Futuristic Pepper's Ghost Approximation

Jeremy McCulloch, Sreya Vangara, Adam Potter

6.111 Fall 2019

Abstract

Inspired by virtual reality games and the Pepper's Ghost system, we wanted to create an interactive projection that effectively optimized the characteristic speed of an FPGA to real-time render 3D objects from the perspective of a moving user. Our project uses a camera mounted overhead to track the user's position and a projector mounted overhead to display a model that is updated to appear correct from the user's perspective. We are able to render complex models that are converted from STLs (a common 3D model format), and we took advantage of this to show some complex models including an iron man suit, a möbius strip, and a cube with the letters FPGA on its faces. We also created a game where Thanos's head is launched towards the user and falls under gravity. Then, they must swipe at it with their hand when it is near its peak. If the user swipes at it with the correct timing then the head breaks in half and falls.

Hardware

The hardware we used includes a projector, a camera, the Nexys video board, and various mounts to attach them all to the ceiling. Additionally, we had some illuminated objects that we used for tracking the user position and allowing the user to interact with the projection.

Projector (Adam)

The first attributes we considered when selecting a projector are video ports and resolution. We developed the first half of our project using a VGA output on the Nexys DDR 4, however, we eventually switched to the Nexys Video which outputs HDMI. VGA is useful because it can output low resolution in RGB without needing tmds encoding. We are limited by board resources to rendering a low resolution image, so a projector that supports low resolutions means we can put our image on a larger percentage of the projected area. In order to send RGB values over HDMI we used a VGA-HDMI converted module which uses a TMDS converter. Externally, we used an HDMI-DVI converter to interface with the projector we chose. We also considered projector size since a smaller projector would be easier to mount.

The last consideration for the projector was the projection angle. The projector has a zoom feature, but only ranged +-10%. Ideally the camera and projector are mounted at the same height so they don't block each other's field of view. The height at which they are mounted should result in a projection a few feet across and an area around the projection a few feet wide in which the camera could still view the user. The measured angle of the projector was about 28 degrees onto a surface 2-3 feet above the

ground. With a 90 degree FOV camera the projector meets the requirements for the hologram rig. This resulted in us using a Dell 3200MP projector as it was the best option available.

Camera and Lens (Sreya)

We used an OV7670 CMOS VGA camera board for our computer vision. The camera operates at 30 fps, and is programmable through Arduino. Originally, with a flat telephoto lens, the field of vision was about 10 degrees. This was an issue, as we wanted to track the user as they moved around a table, and would not be able to achieve that range without mounting the camera extremely high. However, with a fisheye lens, we were able to widen it to about 90 degrees. This was satisfactory for tracking the user in a 3'x3' area, from a mount on the ceiling.

The camera has the ability to view infrared instead of visible light, which we investigated using an infrared LED filtered by a floppy disk. However, the increased accuracy was not significant enough to overhaul the extra effort of using relatively inaccessible materials. In addition, the camera generates 16-bit RGB, but we only used 12 color bits. We also tested adjusting the color gains in the Arduino code, which was very effective for identifying single color objects, but we wanted to be able to detect multiple colors, so we did not pursue this route.

Nexys Board (Sreya)

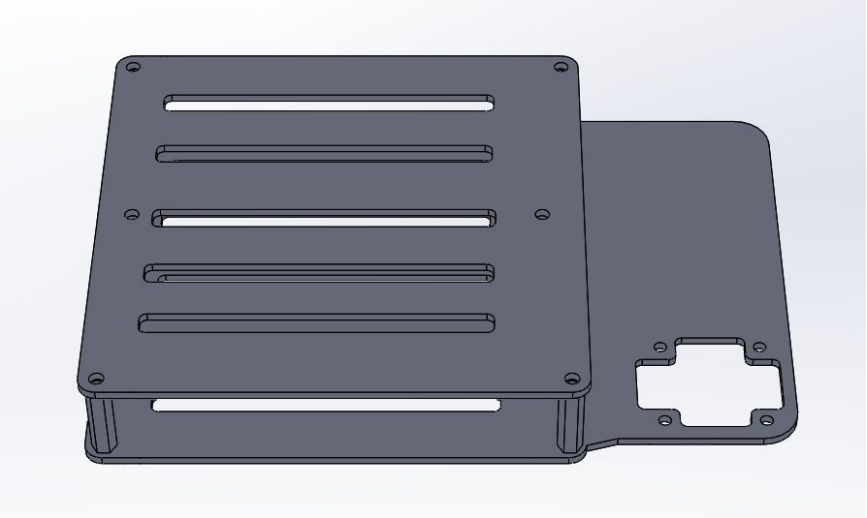
We originally started our project with the Nexys DDR board that populates the lab. This board contains 4,860 KBits of BRAM, which was enough for us to save two 320x240 12-bit color FrameBuffers. This necessitated our computer vision component to not use its own color FrameBuffer, and for our projected images to be smaller than desired. However, we later switched to the Nexys Video board, which has 13MBits of BRAM on board, almost three times as much as the Nexys 4 DDR. This allowed us to store 2 COEs of 12-bit color, with a resolution of 400x400, and generally worry less about memory. We were eventually able to store four COEs for a cube, mobius strip, IronMan suit, and Thanos bust, all colored.

There were a few challenges with switching boards, however. The Nexys DDR had a VGA port for video output, but the Video board uses a HDMI connector. For this, we had to write a VGA to HDMI conversion module. In addition, many of the ports on the DDR board were either in different places on the Video board, or just didn't match up, so we had to rewrite port connections. Also, the Video board has fewer switches than the DDR, and an OLED display instead of an LCD hex display. Since we were previously using the switches to tune thresholds for vision, we changed to using a VIO for calibrating HSV threshold values with the Video board.

Mounts (Adam)

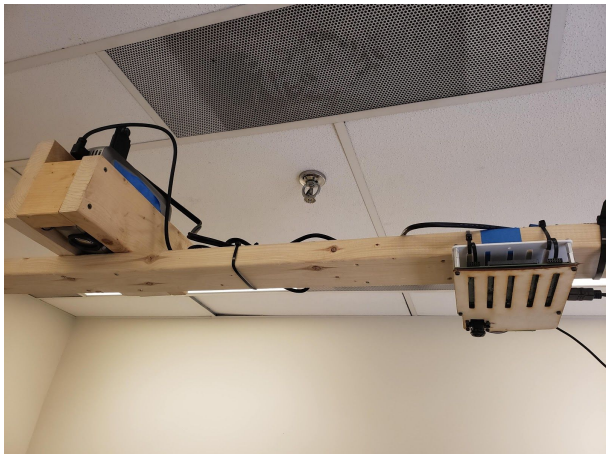
Mounting the projector and the camera over the user's head is critical to project. The camera has to be mounted close to the ceiling or it won't have the field of view to detect the user as they

circumnavigate the projection. The projector must be mounted high enough that the projection is at least a couple feet wide. Mounting the camera is easier because it is much lighter than the projector, however, the ribbon cable connecting the camera gets noisy as the ribbon cable extends several feet to the FPGA. Instead of risking poor communication with the camera, we decided to mount the Nexys board to the ceiling with the camera. The first priority was a case that protected the expensive board should it fall. To do this we designed and laser cut covers for each side of the board suspended with aluminum standoffs. The bottom half of the board extends beyond the boards profile to support the weight of the camera driver board, which is normally held solely with two rows of header pins, and securely attach the camera. The camera is mounted close to the board for minimal noise with necessary USB ports to the FPGA and Arduino board accessible. Additionally, slots in the mount help airflow and make the structure flexible to absorb the impact of a fall.



We secured this mount to the same board on which we secured the projector. After consulting the lab manager on the safety of our rig, we decided on using a 2x4 plank suspended with two ladders to hold the hardware. The second ladder was shorter than the first so we balanced the height with a box made with 2x4s 1x4s and screws. The 10ft 2x4 and box were clamped to the ladders leaving an 8 foot gap 8 foot tall for the user to comfortably walk under and walk around the hologram table. We used several zip ties to secure the camera and Nexys mount to the board directly above the white hologram board and fix assorted wires and cables in place.

The projector sits on the long board held with a box made from 2x4s. The gaps on the bottom leave enough room for the lens and for airflow to the fans. The projector is not placed directly over the hologram table because it projects at an angle.

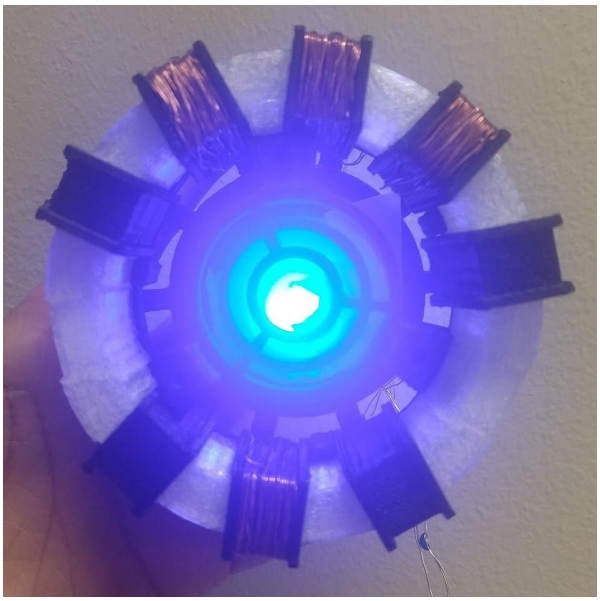


Tracking Objects (Sreya)

As a general piece of advice, an object or color is significantly easier to track if it emits its own light, as opposed to an object that is simply of the desired color. Even though the Hues are both within the same range, it is harder to filter out noise when trying to match the Value and Saturation ranges of an unlit object. However, the brightness of a self-lit object shifts the center of the Value range up, and allows us to cut out a lot of noise.

There are downsides to using a self-lit object though. Firstly, the reflection from the object is substantial, and often doubles the number of colored pixels in the frame. To solve this, the Saturation threshold should be increased. Secondly, if the object is too bright, it will be detected as a white pixel, rather than a colored one. This was an issue when we used LEDs or a flashlight as our colored object, because the bright light would not have the desired Hue since it appeared as white, or, in order to detect the light, we would have to accept very low Saturation values which introduced significant noise. In addition, a large “ring” of color would be detected around the source itself, and the center, as the most intense, would be detected as white, skewing the centroid values. In order to use an LED for detection, we found that dispersing the light using diffusion paper or pointing the LED in an orthogonal direction to the camera both worked well to dull the light. For testing purposes, 75% brightness on a phone served as a great color source.

IronMan Arc Reactor



The arc reactor is used to locate the user within the frame of the camera. The outer frame of the reactor is a 3D printed clear PLA ring with a radius of x, a thickness of y, and a depth of z. The secondary rings around the cylinder are also 3D printed, with black PLA, and hook around the ring. They are wrapped with copper wire to more closely mimic the movie prop. Several 3D printed structures form the backing of the reactor, essentially creating a hollow within the ring to place the light source. We placed a blue LED lamp inside the hollow, with a cone of diffusive paper taped around the LED to disperse the light and lessen the brightness so it didn't appear as white in to our camera.. Another smaller black 3D printed ring holds the LED lamp inside. The arc reactor is placed on the chest of the user.

IronMan Gauntlet

The gauntlet is used to track user interaction. The glove of the gauntlet consists of several 3D printed PLA segments, one for each of the three sections of a finger, and one for the palm. The sections for the fingers are attached to the palm and to each other with elastic bands. There is a circular hole in the palm, in which we place an Adafruit Circuit Playground board, programmable as a SAMD board with Arduino. The board has ten neopixels around the outer circumference, which we program to glow red. The board is connected to a phone power bank kept in the user's pocket.

Special thanks to Shayna Ahteck for lending and printing parts for our object tracking devices.

Computer Vision (Sreya)

Computer Vision Top Level

This module coordinates all the computer vision code, and is responsible for sending the centroid coordinates of the arc reactor and gauntlet to the graphics module. As input, it takes the camera pins (jb, jc). As output, it returns the (x, y) coordinates of the red and green centroids in the camera's field, and two Boolean values for if a green and red object are in the frame. This module interfaces with the Centroid, RGB2HSV, and Camera_Read modules. There are two clocks in this module: the 100MHz system clock, and the ~16.25 MHz pixel clock.

