jbclk),
    .pclk_in(pclk_in2),
    .frame_done_out(frame_done_out2));

//////////end CAMERA_2//////////

////Camera 1 and 2 fusion on display

logic [11:0] cam;

wire phsync,pvsync,pblank;

logic clk_200mhz;
logic clk_200mhz2;
logic clk_200mhz3;
logic inbound1;
logic inbound2;
assign inbound1 = hcount > 11'd20 && hcount < 11'd340 && vcount < 10'd240;
```

```
assign inbound2 = hcount > 11'd340 && hcount < 11'd660 && vcount < 10'd240;

display_select ds(.vclock_in(clk_65mhz),
//      .selectors(sw[15:14]),
      .processing(sw[13:10]),
      .pixel_in(cam1),

      .inBounds(inbound1),
      .hcount_in(hcount),
      .vcount_in(vcount),
      .hsync_in(hsync),
      .vsync_in(vsync),
      .blank_in(blank),
      .world_x(world_x),
      .world_y(world_y),
      .world_z(world_z),
      .distance(distance[31:16]),
      .phsync_out(phsync),
      .pvsync_out(pvsync),
      .pblank_out(pblank),
      .pixel_out(pixel),
      .clk_200mhz(clk_200mhz),
      .center_x(center_x),
      .center_y(center_y),
      .x_acc(x_acc),
      .y_acc(y_acc),
      .bit_count(bit_count));

display_select ds2(.vclock_in(clk_65mhz),
//      .selectors(sw[15:14]),
      .processing(sw[13:10]),
```

```
.pixel_in(cam2),
.inBounds(inbound2),
.hcount_in(hcount),
.vcount_in(vcount),
.hsycn_in(hsync),
.vsync_in(vsync),
.blank_in(blank),
.world_x(world_x),
.world_y(world_y),
.world_z(world_z),
.distance(distance[31:16]),
.phsync_out(phsync),
.pvsync_out(pvsync),
.pblank_out(pblank),
.pixel_out(pixel2),
.clk_200mhz(clk_200mhz2),
.center_x(center_x2),
.center_y(center_y2),
.x_acc(x_acc2),
.y_acc(y_acc2),
.bit_count(bit_count2));
```

```
logic cam3; //not a real camera
```

```
display_select ds3(
```

```
.vclock_in(clk_65mhz),    // 65MHz clock
.hcount_in(hcount), // horizontal index of current pixel (0..1023)
.vcount_in(vcount), // vertical index of current pixel (0..767)
.hsycn_in(hsync),    // XVGA horizontal sync signal (active low)
.vsync_in(vsync),    // XVGA vertical sync signal (active low)
.blank_in(blank),    // XVGA blanking (1 means output black pixel)
.world_x(world_x),
.world_y(world_y),
```

```

.world_z(world_z),
.distance(distance[31:16]),
// .selectors(sw[15:14]), // selects between normal or processed image
.processing(sw[13:10]), // selects which kind of process is being done
.pixel_in(cam3),
.inBounds(vcount >= 10'd240),
.phsync_out(phsync), // pong game's horizontal sync
.pvsync_out(pvsync), // pong game's vertical sync
.pblank_out(pblank), // pong game's blanking
.pixel_out(pixel3), // pong game's pixel // r=11:8, g=7:4, b=3:0
.clk_200mhz(clk_200mhz3),
.center_x(center_x3), ///testing centroid_x
.center_y(center_y3),
.x_acc(x_acc3),
.y_acc(y_acc3),
.bit_count(bit_count3));

wire border = (hcount==0 | hcount==1023 | vcount==0 | vcount==767 |
              hcount == 512 | vcount == 384);

reg b,hs,vs;
always_ff @(posedge clk_65mhz) begin
  if (sw[1:0] == 2'b01) begin
    // 1 pixel outline of visible area (white)
    hs <= hsync;
    vs <= vsync;
    b <= blank;
    rgb <= {12{border}};
  end else if (sw[1:0] == 2'b10) begin
    // color bars
    hs <= hsync;
    vs <= vsync;
  end
end

```

```

    b <= blank;
    rgb <= {{4{hcount[8]}}, {4{hcount[7]}}, {4{hcount[6]}}} ;
end else begin
    // default: pong
    hs <= phsync;
    vs <= pvsync;
    b <= pblank;
    //rgb <= pixel;
    if ((hcount<320) && (vcount<240)) cam <= pixel; //left camera display
    else if ((hcount > 320) && (vcount<240) && (hcount < 641)) cam <= pixel2; //right camera display
//    else if ((hcount<320) && (vcount>240) && (vcount<480)) cam <= pixel; //eroded left camera
    else if (vcount>240 && (vcount<480)) cam <= pixel3; //the blobs
    else cam <= 12'h000;
    rgb <= cam;
end
end

// the following lines are required for the Nexys4 VGA circuit - do not change
assign vga_r = ~b ? rgb[11:8] : 0;
assign vga_g = ~b ? rgb[7:4] : 0;
assign vga_b = ~b ? rgb[3:0] : 0;

assign vga_hs = ~hs;
assign vga_vs = ~vs;

servo_wrapper myservo(.clk(clk_200mhz), .js(jdfour));
endmodule

////////////////////////////////////
//
// display-select: wrapper for object detection

```

```

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

module display_select (
    input vclock_in,    // 65MHz clock
    input [10:0] hcount_in, // horizontal index of current pixel (0..1023)
    input [9:0] vcount_in, // vertical index of current pixel (0..767)
    input hsync_in,    // XVGA horizontal sync signal (active low)
    input vsync_in,    // XVGA vertical sync signal (active low)
    input blank_in,    // XVGA blanking (1 means output black pixel)
// input [1:0] selectors, // selects between normal or processed image
    input [3:0] processing, // selects which kind of process is being done
    input [11:0] pixel_in,
    input inBounds,
    input signed [31:0] world_x,
    input signed [31:0] world_y,
    input signed [31:0] world_z,
    input [15:0] distance,
    output logic phsync_out,    // pong game's horizontal sync
    output logic pvsync_out,    // pong game's vertical sync
    output logic pblank_out,    // pong game's blanking
    output logic [11:0] pixel_out, // pong game's pixel // r=11:8, g=7:4, b=3:0
    output logic clk_200mhz,

    output [24:0] center_x, ///testing centroid_x
    output [24:0] center_y,
    output logic [24:0] x_acc,
    output logic [24:0] y_acc,
    output logic [24:0] bit_count
);

```

```
//centroid stuff
logic [11:0] centroid;
logic [11:0] pixel_outta;

logic [24:0] centroid_x;

logic [24:0] centroid_y;

logic [24:0] valid_center_x;
logic [24:0] valid_center_y;

assign center_x = centroid_x;
assign center_y = centroid_y;
//end centroid stuff

//changing pixel to make it useful for ob det/////

logic [23:0] pixxel_in;
assign pixxel_in = {pixel_in[11:8], 4'b0, pixel_in[7:4], 4'b0, pixel_in[3:0], 4'b0};

always_comb begin
    if(centroid_x > 25'd319)begin
        valid_center_x = centroid_x;
    end
    if(centroid_y > 25'd239)begin
        valid_center_y = centroid_y;
    end
end

/////object_detection///
// logic [11:0] process_pixel;
```



```
object_detection ob_det(.clk(vclock_in),
    .hcount(hcount_in),
    .vcount(vcount_in),
    .inBounds(inBounds),
    .dilate(processing[1]),
    .erode(processing[0]),
    .thresholds(processing[3:2]),
    .pixel_in(pixel_in),
    .centroid_x(centroid_x),
    .centroid_y(centroid_y),
//    .pixel_out(process_pixel),
    .pixel_out(pixel_outta),
    .x_acc(x_acc),
    .y_acc(y_acc),
    .bit_count(bit_count));

assign phsync_out = hsync_in;
assign pvsync_out = vsync_in;
assign pblank_out = blank_in;

picture_blob dulcecito(.pixel_clk_in(vclock_in),
    .x_in(11'd200),
    .hcount_in(hcount_in),
    .y_in(10'd200),
    .vcount_in(vcount_in),
    .pixel_in(pixel_in),
//    .original(selectors[1]),
//    .processed(selectors[0]),
    .process_selects(processing),
//    .pixel_out(pixel_outta),
```

```
.clk_260mhz(clk_200mhz));

// logic [24:0] centroid_x_in;
// logic [24:0] centroid_y_in;

blob #(.WIDTH(16),.HEIGHT(16),.COLOR(12'hFF0)) // yellow!

the_centroid(.pixel_clk_in(vclock_in),
             .hcount_in(hcount_in),
             .vcount_in(vcount_in),
             .centroid_x(centroid_x),
             .centroid_y(centroid_y),
//             .centroid_x(centroid_x_in),
//             .centroid_y(centroid_y_in),
//             .original(selectors[1]),
//             .processed(selectors[0]),
//             .process_selects(processing),
             .pixel_out(centroid));

// SIDE VIEW
// z bounds from camera: [-2083, 0]
// -y bounds from camera: 185 -> 0
logic [15:0] side_ball_x;
// logic signed [31:0] side_ball_y;

// logic signed [31:0] top_ball_x;
logic [15:0] top_ball_y;

// assign side_ball_x = (-world_z[30:16]*320) >>> 11; // change denom
// assign side_ball_y = (240*300 - world_y[31:16]*240) >>> 9;
```

```

// TOP VIEW
// x bounds from camera: 24, 292, 0
// z bounds from camera: -2083, -2075, 0 --> [2083,0

//  assign top_ball_x = (world_x[31:16] * 320 + 320*300) >>> 9 ;
//  assign top_ball_y = (-world_z[31:16] * 240) >>> 11;

always_ff@(posedge vclock_in)begin
    top_ball_y <= ((distance - 16'h600) ) * 240 >> 6;
    side_ball_x <= ((distance - 16'h600) << 2);
end

///interesting math to map distance module values to pixels/////
logic [11:0] side_view;
blob #(.WIDTH(320), .HEIGHT(240), .COLOR(12'h00F))
    side_v(.pixel_clk_in(vclock_in), .hcount_in(hcount_in), .vcount_in(vcount_in),
        .centroid_x(25'd0), .centroid_y(25'd240), .pixel_out(side_view));

logic [11:0] top_view;
blob #(.WIDTH(320), .HEIGHT(240), .COLOR(12'h0F0))
    top_v(.pixel_clk_in(vclock_in), .hcount_in(hcount_in), .vcount_in(vcount_in),
        .centroid_x(25'd320), .centroid_y(25'd240), .pixel_out(top_view));

logic [11:0] side_or;
blob #(.WIDTH(12), .HEIGHT(12), .COLOR(12'hFF0))
    side_origin(.pixel_clk_in(vclock_in), .hcount_in(hcount_in), .vcount_in(vcount_in),
        .centroid_x(25'd300), .centroid_y(25'd360), .pixel_out(side_or));

logic [11:0] side_cam;
blob #(.WIDTH(8), .HEIGHT(8), .COLOR(12'hF00))
    side_camera(.pixel_clk_in(vclock_in), .hcount_in(hcount_in), .vcount_in(vcount_in),

```