The module creates an instance of Camera_Read, passes in data from jb and jc, and receives a 16-bit raw pixel value and a valid_pixel pulse. To process the pixel, on every 100MHz clock cycle, pixel[15:12] is taken as the red value, [10:7] is the green, and [4:1] is blue. To assign the pixel an (x, y) coordinate, an always_ff block increments a hcount_camera and v_count camera counter on every rising edge of the camera clock, which synchronizes the pixel location with the pixel being processed at any given time.

Ideally, all the pixels would be stored in full color in BRAM, to allow for accurate object detection with an erosion and dilation filter. However, this would take 920 Kbits of BRAM, which would not fit on the Nexys when accounting for the FrameBuffers within the graphics module. If each pixel was passed through the RGB2HSV module, we could store two binary images instead, each with a 1 corresponding to a blue or red pixel, and 0 otherwise. However, with initial testing, an averaging filter with proper thresholding calculated the centroid rather accurately, and negated the need for the storage of an entire image. Therefore, we created a FIFO of write depth 512 (minimum write depth) with two independent clocks. The FIFO width is 33 bits in order to store the processed pixels, hcount_camera, and vcount_camera, all conjoined. At a pulse of the pixel clock, we write the processed input pixel with its (x, y) coordinate into the FIFO, with the valid_pixel pulse as the write enable. The read-out clock is the system clock of 100MHz, but the read enable is only high when the FIFO is not empty. Since the pixel clock is slower than the system clock, the throughput of RGB2HSV is always one. Therefore, at any point in time we only have a few full-color pixels stored in the FIFO, greatly reducing the data storage of the computer vision component!

From the data read out of the FIFO, we extract the pixel RGB value. We also extract the hcount_fifo, and v_count fifo, which are the original (x, y) coordinates of the pixel, but differ from the hcount_camera and vcount_camera in the module at that time, which correspond to the pixel just read by the camera. We use the hcount_fifo and vcount_fifo to pulse a frame_done register if the output pixel is the last in a frame, i.e. if hcount_fifo==319 && vcount_fifo==239.

We send the pixel RGB through two instances of RGB2HSV, to calculate two Boolean values, blue and red, each of which is 1 if the pixel is that color. In order to threshold color detection, we create upper and lower bounds for Hue, Saturation, and Value for blue and red objects. These values were determined manually. For testing purposes, they may also be tuned with a VIO. All other values associated with the pixel are pipelined for 22 cycles to account for the latency from RGB2HSV.

The color Booleans of the pixel, along with its (x, y) coordinates, and the frame_done register are then sent to two instances of the Centroid module, one of which returns the blue object centroid (x, y) coordinates, and the other of which returns the red centroid, both updated on a rising high of frame_done. However, in order to avoid counting the same pixel multiple times, we send in a 0 for the color Boolean if the FIFO is empty, indicating that the pixel has not updated. The Centroid module, in practice, only updates the centroid values 18 clock cycles after frame_done is pulsed to account for the latency of division calculation in the module, which takes 18 clock cycles.

Camera Read

This module takes as input the camera clock and 8-bit camera data. It outputs a 16-bit pixel value, a pulse when a valid pixel is processed, and a pulse when a frame is fully processed.

This module was provided by Joe Steinmeyer for interfacing with the OV7670.

RGB to HSV

The RGB to HSV module takes as input an RGB color of a pixel and HSV thresholds. It outputs a Boolean value corresponding to if the HSV values are within the HSV thresholds.

Converting the RGB values of the pixel to an HSV representation takes 22 clock cycles. We find the minimum and maximum of the R, G, B values, and then calculate the difference (delta). Value is delayed 18 clock cycles, and passed through as the maximum. Saturation is the delta divided by the maximum. The Hue calculation is more complicated, and depends on the maximum RGB color. The mathematical formulas are described below for interested students, though inspecting the Verilog module would likely be more helpful. Note that checking for division by zero is not included, but should be evaluated. Division takes 18 clock cycles, which necessitates the pipelining of Value.

$$Max = Max(R, G, B)$$

$$Min = Min(R, G, B)$$
$$\Delta = Max - Min$$
$$V = Max$$
$$S = Max\,!=0\ ?\ \Delta/Max\ :\ 0$$
$$H = \{R\ is\ Max => G \geq B\ ?\ 255 - (|G-B| * 8)/(\Delta * 6)\ :\ (|G-B| * 8)/(\Delta * 6),$$
$$G\ is\ Max => B \geq R\ ?\ (|B-R| * 8)/(\Delta * 6) > 85\ ?\ 255 - (|B-R| * 8)/(\Delta * 6) + 85\ :$$
$$85 - (|B-R| * 8)/(\Delta * 6)\ :\ 85 + (|B-R| * 8)/(\Delta * 6),$$
$$B\ is\ Max => R \geq G\ ?\ (|G-R| * 8)/(\Delta * 6) > 170\ ?\ 255 - (|G-R| * 8)/(\Delta * 6) + 170$$
$$170 - (|B-R| * 8)/(\Delta * 6)\ :\ 170 + (|B-R| * 8)/(\Delta * 6),\ B\ is\ Max\ \}$$

Our Hue, Saturation, and Value thresholds vary depending on the ambient lighting of the room. For instance, our lower Value threshold in daylight is 150, versus a threshold of 127 at night. This is because more brightness in the room introduces more noise. In addition, the reflectivity of the material in the room influences the Saturation thresholds. If the material is very reflective, it will easily take the Hue of the light source, and necessitates a higher minimum Saturation threshold.

The overall latency of this module is 22 clock cycles, incorporating the delta, maximum and minimum calculations, and the latency of division. The module is largely adapted from Kevin Zheng, Class of 2012.

Centroid Detection

The Centroid module processes the camera image one pixel at a time to determine the centroid coordinates of all elements in the frame of a given color. It takes as input an x-coordinate, a y-coordinate, end-of-frame pulse, Boolean color value, and a counter threshold. The color value is a Boolean indicating whether the pixel at the input (x, y) is of the desired color, and the counter threshold discerns the number of colored pixels in the frame necessary to confirm that the desired object is actually in the camera's field of vision, and the colored pixels are not just noise.

Our outputs are an (x,y) coordinate for the centroid, a Boolean value indicating whether an object of the input color was detected, and a counter of the number of pixels in the frame of the input color.

At reset, the module initializes to 0 an x-accumulator, y-accumulator, counter, x-centroid coordinate, and y-centroid coordinate. The accumulators add up the x and y values of any pixels that are the input color, and the counter keeps track of the number of pixels in the camera frame of the given color. The module contains two Divider IPs that average the accumulator values by the number of colored pixels to determine the centroid (x, y) coordinate.

During runtime, we can be in one of three states.

The first state is if the end-of-frame pulse is high. We first check if the counter is above the input threshold, and if so, we assert that our object was detected, and update the centroid coordinates to the

values in the x and y dividers. If not, we simply leave the centroid to be the last calculated value, and leave our detected output as low.

The second state is if the previous end-of-frame pulse was high, indicating that we are the clock cycle just after finishing a frame. We reinitialize our accumulators and counter.

Our third state is if the end-of-frame pulse, and previous end-of-frame pulse are both low. We check the color value of the input pixel, and if it matches, we add the x and y coordinates to the proper accumulators and increment the counter.

To account for the latency of the division calculation, which takes 18 clock cycles, the frame_done input used to trigger the first state is actually pipelined by 18 cycles. This allows all the pixels in the frame to be processed in the division of the accumulators.

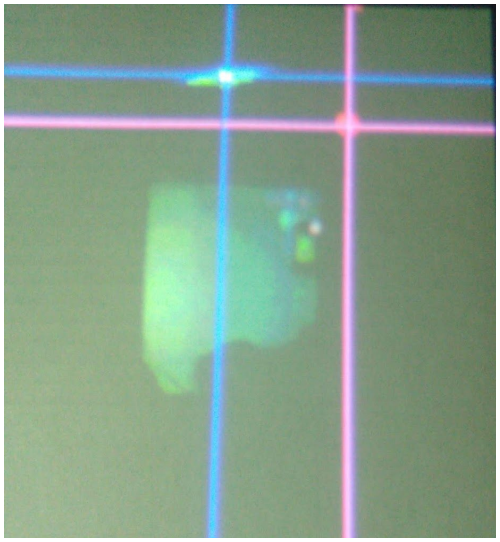


Image: This is the camera output projected onto a posterboard. The lights are off and the blinds are closed, so the user is not visible. However, the blue and red lights from the user tracking devices are visible, as well as the reflection on the white posterboard. The centroid of the blue and red pixels is marked with the crosshairs, and dynamically tracked as the user moves around the board. Saturation thresholds were calculated so that the reflection of these lights did not contribute to blue or red pixel counts.

Graphics

Graphics FSM (Jeremy)

Since our graphics subsystem has many modules that interact, we use an FSM to manage the transfer of data between these modules. In particular, the graphics FSM takes as inputs a finished signal from each module, which is high for a single clock cycle when the module finishes. It also outputs a new data signal which pulses high for one clock cycle when data is available for a given module. It also has a few additional inputs / outputs that interface with the triangle source (next_triangle, data_available, and next_frame).

The way this module works is it keeps track of whether data is available for each module, and whether each module is finished. When a module returns finished, finished gets set to true, and when all modules have finished then we proceed to the next triangle. Whenever we move to a new triangle, new data pulses high for a module if it has data available, which tells the module that it should begin working and return finished when it is done. Additionally, for any module with data available, finished is set to false when we move to the next triangle. All others remain true so we don't wait for them to finish.

We update whether a module has data available each time there is a new triangle. Data is available for a given module if it was available for the previous module on the previous triangle. At the beginning of a frame, data available starts as false for all modules except transform (the first module).

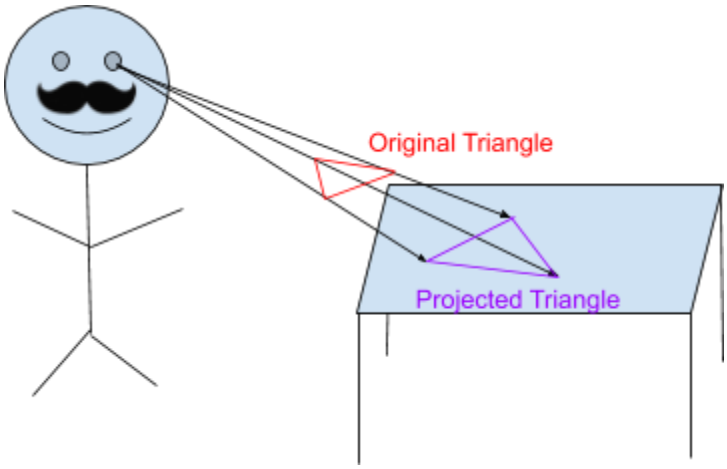
Triangle Source (Adam)

The triangle source module acts as the gate between loaded models stored in ROMs and the graphics subsystem. We used single port ROMS to store large 3D models allowing us to render more complicated models. Each line in the ROM has a width of 156 consisting of 36 bits (3*12 bits) for a normal vector, 12 bits for a RGB values, and 108 bits (3*3*12bits) for the three coordinates of three triangle. The module outputs a triangle and its properties when the next_frame input pulses.

In the triangle source there are five ROMS: a mobius strip, a cube, an Iron Man suit, and two halves of out game model, Thanos. The outputs of the module are controlled by switching between the ROMS using an object select register. Each time a new frame pulses, the object selector chooses an object based on the game state. Each time the next triangle input pulses, the module passes triangles from the selected module one by one. The module extracts and outputs triangle vertices, color, and normal vector from the ROMS.

Game state is passed from the game module into triangle source and has four states. The first three consist of transitioning between three objects every 7 seconds and the last is the game state where two models are rendered adjacent and break apart. For the first three states, triangle source selects the appropriate object and iterates through its triangles until it asserts that no triangles are available. For the final game state, the first of the two objects is selected (eg. half of Thanos' head) and respective triangles are returned. Upon reaching the end of the first model, triangle source instead switches to the second model and returns each triangle before asserting no more triangles are available.

Projection (Jeremy)



The projection module takes as inputs the user's position and the location of the vertices of a triangle and outputs the positions that the vertices of the triangle should be drawn on the screen. In order to do this, we calculate the position of each vertex simultaneously. To find where a given vertex should be drawn, we find where the z = 0 plane (the plane of the table) intersects the line through the user position and the vertex. The math for this evaluates to the following:

$$p_x = \frac{v_z u_x - u_z v_x}{v_z - u_z}$$
$$p_y = \frac{v_z u_y - u_z v_y}{v_z - u_z}$$

In order to implement this, we create two dividers per vertex and pipeline the remaining calculations. The dividers are signed and have a numerator of width 24 and denominator of width 16. The total latency for this module is 23 clock cycles, 21 of which come from the divider.

Shading (Adam)

The shading module takes in transformed triangles, user position, and RGB values and outputs shaded RGB values for each triangle. This is what allows a single model with a uniform color to have texture. This shader module assumes a light source behind the user meaning that the shading of each triangle is based on the angle between the triangle and the user, or more precisely the normal vector and the user vector. We set the color scaler to:

$$Scaler_{color} = cos(\theta_{triag})$$

Because it scales from zero to one and makes the math easier. The formula for this angle is:

$$T = Triangle\ Normal$$
$$U = User\ Vector$$
$$cos(\theta_{triag}) = (T \bullet U) / (|T| * |U|)$$

Rewritten using the vector magnitude formula:

$$cos(\theta_{triag}) = \sqrt{(T \bullet V)} / \sqrt{(T_x^2 + T_y^2 + T_z^2) * (V_x^2 + V_y^2 + V_z^2)}$$

For this module we realized that the resulting scalar from this equation would ultimately scale each of the 4 bit values in the 12 bit RGB color. This means the scalar only requires accuracy to 4 bits. To take advantage of this we used a division and square root lookup tables for the final steps of the calculation.

The total calculation for the shading of a triangle takes 13 clock cycles. The first is used to find the user vector using user location and one vertex of the triangle and the triangle normal vector which is passed from triangle source. The next clock cycle finds the dot product of the two vectors as well as the summed squares of the components in each vector. These values are the squared divisor and dividend of the next division step. These top and bottom values are the result of several multiplications and additions, so we allocate them 64 bits. However the scalar needs only 4 bit accuracy meaning the division can have a 4 bit divisor and dividend. Before the square root, these values only need to be 8 bits long in order to achieve our desired accuracy. We only care about the ratio of these numbers, so we can take the 8 most significant bits of each starting with the first positive bit of the dividend.

The next seven clock cycles are used to reduce the two 64 bit number to two 8 bit numbers. The dividend squared is sent to a module that uses a binary search to find the position of the first one be comparing part of the bit array to zero. For example, if the 32 bits of the number equals zero, the first one but lie in the second half. Once the location is determined, the module returns eight bits of the top and bottom number starting at the determined location.

Finally, each of the eight bit numbers are used as addresses to a square root lookup ROM which returns a 4 bit result in two clock cycles. These two numbers are concatenated together and used as an address to the division lookup ROM. The output of this ROM is our scalar. We multiply our RGB values by it and bit shift appropriately to obtain the shaded values for that triangle.

The lookup tables were created using a python sketch that took linearly spaced values in 8 bits for the square root ROM and 2x 4 bits for the division and returned a rounded output in binary. This resulted in ROMs with 2^8 lines of 4 bits, a file about 1kb large. For any function that only has to process an 8 bit value or smaller, this is a very efficient method. For values much larger than 8 bits, the ROM file size increases at O(2^N) so lookup tables might not be the best fit.

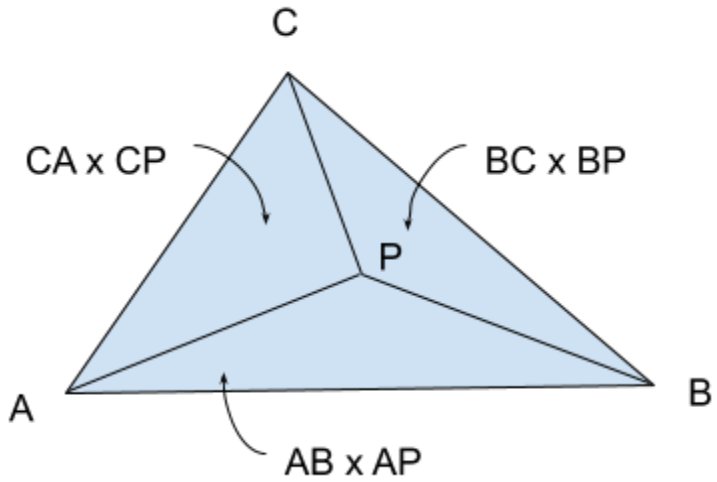
Rasterize (Jeremy)

The rasterize module takes as an input the vertices of a triangle to rasterize and its color, and it's remaining input and outputs remaining inputs and outputs interface with the framebuffer. When it first receives new data, the module begins by calculating the min and max x and y coordinates, and then iterates through all points in the bounding box of the triangle. For each point, it determines whether the point is inside the triangle and interpolates the z coordinate of that point.

To determine whether the point is within the triangle we calculate the cross product of each edge of the triangle with a vector from one of the vertices to the point we are testing. If all of these cross products have the same sign it is inside the triangle. Additionally, we use the magnitude of these cross products to interpolate the z coordinate. This works because each cross product has magnitude equal to twice the area of a triangle formed by the point we are testing and two of the vertices of the triangle. If we multiply the z value at each vertex by the cross product corresponding to the opposite side and divide by the total area of the triangle, we get an interpolated z value (see diagram + math below). The cross product and the interpolation each take a single clock cycle, and the divider takes 30 clock cycles. All of this is pipelined.

Once we interpolate the z coordinate and determine whether the point is in the triangle, we read the z coordinate from the framebuffer. Then we compare the calculated z coordinate from the one we

read. If the one we interpolated is larger than the one we measure and our point is inside the triangle then we write the specified color and calculated z coordinate to the framebuffer.



$$z = \frac{z_A(BC \times BP) + z_B(CA \times CP) + z_C(AB \times AP)}{AC \times AB}$$

Framebuffer (Jeremy)

The framebuffer consists of two simple dual-port BRAM modules each of which has a depth of 160,000 and a width of 24. The depth is 160,000 because we render a 400 x 400 pixel frame, and the width is 24 because in each position we must store a 12 bit color and a 12 bit signed z coordinate. This uses a total of 7.68 Mb of BRAM, which a majority of the 13 Mb available of BRAM. Increasing the amount of BRAM used led to routing / timing problems so we stopped at 400x400.

Additionally, this module keeps track of which of the two BRAM modules corresponds to the active frame and which corresponds to the inactive frame. This switches whenever a next_frame signal is received. For the inactive frame, the framebuffer takes in an x and y coordinate to read the z coordinate of, and x and y coordinate to write the z coordinate and color of. It then outputs the z coordinate that it reads. These inputs come from the rasterize module which is responsible for rendering the inactive frame.

For the active frame, the framebuffer takes in an x and y coordinate to read the color of and it returns a 12 bit rgb value. Subsequently it clears the memory in that location so it is ready to render the next frame when it becomes the inactive frame. To clear a memory location, we set the color to black and the z coordinate to the smallest possible z coordinate.

Transformations (Jeremy)

The transformation is described by 9 numbers, each of which is 12 bits wide and signed. The first three represent the x, y, and z model translation, which is a translation that we apply to all points in the model before we apply any rotations. This takes a single clock cycle and only requires addition.

The next three represent the roll, pitch, and yaw, and are given as a fraction of pi radians. In particular 0x7FF = 2047 corresponds to a rotation of +pi radians, and 0x800 = -2048 corresponds to a

rotation of -pi radians. These rotations are applied sequentially, with roll applied first as a rotation around the x axis, then pitch as a rotation around the y axis, and finally yaw as a rotation around the z axis. In order to perform these rotations, we use a CORDIC IP module in rotate mode which takes in an x and y coordinate and and an angle, and outputs a transformed x and y coordinate. The CORDIC module has a latency of 19, and each transformation is applied sequentially, so the rotations take a total of 63 clock cycles (just over 3 * the latency of the CORDIC).

The final three numbers represent the x, y, and z world translations, which is a translation that we apply to all points after we apply the rotations. This takes a single clock cycles and also only requires addition. Thus, the total latency of the transform module is 65 clock cycles.

The transformation module independently transforms each of the three vertices of the triangle and its normal vector. Each of these transformations is the same, except that the transformation for the normal vector does not include the translations.

Game (Jeremy)

The game module is responsible for keeping track of the game state, detecting motion of the gauntlet, and providing the transform values for whichever object(s) are being displayed)

In order to track motion of the gauntlet, the game module takes the calculated centroid x and y as inputs, as well as a single bit which indicates whether the gauntlet was detected. It samples the centroid at a specified frequency (5 Hz) and calculates the displacement of the centroid since it was last sampled. If the distance moved exceeds 25 pixels and the gauntlet was detected in both this sample and the last, then we declare that the gauntlet moved.

When the FPGA is reset, it sets the game state to 0, which indicates that it is showing the first demo model (mobius strip). Then it updates the game state every 5 seconds in order to show the next model. After the mobius strip, it shows an iron man suit, then a cube with the letters FPGA inscribed on it, and then returns to the mobius strip. This cycle continues until wand motion is detected while the FPGA cube is being displayed, at which point we transition to game state 3, which corresponds to playing the game. Once we enter game state 3 we stay in that state until the FPGA is reset.

If we are displaying the iron man suit or the mobius strip, then swiping the gauntlet causes the object to spin. We want to achieve a spin that decays with time, so we calculate a spin rate which starts at about pi / 16 radians per frame, or about two revolutions per second. Then, we decrease the spin rate every tenth of a second so that it comes to rest after three seconds. Once we calculate the spin rate, we use it to update the yaw of the object every frame.

Once we are playing the game, the game module calculates a model translation, rotation, and world translation for each half of Thanos. The model translation is always set to 0 for both objects. The world translation z component is a quadratic function of time in order to simulate the object in projectile motion. This projectile motion can be configured by adjusting the min and max z coordinates and the time of flight. The y coordinate of the world translation is always 0, and the x coordinate is 0 until gauntlet motion is detected near the peak of the objects arc, at which point the x coordinate increases linearly for one half of the fruit and decreases linearly for the other. Finally, the yaw increases linearly, the pitch is identically zero, and the roll increases linearly once Thanos is sliced in half.

Miscellaneous

VGA to HDMI (Jeremy)

In order to convert a VGA signal to HDMI, we use an IP module created by Digilent. It takes as an input a clock with the appropriate period, the color as 24 bit RBG (not RGB), vsync, hsync, and blanking. Its outputs correspond directly to the outputs of the nexys video board. The VGA signals are initially generated by the VGA module which is clocked at 25 MHz. Then, these are sent through a few registers and sent to the graphics subsystem, which is clocked at 100 MHz and returns a color value for the given vcount and hcount. Then, the signals are sent through more registers and sent to the HDMI module which is clocked at 25 MHz. Transferring data between clock domains is simplified by the fact that one clock domain is a multiple of the other, but synchronize registers are still used to avoid metastability.

STL to COE (Adam)

It quickly became clear that creating 3D models by manually writing coordinates in hex to a COE file would not be scalable. We considered writing code that could generate abstract shapes like spheres and cylinders, but it seemed better to be able to interface with existing CAD software. Most 3D model file formats are abstract and difficult to parse such as OBJ and STEP. These describe curved surfaces and large polygon facets on model which is efficient but difficult to turn into triangles. This led us to parsing STL or stereolithography files. When compiled into binary these files are uninterpretable, but when converted to ASCII they list the vertex coordinates of every triangle in the model and its normal vector, which makes the parsing simple. We wanted to be able to extract multiple colors from a model file; however only one format, STEP AP214, supports color and its formatting was too abstract to extract triangles. We settled for a single color per model passed as an argument to the python script.