```

        .centroid_x(25'd0), .centroid_y(25'd387), .pixel_out(side_cam));

logic [11:0] side_obj;
blob #(.WIDTH(16), .HEIGHT(16), .COLOR(12'h0F0))
    side_object(.pixel_clk_in(vclock_in), .hcount_in(hcount_in), .vcount_in(vcount_in),
//    .centroid_x(side_ball_x[24:0]), .centroid_y(side_ball_y[24:0] + 25'd240), .pixel_out(side_obj));
    .centroid_x({9'b0, side_ball_x}), .centroid_y(25'd120 + 25'd240), .pixel_out(side_obj));

logic [11:0] top_or;
blob #(.WIDTH(12), .HEIGHT(12), .COLOR(12'hF00))
    top_origin(.pixel_clk_in(vclock_in), .hcount_in(hcount_in), .vcount_in(vcount_in),
    .centroid_x(25'd480), .centroid_y(25'd250), .pixel_out(top_or));

logic [11:0] top_cam1;
blob #(.WIDTH(8), .HEIGHT(8), .COLOR(12'hF0F))
    top_camL(.pixel_clk_in(vclock_in), .hcount_in(hcount_in), .vcount_in(vcount_in),
    .centroid_x(25'd510), .centroid_y(25'd458), .pixel_out(top_cam1));

logic [11:0] top_cam2;
blob #(.WIDTH(8), .HEIGHT(8), .COLOR(12'hF0F))
    top_camR(.pixel_clk_in(vclock_in), .hcount_in(hcount_in), .vcount_in(vcount_in),
    .centroid_x(25'd537), .centroid_y(25'd459), .pixel_out(top_cam2));

logic [11:0] top_obj;
blob #(.WIDTH(16), .HEIGHT(16), .COLOR(12'h00F))
    top_object(.pixel_clk_in(vclock_in), .hcount_in(hcount_in), .vcount_in(vcount_in),
//    .centroid_x(top_ball_x[24:0] + 25'd320), .centroid_y(top_ball_y[24:0] + 25'd240),
    .pixel_out(top_obj));
    .centroid_x(25'd160 + 25'd320), .centroid_y({9'b0, top_ball_y} + 25'd240), .pixel_out(top_obj));

// always_ff @(posedge vclock_in) begin

```

```

//  if (selectors == 2'b10) begin
//      if ((hcount_in >= centroid_x_in && hcount_in <= centroid_x_in + 25'd16)
//          && (vcount_in >= centroid_y_in && vcount_in <= centroid_y_in + 25'd16)) begin
//          pixel_outta <= 12'd0;
//      end else pixel_outta <= process_pixel;
//      centroid_x_in <= centroid_x;
//      centroid_y_in <= centroid_y;
//  end else if (selectors == 2'b01) begin
//      if ((hcount_in >= centroid_x_in && hcount_in <= centroid_x_in + 25'd16)
//          && (vcount_in >= centroid_y_in && vcount_in <= centroid_y_in + 25'd16)) begin
//          pixel_outta <= 12'd0;
//      end else pixel_outta <= pixel_in;
//      centroid_x_in <= centroid_x - 25'd20;
//      centroid_y_in <= centroid_y - 25'd20;
//  end
//  end
//  end

```

```

assign pixel_out = centroid + pixel_outta + side_cam +
side_or + side_obj + top_or + top_cam1 + top_cam2 + top_obj + top_view + side_view;

```

```
endmodule
```

```

////////////////////////////////////
//
// picture_blob: displays the image manipu40.88lated
//
////////////////////////////////////

```

```

module picture_blob
    #(parameter WIDTH = 320,    // default picture width
        HEIGHT = 240) // default picture height
    (input pixel_clk_in,
     input [10:0] x_in,hcount_in,
     input [9:0] y_in,vcount_in,
     input [11:0] pixel_in,
     // input original, //selection of original or processed image
     // input processed,
     input [3:0] process_selects, // allows us to see erosion and dilation, and to choose hue thresholds
     //output logic [11:0] pixel_out,
     output logic clk_260mhz);

    clk_wiz_0 clkmulti(.clk_in1(pixel_clk_in), .clk_out1(clk_260mhz));

endmodule

/////////BLOB/////////

module blob
    #(parameter WIDTH = 64,      // default width: 64 pixels
        HEIGHT = 64,           // default height: 64 pixels
        COLOR = 12'hFFF) // default color: white
    (input pixel_clk_in,
     input [10:0] hcount_in,
     input [9:0] vcount_in,
     input [24:0] centroid_x,
     input [24:0] centroid_y,

```

```

// input original, //selection of original or processed image
// input processed,
//input [3:0] process_selects, // allows us to see erosion and dilation, and to choose hue
thresholds
output logic [11:0] pixel_out);

logic clk_200mhz;
clk_wiz_0 clkmulti(.clk_in1(pixel_clk_in), .clk_out1(clk_200mhz));

always_ff @(posedge pixel_clk_in) begin
    if ((hcount_in >= centroid_x && hcount_in < (centroid_x+WIDTH)) &&
        (vcount_in >= centroid_y && vcount_in < (centroid_y+HEIGHT))) begin
        pixel_out <= COLOR;
    end else begin
        pixel_out <= 0;
    end
end

endmodule

/////////ENDBLOB/////////

/////////
//
// Pushbutton Debounce Module (video version - 24 bits)
//
/////////

module debounce (input reset_in, clock_in, noisy_in,
                 output reg clean_out);

```

```

reg [19:0] count;
reg new_input;

always_ff @(posedge clock_in)
  if (reset_in) begin
    new_input <= noisy_in;
    clean_out <= noisy_in;
    count <= 0; end
  else if (noisy_in != new_input) begin new_input<=noisy_in; count <= 0; end
  else if (count == 650000) clean_out <= new_input;
  else count <= count+1;

endmodule

```

```

/////////////////////////////////////////////////////////////////
// Update: 8/8/2019 GH
// Create Date: 10/02/2015 02:05:19 AM
// Module Name: xvga
//
// xvga: Generate VGA display signals (1024 x 768 @ 60Hz)
//
//          ---- HORIZONTAL ----   -----VERTICAL -----
//          Active          Active
//          Freq  Video  FP Sync  BP   Video  FP Sync  BP
// 640x480, 60Hz  25.175  640   16  96  48   480  11  2  31
// 800x600, 60Hz  40.000  800   40 128  88   600   1  4  23
// 1024x768, 60Hz 65.000 1024  24 136 160   768   3  6  29
// 1280x1024, 60Hz 108.00 1280  48 112 248   768   1  3  38
// 1280x720p 60Hz  75.25  1280  72  80 216   720   3  5  30
// 1920x1080 60Hz 148.5  1920  88  44 148  1080  4  5  36

```



```

//
// change the clock frequency, front porches, sync's, and back porches to create
// other screen resolutions
////////////////////////////////////

module xvga(input vclock_in,
            output reg [10:0] hcount_out, // pixel number on current line
            output reg [9:0] vcount_out, // line number
            output reg vsync_out, hsync_out,
            output reg blank_out);

parameter DISPLAY_WIDTH = 1024; // display width
parameter DISPLAY_HEIGHT = 768; // number of lines
parameter H_FP = 24; // horizontal front porch
parameter H_SYNC_PULSE = 136; // horizontal sync
parameter H_BP = 160; // horizontal back porch
parameter V_FP = 3; // vertical front porch
parameter V_SYNC_PULSE = 6; // vertical sync
parameter V_BP = 29; // vertical back porch
// horizontal: 1344 pixels total
// display 1024 pixels per line

reg hblank,vblank;
wire hsyncon,hsyncoff,hreset,hblankon;
assign hblankon = (hcount_out == (DISPLAY_WIDTH - 1));
assign hsyncon = (hcount_out == (DISPLAY_WIDTH + H_FP - 1)); //1047
assign hsyncoff = (hcount_out == (DISPLAY_WIDTH + H_FP + H_SYNC_PULSE - 1)); // 1183
assign hreset = (hcount_out == (DISPLAY_WIDTH + H_FP + H_SYNC_PULSE + H_BP - 1)); //1343

// vertical: 806 lines total
// display 768 lines

```

```

wire vsyncon,vsyncoff,vreset,vblankon;
assign vblankon = hreset & (vcount_out == (DISPLAY_HEIGHT - 1)); // 767
assign vsyncon = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP - 1)); // 771
assign vsyncoff = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP + V_SYNC_PULSE - 1)); // 777
assign vreset = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP + V_SYNC_PULSE + V_BP - 1)); //
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_ff @(posedge vclock_in) begin
    hcount_out <= hreset ? 0 : hcount_out + 1;
    hblank <= next_hblank;
    hsync_out <= hsyncon ? 0 : hsyncoff ? 1 : hsync_out; // active low
    vcount_out <= hreset ? (vreset ? 0 : vcount_out + 1) : vcount_out;
    vblank <= next_vblank;
    vsync_out <= vsyncon ? 0 : vsyncoff ? 1 : vsync_out; // active low
    blank_out <= next_vblank | (next_hblank & ~hreset);
end
endmodule

```

Camera_read.sv

```

module camera_read(
    input p_clock_in,
    input vsync_in,
    input href_in,
    input [7:0] p_data_in,
    output logic [15:0] pixel_data_out,
    output logic pixel_valid_out,
    output logic frame_done_out
);

```

```
logic [1:0] FSM_state = 0;
logic pixel_half = 0;

localparam WAIT_FRAME_START = 0;
localparam ROW_CAPTURE = 1;

always_ff@(posedge p_clock_in)
begin
case(FSM_state)

WAIT_FRAME_START: begin //wait for VSYNC
    FSM_state <= (!vsync_in) ? ROW_CAPTURE : WAIT_FRAME_START;
    frame_done_out <= 0;
    pixel_half <= 0;
end

ROW_CAPTURE: begin
    FSM_state <= vsync_in ? WAIT_FRAME_START : ROW_CAPTURE;
    frame_done_out <= vsync_in ? 1 : 0;
    pixel_valid_out <= (href_in && pixel_half) ? 1 : 0;
    if (href_in) begin
        pixel_half <= ~ pixel_half;
        if (pixel_half) pixel_data_out[7:0] <= p_data_in;
        else pixel_data_out[15:8] <= p_data_in;
    end
end
endcase
end

endmodule
```

Servo_controller.sv

```
/*
servo controller
Based on code from Mojo tutorial
https://embeddedmicro.com/tutorials/mojo/servos

Takes an 8-bit position as an input
Output a single pwm signal with period of ~20ms
Pulse width = 1ms -> 2ms full scale. 1.5ms is center position
*/
module servo_controller (
    input clk,
    input rst,
    input [7:0] position,
    output servo
);

reg pwm_q, pwm_d;
reg [19:0] ctr_q, ctr_d;
assign servo = pwm_q;
//position (0-255) maps to 50,000-100,000 (which corresponds to 1ms-2ms @ 50MHz)
//this is approximately (position+165)<<8
//The servo output is set by comparing the position input with the value of the counter (ctr_q)
always @(*) begin
    ctr_d = ctr_q + 1'b1;
    if (position + 9'd165 > ctr_q[19:8]) begin
        pwm_d = 1'b1;
    end else begin
        pwm_d = 1'b0;
    end
end
```

```
end

always @(posedge clk) begin
  if (rst) begin
    ctr_q <= 1'b0;
  end else begin
    ctr_q <= ctr_d;
  end
  pwm_q <= pwm_d;
end
endmodule
```

Servo_wrapper.sv

```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////
//
// Updated 8/10/2019 Lab 3
// Updated 8/12/2018 V2.lab5c
// Create Date: 10/1/2015 V1.0
// Design Name:
// Module Name: labkit
//
/////////////////////////////////////////////////////////////////

module servo_wrapper(
  input clk, //200mhz
  output logic js //outputting the serial here
);

//  logic clk_65mhz;
//  create 65mhz system clock, happens to match 1024 x 768 XGA timing
```

```
// clk_wiz_lab3 clkdivider(.clk_in1(clk_100mhz), .clk_out1(clk_65mhz));
```

```
logic [7:0] myang = 8'd255;
```

```
logic clk_50mhz = 1'b0;
```

```
logic [2:0] county = 3'b0;
```

```
always_ff @(posedge clk) begin
```

```
    if (county == 3'd4) begin
```

```
        county <= 1'b0;
```

```
        clk_50mhz <= 1'b1;
```

```
    end else begin
```

```
        county <= county + 4'd1;
```

```
        clk_50mhz <= 1'b0;
```

```
    end
```

```
end
```

```
// clk_wiz_0 clkmulti(.clk_in1(clk_100mhz), .clk_out1(clk_50mhz));
```

```
//code to make servo sweep
```

```
logic [27:0] hz_count = 28'b0;
```

```
logic mydir = 0; //direction of the servo, 0 is left, 1 is right
```

```
always_ff @(posedge clk) begin
```

```
    if (hz_count == 28'd100000000) begin
```

```
        hz_count <= 28'b0;
```

```
        if (myang >= 8'd250) mydir <= 0;
```

```
        else if (myang <= 8'd5) mydir <= 1;
```

```
        if (mydir == 0) myang <= myang-28'd5;
        else if (mydir == 1) myang <= myang +28'd5;
    end else begin
        hz_count <= hz_count+28'd1;
    end
end
end

servo_controller mysc(.clk(clk_50mhz),
                    .rst(1'b0),
                    .position(myang),
                    .servo(js));

// servo my_servo(.clk(clk_100mhz), .angle(myang), .servo_pulse(jc[0]));
// always_ff @(posedge clk_65mhz) begin
// end
endmodule
```

Fsm.sv

```
module fsm(input clk,
           input rst,
           input btnleft,
           input btnright,
           input btncalc,
           output logic servo_dir,
           output logic servo_stop);

    //define the possible states
    logic [1:0] IDLE = 2'b00;
    logic [1:0] LEFTMOVE = 2'b01;
    logic [1:0] RIGHTMOVE = 2'b10;
```

```
logic [1:0] CALC = 2'b11;
logic [1:0] state = 2'b00;

always_ff @(posedge clk) begin
    if (rst) state <= IDLE;
    else begin
        case (state)
            IDLE: begin
                if (btnleft == 1'b1) state <= LEFTMOVE;
                else if (btnright == 1'b1) state <= RIGHTMOVE;
                else if (btncalc == 1'b1) state <= CALC;
                servo_stop <= 1'b1;
            end

            LEFTMOVE: begin
                //do stuff to actually move the servo left
                servo_stop <= 1'b0;
                servo_dir <= 1'b0;

                if (btnleft == 1'b0 && btnright == 1'b0) state <= IDLE;
                else if (btnright == 1'b1) state <= RIGHTMOVE;
            end

            RIGHTMOVE: begin
                //do stuff to actually move the servo right
                servo_stop <= 1'b0;
                servo_dir <= 1'b1;

                if (btnright == 1'b0 && btnleft == 1'b0) state <= IDLE;
                else if (btnleft == 1'b1) state <= LEFTMOVE;
            end
        endcase
    end
end
```



```

        CALC: begin
            //make sure the servo is stopped
            servo_stop <= 1'b1;
            //display the centroid
            //display the distance
            if (rst == 1'b1) state <= IDLE;
        end
        default begin
            state <= IDLE;
        end
    endcase
end
end //end of always block
endmodule //end of fsm

```

Jeana:

Serial_tx.sv

```

module serial_tx( input    clk_in,
                 input    rst_in,
                 input    trigger_in,
                 input [7:0] val_in,
                 output logic done,
                 output logic data_out);

    parameter DIVISOR = 564; // to account for 65 mhz clock 1/ baud rate * clock rate

    logic [9:0]    shift_buffer; //10 bits...interesting
    logic [31:0]   count;
    logic [3:0]    count_bits;

    always_ff @(posedge clk_in)begin

```

```

if(rst_in)begin
    count <= 32'd0; //reset count
    shift_buffer <= 10'b1111111111; // reset to 1
    done <= 0;
    count_bits <= 0;
end else begin
    if(trigger_in)begin
        shift_buffer <= {1'b1, val_in, 1'b0}; // prepend val_in with 0 and append with 1, store in
shift_buffer
        count <= DIVISOR;
        count_bits <= 0;
        done <= 0;
    end else begin
        if(count_bits == 11)begin
            count_bits <=0;
            done <= 1;
        end else if(count==DIVISOR)begin // if count == DIVISOR
            done <= 0;
            count <= 32'd0; //reset count
            data_out <= shift_buffer[0]; // send least significant bit first
            shift_buffer <= {1'b1, shift_buffer[9:1]}; // shift to right, pad the unused bits as HIGH
            count_bits <= count_bits + 1;
        end else begin
            done <= 0;
            count <= count + 1; // else keep counting
        end
    end
end
end
end
endmodule //serial_tx

```

Serial transmission state machine (in an alternative version of top_level.sv):

```
//////////////////////////////// SERIAL SENDING //////////////////////////////////
```

```
logic send_serial;
logic trigger_in = 0;
logic done;
logic [7:0] rgb_input;
logic out1;

debounce
send_debounce(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnl),.clean_out(send_serial));

serial_tx my_tx(.clk_in(clk_65mhz), .rst_in(reset), .trigger_in(trigger_in),
               .val_in(rgb_input), .done(done), .data_out(out1));

logic send_serial2;
logic trigger_in2 = 0;
logic done2;
logic [7:0] rgb_input2;
logic out2;

debounce
send_debounce2(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnr),.clean_out(send_serial2));

serial_tx my_tx2(.clk_in(clk_65mhz), .rst_in(reset), .trigger_in(trigger_in2),
                .val_in(rgb_input2), .done(done2), .data_out(out2));

// Serial FSM for Camera 1
parameter IDLE = 10'b1000000000;
parameter WAIT1 = 10'b0100000000;
parameter SEND_RED = 10'b0010000000;
parameter WAIT2 = 10'b0001000000;
parameter SEND_GREEN = 10'b0000100000;
```

```
parameter WAIT3 = 10'b0000010000;
parameter SEND_BLUE = 10'b0000001000;
parameter PASS1 = 10'b0000000100;
parameter PASS2 = 10'b0000000010;
parameter PASS3 = 10'b0000000001;
parameter LEFT = 2'b10;
parameter RIGHT = 2'b01;

parameter PIXELS = 'd76800; // 240 * 320 pixels

logic [9:0] state = IDLE;
logic [1:0] camera;

always_ff@(posedge clk_65mhz)begin
    if(reset)begin
        state <= IDLE;
    end else begin
        if(camera == LEFT)begin
            jdtx <= out1;
        end else if(camera == RIGHT)begin
            jdtx <= out2;
        end
        if(sw[15]) begin
            pixel_addr_out <= hcount+vcount*32'd320;
            pixel_addr_out2 <= hcount+vcount*32'd320;
        end else begin
            case (state)
                IDLE: begin
                    if(send_serial)begin
                        trigger_in <= 0;
                        pixel_addr_out <= 0;
                        camera <= LEFT;
                    end
                end
            endcase
        end
    end
end
```

```
        state <= PASS1;
    end

    if(send_serial2)begin
        trigger_in2 <= 0;
        pixel_addr_out2 <= 0;
        camera <= RIGHT;
        state <= PASS1;
    end
end

PASS1: begin
    if(camera == LEFT)begin
        trigger_in <= 0;
    end else if(camera == RIGHT)begin
        trigger_in2 <= 0;
    end

    state <= WAIT1;
end

WAIT1: begin
    if(camera == LEFT)begin
        trigger_in <= 1;
        rgb_input <= {4'b0000, frame_buff_out[11:8]};
    end else if(camera == RIGHT)begin
        trigger_in2 <= 1;
        rgb_input2 <= {4'b0000, frame_buff_out2[11:8]};
    end

    state <= SEND_RED;
end
```

```
SEND_RED: begin
  if(camera == LEFT)begin
    trigger_in <= 0;
    if(pixel_addr_out == PIXELS)begin
      state <= IDLE;
      pixel_addr_out <= 0;
      camera <= 0;
    end else if(done & ~send_serial)begin
      state <= PASS2;
    end
  end else if(camera == RIGHT)begin
    trigger_in2 <= 0;
    if(pixel_addr_out2 == PIXELS)begin
      state <= IDLE;
      pixel_addr_out2 <= 0;
      camera <= 0;
    end else if(done2 & ~send_serial2)begin
      state <= PASS2;
    end
  end
end

PASS2: begin
  if(camera == LEFT)begin
    trigger_in <= 0;
  end else if(camera == RIGHT)begin
    trigger_in2 <= 0;
  end
end

state <= WAIT2;
end
```

```
WAIT2: begin
  if(camera == LEFT)begin
    trigger_in <= 1;
    rgb_input <= {4'b0, frame_buff_out[7:4]};
  end else if(camera == RIGHT)begin
    trigger_in2 <= 1;
    rgb_input2 <= {4'b0, frame_buff_out2[7:4]};
  end

  state <= SEND_GREEN;
end

SEND_GREEN: begin
  if(camera == LEFT)begin
    trigger_in <= 0;
    if(pixel_addr_out == PIXELS)begin
      state <= IDLE;
      pixel_addr_out <= 0;
      camera <= 0;
    end else if(done & ~send_serial)begin
      state <= PASS3;
    end
  end else if(camera == RIGHT)begin
    trigger_in2 <= 0;
    if(pixel_addr_out2 == PIXELS)begin
      state <= IDLE;
      pixel_addr_out2 <= 0;
      camera <= 0;
    end else if(done2 & ~send_serial2)begin
      state <= PASS3;
    end
  end
end
```

```
    end
end

PASS3: begin
    if(camera == LEFT)begin
        trigger_in <= 0;
    end else if(camera == RIGHT)begin
        trigger_in2 <= 0;
    end

    state <= WAIT3;
end

WAIT3: begin
    if(camera == LEFT)begin
        trigger_in <= 1;
        rgb_input <= {4'b0, frame_buff_out[3:0]};
    end else if(camera == RIGHT)begin
        trigger_in2 <= 1;
        rgb_input2 <= {4'b0, frame_buff_out2[3:0]};
    end

    state <= SEND_BLUE;
end

SEND_BLUE: begin
    if(camera == LEFT)begin
        trigger_in <= 0;
        if(pixel_addr_out == PIXELS)begin
            state <= IDLE;
            pixel_addr_out <= 0;
            camera <= 0;
        end
    end
end
```