Models passed into the stl_parser python sketch are scaled based on a either a desired size in the graphics coordinate frame or some scale which we multiply all the vertex coordinates by. The sketch parses through the STL file and extracts vertexes and normal vectors as floats using known delimiters. The normal vectors for each triangle are stored to later be used for the shader. It finds maxes and minimums of the model and then scales to the desired size. Optionally, the model can be centered based on the average of the max and mins. The coordinates for each triangle are cast to integers and turned into a binary string where each coordinate is given 12 bits. A method turns negative values into two's complement so when triangle source reads the COE as signed values it correctly interprets negative values.

The last tool in the STL parser is the fragment detection tool. It assumes that many of the smaller triangles in the model when scaled and cast to integers converge to points or lines instead of triangles. It checks if the three points in each triangle all share two or more coordinate values in which case the triangle has converged and it is omitted from the COE. Each line is appended with the same user-defined color. Finally, the script names and writes the file. This script is fully interfaceable from shell.

Vision Tuning (Sreya)

Determining the ideal Hue, Value, and Saturation thresholds is a time-consuming, rather irritating process that varies significantly if the environment changes. We originally tuned thresholds using the switches on the Nexys 4 DDR, but after switching to the Nexys Video and mounting the board on the ceiling, it was not very practical to continue this method of calibrating, especially since we moved to a room with windows, where the ambient light depended greatly on the time of day. However, we discovered a handy IP tool that should save future teams significant time in regards to computer vision: VIO.

VIO is a virtual input/output IP. We can specify the number of input and output probes when customizing the IP. Then for each probe, we can assign it a width (number of bits it should maximally be). For the output probes, we can also assign an initial value in hex. After programming the device, the IP launches a window that allows us to edit the output probes, and view the values of the input probes.

Our input probes are are the red and blue centroid (x, y) coordinates, and counts of red and blue pixels. Our output probes were the upper and lower Hue, Value, and Saturation thresholds for red and blue, and the counter thresholds for red and blue pixels.

Using the VIO greatly reduced our threshold calibration time, and allowed us to continue tuning even after the mounting of the camera and board on the ceiling removed our ability to use the switches and hex display.

HSV Thresholds (Sreya)

For the sake of future 6.111 vision projects, here are some thresholds we've found experimentally:

		In Lab
Green	Hue Range	(211, 63)
	Saturation Range	(100, 255)
	Value Range	(224, 127)
Blue	Hue Range	(96, 30)
	Saturation Range	(100, 255)
	Value Range	(127, 255)
	Hue Range	(10, 0)

Red	Saturation Range	(100, 255)
	Value Range	(224, 127)

Camera Arduino Code (Sreya)

Although we implemented object recognition by thresholding on HSV representations of RGB pixels, there were many iterations of computer vision we tried before deciding on this method. However, different methods of identification could be more effective in certain situations. These can be adjusted for in the OV7670 Arduino code.

For detecting a single color, adjusting the color gains of the camera is very effective for detecting a single-color object. Essentially, you would no longer need to convert to HSV and calculate division if that is pertinent to your project. Instead, objects of the opposing color of the adjusted gain (ex. Orange objects with high green gain) would have significantly differing RGB values.. Within the settings array, register 1 (0x01) tunes blue gain, register 2 tunes red gain, and register 3 tunes green gain.

For very accurate detection in a noisy environment, infrared detection is very effective. The OV7670 can also capture infrared images. A possible object of detection is an infrared LED behind a filter, such as a floppy disk.

If you'd like to experiment with different pixel encodings, the COM7 port (register 18, 0x12) controls the pixel format. The different permutations of bits[2:0] of the register produce the following: 00: YUV, 01: RGB, 10: Bayer raw, and 11: Processed Bayer raw. Enabling bit[3] uses the QCIF format, bit[4] uses QVGA, and bit[5] uses CIF.

Pipelining (Jeremy)

Since we make frequent use of dividers and other IP modules that have several clock cycles of latency, we needed an easy way to delay signals by a given number of clock cycles. Additionally, we want to be able to easily delay signals of specified widths. Thus, we created a module which takes as a parameter the width of the signal to pipeline and the number of clock cycles by which to delay it. As inputs, this module takes a clock, a reset, and the input signal of the given width, and it outputs a signal of the same width delayed by the specified number of clock cycles. This module saves us from needing to manually create numerous buffers to pipeline signals, which is time consuming and clutters other modules.

CV-Graphics Transform (Adam)

One advantage of how we treat models is that all the math for projection, shading, and rendering is done in the same coordinate space. Given a user position in that coordinate space, the projection and shading modules can accurately estimate how a 3D object should appear from that perspective. The difficulty that comes with aligning the coordinate spaces of the camera and graphics subsystems to correctly place the user in 3D. This is accomplished by comparing both the resolution and projected size (FOV) of the camera and the projector. The coordinate system of the camera is first centered to the center of the image by subtracting half the resolution in each direction. The CV centroid location is then scaled by some value. The scaling factor is declared using parameters and is implemented by multiplying by some integer M and bit shifting by N to approximate some float. The user height is passed into the function from top level and its value is appended to the 3 dimensional user position output.

$$Scale = (ProjectorRes * ProjectorSize / CamRes * CamSize)$$
$$X_{user} = Scale * (X_{centroid} - CamRes_X / 2)$$
$$Y_{user} = Scale * (Y_{centroid} - CamRes_Y / 2)$$

The second part of this module is limiting the movement of the CV centroid using an FIR filter. Imperfections in using computer vision for centroid detection can cause high frequency noise in the centroid measurements which creates glitchy rendering. Rapid movements room aren't possible for the user so this high frequency signal can be filtered out. The filter works as so:

$$Diff = Weight * (User_{calculated} - User)$$
$$User = User + Diff$$

Based on the weight of the filter, a real number between zero and one executed with multiplication and bit shifts, the user position is smoothed over time as user velocity is reduced.

Challenges and Advice

We had lots of trouble with our shader at first because it was initially implemented using all combinational logic. While this worked in simulation and led to very small latency, it also led to timing violations which we had to resolve. The end result of this was that we had to pipeline this module, which we could have done in the first place. From this we learned to pipeline everything, because although combinational logic may seem convenient when you first write it, it will likely cause you problems down the line.

Nearer the end of the project, it was also difficult to make even small changes because the time to generate bitstream for the project was close to half an hour. Some things that we learned helped minimize the number of times we needed to generate bitstream include making numerous testbenches. Creating unit testbenches for each module and integration testbenches for larger modules can save a lot of time, and catch mistakes in logic. This is especially true with signed logic, which seems to never work as expected the first time around. Another thing that helped save time was to run synthesis only, which is relatively

quick, and then check through any warnings that are generated. While some of them can be safely ignored, others, like unconnected ports, can be indicative of a problem.

When doing extensive signed math in Verilog, such as the vector math in this project, we found if very helpful to repeatedly cast registers using ‘ $signed(reg_name) ‘. This ensures the values are read as two’s complement and the multiplication and addition treats the values as such. We ran into several problems where we thought signed was implied yet it wouldn’t work until we cast the register. It’s not worth generating another bitstream, we’d recommend using $signed() all the time.

Finally, a very useful debugging feature we discovered is a VIO (virtual input / output) to inspect values of some registers and set values of others from your computer. This was useful for vision tuning, but also for testing when some modules are incomplete. In general, searching for IP modules that do what you are trying to do is worthwhile since they often exist. Other we use include a CORDIC module for rotating a point in 2D space and a VGA to HDMI module made by Digilent.

Another great resource is previous teams’ projects. Reading the reports for projects that involve similar features to yours helps guide structural thinking. Taking a look at other students’ Verilog is a great starting block to writing more complex modules.

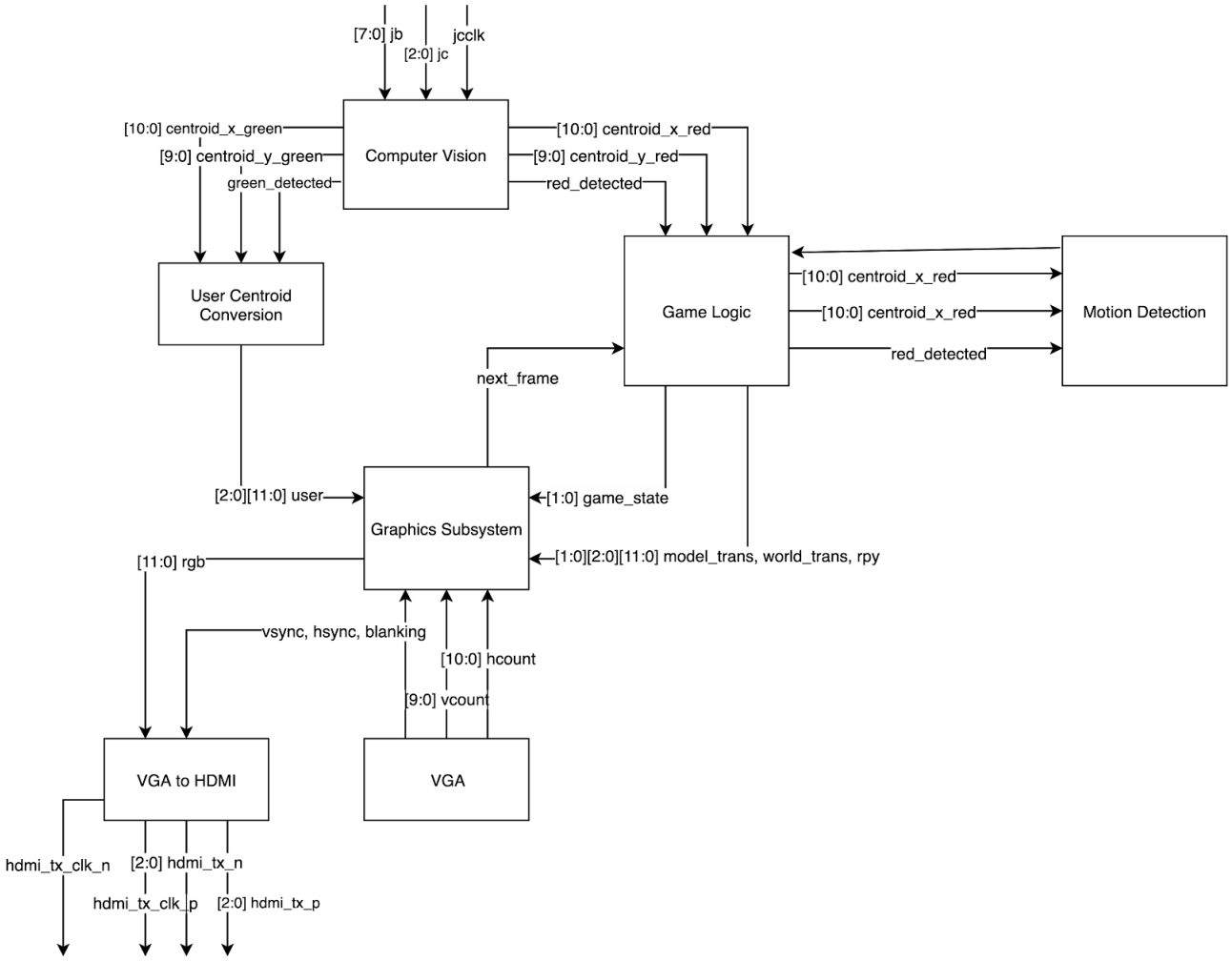
Conclusion

Our project accomplished our main goals of rendering an object from the user’s perspective as they walk around, allowing the user to interact with the model, and animating the model so it rotates / translates over time. We even achieved other reach goals, like displaying complex models with thousands of triangles. In doing so, we learned about computer vision, graphics rendering, and memory management on FPGAs. One reach goal which we did not achieve was projecting our image onto a cone instead of a flat surface so it looks more like a hologram. We hope to achieve this over IAP.