```

        end else if(done & ~send_serial)begin
            pixel_addr_out <= pixel_addr_out + 1;
            state <= PASS1;
        end
    end else if(camera == RIGHT)begin
        trigger_in2 <= 0;
        if(pixel_addr_out2 == PIXELS)begin
            state <= IDLE;
            pixel_addr_out2 <= 0;
            camera <= 0;
        end else if(done2 & ~send_serial2)begin
            pixel_addr_out2 <= pixel_addr_out2 + 1;
            state <= PASS1;
        end
    end
end
end
endcase
end
end
end
end

```

Distance.sv

```

module distance(
    input clk_in,
    input rst_in,
    input [24:0] x1, // make these 25 bits
    input [24:0] y1,
    input [24:0] x2,
    input [24:0] y2,
    // input [9:0] servo_angle,
    output logic signed [31:0] distance,

```

```

output logic signed [31:0] world_x,
output logic signed [31:0] world_y,
output logic signed [31:0] world_z
);

```

```

///// LOGIC to convert 25 bit unsigned int to 32 bit fixed point

```

```

logic signed [31:0] converted_x1;
logic signed [31:0] converted_y1;
logic signed [31:0] converted_x2;
logic signed [31:0] converted_y2;
assign converted_x1 = {x1[15:0], 16'b0};
assign converted_y1 = {y1[15:0], 16'b0};
assign converted_x2 = {x2[15:0], 16'b0};
assign converted_y2 = {y2[15:0], 16'b0};

```

```

// 16 bits to decimal: 2^16 = 65536

```

```

/* LEFT CAMERA */

```

```

//inverted camera 1 matrix

```

```

parameter signed P1_inv11 = {16'b0,          16'b0000_0001_1111_1000}; // 0.0077
parameter signed P1_inv12 = {16'b0,          16'b0000_0000_0000_0000}; //d64;
parameter signed P1_inv13 = {16'b0,          16'b0};
parameter signed P1_inv21 = {1'b0, 15'b0, 16'b000000000000001101}; //-'d59;
parameter signed P1_inv22 = {1'b0, 15'b0,      16'b0000000111101011}; //d2007;
parameter signed P1_inv23 = {1'b0, 15'b0, 16'b0};
parameter signed P1_inv31 = {1'b1, 15'b1111111111111110, 16'b0100100011110101};
parameter signed P1_inv32 = {1'b1, 15'b1111111111111110, 16'b1110111101000001};
parameter signed P1_inv33 = {1'b0, 15'b0,      16'b0000000010000011};

```

```

//real world coords of camera 1

```

```

parameter signed C1_1 = {1'b0, 15'b101001111, 16'b0}; //d292;

```

```
parameter signed C1_2 = {1'b0, 15'b1100001, 16'b0};//d73;
parameter signed C1_3 = {1'b1, 15'b111100101000101, 16'b1111_1111_1111_1111};//-d2075;


/* RIGHT CAMERA */

//inverted camera 2 matrix
parameter signed P2_inv11 = {1'b0, 15'b0, 16'b0000000111101011};//32'd1970;
parameter signed P2_inv12 = {1'b1, 15'b1111111111111111, 16'b1111_1111_1111_0010};
parameter signed P2_inv13 = 32'b0;
parameter signed P2_inv21 = {1'b0, 15'b0, 16'b000000000000001101};
parameter signed P2_inv22 = {1'b0, 15'b0, 16'b0000000111101011};
parameter signed P2_inv23 = 32'd0;
parameter signed P2_inv31 = {1'b1, 15'b1111111111111111, 16'b1010111001010101};
parameter signed P2_inv32 = {1'b1, 15'b1111111111111111, 16'b0000101001000011};
parameter signed P2_inv33 = {1'b0, 15'b0, 16'b0000000010010110};

//real world coords for camera 2
parameter signed C2_1 = {1'b0, 15'b10110010, 16'b0};//32'd24;
parameter signed C2_2 = {1'b0, 15'b10010001, 16'b0};
parameter signed C2_3 = {1'b1, 15'b111100100110111, 16'b1111_1111_1111_1111};//-32'd2083;

// STAGE 1
logic signed [31:0] s1_world1_x1;
logic signed [31:0] s1_world1_x2;
logic signed [31:0] s1_world1_x3;

logic signed [31:0] s1_world1_y1;
logic signed [31:0] s1_world1_y2;
logic signed [31:0] s1_world1_y3;
```



```
logic signed [31:0] s1_scaling_1 ;  
logic signed [31:0] s1_scaling_2;  
logic signed [31:0] s1_scaling_3;
```

```
logic signed [31:0] s2_world1_x1;  
logic signed [31:0] s2_world1_x2;  
logic signed [31:0] s2_world1_x3;
```

```
logic signed [31:0] s2_world1_y1;  
logic signed [31:0] s2_world1_y2;  
logic signed [31:0] s2_world1_y3;
```

```
logic signed [31:0] s2_scaling_1;  
logic signed [31:0] s2_scaling_2;  
logic signed [31:0] s2_scaling_3;
```

```
logic signed [63:0] s1_world1_x1_temp;  
logic signed [63:0] s1_world1_x2_temp;
```

```
logic signed [63:0] s1_world1_y1_temp;  
logic signed [63:0] s1_world1_y2_temp;
```

```
logic signed [63:0] s1_scaling_1_temp;  
logic signed [63:0] s1_scaling_2_temp;
```

```
logic signed [63:0] s2_world1_x1_temp;  
logic signed [63:0] s2_world1_x2_temp;
```

```
logic signed [63:0] s2_world1_y1_temp;  
logic signed [63:0] s2_world1_y2_temp;
```

```
logic signed [63:0] s2_scaling_1_temp;
logic signed [63:0] s2_scaling_2_temp;

/////////////////////////////////////////////////////////////////
// STAGE 2:
// Camera 1 world vector
logic signed [31:0] world1_x;
logic signed [31:0] world1_y;
logic signed [31:0] scaling1 = 1;

//Camera 2 world vector
logic signed [31:0] world2_x;
logic signed [31:0] world2_y;
logic signed [31:0] scaling2 = 1;
/////////////////////////////////////////////////////////////////
// STAGE 3: scaling
// Camera 1 world vector
logic signed [31:0] world1_x_scaled;
logic signed [31:0] world1_y_scaled;

//Camera 2 world vector
logic signed [31:0] world2_x_scaled;
logic signed [31:0] world2_y_scaled;

my_division world1x(
    .clk_in(clk_in),
    .dividend(world1_x),
    .divisor(scaling1),
    .valid_signal(1),
```

```
.quotient(world1_x_scaled)
);

my_division world1y(
    .clk_in(clk_in),
    .dividend(world1_y),
    .divisor(scaling1),
    .valid_signal(1),
    .quotient(world1_y_scaled)
);

my_division world2x(
    .clk_in(clk_in),
    .dividend(world2_x),
    .divisor(scaling2),
    .valid_signal(1),
    .quotient(world2_x_scaled)
);

my_division world2y(
    .clk_in(clk_in),
    .dividend(world2_y),
    .divisor(scaling2),
    .valid_signal(1),
    .quotient(world2_y_scaled)
);

/////////////////////////////////////////////////////////////////
// STAGE 4 u, v calculations
// Midpoint Calculation Variables
//Camera 1 unit vector
logic signed [31:0] u1;
logic signed [31:0] u2;
```

```
logic signed [31:0] u3;
```

```
logic signed [31:0] u1_delay;
```

```
logic signed [31:0] u2_delay;
```

```
logic signed [31:0] u3_delay;
```

```
logic signed [31:0] u1_delay2;
```

```
logic signed [31:0] u2_delay2;
```

```
logic signed [31:0] u3_delay2;
```

```
logic signed [31:0] u1_delay3;
```

```
logic signed [31:0] u2_delay3;
```

```
logic signed [31:0] u3_delay3;
```

```
logic signed [31:0] u1_delay4;
```

```
logic signed [31:0] u2_delay4;
```

```
logic signed [31:0] u3_delay4;
```

```
// Camera 2 unit vector
```

```
logic signed [31:0] v1;
```

```
logic signed [31:0] v2;
```

```
logic signed [31:0] v3;
```

```
logic signed [31:0] v1_delay;
```

```
logic signed [31:0] v2_delay;
```

```
logic signed [31:0] v3_delay;
```

```
logic signed [31:0] v1_delay2;
```

```
logic signed [31:0] v2_delay2;
```

```
logic signed [31:0] v3_delay2;
```

```
logic signed [31:0] v1_delay3;
```

```
logic signed [31:0] v2_delay3;
```

```
logic signed [31:0] v3_delay3;
```

```
logic signed [31:0] v1_delay4;
```

```
logic signed [31:0] v2_delay4;
```

```
logic signed [31:0] v3_delay4;
```

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

```
// STAGE 5: t_cpa breakdown
```

```
logic signed [31:0] t_subtraction_1;
```

```
logic signed [31:0] t_subtraction_2;
```

```
logic signed [31:0] t_subtraction_3;
```

```
logic signed [31:0] t_subtraction_4;
```

```
logic signed [31:0] t_subtraction_5;
```

```
logic signed [31:0] t_subtraction_6;
```

```
logic signed [31:0] t_multiplication_1;
```

```
logic signed [31:0] t_multiplication_2;
```

```
logic signed [31:0] t_multiplication_3;
```

```
logic signed [63:0] t_multiplication_1_temp;
```

```
logic signed [63:0] t_multiplication_2_temp;
```

```
logic signed [63:0] t_multiplication_3_temp;
```

```
logic signed [31:0] t_divisor_1;
```

```
logic signed [31:0] t_divisor_2;
```

```
logic signed [31:0] t_divisor_3;
```

```
logic signed [63:0] t_divisor_1_temp;
```

```
logic signed [63:0] t_divisor_2_temp;
```

```
logic signed [63:0] t_divisor_3_temp;
```



```
// STAGE 7
    logic signed [31:0] t_cpa_numerator;
    logic signed [31:0] t_cpa_denominator;

// STAGE 8
    // scaling factor of where the two lines come closest
    logic signed [31:0] t_cpa;
    my_division get_tcpa(
        .clk_in(clk_in),
        .dividend(t_cpa_numerator),
        .divisor(t_cpa_denominator),
        .valid_signal(1),
        .quotient(t_cpa)
    );
    ///////////////////////////////////////////////////////////////////

// STAGE 9
    logic signed [31:0] world_x_multiplication_1;
    logic signed [31:0] world_x_multiplication_2;

    logic signed [31:0] world_y_multiplication_1;
    logic signed [31:0] world_y_multiplication_2;

    logic signed [31:0] world_z_multiplication_1;
    logic signed [31:0] world_z_multiplication_2;

    logic signed [63:0] world_x_multiplication_1_temp;
    logic signed [63:0] world_x_multiplication_2_temp;

    logic signed [63:0] world_y_multiplication_1_temp;
    logic signed [63:0] world_y_multiplication_2_temp;

    logic signed [63:0] world_z_multiplication_1_temp;
```

```
logic signed [63:0] world_z_multiplication_2_temp;

/////////////////////////////////////////////////////////////////
// STAGE 10
logic signed [31:0] world_x_numerator;
logic signed [31:0] world_y_numerator;
logic signed [31:0] world_z_numerator;
/////////////////////////////////////////////////////////////////
logic signed [63:0] world_x_sq;
logic signed [63:0] world_y_sq;
logic signed [63:0] world_z_sq;

logic signed [63:0] world_x_sq_temp;
logic signed [63:0] world_y_sq_temp;
logic signed [63:0] world_z_sq_temp;