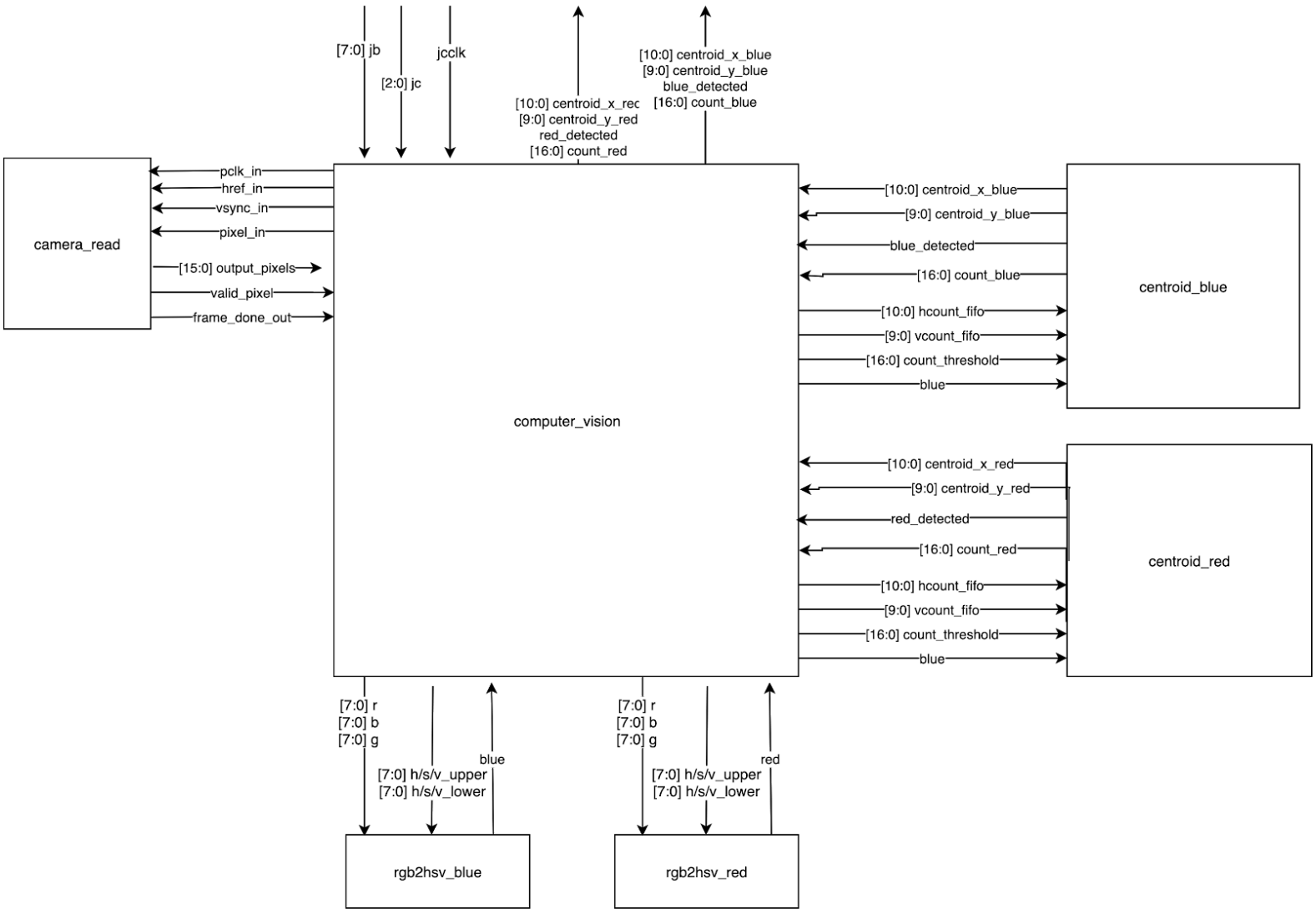
Appendix A (Block Diagrams)

Pipelining of all signals, debouncing, and the passing of clocks and resets were omitted from the block diagrams for clarity.

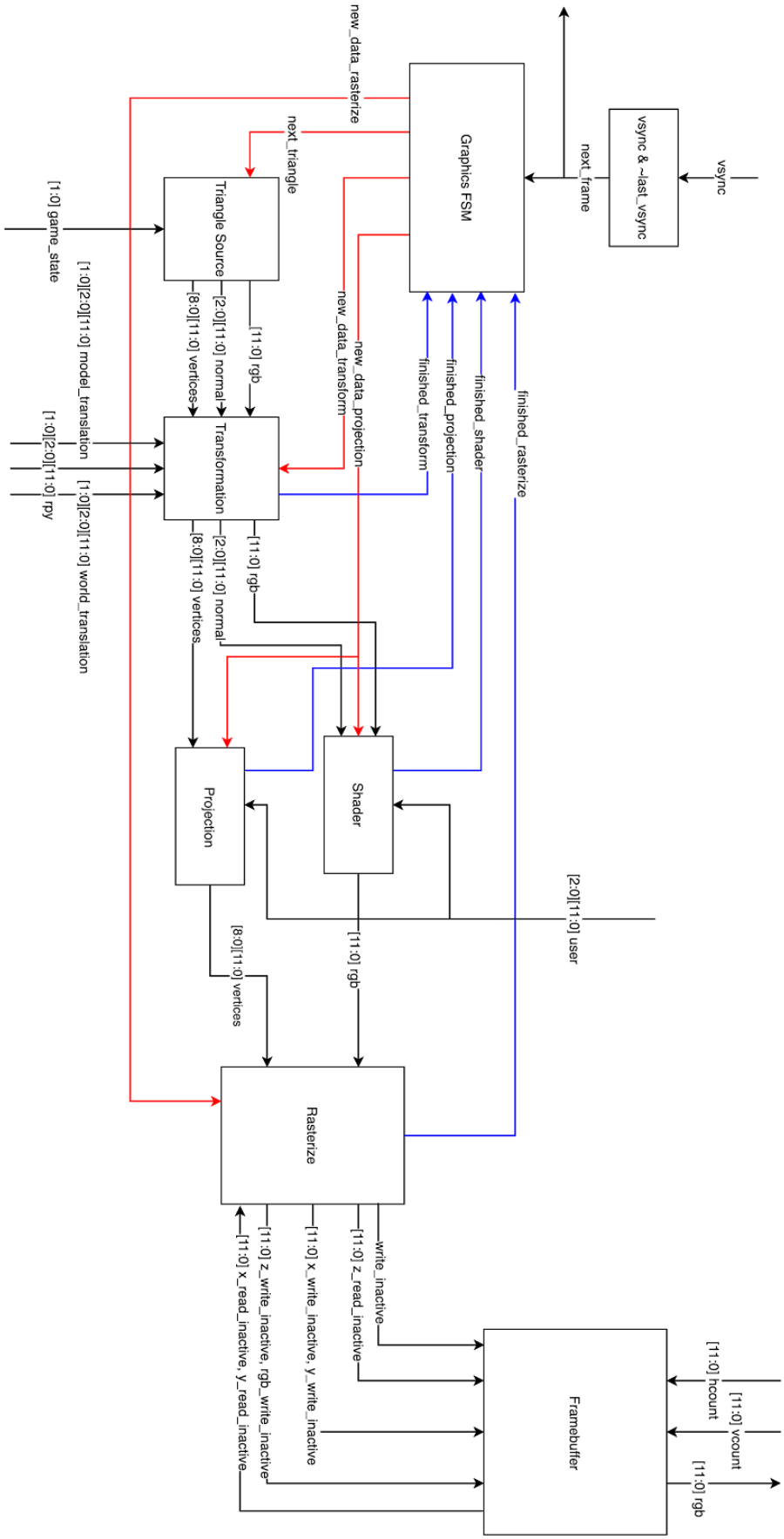
Top Level (Jeremy)



Computer Vision Subsystem (Sreya)



Graphics Subsystem (Jeremy)



Appendix B (Github Links)

The link to our main repository is https://github.com/jmcculloch2018/6111
The link to our vision repository which is setup for tuning the camera using a VIO on the nexys video board is https://github.com/SreyaV/6111-cv/tree/tuning-nexys-video
Both of the above links are for the Nexys video board. If you would like an older version of the camera repository that supports the Nexys DDR4, see https://github.com/SreyaV/6111-cv/tree/unified2

top_level.sv

```
`timescale 1ns / 1ps

module top_level(
  input clk,
  input [7:0] sw,
  input [7:0] jb,
  input [2:0] jc,
  output jcclk,
  output logic hdmi_tx_clk_n,
   output logic hdmi_tx_clk_p,
   output logic [2:0] hdmi_tx_n,
   output logic [2:0] hdmi_tx_p

   );

   logic reset;

   // Computer vision outputs
   logic [10:0] centroid_x_user;
   logic [9:0] centroid_y_user;
   logic [10:0] centroid_x_saber;
   logic [9:0] centroid_y_saber;
   logic saber_detected;

   // Converted centroid to world frame
   logic signed [2:0][11:0] user;
   logic signed [11:0] user_z;

   // VGA signals
   logic [1:0] vclock_count;
   logic [11:0] vcount;
   logic [11:0] hcount;
   logic vsync, hsync, blank;
   logic pixel_clk;

   // graphics subsystem outputs
   logic b,hs,vs;
   logic [11:0] rgb;
   logic next_frame;

   // Game outputs
   logic [1:0] game_state;
   logic clk_5sec;
```

```
logic signed [1:0][2:0][11:0] model_trans;
logic signed [1:0][2:0][11:0] rpy;
logic signed [1:0][2:0][11:0] world_trans;

// RGB to HDMI inputs
logic [23:0] rbg24;
logic vga_b, vga_hs, vga_vs;

assign reset = sw[7];

pipeline #(.N_BITS(24), .N_REGISTERS(3)) pipeline_rgb(
    .clk_in(pixel_clk), .rst_in(reset),
    .data_in({rgb[11:8], 4'hF, rgb[3:0], 4'hF, rgb[7:4], 4'hF}),
    .data_out(rbg24));
pipeline #(.N_BITS(1), .N_REGISTERS(3)) pipeline_b(
    .clk_in(pixel_clk), .rst_in(reset),
    .data_in(~b), .data_out(vga_b)
);
pipeline #(.N_BITS(1), .N_REGISTERS(3)) pipeline_vs(
    .clk_in(pixel_clk), .rst_in(reset),
    .data_in(~vs), .data_out(vga_vs)
);
pipeline #(.N_BITS(1), .N_REGISTERS(3)) pipeline_hs(
    .clk_in(pixel_clk), .rst_in(reset),
    .data_in(~hs), .data_out(vga_hs)
);


graphics_subsystem my_graphics(
    .clk(clk),
    .reset(reset),
    .user(user),
    .model_trans(model_trans),
    .rpy(rpy),
    .world_trans(world_trans),
    .game_state(game_state),
    .vcount_in(vcount),
    .hcount_in(hcount),
    .hsync_in(hsync),
    .vsync_in(vsync),
    .blank_in(blank),
    .rgb_out(rgb),
    .hsync_out(hs),
    .vsync_out(vs),
```

```
    .blank_out(b),
    .next_frame(next_frame)
);

computer_vision my_cv(
    .clk_100mhz(clk),
    .btnc(reset),
    .ja(jb),
    .jb(jc),
    .jbclk(jcclk),
    .night(sw[0]),
    .centroid_x_blue(centroid_x_user),
    .centroid_y_blue(centroid_y_user),
    .blue_detected(),
    .centroid_x_red(centroid_x_saber),
    .centroid_y_red(centroid_y_saber),
    .red_detected(saber_detected)

);

cv2render my_converter(
    .blob_x(centroid_x_user),
    .blob_y(centroid_y_user),
    .next_frame(next_frame),
    .user_z(user_z),
    .clk_in(clk),
    .user(user)
);

xvga my_vga(
        .vclock_in(pixel_clk),
        .rst_in(reset),
        .hcount_out(hcount),
        .vcount_out(vcount),
        .vsync_out(vsync),
        .hsync_out(hsync),
        .blank_out(blank));

hdmi_render hdmi (
  .TMDS_Clk_p(hdmi_tx_clk_p),    // output wire TMDS_Clk_p
  .TMDS_Clk_n(hdmi_tx_clk_n),    // output wire TMDS_Clk_n
  .TMDS_Data_p(hdmi_tx_p),  // output wire [2 : 0] TMDS_Data_p
  .TMDS_Data_n(hdmi_tx_n),  // output wire [2 : 0] TMDS_Data_n
  .aRst(reset),              // input wire aRst
```

```
  .vid_pData(rbg24),      // input wire [23 : 0] vid_pData
  .vid_pVDE(vga_b),       // input wire vid_pVDE
  .vid_pHSync(vga_hs),    // input wire vid_pHSync
  .vid_pVSync(vga_vs),    // input wire vid_pVSync
  .PixelClk(pixel_clk)       // input wire PixelClk
);

display_height my_height_disp(
    .clk_in(clk), .rst_in(reset),
    .sw(8'd36),
    .height(user_z),
    .seg_out(),
    .dp(),
    .strobe_out());

five_sec_clk my_slow_clock(
.clk_in(clk),
.rst_in(reset),
.clk_5sec(clk_5sec)
);

game_logic my_game(
    .clk_in(clk),
    .clk_5sec(clk_5sec),
    .rst_in(reset),
    .centroid_x(centroid_x_saber),
    .centroid_y(centroid_y_saber),
    .saber_detected(saber_detected),
    .next_frame(next_frame),
    .model_trans(model_trans),
    .rpy(rpy),
    .world_trans(world_trans),
    .game_state(game_state)
);




// Creates 25 MHz clock for Display (since 640 x 480)
assign pixel_clk = vclock_count[1];
always_ff @(posedge clk) begin
    if (reset) begin
        vclock_count <= 0;
    end else begin
```

```
            vclock_count <= vclock_count + 1;
        end
    end

endmodule
```

computer_vision.sv

```
    module computer_vision(
    input clk_100mhz, //system clock
    input btnc, //reset
    input [7:0] ja, //camera input
    input [2:0] jb, //camera input
    output jbclk,
    //used to track the arc reactor/user position
    output logic [10:0] centroid_x_blue,
    output logic [9:0] centroid_y_blue,
    output logic blue_detected,
    output logic [16:0] count_blue,
    //used to track gauntlet/user interaction
    output logic [10:0] centroid_x_red,
    output logic [9:0] centroid_y_red,
    output logic red_detected,
    output logic [16:0] count_red
     );

     wire [10:0] hcount;    // pixel on current line
     wire [9:0] vcount;     // line number
     wire hsync, vsync, blank;
     wire [11:0] pixel;
     reg [11:0] rgb;

     // btnc button is user reset
     wire reset;
     debounce db1(.reset_in(reset),.clock_in(clk_100mhz),.noisy_in(btnc),.clean_out(reset));
//debounce button

     logic xclk;
     logic[1:0] xclk_count;

     //for handling the pixel clock, syncs from the camera
     logic pclk_buff, pclk_in;
     logic vsync_buff, vsync_in;
     logic href_buff, href_in;
```

```
    logic[7:0] pixel_buff, pixel_in;

    logic [11:0] cam;
    logic [11:0] frame_buff_out;
    logic [15:0] output_pixels;
    logic [15:0] old_output_pixels;
    logic [11:0] processed_pixels; //changed from 12:0
    logic valid_pixel;
    logic frame_done_out;

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

    wire blue;
    wire red;
    logic frame_done;

    //registers for hue, value, and saturation thresholds
    logic [7:0] h_upper_blue;
    logic [7:0] h_lower_blue;
    logic [7:0] h_upper_red;
    logic [7:0] h_lower_red;
    logic [7:0] v_upper_blue, v_upper_red;
    logic [7:0] v_lower_blue, v_lower_red;
    logic [7:0] s_upper_red, s_upper_blue;
    logic [7:0] s_lower_red, s_lower_blue;
    //thresholds for the number of colored pixels necessary to determine if an object is in the
field, or if it is just noise
    logic [16:0] count_threshold_red, count_threshold_blue;

    //hcount and vcount for the pixel coming in from the camera
    logic [10:0] hcount_camera;
    logic [9:0] vcount_camera;

    //hcount and vcount for the pixel being read from the fifo
    logic [10:0] hcount_fifo;
    logic [9:0] vcount_fifo;

    assign xclk = (xclk_count >2'b01);
    assign jbclk = xclk;

    //high when the last pixel is read out of the fifo, as opposed to when the camera is done
reading a frame
    assign frame_done = (hcount_fifo==319 && vcount_fifo==239) ? 1 : 0;
```

```
   //increments hcount and vcount of camera pixel on posedge of the camera clock, gives each
pixel an (x, y) coordinate
   always_ff @(posedge pclk_in)begin
      if (frame_done_out)begin
         vcount_camera <= 0;
         hcount_camera<=0;
        // pixel_addr_in <= 17'b0;
      end else if (valid_pixel)begin
         if (hcount_camera == 319) begin
            hcount_camera <= 0;
            vcount_camera <= vcount_camera+1;
         end else begin
            hcount_camera <= hcount_camera+1;
         end
      end
   end

   //registers to store FIFO data
   logic full;
   logic empty;
   logic [32:0] fifo_temp;

    //store a few pixels at a time, between when the camera reads them and when they are
processed, to avoid storing
    //an entire framebuffer
    fifo my_fifo(
    .wr_clk(pclk_in),  // input wire wr_clk
  .rd_clk(clk_100mhz),  // input wire rd_clk
  .rst(0),    // input wire srst
  .din({processed_pixels, hcount_camera, vcount_camera}),      // input wire [32 : 0] din
  .wr_en(valid_pixel),  // input wire wr_en
  .rd_en(!empty),  // input wire rd_en
  .dout(fifo_temp),    // output wire [32 : 0] dout
  .full(full),    // output wire full
  .empty(empty)  // output wire empty
);

   //framebuffer of single pixel is read from the FIFO
   assign frame_buff_out = fifo_temp[32:21];
   assign hcount_fifo = fifo_temp[20:10];
   assign vcount_fifo = fifo_temp[9:0];

   //address of pixel is essentially its "index" within the frame
```

```
   assign pixel_addr_in = hcount_fifo+vcount_fifo*32'd320;

   assign cam = frame_buff_out; //was set to 'cam' previously for VGA output

   //to process camera inputs
   always_ff @(posedge clk_100mhz) begin
      pclk_buff <= jb[0];//WAS JB
      vsync_buff <= jb[1]; //WAS JB
      href_buff <= jb[2]; //WAS JB
      pixel_buff <= ja;
      pclk_in <= pclk_buff;
      vsync_in <= vsync_buff;
      href_in <= href_buff;
      pixel_in <= pixel_buff;
      old_output_pixels <= output_pixels;
      xclk_count <= xclk_count + 2'b01;
      processed_pixels = {output_pixels[15:12],output_pixels[10:7],output_pixels[4:1]};
   end

   //initial settings for threshold values
   assign h_upper_blue = 230; // blue = blue
   assign h_lower_blue = 150;
   assign h_upper_red = 10;
   assign h_lower_red = 0;
   assign v_upper_red = 255;
   assign v_lower_red = 127;
   assign v_upper_blue = 255;
   assign v_lower_blue = 127;
   assign s_upper_blue = 255;
   assign s_lower_blue = 200;
   assign s_upper_red = 255;
   assign s_lower_red = 200;
   assign count_threshold_red = 15;
   assign count_threshold_blue = 20;

   logic empty_delay; //pipelined "empty" signal for pixel taken from FIFO, delayed 22 cycles
to account for rgb2hsv module
   logic [10:0] hcount_fifo_delay; //pipelined hcount_fifo, delayed 22 cycles for rgb2hsv
module
   logic [9:0] vcount_fifo_delay;  //pipelined vcount_fifo, delayed 22 cycles for rgb2hsv
module

    pipeline #(.N_BITS(1), .N_REGISTERS(22)) pipeline_empty(
      .clk_in(clk_100mhz),
```

```
      .rst_in(reset),
      .data_in(empty),
      .data_out(empty_delay));

    pipeline #(.N_BITS(11), .N_REGISTERS(22)) pipeline_hcount(
      .clk_in(clk_100mhz),
      .rst_in(reset),
      .data_in(hcount_fifo),
      .data_out(hcount_fifo_delay));

    pipeline #(.N_BITS(10), .N_REGISTERS(22)) pipeline_vcount(
      .clk_in(clk_100mhz),
      .rst_in(reset),
      .data_in(vcount_fifo),
      .data_out(vcount_fifo_delay));

   //determines if pixel is red
   rgb2hsv rgb2hsv_red (.clock(clk_100mhz), .reset(reset), .r(cam[11:8]<<4),
.g(cam[7:4]<<4),
      .b(cam[3:0]<<4), .color(red), .h_upper(h_upper_red), .h_lower(h_lower_red),
         .v_upper(v_upper_red), .v_lower(v_lower_red), .s_upper(s_upper_red),
.s_lower(s_lower_red), .out_h());

   //determines if pixel is blue
   rgb2hsv rgb2hsv_blue (.clock(clk_100mhz), .reset(reset), .r(cam[11:8]<<4),
.g(cam[7:4]<<4),
      .b(cam[3:0]<<4), .color(blue), .h_upper(h_upper_blue), .h_lower(h_lower_blue),
         .v_upper(v_upper_blue), .v_lower(v_lower_blue), .s_upper(s_upper_blue),
.s_lower(s_lower_blue), .out_h());

  //takes pixel (x, y) and Boolean (is color red?) input, at frame_done, updates centroid of red
pixels if a red object is in frame
  centroid centroid_red (.clock(clk_100mhz), .reset(reset), .x(hcount_fifo_delay),
.y(vcount_fifo_delay),
      .color(!empty_delay ? red : 1'b0), .frame_done(frame_done), .centroid_x(centroid_x_red),
         .centroid_y(centroid_y_red), .count_out(count_red), .detected(red_detected),
.count_threshold(count_threshold_red));

  //takes pixel (x, y) and Boolean (is color blue?) input, at frame_done, updates centroid of
blue pixels if a blue object is in frame
  centroid centroid_blue (.clock(clk_100mhz), .reset(reset), .x(hcount_fifo_delay),
.y(vcount_fifo_delay),
       .color(!empty_delay ? blue : 1'b0), .frame_done(frame_done),
.centroid_x(centroid_x_blue),
```

```
        .centroid_y(centroid_y_blue), .count_out(count_blue), .detected(blue_detected),
.count_threshold(count_threshold_blue));

  //use camera_read module to input camera data
   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
```

rgb2hsv.sv

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//Largely adapted from Kevin Zheng, Class of 2012
//////////////////////////////////////////////////////////////////////////////////
module rgb2hsv(clock, reset, r, g, b, color, h_upper, h_lower, v_upper, v_lower, s_upper,
s_lower);
                        input wire clock;
                        input wire reset;
                        input wire [7:0] r;
                        input wire [7:0] g;
                        input wire [7:0] b;
                        //Thresholds for color identification
                        input logic [7:0] h_upper;
                        input logic [7:0] h_lower;
                        input logic [7:0] v_upper;
                        input logic [7:0] v_lower;
                        input logic [7:0] s_upper;
                        input logic [7:0] s_lower;

                        //Returns high if color is matched
                        output wire color;



                        reg [7:0] h;
                        reg [7:0] s;
                        reg [7:0] v;
                        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;
                logic s_ready;
                logic h_ready;
                logic [31:0] s_quotient_temp;
                logic [31:0] h_quotient_temp;