// my_division worldz(
//   .clk_in(clk_in),
//   .dividend(world_z_numerator),
//   .divisor({1'b0, 13'b0, 1'b1, 17'b0}),
//   .valid_signal(1),
//   .quotient(world_z)
//   );

/////////////////////////////////////////////////////////////////
// FINAL STAGE:
logic [63:0] squared_distance;

my_sqrt sqrt_function(
  .clk_in(clk_in),
  .valid_signal(1),
  .squared_distance(squared_distance),
```

```
.distance_output(distance)
);
////////////////////////////////////////////////////////////////

always_comb begin
    s1_world1_x1 = s1_world1_x1_temp[47:16];
    s1_world1_x2 = s1_world1_x2_temp[47:16];

    s1_world1_y1 = s1_world1_y1_temp[47:16];
    s1_world1_y2 = s1_world1_y2_temp[47:16];

    s1_scaling_1 = s1_scaling_1_temp[47:16];
    s1_scaling_2 = s1_scaling_2_temp[47:16];

    s2_world1_x1 = s2_world1_x1_temp[47:16];
    s2_world1_x2 = s2_world1_x2_temp[47:16];

    s2_world1_y1 = s2_world1_y1_temp[47:16];
    s2_world1_y2 = s2_world1_y2_temp[47:16];

    s2_scaling_1 = s2_scaling_1_temp[47:16];
    s2_scaling_2 = s2_scaling_2_temp[47:16];

    t_multiplication_1 = t_multiplication_1_temp[47 + 7:16 + 7];
    t_multiplication_2 = t_multiplication_2_temp[47 + 7:16 + 7];
    t_multiplication_3 = t_multiplication_3_temp[47 + 7:16 + 7];

    t_divisor_1 = t_divisor_1_temp[47 + 7:16 + 7];
    t_divisor_2 = t_divisor_2_temp[47 + 7:16 + 7];
    t_divisor_3 = t_divisor_3_temp[47 + 7:16 + 7];

    world_x_multiplication_1 = world_x_multiplication_1_temp[47:16];
```

```
world_x_multiplication_2 = world_x_multiplication_2_temp[47:16];

world_y_multiplication_1 = world_y_multiplication_1_temp[47:16];
world_y_multiplication_2 = world_y_multiplication_2_temp[47:16];

world_z_multiplication_1 = world_z_multiplication_1_temp[47:16];
world_z_multiplication_2 = world_z_multiplication_2_temp[47:16];

// world_x_sq = world_x_sq_temp[47:16];
// world_y_sq = world_y_sq_temp[47:16];
// world_z_sq = world_z_sq_temp[47:16];
end

always_ff @(posedge clk_in)begin
    // STAGE 1 [x y 1]* P_inv to get world coord X, Y (scale by third value)
    s1_world1_x1_temp <= converted_x1*P1_inv11;
    s1_world1_x2_temp <= converted_y1*P1_inv21;
    s1_world1_x3 <= P1_inv31;

    s1_world1_y1_temp <= converted_x1*P1_inv12;
    s1_world1_y2_temp <= converted_y1*P1_inv22;
    s1_world1_y3 <= P1_inv32;

    s1_scaling_1_temp <= converted_x1*P1_inv13;
    s1_scaling_2_temp <= converted_y1*P1_inv23;
    s1_scaling_3 <= P1_inv33;

    s2_world1_x1_temp <= converted_x2*P2_inv11;
    s2_world1_x2_temp <= converted_y2*P2_inv21;
    s2_world1_x3 <= P2_inv31;
```

```
s2_world1_y1_temp <= converted_x2*P2_inv12;
s2_world1_y2_temp <= converted_y2*P2_inv22;
s2_world1_y3 <= P2_inv32;
```

```
s2_scaling_1_temp <= converted_x2*P2_inv13;
s2_scaling_2_temp <= converted_y2*P2_inv23;
s2_scaling_3 <= P2_inv33;
```

```
// STAGE 2
```

```
world1_x <= s1_world1_x1 + s1_world1_x2 + s1_world1_x3;
world1_y <= s1_world1_y1 + s1_world1_y2 + s1_world1_y3;
scaling1 <= s1_scaling_1 + s1_scaling_2 + s1_scaling_3;
```

```
world2_x <= s2_world1_x1 + s2_world1_x2 + s2_world1_x3;
world2_y <= s2_world1_y1 + s2_world1_y2 + s2_world1_y3;
scaling2 <= s2_scaling_1 + s2_scaling_2 + s2_scaling_3;
```

```
// STAGE 3
```

```
// divide both world1_x and world1_y by scaling1 = world1_x_scaled, world1_y_scaled
```

```
// divide both world2_x and world2_y by scaling2 = world2_x_scaled, world2_y_scaled
```

```
/*
```

```
world1_x_scaled <= world1_x / scaling1;
```

```
world1_y_scaled <= world1_y / scaling1;
```

```
world2_x_scaled <= world2_x / scaling2;
```

```
world2_y_scaled <= world2_y / scaling2;
```

```
*/
```

```
//STAGE 4
u1 <= world1_x_scaled - C1_1;
u2 <= world1_y_scaled - C1_2;
u3 <= - C1_3;

v1 <= world2_x_scaled - C2_1;
v2 <= world2_y_scaled - C2_2;
v3 <= - C2_3;

// STAGE 5
// t_cpa <= -( (C1_1 - C2_1)*(u1 - v1) + (C1_2 - C2_2)*(u2 - v2) + (C1_3 - C2_3)*(u3 - v3) ); // break
these up
// t_cpa <= (C1_1 - C2_1)*(v1 - u1) + (C1_2 - C2_2)*(v2 - u2) + (C1_3 - C2_3)*(v3 - u3) ;
t_subtraction_1 <= C1_1 - C2_1;
t_subtraction_2 <= v1 - u1;
t_subtraction_3 <= C1_2 - C2_2;
t_subtraction_4 <= v2 - u2;
t_subtraction_5 <= C1_3 - C2_3;
t_subtraction_6 <= v3 - u3;

u1_delay <= u1;
u2_delay <= u2;
u3_delay <= u3;
v1_delay <= v1;
v2_delay <= v2;
v3_delay <= v3;

// STAGE 6
t_multiplication_1_temp <= t_subtraction_1 * t_subtraction_2;
t_multiplication_2_temp <= t_subtraction_3 * t_subtraction_4;
```

```
t_multiplication_3_temp <= t_subtraction_5 * t_subtraction_6;

u1_delay2 <= u1_delay;
u2_delay2 <= u2_delay;
u3_delay2 <= u3_delay;
v1_delay2 <= v1_delay;
v2_delay2 <= v2_delay;
v3_delay2 <= v3_delay;

// divide t_cpa by ((u1 - v1)*(u1 - v1) + (u2 - v2)*(u2 - v2) + (u3 - v3)*(u3 - v3))
t_divisor_1_temp <= t_subtraction_2 * t_subtraction_2;
t_divisor_2_temp <= t_subtraction_4 * t_subtraction_4;
t_divisor_3_temp <= t_subtraction_6 * t_subtraction_6;

// STAGE 7
t_cpa_numerator <= t_multiplication_1 + t_multiplication_2 + t_multiplication_3;
t_cpa_denominator <= t_divisor_1 + t_divisor_2 + t_divisor_3;

u1_delay3 <= u1_delay2;
u2_delay3 <= u2_delay2;
u3_delay3 <= u3_delay2;
v1_delay3 <= v1_delay2;
v2_delay3 <= v2_delay2;
v3_delay3 <= v3_delay2;

// STAGE 8:
// t_cpa <= t_cpa_numerator / t_cpa_denominator;

u1_delay4 <= u1_delay3;
u2_delay4 <= u2_delay3;
```

```
u3_delay4 <= u3_delay3;
v1_delay4 <= v1_delay3;
v2_delay4 <= v2_delay3;
v3_delay4 <= v3_delay3;

// STAGE 9: individual multiplications
world_x_multiplication_1_temp <= t_cpa * u1_delay4;
world_x_multiplication_2_temp <= t_cpa * v1_delay4;

world_y_multiplication_1_temp <= t_cpa * u2_delay4;
world_y_multiplication_2_temp <= t_cpa * v2_delay4;

world_z_multiplication_1_temp <= t_cpa * u3_delay4;
world_z_multiplication_2_temp <= t_cpa * v3_delay4;

// STAGE 10: add to get world_x, world_y, and world_z
world_x_numerator <= C1_1 + world_x_multiplication_1 + C2_1 + world_x_multiplication_2; //
change these to delays,
world_y_numerator <= C1_2 + world_y_multiplication_1 + C2_2 + world_y_multiplication_2;
world_z_numerator <= C1_3 + world_z_multiplication_1 + C2_3 + world_z_multiplication_2;

// world_x <= (C1_1 + t_cpa*u1 + C2_1 + t_cpa*v1) >> 1;
// world_y <= (C1_2 + t_cpa*u2 + C2_2 + t_cpa*v2) >> 1;
// world_z <= (C1_3 + t_cpa*u3 + C2_3 + t_cpa*v3)16 >> 1;

// STAGE 11: divide world_x, world_y, and world_z by 2
world_x <= world_x_numerator >>> 1;
world_y <= world_y_numerator >>> 1;
world_z <= world_z_numerator >>> 1; // use divider
```



```
// STAGE 12
// do individual multiplications
world_x_sq <= world_x * world_x;
world_y_sq <= world_y * world_y;
world_z_sq <= world_z * world_z;

// STAGE 13
// add to get distance (change below to use _sq signals)
squared_distance <= world_x_sq + world_y_sq + world_z_sq;

// STAGE 14
// compute sqrt

end

endmodule

module my_division(
    input clk_in,
    input signed[31:0] dividend,
    input signed [31:0] divisor,
    input valid_signal,
    output signed [31:0] quotient
);

    logic signed [63:0] dividend_float64;
    logic signed [63:0] divisor_float64;
    logic signed [63:0] division_result;
    logic dividend_result_valid;
    logic divisor_result_valid;
    logic division_result_valid;
```

```
logic quotient_result_valid;
```

```
floating_point_0 conv1(  
    .aclk(clk_in),           //: IN STD_LOGIC;  
    .s_axis_a_tvalid(valid_signal), //: IN STD_LOGIC;  
    .s_axis_a_tready( ),     //: OUT STD_LOGIC;  
    .s_axis_a_tdata(dividend), //: IN STD_LOGIC_VECTOR(31 DOWNT0 0);  
    .m_axis_result_tvalid(dividend_result_valid), //: OUT STD_LOGIC;  
    .m_axis_result_tready(1), //: IN STD_LOGIC;  
    .m_axis_result_tdata(dividend_float64) //: OUT STD_LOGIC_VECTOR(63 DOWNT0 0)  
);
```

```
floating_point_0 conv2(  
    .aclk(clk_in),           //: IN STD_LOGIC;  
    .s_axis_a_tvalid(valid_signal), //: IN STD_LOGIC;  
    .s_axis_a_tready( ),     //: OUT STD_LOGIC;  
    .s_axis_a_tdata(divisor), //: IN STD_LOGIC_VECTOR(31 DOWNT0 0);  
    .m_axis_result_tvalid(divisor_result_valid), //: OUT STD_LOGIC;  
    .m_axis_result_tready(1), //: IN STD_LOGIC;  
    .m_axis_result_tdata(divisor_float64) //: OUT STD_LOGIC_VECTOR(63 DOWNT0 0)  
);
```

```
division div1(  
    .aclk(clk_in),  
    .s_axis_a_tvalid(dividend_result_valid), // IN STD_LOGIC;  
    .s_axis_a_tready( ), // OUT STD_LOGIC;  
    .s_axis_a_tdata(dividend_float64), // IN STD_LOGIC_VECTOR(63 DOWNT0 0);  
    .s_axis_b_tvalid(divisor_result_valid), // IN STD_LOGIC;  
    .s_axis_b_tready( ), // OUT STD_LOGIC;  
    .s_axis_b_tdata(divisor_float64), //: IN STD_LOGIC_VECTOR(63 DOWNT0 0);  
    .m_axis_result_tvalid(division_result_valid), //: OUT STD_LOGIC;
```

```

.m_axis_result_tready(1),          //: IN STD_LOGIC;
.m_axis_result_tdata(division_result) //: OUT STD_LOGIC_VECTOR(63 DOWNT0 0)
);

floating_point_1 convertBack(
  .aclk(clk_in),                    //: IN STD_LOGIC;
  .s_axis_a_tvalid(division_result_valid), //: IN STD_LOGIC;
  .s_axis_a_tready( ),              //: OUT STD_LOGIC;
  .s_axis_a_tdata(division_result), //: IN STD_LOGIC_VECTOR(31 DOWNT0 0);
  .m_axis_result_tvalid(quotient_result_valid), //: OUT STD_LOGIC;
  .m_axis_result_tready(1),         //: IN STD_LOGIC;
  .m_axis_result_tdata(quotient) //: OUT STD_LOGIC_VECTOR(63 DOWNT0 0)
);

endmodule

module my_sqrt(
  input clk_in,
  input valid_signal,
  input [63:0] squared_distance,
  output [31:0] distance_output
);
logic sqrt_result_valid;
logic actual_sqrt_result_valid;
logic [63:0] squared_distance_float64;
logic [63:0] sqrt_float64;

sqrt_conversion convert(
  .aclk(clk_in),                    //: IN STD_LOGIC;
  .s_axis_a_tvalid(valid_signal),    //: IN STD_LOGIC;
  .s_axis_a_tready( ),              //: OUT STD_LOGIC;
  .s_axis_a_tdata(squared_distance), //: IN STD_LOGIC_VECTOR(31 DOWNT0 0);

```

```

.m_axis_result_tvalid(sqrt_result_valid), //: OUT STD_LOGIC;
.m_axis_result_tready(1), //: IN STD_LOGIC;
.m_axis_result_tdata(squared_distance_float64) //: OUT STD_LOGIC_VECTOR(63 DOWNT0 0)
);

square_root my_sqrt(
.aclk(clk_in),
.s_axis_a_tvalid(sqrt_result_valid), //: IN STD_LOGIC;
.s_axis_a_tready( ), //: OUT STD_LOGIC;
.s_axis_a_tdata(squared_distance_float64), //: IN STD_LOGIC_VECTOR(31 DOWNT0 0);
.m_axis_result_tvalid(actual_sqrt_result_valid), //: OUT STD_LOGIC;
.m_axis_result_tready(1), //: IN STD_LOGIC;
.m_axis_result_tdata(sqrt_float64) //: OUT STD_LOGIC_VECTOR(63 DOWNT0 0)
);

logic final_result_valid;
floating_point_1 convertBack(
.aclk(clk_in), //: IN STD_LOGIC;
.s_axis_a_tvalid(actual_sqrt_result_valid), //: IN STD_LOGIC;
.s_axis_a_tready( ), //: OUT STD_LOGIC;
.s_axis_a_tdata(sqrt_float64), //: IN STD_LOGIC_VECTOR(31 DOWNT0 0);
.m_axis_result_tvalid(final_result_valid), //: OUT STD_LOGIC;
.m_axis_result_tready(1), //: IN STD_LOGIC;
.m_axis_result_tdata(distance_output) //: OUT STD_LOGIC_VECTOR(63 DOWNT0 0)
);

endmodule

module get_divisor(

```

```

input clk_in,
input [16:0] number,
output logic [16:0] divisor
);
logic [16:0] curr_num = number;
logic [16:0] temp_divisor = 'd1;

always_ff@(posedge clk_in)begin
    if(curr_num >> 2 == 0)begin
        if(number - temp_divisor < temp_divisor*2 - number)begin
            divisor <= temp_divisor;
        end else begin
            divisor <= temp_divisor * 2;
        end
        curr_num <= number;
        temp_divisor <= 1;
    end else begin
        curr_num <= curr_num >> 2;
        temp_divisor <= temp_divisor * 2;
    end
end

endmodule // get_divisor

```

Ryan:

Object_detection.sv

```

module object_detection (input clk,
                        input dilate,
                        input erode, //switch operated mechanism that allows us to see either erosion or
dilation---temporary testing input
                        input [1:0] thresholds, //switch-based thresholding alternators
                        input [23:0] pixel_in, //pixel that goes in

```

```
        input [10:0] hcount,
        input [9:0] vcount,
        input inBounds,
        output logic [15:0] centroid_x,
        output logic [15:0] centroid_y,
        output logic [11:0] pixel_out,
        output logic [24:0] x_acc,
        output logic [24:0] y_acc,
        output logic [24:0] bit_count
    );

    logic [23:0] pixel; //pixel that goes in module

    logic thresh_out; //bit leaving hue thresholding
    logic erosion_out; //bit leaving erosion
    logic dilation_out; //bit leaving dilation

    logic thresh_valid;
    logic erode_valid;
    logic dilate_valid;

    logic [7:0] hue; //hsv
    logic [7:0] sat;
    logic [7:0] val;
    logic hugh_valid;

    //for localizer
    logic frame_over;
    logic [15:0] averaging;
```

```

    rgb2hsv convert(.clock(clk), .reset(0), .r(pixel[23:16]), .g(pixel[15:8]), .b(pixel[7:0]), .h(hue), .s(sat),
    .v(val), .hue_valid(hugh_valid));

    hue_thresholding thresh(.clk(clk), .threshes(thresholds), .hue_val(hue), .isValid(hugh_valid),
    .thresh_bit(thresh_out), .valid(thresh_valid));

    erosion eroding(.clk(clk), .bit_in(thresh_out), .isValid(thresh_valid), .eroded_bit(erosion_out),
    .valid(erode_valid));

    dilation dilating(.clk(clk), .bit_in(erosion_out), .isValid(erode_valid), .dilated_bit(dilation_out),
    .valid(dilate_valid));

    localizer centroid(.clk(clk), .erode_bit(erosion_out), .isValid(erode_valid), .hcount(hcount),
    .vcount(vcount), .inBounds(inBounds), .x_center(centroid_x),
    .y_center(centroid_y), .frame_blink(frame_over), .x_accumulator(x_acc), .y_accumulator(y_acc),
    .bit_count(bit_count));
// centroid busqueda(.clock(clk), .reset(0), .x(hcount), .y(vcount), .green(erosion_out),
// .centroid_x(centroid_x[10:0]), .centroid_y(centroid_y[9:0]), .frame_done(frame_over),
    .averaging(averaging));

always_ff @(posedge clk) begin
    pixel <= pixel_in;
    if (hcount == 11'd319 && vcount == 10'd239) frame_over <= 1;
    else frame_over <= 0;
    if (dilate) begin
        if (dilation_out == 1'b1) begin
            pixel_out <= 12'b1111_1111_1111;
        end else if (dilation_out == 1'b0) begin
            pixel_out <= 12'b0000_0000_0000;
        end
    end else if (erode) begin
        if (erosion_out == 1'b1) begin
            pixel_out <= 12'b1111_1111_1111;
        end else if (erosion_out == 1'b0) begin
            pixel_out <= 12'b0000_0000_0000;
        end
    end else begin
        if (thresh_out == 1'b1) begin

```

```
        pixel_out <= 12'b1111_1111_1111;
    end else if (thresh_out == 1'b0) begin
        pixel_out <= 12'b0000_0000_0000;
    end
end
end
end

endmodule

module hue_thresholding (input clk,
                        input [1:0] threshes,
                        input [7:0] hue_val,
                        input isValid,
                        output logic thresh_bit,
                        output logic valid
                        );

    always_ff @(posedge clk) begin
        if (isValid) begin
            if (threshes == 2'b00) begin
                if (hue_val > 8'd172 || hue_val < 8'd165) begin
                    thresh_bit <= 0;
                    valid <= 1;
                end else begin
                    thresh_bit <= 1;
                    valid <= 1;
                end
            end
            end else if (threshes == 2'b01) begin
                if (hue_val > 8'd87 || hue_val < 8'd82) begin
                    thresh_bit <= 0;
                    valid <= 1;
                end
            end
        end
    end
end
```



```
        end else begin
            thresh_bit <= 1;
            valid <= 1;
        end
    end else if (threshes == 2'b10) begin
        if (hue_val > 8'd87 || hue_val < 8'd82) begin
            thresh_bit <= 0;
            valid <= 1;
        end else begin
            thresh_bit <= 1;
            valid <= 1;
        end
    end
end else valid <= 0;
end
endmodule

module erosion(input clk,
               input bit_in,
               input isValid,
               output logic eroded_bit,
               output logic valid
);

    logic [9:0] ycounter;
    logic [9:0] xcounter;
    logic [321:0] kernel_workspace; //workspace register
    logic erosion_trigger; //tells the module when to start eroding

    initial begin
        valid = 0;
```

```
xcounter = 10'd1;
ycounter = 10'd0;
erosion_trigger = 1'b0;
kernel_workspace[321] = 1'b1;
kernel_workspace[0] = 1'b1;
end

logic kernel; // result of erosion
assign kernel = kernel_workspace[xcounter-9] && kernel_workspace[xcounter-8] &&
kernel_workspace[xcounter-7] && kernel_workspace[xcounter-6] &&
kernel_workspace[xcounter-5] && kernel_workspace[xcounter-4] &&
kernel_workspace[xcounter-3] && kernel_workspace[xcounter-2] &&
kernel_workspace[xcounter-1];

always_ff @(posedge clk) begin
if (isValid) begin
if (ycounter == 10'd239) begin
ycounter <= 10'd0;
end else if (xcounter == 10'd320) begin
xcounter <= 10'd1;
ycounter <= ycounter + 1'd1;
end else if (xcounter < 10'd320) begin
xcounter <= xcounter + 1'd1;

end if (xcounter == 10'd8) begin
erosion_trigger <= 1;
end if (erosion_trigger) begin
eroded_bit <= kernel;
valid <= 1;
end

kernel_workspace[xcounter] <= bit_in;
```

```
end else valid <= 0;
end
endmodule

module dilation(input clk,
    input bit_in,
    input isValid,
    output logic dilated_bit,
    output logic valid
);

    logic [9:0] xcount;
    logic [9:0] ycount;
    logic [321:0] kernel_workspace; //workspace register

    logic dilation_trigger; //tells the module when to start dilating

    logic kernel;
    assign kernel = kernel_workspace[xcount-5] || kernel_workspace[xcount-4] ||
kernel_workspace[xcount-3]
    || kernel_workspace[xcount-2] || kernel_workspace[xcount-1];

    initial begin
        valid = 0;
        xcount = 10'd1;
        ycount = 10'd0;
        kernel_workspace[0] = 1'b1;
        kernel_workspace[321] = 1'b1;
        dilation_trigger = 1'b0;
    end

    always_ff @(posedge clk) begin
```

```
if (isValid) begin
    if (ycount == 10'd239) begin
        ycount <= 10'd0;
    end else if (xcount == 10'd320) begin
        xcount <= 10'd1;
        ycount <= ycount + 1'd1;
    end else if (xcount < 10'd320) begin
        xcount <= xcount + 1;
    end

    end if (xcount == 10'd4) begin
        dilation_trigger <= 1;
    end if (dilation_trigger) begin
        dilated_bit <= kernel;
        valid <= 1;
    end
end

    kernel_workspace[xcount] <= bit_in;
end else valid <= 0;
end
endmodule

module localizer( input clk,
                 input erode_bit,
                 input isValid,
                 input frame_blink,
                 input [10:0] hcount,
                 input [9:0] vcount,
                 input inBounds,
                 output logic [15:0] x_center,
                 output logic [15:0] y_center,
                 output logic [24:0] x_accumulator,
                 output logic [24:0] y_accumulator,
```

```
        output logic [24:0] bit_count
    );

    //logic [24:0] x_accumulator; // accumulates all values of x
    //logic [24:0] y_accumulator; //and y
    //logic [24:0] bit_count; // counts number of bits that enter the stream
    //logic [4:0] bit_shift;
    //logic div_start; //tells the divider to start dividing
    logic a_frame_passed; //true the cycle after frame_blink?