                // Clocks 4-18: perform all the divisions
                //the s_div (16/16) has delay 18
                //the h_div (16/16) has delay 18

   div_16 s_div (
  .aclk(clock),                                 // input wire aclk
  .s_axis_divisor_tvalid(1'b1),    // input wire s_axis_divisor_tvalid
  .s_axis_divisor_tdata(s_bottom),      // input wire [15 : 0] s_axis_divisor_tdata
  .s_axis_dividend_tvalid(1'b1),  // input wire s_axis_dividend_tvalid
  .s_axis_dividend_tdata(s_top),    // input wire [15 : 0] s_axis_dividend_tdata
  .m_axis_dout_tvalid(),          // output wire m_axis_dout_tvalid
  .m_axis_dout_tdata(s_quotient_temp)           // output wire [31 : 0] m_axis_dout_tdata
);


          div_16 h_div (
  .aclk(clock),                                 // input wire aclk
  .s_axis_divisor_tvalid(1'b1),    // input wire s_axis_divisor_tvalid
  .s_axis_divisor_tdata(h_bottom),      // input wire [15 : 0] s_axis_divisor_tdata
  .s_axis_dividend_tvalid(1'b1),  // input wire s_axis_dividend_tvalid
  .s_axis_dividend_tdata(h_top),    // input wire [15 : 0] s_axis_dividend_tdata
  .m_axis_dout_tvalid(),          // output wire m_axis_dout_tvalid
  .m_axis_dout_tdata(h_quotient_temp)           // output wire [31 : 0] m_axis_dout_tdata
```

```
);

   assign s_quotient = s_quotient_temp[31:16];
   assign h_quotient = h_quotient_temp[31:16];

                always @ (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];
                        end
                        else if(h_negative[18]) begin
                                h <= h_add[18] - h_quotient[7:0];
                        end
                        else begin
                                h <= h_quotient[7:0] + h_add[18];
                        end

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

                end

                //check if HSV matches thresholds
```

```
                assign color = ((h <= h_upper) && (h >= h_lower) && (v <= v_upper) && (v
>= v_lower) && (s <= s_upper) && (s >= s_lower));

endmodule
```

centroid.sv

```
  `timescale 1ns / 1ps

module centroid(clock, reset, x, y, color, centroid_x, centroid_y, frame_done, count_out,
detected, count_threshold);
    input logic clock; //system clock 100mghz
    input logic reset; //btnc
    input logic [10:0] x; //hcount of pixel
    input logic [9:0] y; //vcount of pixel
    input logic color; //boolean, high if the color of pixel matches the color we want to track
    input logic frame_done; //high if this pixel is the last to be removed from the FIFO
    input [16:0] count_threshold; //number of colored pixels in a frame to justify that an object
is being tracked, not noise
    output logic [10:0] centroid_x; //x coordinate of object's centroid
    output logic [9:0] centroid_y; //y coordinate of object's centroid
    output logic detected; //Boolean for whether enough colored pixels were on screen to
guarantee an object
    output logic [16:0] count_out; //return number of colored pixels, mostly used for debugging
    logic [55:0] centroid_x_temp; //output of divider
    logic [55:0] centroid_y_temp; //output of divider
    logic [26:0] x_acc; //keeps sum of all x-coordinates colored pixels are at
    logic [26:0] y_acc; //keeps sum of all y-coordinates colored pixels are at
    logic [16:0] count; //count of number of colored pixels
    logic last_frame_done; //for fsm
    logic frame_done_delay; //to account for latency of division

    always_ff @(posedge clock) begin
        if (reset) begin //zero accumulators and output
            x_acc <= 0;
            y_acc <= 0;
            count <= 0;
            centroid_x <= 0;
            centroid_y <= 0;
        end
        else if (frame_done_delay) begin
           count_out <= count; //update count at end of frame
            if (count > count_threshold) begin //only update centroid values if not viewing noise
                centroid_x <= centroid_x_temp[33:24];
```

```
            centroid_y <= centroid_y_temp[32:24];
            detected<=1;
          end
          else begin
            detected<=0;
          end
        end
        else if (!frame_done && last_frame_done) begin //next state: zero accumulators and
count
          x_acc <= 0;
          y_acc <= 0;
          count <= 0;
        end
        else begin
          if (color) begin //add pixels to accumulator if colored
            x_acc <= x_acc + x;
            y_acc <= y_acc + y;
            count <= count + 1;
          end
        end
        last_frame_done <= frame_done;
    end

    //pipeline frame_done for 18 counts to account for division latency, only assign new
centroid values after all pixels in frame are processed
     pipeline #(.N_BITS(1), .N_REGISTERS(18)) pipeline_frame_done(
    .clk_in(clock),
    .rst_in(reset),
    .data_in(frame_done),
    .data_out(frame_done_delay));

    //dividers for finding (x, y) of centroid from accumulators and count
    centroid_div xcenter(
    .aclk(clock),
    .s_axis_divisor_tdata(count),
    .s_axis_divisor_tvalid(1),
    .s_axis_dividend_tdata(x_acc),
    .s_axis_dividend_tvalid(1),
    .m_axis_dout_tdata(centroid_x_temp),
    .m_axis_dout_tvalid()
  );

  centroid_div ycenter(
    .aclk(clock),
```

```
    .s_axis_divisor_tdata(count),
    .s_axis_divisor_tvalid(1),
    .s_axis_dividend_tdata(y_acc),
    .s_axis_dividend_tvalid(1),
    .m_axis_dout_tdata(centroid_y_temp),
    .m_axis_dout_tvalid()
  );


endmodule
```

camera_read.sv

```
////////////////////////////////////////////////////////////////////////////////
//Module from Joe Steinmeyer
////////////////////////////////////////////////////////////////////////////////
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;
             if (pixel_half) begin
                pixel_data_out[7:0] <= p_data_in;
             end
             else begin
                pixel_data_out[15:8] <= p_data_in;
             end
          end
        end
        endcase
        end

endmodule
```

graphics_subsystem.sv

```
`timescale 1ns / 1ps

 module graphics_subsystem(
   input clk,
   input reset,
   input signed [2:0][11:0] user,
   input signed [1:0][2:0][11:0] model_trans,
   input signed [1:0][2:0][11:0] rpy,
   input signed [1:0][2:0][11:0] world_trans,
   input [1:0] game_state,
   input [11:0] vcount_in,
   input [11:0] hcount_in,
   input hsync_in,
   input vsync_in,
   input blank_in,
   output logic [11:0] rgb_out,
   output logic hsync_out,
   output logic vsync_out,
   output logic blank_out,
   output logic next_frame
```

```
    );

    parameter SCREEN_WIDTH = 400;
    parameter SCREEN_HEIGHT = 400;

    // New Data signals (from FSM to each module)
    logic new_data_rasterize;
    logic new_data_projection;
    logic new_data_transform;

    // Finish signals (to FSM from each module)
    logic projection_finish;
    logic rasterize_finish;
    logic shader_finish;
    logic transform_finish;

    // RGB signals (passed out of given module i.e. rgb_triangle source comes from triangle
source)
    logic [11:0] rgb_triangle_source;
    logic [11:0] rgb_transform;
    logic [11:0] rgb_shader;
    logic [11:0] rgb_rasterize;

    // Vertex coordinates (passed out of given module i.e. vertices_triangle source comes from
triangle source)
    logic signed [8:0][11:0] vertices_triangle_source;
    logic signed [8:0][11:0] vertices_transform;
    logic signed [8:0][11:0] vertices_projection_out;
    logic signed [8:0][11:0] vertices_rasterize;

    // Normal vectors (passed out of given module i.e. normal_triangle source comes from
triangle source)
    logic signed [2:0][11:0] normal_triangle_source;
    logic signed [2:0][11:0] normal_transform;

    // Framebuffer / Rasterize I/O signals
    logic [11:0] x_read_inactive_frame;
    logic [11:0] y_read_inactive_frame;
    logic [11:0] x_write_inactive_frame;
    logic [11:0] y_write_inactive_frame;
    logic signed [11:0] z_write_inactive_frame;
    logic signed [11:0] z_read_inactive_frame;
    logic write_inactive_frame;
    logic [11:0] rgb_write_inactive_frame;
```

```
// Triangle Source inputs
logic triangles_available;
logic next_triangle;

// Other
logic last_vsync_in;
logic obj_sel;

graphics_fsm my_graphics_fsm(
    .clk_in(clk),
    .rst_in(reset),
    .finish_rasterize(rasterize_finish),
    .finish_projection(projection_finish),
    .finish_shader(shader_finish),
    .finish_transform(transform_finish),
    .data_available_triangle_source(triangles_available),
    .next_frame(next_frame),
    .next_triangle(next_triangle),
    .new_data_projection(new_data_projection),
    .new_data_rasterize(new_data_rasterize),
    .new_data_transform(new_data_transform)
);

triangle_source my_tri_source(
    .clk_in(clk),
    .rst_in(reset),
    .next_triangle(next_triangle),
    .next_frame(next_frame),
    .game_state(game_state),
    .tf_sel(obj_sel),
    .triangles_available(triangles_available),
    .rgb_out(rgb_triangle_source),
    .vertices_out(vertices_triangle_source),
    .normal(normal_triangle_source)
);

transformation my_trans(
    .clk_in(clk),
    .rst_in(reset),
    .vertices_in(vertices_triangle_source),
    .model_translation(model_trans[obj_sel]),
    .rpy(rpy[obj_sel]),
    .world_translation(world_trans[obj_sel]),
```

```
        .normal_in(normal_triangle_source),
        .new_data_in(new_data_transform),
        .vertices_out(vertices_transform),
        .finished_out(transform_finish),
        .normal_out(normal_transform)
    );

    projection my_projection(
        .clk_in(clk),
        .rst_in(reset),
        .vertices_in(vertices_transform),
        .user_in(user),
        .new_data_in(new_data_projection),
        .vertices_out(vertices_projection_out),
        .finished_out(projection_finish)
    );

    shader my_shader(
        .clk_in(clk),
        .new_data(new_data_projection),
        .rgb_in(rgb_transform),
        .triag(vertices_transform),
        .user_pos(user),
        .rgb_out(rgb_shader),
        .normal(normal_transform),
        .finished(shader_finish)
    );

    rasterize #(.SCREEN_WIDTH(SCREEN_WIDTH),
.SCREEN_HEIGHT(SCREEN_HEIGHT)) my_rasterize(
        .clk_in(clk),
        .rst_in(reset),
        .rgb_in(rgb_rasterize),
        .vertices(vertices_rasterize),
        .new_data(new_data_rasterize),
        .finished(rasterize_finish),
        .z_read(z_read_inactive_frame),
        .write_ram(write_inactive_frame),
        .x_write(x_write_inactive_frame),
        .y_write(y_write_inactive_frame),
        .x_read(x_read_inactive_frame),
        .y_read(y_read_inactive_frame),
        .rgb_write(rgb_write_inactive_frame),
        .z_write(z_write_inactive_frame)
```

```
    );

    frame_buffer_manager #(.SCREEN_WIDTH(SCREEN_WIDTH),
.SCREEN_HEIGHT(SCREEN_HEIGHT)) my_manager(
        .clk_in(clk),
        .rst_in(reset),
        .next_frame(next_frame),
        .write_inactive_frame(write_inactive_frame),
        .x_read_inactive_frame(x_read_inactive_frame),
        .y_read_inactive_frame(y_read_inactive_frame),
        .x_write_inactive_frame(x_write_inactive_frame),
        .y_write_inactive_frame(y_write_inactive_frame),
        .rgb_write_inactive_frame(rgb_write_inactive_frame),
        .z_write_inactive_frame(z_write_inactive_frame),
        .z_read_inactive_frame(z_read_inactive_frame),
        .hcount_in(hcount_in),
        .vcount_in(vcount_in),
        .rgb_active_frame(rgb_out)
    );


    pipeline #(.N_BITS(1), .N_REGISTERS(2)) pipeline_vs (
        .clk_in(clk), .rst_in(reset),
        .data_in(vsync_in), .data_out(vsync_out)
    );
    pipeline #(.N_BITS(1), .N_REGISTERS(2)) pipeline_hs (
        .clk_in(clk), .rst_in(reset),
        .data_in(hsync_in), .data_out(hsync_out)
    );
    pipeline #(.N_BITS(1), .N_REGISTERS(2)) pipeline_b (
        .clk_in(clk), .rst_in(reset),
        .data_in(blank_in), .data_out(blank_out)
    );

    // Calculate next_frame from VGA signals
    assign next_frame = vsync_in && ~ last_vsync_in;
    always_ff @(posedge clk) begin
        if (reset) begin
            rgb_rasterize <= 12'h0;
            vertices_rasterize <= 0;
            rgb_transform <= 12'h0;
        end else begin
            rgb_transform <= new_data_transform ? rgb_triangle_source : rgb_transform;
            rgb_rasterize <= next_triangle ? rgb_shader : rgb_rasterize;
```