    //for division
    //  logic [24:0] y_total;
    //  logic [24:0] x_total;

    logic [63:0] x_center_BIG;
    logic [63:0] y_center_BIG;
    logic add_break;
    logic [9:0] break_counter;
    //  assign x_center = x_center_BIG[47:32];
    //  assign y_center = y_center_BIG[47:32];

    initial begin
        add_break = 0;
        break_counter = 0;
        x_center_BIG = 64'h1111_0011_1111_1111;
        y_center_BIG = 64'd100;
        x_accumulator = 25'd100;
        y_accumulator = 25'd100;
        bit_count = 25'd0;
        //div_start = 0;
        a_frame_passed = 0;
    end
```

```

// divider #(.WIDTH(25)) ydivide(.clk(clk), .start(div_start), .dividend(y_accumulator),
// .divider(bit_count), .quotient(y_center));

// divider #(.WIDTH(25)) xdivide(.clk(clk), .start(div_start), .dividend(x_accumulator),
// .divider(bit_count), .quotient(x_center));

logic xcenter_valid;
logic ycenter_valid;
logic [24:0] x_feed;
logic [24:0] y_feed;

divvy_itup xcenter (.aclk(clk),
    .s_axis_divisor_tdata({7'b0000_000, bit_count}),
    .s_axis_divisor_tvalid(a_frame_passed),
    .s_axis_dividend_tdata({7'b0000_000, x_feed}),
    .s_axis_dividend_tvalid(a_frame_passed),
    .m_axis_dout_tdata({x_center_BIG}),
    .m_axis_dout_tvalid(xcenter_valid));

divvy_itup ycenter (.aclk(clk),
    .s_axis_divisor_tdata({7'b0000_000, bit_count}),
    .s_axis_divisor_tvalid(a_frame_passed),
    .s_axis_dividend_tdata({7'b0000_000, y_feed}),
    .s_axis_dividend_tvalid(a_frame_passed),
    .m_axis_dout_tdata({y_center_BIG}),
    .m_axis_dout_tvalid(ycenter_valid));

always_ff @(posedge clk) begin
    if (xcenter_valid) x_center <= x_center_BIG[47:32];
    if (ycenter_valid) y_center <= y_center_BIG[47:32];
end

```

```
if (frame_blink) begin //start centroid calculation
    x_feed <= x_accumulator;
    y_feed <= y_accumulator;
    a_frame_passed <= 1;
    //add_break <= 1;
    //div_start <= 1;
end else if (!frame_blink && a_frame_passed) begin
    bit_count <= 0;
    //div_start <= 0;
    a_frame_passed <= 0;
    y_accumulator <= 25'd0;
    x_accumulator <= 25'd0;
end else if (inBounds && !add_break && erode_bit) begin
    y_accumulator <= y_accumulator + vcount;
    x_accumulator <= x_accumulator + hcount;
//    if (break_counter == 10'd100) begin
//        add_break <= 1;
//    end
    bit_count <= bit_count + 25'd1;
end
end
endmodule

//inBounds && erode_bit && isValid

////////////////////////////////////
// Company:
// Engineer: Kevin Zheng Class of 2012
// Dept of Electrical Engineering & Computer Science
//
// Create Date: 18:45:01 11/10/2010
```

```
// Design Name:
// Module Name:  rgb2hsv
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////
module rgb2hsv(clock, reset, r, g, b, h, s, v, hue_valid);
    input wire clock;
    input wire reset;
    input wire [7:0] r;
    input wire [7:0] g;
    input wire [7:0] b;
    output reg [7:0] h;
    output reg [7:0] s;
    output reg [7:0] v;
    output reg hue_valid;
    reg [7:0] my_r_delay1, my_g_delay1, my_b_delay1;
    reg [7:0] my_r_delay2, my_g_delay2, my_b_delay2;
    reg [7:0] my_r, my_g, my_b;
    reg [7:0] min, max, delta;
    reg [15:0] s_top;
    reg [15:0] s_bottom;
    reg [15:0] h_top;
    reg [15:0] h_bottom;
```



```
wire [15:0] s_quotient;
wire [15:0] s_remainder;
wire s_rfd;
wire [15:0] h_quotient;
wire [15:0] h_remainder;
wire h_rfd;
reg [7:0] v_delay [19:0];
reg [18:0] h_negative;
reg [15:0] h_add [18:0];
reg [4:0] i;
// Clocks 4-18: perform all the divisions
//the s_divider (16/16) has delay 18
//the hue_div (16/16) has delay 18

divider hue_div1(
    .clk(clock),
    .dividend(s_top),
    .divider(s_bottom),
    .quotient(s_quotient),
// note: the "fractional" output was originally named "remainder" in this
// file -- it seems coregen will name this output "fractional" even if
// you didn't select the remainder type as fractional.
    .remainder(s_remainder),
    .ready(s_rfd)
);
divider hue_div2(
    .clk(clock),
    .dividend(h_top),
    .divider(h_bottom),
    .quotient(h_quotient),
    .remainder(h_remainder),
    .ready(h_rfd)
```

```

);
always_ff @ (posedge clock) begin

    // Clock 1: latch the inputs (always positive)
    {my_r, my_g, my_b} <= {r, g, b};

    // Clock 2: compute min, max
    {my_r_delay1, my_g_delay1, my_b_delay1} <= {my_r, my_g, my_b};

    if((my_r >= my_g) && (my_r >= my_b)) //(B,S,S)
        max <= my_r;
    else if((my_g >= my_r) && (my_g >= my_b)) //(S,B,S)
        max <= my_g;
    else    max <= my_b;

    if((my_r <= my_g) && (my_r <= my_b)) //(S,B,B)
        min <= my_r;
    else if((my_g <= my_r) && (my_g <= my_b)) //(B,S,B)
        min <= my_g;
    else
        min <= my_b;

    // Clock 3: compute the delta
    {my_r_delay2, my_g_delay2, my_b_delay2} <= {my_r_delay1, my_g_delay1,
my_b_delay1};

    v_delay[0] <= max;
    delta <= max - min;

    // Clock 4: compute the top and bottom of whatever divisions we need to do
    s_top <= 8'd255 * delta;
    s_bottom <= (v_delay[0]>0)?{8'd0, v_delay[0]}: 16'd1;

```

```

        if(my_r_delay2 == v_delay[0]) begin
            h_top <= (my_g_delay2 >= my_b_delay2)?(my_g_delay2 - my_b_delay2)
* 8'd255:(my_b_delay2 - my_g_delay2) * 8'd255;
            h_negative[0] <= (my_g_delay2 >= my_b_delay2)?0:1;
            h_add[0] <= 16'd0;
        end
        else if(my_g_delay2 == v_delay[0]) begin
            h_top <= (my_b_delay2 >= my_r_delay2)?(my_b_delay2 - my_r_delay2)
* 8'd255:(my_r_delay2 - my_b_delay2) * 8'd255;
            h_negative[0] <= (my_b_delay2 >= my_r_delay2)?0:1;
            h_add[0] <= 16'd85;
        end
        else if(my_b_delay2 == v_delay[0]) begin
            h_top <= (my_r_delay2 >= my_g_delay2)?(my_r_delay2 - my_g_delay2)
* 8'd255:(my_g_delay2 - my_r_delay2) * 8'd255;
            h_negative[0] <= (my_r_delay2 >= my_g_delay2)?0:1;
            h_add[0] <= 16'd170;
        end

        h_bottom <= (delta > 0)?delta * 8'd6:16'd6;

        //delay the v and h_negative signals 18 times
        for(i=1; i<19; i=i+1) begin
            v_delay[i] <= v_delay[i-1];
            h_negative[i] <= h_negative[i-1];
            h_add[i] <= h_add[i-1];
        end

        v_delay[19] <= v_delay[18];
        //Clock 22: compute the final value of h

```

```

//depending on the value of h_delay[18], we need to subtract 255 from it to
make it come back around the circle

```

```

    if(h_negative[18] && (h_quotient > h_add[18])) begin
        h <= 8'd255 - h_quotient[7:0] + h_add[18];
        hue_valid <= 1;
    end
    else if(h_negative[18]) begin
        h <= h_add[18] - h_quotient[7:0];
        hue_valid <= 1;
    end
    else if(!h_negative[18]) begin
        h <= h_quotient[7:0] + h_add[18];
        hue_valid <= 1;
    end else hue_valid <= 0;

```

```

//pass out s and v straight
s <= s_quotient;
v <= v_delay[19];

```

```

end

```

```

endmodule

```

```

// The divider module divides one number by another. It
// produces a signal named "ready" when the quotient output
// is ready, and takes a signal named "start" to indicate
// the the input dividend and divider is ready.
// sign -- 0 for unsigned, 1 for twos complement

```

```

// It uses a simple restoring divide algorithm.
// http://en.wikipedia.org/wiki/Division\_\(digital\)#Restoring\_division
//
// Author Logan Williams, updated 11/25/2018 gph

```

```

module divider #(parameter WIDTH = 8)
  (input clk, sign, start,
   input [WIDTH-1:0] dividend,
   input [WIDTH-1:0] divider,
   output reg [WIDTH-1:0] quotient,
   output [WIDTH-1:0] remainder,
   output ready);

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

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

  reg [5:0] a_bit = 0;
  reg del_ready = 1;
  assign ready = (a_bit==0) & ~del_ready;

  wire [WIDTH-2:0] zeros = 0;
  initial a_bit = 0;
  initial negative_output = 0;
  always @(posedge clk) begin
    del_ready <= (a_bit==0);
    if( start ) begin

      a_bit = WIDTH;
      quotient = 0;
      quotient_temp = 0;
      dividend_copy = (!sign || !dividend[WIDTH-1]) ?
        {1'b0,zeros,dividend} :
        {1'b0,zeros,~dividend + 1'b1};
    end
  end
endmodule

```

```

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

negative_output = sign &&
                  ((divider[WIDTH-1] && !dividend[WIDTH-1])
                   | (!(divider[WIDTH-1] && dividend[WIDTH-1]]));
end
else if ( a_bit > 0 ) begin
    diff = dividend_copy - divider_copy;
    quotient_temp = quotient_temp << 1;
    if( !diff[WIDTH*2-1] ) begin
        dividend_copy = diff;
        quotient_temp[0] = 1'd1;
    end
    quotient = (!negative_output) ?
              quotient_temp :
              ~quotient_temp + 1'b1;
    divider_copy = divider_copy >> 1;
    a_bit = a_bit - 1'b1;
end
end
endmodule

```

Appendix B - Non-Verilog Code

Create_image.py (Python script that receives the serial transmission)

```

"""Automatically find USB Serial Port (jodalyst 8/2019)
"""
import serial.tools.list_ports
from PIL import Image
import numpy as np

```

```
def get_usb_port():
    usb_port = list(serial.tools.list_ports.grep("USB"))
    if len(usb_port) == 1:
        print("Automatically found USB-Serial Controller: {}".format(usb_port[0].description))
        return usb_port[0].device
    else:
        ports = list(serial.tools.list_ports.comports())
        port_dict = {i:[ports[i],ports[i].vid] for i in range(len(ports))}
        usb_id=None
        for p in port_dict:
            #print("{}: {} (Vendor ID: {})".format(p,port_dict[p][0],port_dict[p][1]))
            #print(port_dict[p][0],"UART")
            print("UART" in str(port_dict[p][0]))
            if port_dict[p][1]==1027 and "UART" in str(port_dict[p][0]): #for generic USB Devices
                usb_id = p
        if usb_id== None:
            return False
        else:
            print("Found it")
            print("USB-Serial Controller: Device {}".format(p))
            return port_dict[usb_id][0].device

s = get_usb_port() #grab a port
print("USB Port: "+str(s)) #print it if you got
if s:
    ser = serial.Serial(port = s,
        baudrate=115200,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        bytesize=serial.EIGHTBITS,
        timeout=0.01) #auto-connects already I guess?
```

```
print("Serial Connected!")
if ser.isOpen():
    print(ser.name + ' is open...')
else:
    print("No Serial Device :/ Check USB cable connections/device!")
    exit()

j = 0
k = 0
for i in range(30):
    test_data = []
    count = 0
    temp = []
    try:
        print("Reading...")

        while True:
            data = ser.read(1) #read the buffer (99/100 timeout will hit)
            if data != b"": #if not nothing there.
                print("data", data[0] << 4, "num data received", count)
                count += 1

            #RED
            # if count <= 320*240:
            #     # print("current count", count)
            #     test_data.append((data[0] << 4, 0, 0))
            # else:
            #     print("uh oh" ,count)
            #     break

        # Actual
```



```

temp.append(data[0] << 4)
if count % 3 == 0:
    test_data.append(tuple(temp))
    temp = []

if count == 240*320*3:
    break
print("done!")
# print("v1", test_data)
test_data2 = np.array(test_data, dtype=np.uint8).reshape(240, 320, 3)
# print("reshaped", test_data2)
new_image = Image.fromarray(test_data2, mode='RGB')
if i % 2 == 0:
    j+=1
    new_image.save('k_left_final16.png'.format(j))
else:
    k+=1
    new_image.save('k_right_final16.png'.format(k))

except Exception as e:
    print(e)

```

Distance_calculation.m (MatLab script that we used to confirm our calculations before we converted it to Verilog)

% NOTE: images used for extrinsics: kleft_11.png and kright_11.png

```

%%%%%%%%% INTRINSIC %%%%%%%%%%
%
% K = [516.803363075377,0,0;
%      0,522.531338207211,0;

```

```
% 176.054418694951,125.385553812360,1]; % combined_2
```

```
K = [528.387846862879,0,0;
```

```
0,534.167521736855,0;
```

```
178.314289048315,123.438835414497,1];
```

```
% combined2_ with combined_1
```

```
% K = [530.908953461668,0,0;
```

```
% 0,535.774246324611,0;
```

```
% 174.570734765001,125.908473105822,1];
```

```
%%%%%%%% LEFT CAMERA %%%%%%%%%
```

```
% t1 = [316.881687216546, 28.5629433008597, 2072.02752489358]; % combined_2
```

```
% t1 = [322.427356065384,26.5027310146368,2125.89263907133]; % combined_1 and combined_2
```

```
t1 = [141.723252846619,66.1454836638422,1750.40978151427];
```

```
% R1 = [0.9573 -0.0356 -0.2869;
```

```
% 0.0214 0.9984 -0.0522;
```

```
% 0.2883 0.0438 0.9565];
```

```
R1 = [0.9630 -0.0037 -0.2693;
```

```
-0.0216 0.9956 -0.0910;
```

```
0.2685 0.0935 0.9587];
```

```
C1 = round(-t1/R1) %/ 25.4 % -t * inv(R)
```

```
% R2 = [0.9738 -0.0355 0.2247;
```

```
% 0.0492 0.9972 -0.0556;
```

```
% -0.2221 0.0652 0.9728];
```

```
R2 = [0.9883 0.0103 0.1524;
```

```
0.0040 0.9956 -0.0932;
```

```
-0.1527 0.0927 0.9839];
```

```
% t2 = [-495.460497629209,-48.1755393560359,2031.36032904293]; % combined_2
t2 = [-441.178698771649,14.8160945906379,1694.52827944107];
C2 = round(-t2/R2)

%%% coordinates to choose from
imagePoints_left = [253 131; 302 130; 304 162; 254 163];
origin_left = [253 131 1]; % actual: 255 132.5
orange_left = [268 116 1]; %[147 160 1];
% example_left = [211 120 1];
example_left = [230 117 1];
example_left3 = [216 98 1];
example_left4 = [208 131 1];

%%%
% example_right = [37 98 1];
example_right = [167 93 1];
example_right3 = [116 79 1];
example_right4 = [104 110 1];

inv1 = inv([R1(1, :); R1(2,:); t1]*K) * 2.^2

inv2 = inv([R2(1, :); R2(2,:); t2]*K) * 2.^2

% STAGE 1
P1 = origin_left * inv1; % Actual
s1_world1_x1 = example_left(1)*inv1(1, 1)
s1_world1_x2 = example_left(2)*inv1(2, 1)
s1_world1_x3 = inv1(3, 1)
```

```
s1_world1_y1 = example_left(1)*inv1(1, 2)
s1_world1_y2 = example_left(2)*inv1(2, 2)
s1_world1_y3 = inv1(3, 2)

s1_scaling_1 = example_left(1)*inv1(1, 3)
s1_scaling_2 = example_left(2)*inv1(2, 3)
s1_scaling_3 = inv1(3, 3)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

P2 = origin_right * inv2; % Actual

s2_world1_x1 = example_right(1)*inv2(1, 1)
s2_world1_x2 = example_right(2)*inv2(2, 1)
s2_world1_x3 = inv2(3, 1)

s2_world1_y1 = example_right(1)*inv2(1, 2)
s2_world1_y2 = example_right(2)*inv2(2, 2)
s2_world1_y3 = inv2(3, 2)

s2_scaling_1 = example_right(1)*inv2(1, 3)
s2_scaling_2 = example_right(2)*inv2(2, 3)
s2_scaling_3 = inv2(3, 3)

% STAGE 2
world1_x = s1_world1_x1 + s1_world1_x2 + s1_world1_x3
world1_y = s1_world1_y1 + s1_world1_y2 + s1_world1_y3
scaling1 = s1_scaling_1 + s1_scaling_2 + s1_scaling_3

world2_x = s2_world1_x1 + s2_world1_x2 + s2_world1_x3
world2_y = s2_world1_y1 + s2_world1_y2 + s2_world1_y3
scaling2 = s2_scaling_1 + s2_scaling_2 + s2_scaling_3
```

```
% STAGE 3
world1_x_scaled = world1_x / scaling1
world1_y_scaled = world1_y / scaling1
world2_x_scaled = world2_x / scaling2
world2_y_scaled = world2_y / scaling2

% STAGE 4
u1 = world1_x_scaled - C1(1)
u2 = world1_y_scaled - C1(2)
u3 = - C1(3)

v1 = world2_x_scaled - C2(1)
v2 = world2_y_scaled - C2(2)
v3 = - C2(3)

% STAGE 5
% t_subtraction_1 = C1(1) - C2(1);
% t_subtraction_2 = u1 - v1;
% t_subtraction_3 = C1(2) - C2(2);
% t_subtraction_4 = u2 - v2;
% t_subtraction_5 = C1(3) - C2(3);
% t_subtraction_6 = u3 - v3;

t_subtraction_1 = C1(1) - C2(1) % divide everything by 8 here
t_subtraction_2 = v1 - u1
t_subtraction_3 = C1(2) - C2(2)
t_subtraction_4 = v2 - u2
t_subtraction_5 = C1(3) - C2(3)
t_subtraction_6 = v3 - u3
```

```
% STAGE 6
% t_multiplication_1 = -(t_subtraction_1 * t_subtraction_2);
% t_multiplication_2 = -(t_subtraction_3 * t_subtraction_4);
% t_multiplication_3 = -(t_subtraction_5 * t_subtraction_6);

t_multiplication_1 = (t_subtraction_1 * t_subtraction_2)
t_multiplication_2 = (t_subtraction_3 * t_subtraction_4)
t_multiplication_3 = (t_subtraction_5 * t_subtraction_6)

t_divisor_1 = t_subtraction_2 * t_subtraction_2
t_divisor_2 = t_subtraction_4 * t_subtraction_4
t_divisor_3 = t_subtraction_6 * t_subtraction_6

% STAGE 7
t_cpa_numerator = t_multiplication_1 + t_multiplication_2 + t_multiplication_3
t_cpa_denominator = t_divisor_1 + t_divisor_2 + t_divisor_3

% STAGE 8
t_cpa = t_cpa_numerator / t_cpa_denominator;

% STAGE 9

world_x_multiplication_1 = t_cpa * u1
world_x_multiplication_2 = t_cpa * v1

world_y_multiplication_1 = t_cpa * u2
world_y_multiplication_2 = t_cpa * v2

world_z_multiplication_1 = t_cpa * u3
world_z_multiplication_2 = t_cpa * v3
```



```
% STAGE 10
```

```
world_x_numerator = C1(1) + world_x_multiplication_1 + C2(1) + world_x_multiplication_2
```

```
world_y_numerator = C1(2) + world_y_multiplication_1 + C2(2) + world_y_multiplication_2
```

```
world_z_numerator = C1(3) + world_z_multiplication_1 + C2(3) + world_z_multiplication_2
```

```
% STAGE 11
```

```
world_x = world_x_numerator / 2
```

```
world_y = world_y_numerator / 2
```

```
world_z = world_z_numerator / 2
```

```
% STAGE 12
```

```
world_x_sq = world_x * world_x
```

```
world_y_sq = world_y * world_y
```

```
world_z_sq = world_z * world_z
```

```
% STAGE 13
```

```
squared_distance = world_x_sq + world_y_sq + world_z_sq
```

```
% STAGE 14
```

```
distance = sqrt(squared_distance)
```

Appendix C - CAD Files

Camera L-Bracket



Camera Base