```
        vertices_rasterize <= next_triangle ? vertices_projection_out : vertices_rasterize;
      end
      last_vsync_in <= vsync_in;
   end
endmodule
```

shader.sv

```
module shader(
input clk_in,
input [11:0] rgb,
input [8:0] [15:0] triag, //pnts 1,2,3  1,2,3  1,2,3
input [2:0] [11:0] user_pos,
input new_data,
output logic finished,
output logic [11:0] rgb_out
   );
//Assign values to user and traingle coords
logic signed  [2:0] [15:0] t1;
logic signed [2:0] [15:0] t2;
logic signed [2:0] [15:0] t3;
assign t1 [0] = triag[0];
assign t1 [1] = triag[1];
assign t1 [2] = triag[2];
assign t2 [0] = triag[3];
assign t2 [1] = triag[4];
assign t2 [2] = triag[5];
assign t3 [0] = triag[6];
assign t3 [1] = triag[7];
assign t3 [2] = triag[8];
//Vectors for user and Triangle Normal
logic signed [2:0] [31:0] T;
logic signed [2:0] [19:0] V;
//Top and bottom sum
logic signed [59:0] top;
logic signed [59:0] bottom;
//Sqrt of 8 msb
logic [7:0] top_short;
logic [7:0] bottom_short;
//Num and Denom does into divider
logic [3:0] div_top;
logic [3:0] div_bottom;
//4 bit scaler for rgb
logic [3:0] scale; //unsigned
```

```
//Stored rgb values
logic [7:0] r;
logic [7:0] g;
logic [7:0] b;

//Find absolute value to get 8 bits
eight_msb mod(.clk(clk_in), .in_top(top), .in_bot(bottom), .out_top(top_short),
.out_bot(bottom_short));

//Sqrt of top and botom using 256 lookup table
sqrt_rom my_sqrt_rom_top(.addra(top_short), .clka(clk_in), .douta(div_top), .ena(1)); //8bit in,
4 bit out
sqrt_rom my_sqrt_rom_bottom(.addra(bottom_short), .clka(clk_in), .douta(div_bottom),
.ena(1));

//Divide using lookup table
div_rom my_div_rom(
   .addra( {div_top,div_bottom} ),
   .clka(clk_in),
   .douta(scale), .ena(1));

//Adjust RGB based on this scalar
assign r = rgb[11:8]*scale;
assign g = rgb[7:4]*scale;
assign b = rgb[3:0]*scale;

//FSM for trigger and output
logic [2:0] counter = 0;
logic state = 0;
always_ff @(posedge clk_in) begin
   if (new_data==1 && state==0) begin //Trigger starts process
      state <= 1;
   end else if (counter>5) begin //Finished
      finished <= 1;
      state <= 0;
      counter <= 0;
      rgb_out <= {r[7:4], g[7:4], b[7:4]}; // set output
   end else if (state==1) begin //Increment during run
      counter <= counter +1;
   end else begin // Not running
      finished <= 0;
   end

   case (state) //Does a multiplication in steps
```

```
      3'b001: begin //Find Vectors
         //Cross product of two triangles (27 bits)
         T[0] <= (t2[1]-t1[1])*(t3[2]-t1[2])-(t3[1]-t1[1])*(t2[2]-t1[2]);
         T[1] <= -(t2[0]-t1[0])*(t3[2]-t1[2])+(t3[0]-t1[0])*(t2[2]-t1[2]);
         T[2] <= (t2[0]-t1[0])*(t3[1]-t1[1])-(t3[0]-t1[0])*(t2[1]-t1[1]);
         //User perspective vector (13 bits)
         V[0] <= ($signed(t1[0])-$signed(user_pos[0]));
         V[1] <= ($signed(t1[1])-$signed(user_pos[1]));
         V[2] <= ($signed(t1[2])-$signed(user_pos[2]));
         end
      3'b010: begin //Compute top and bottom
         //Dot product between the vectors, result squared (42)->84
         top <= ($signed(T[0])*$signed(V[0]) + $signed(T[1])*$signed(V[1]) +
$signed(T[2])*$signed(V[2]))
         *($signed(T[0])*$signed(V[0]) + $signed(T[1])*$signed(V[1]) +
$signed(T[2])*$signed(V[2]));
         //Magnitude of two vectors squared (84 bits)
         bottom <=
($signed(T[0])*$signed(T[0])+$signed(T[1])*$signed(T[1])+$signed(T[2])*$signed(T[2]))

*($signed(V[0])*$signed(V[0])+$signed(V[1])*$signed(V[1])+$signed(V[2])*$signed(V[2]));
         end
      // Eight MSB (8 clks)
      //Sqrt Rom (1 clk)
      //Div Rom (1 clk)
      //Not Running
   endcase
end

endmodule


// Module takes bits 8 msb from bottom excluding zeros, takes corresponding segment in top
module eight_msb(
input clk,
input signed [59:0] in_top,
input signed [59:0] in_bot,
output logic [7:0] out_top,
output logic [7:0] out_bot
);
logic [59:0] top;
logic [59:0] bottom;
assign top = in_top[59]==1 ? -in_top : in_top;
assign bottom = in_bot[59]==1 ? -in_bot : in_bot;
```

```
logic [7:0] loc;
msb my_msb(.number(bottom), .msb(loc));

//Assign outputs
assign out_top = top[loc -: 8];
assign out_bot = bottom[loc -: 8];

endmodule

//MSB Finder
module msb(
   input [59:0] number,
   output logic [7:0] msb);

   logic counter [7:0];

   always_comb begin
      for(integer i=0; i<60; i=i+1) begin
         if (number[i]==1) begin
            msb = i;
         end
      end
   end

endmodule
```

triangle_source.sv

```
`timescale 1ns / 1ps

module triangle_source(
      input clk_in,
      input rst_in,
      input next_triangle,
      input next_frame,
      input [1:0] game_state,
      output logic tf_sel,
      output logic triangles_available,
      output logic [11:0] rgb_out,
      output logic [8:0][11:0] vertices_out,
      output logic signed [2:0] [11:0] normal
   );
   //Cube, Suit, Mobius, Thanos 2, Thanos 1
   //Max triangles for Nexys Video <25000
```

```
   parameter integer NUM_TRIANGLES [4:0] = {14'd560, 14'd12754, 14'd1677, 14'd6160,
14'd6152} ;
   logic [15:0] tri_count;
   logic [4:0][155:0] data_out; //One for each model
   logic [2:0] obj_select;
   //First Half of Model
   banana_rom1 thanos1 ( //*these roms used to be for bananas
     .clka(clk_in),    // input wire clka
     .addra(tri_count),  // input wire [3 : 0] addra
     .douta(data_out[0]),  // output wire [119 : 0] douta
     .ena(1)
   );
   //Seconds Half of Model
   banana_rom2 thanos2 (
     .clka(clk_in),    // input wire clka
     .addra(tri_count),  // input wire [3 : 0] addra
     .douta(data_out[1]),  // output wire [119 : 0] douta
     .ena(1)
   );

   demo_rom1 mobius ( //Mobius Strip
     .clka(clk_in),
     .addra(tri_count),
     .douta(data_out[2]),
     .ena(1)
   );

   demo_rom2 suit ( //Iron Man suit
     .clka(clk_in),
     .addra(tri_count),
     .douta(data_out[3]),
     .ena(1)
   );

   demo_rom3 cube ( //FPGA Cube
     .clka(clk_in),
     .addra(tri_count),
     .douta(data_out[4]),
     .ena(1)
   );

   always_ff @(posedge clk_in) begin
      if (rst_in) begin
         tri_count <= 0;
```

```
            obj_select <= 0;
        end else if (next_frame) begin //Set object based on game state
            case(game_state)
                2'b00: obj_select <= 2;
                2'b01: obj_select <= 3;
                2'b10: obj_select <= 4;
                2'b11: obj_select <= 0;
            endcase
            tri_count <= 0;
        end else if (next_triangle) begin //Pass ecah triangle one by one
            if (game_state==2'b11) begin //switches between model halves on fruit
                obj_select <= tri_count>=(NUM_TRIANGLES[obj_select] - 1) || (obj_select==1) ?
1:0;
                tri_count <= tri_count<(NUM_TRIANGLES[obj_select] - 1) ? (tri_count + 1):
obj_select==0 ? 0:tri_count;
            end else begin
                tri_count <= tri_count<(NUM_TRIANGLES[obj_select] - 1) ? (tri_count + 1):
tri_count;
            end
        end
    end

    //Indicates module has run out of triangles
    assign triangles_available = tri_count < (NUM_TRIANGLES[obj_select] - 1) ||
(obj_select==0);
    assign normal = data_out[obj_select][155:120];
    assign rgb_out = data_out[obj_select][119:108];
    assign vertices_out = data_out[obj_select][107:0];
    assign tf_sel = obj_select==1;
endmodule
```

transformation.sv

```
`timescale 1ns / 1ps

module transformation(
    input clk_in,
    input rst_in,
    input signed [8:0][11:0] vertices_in,
    input signed [2:0][11:0] model_translation,
    input signed [2:0][11:0] rpy,
    input signed [2:0][11:0] world_translation,
    input new_data_in,
    input [2:0] [11:0] normal_in,
```

```
    output logic signed [2:0] [11:0] normal_out,
    output logic signed [8:0][11:0] vertices_out,
    output logic finished_out
);
    // We transform each vertex and the normal vector separately using the values passed in
    // Translation is always set to 0 for the normal vector
    parameter ROTATION_LATENCY = 19;
    parameter TOTAL_LATENCY = 3 * ROTATION_LATENCY + 8;
    transformation_vertex #(.ROTATION_LATENCY(ROTATION_LATENCY),
        .TOTAL_LATENCY(TOTAL_LATENCY)) vertex1(
        .clk_in(clk_in), .rst_in(rst_in),
        .vertex_in(vertices_in[8:6]),
        .model_translation(model_translation),
        .rpy(rpy),
        .world_translation(world_translation),
        .new_data_in(new_data_in),
        .vertex_out(vertices_out[8:6])
    );
    transformation_vertex #(.ROTATION_LATENCY(ROTATION_LATENCY),
        .TOTAL_LATENCY(TOTAL_LATENCY)) vertex2(
        .clk_in(clk_in), .rst_in(rst_in),
        .vertex_in(vertices_in[5:3]),
        .model_translation(model_translation),
        .rpy(rpy),
        .world_translation(world_translation),
        .new_data_in(new_data_in),
        .vertex_out(vertices_out[5:3])
    );
    transformation_vertex #(.ROTATION_LATENCY(ROTATION_LATENCY),
        .TOTAL_LATENCY(TOTAL_LATENCY)) vertex3(
        .clk_in(clk_in), .rst_in(rst_in),
        .vertex_in(vertices_in[2:0]),
        .model_translation(model_translation),
        .rpy(rpy),
        .world_translation(world_translation),
        .new_data_in(new_data_in),
        .vertex_out(vertices_out[2:0])
    );
        transformation_vertex #(.ROTATION_LATENCY(ROTATION_LATENCY),
        .TOTAL_LATENCY(TOTAL_LATENCY)) normal1(
        .clk_in(clk_in), .rst_in(rst_in),
        .vertex_in(normal_in),
        .model_translation(36'b0),
        .rpy(rpy),
```

```
        .world_translation(36'b0),
        .new_data_in(new_data_in),
        .vertex_out(normal_out)
    );
    pipeline #(.N_BITS(1), .N_REGISTERS(TOTAL_LATENCY)) pipeline_finished(
        .clk_in(clk_in), .rst_in(rst_in),
        .data_in(new_data_in), .data_out(finished_out)
    );


endmodule


module transformation_vertex(
    input clk_in,
    input rst_in,
    input signed [2:0][11:0] vertex_in,
    input signed [2:0][11:0] model_translation,
    input signed [2:0][11:0] rpy,
    input signed [2:0][11:0] world_translation,
    input new_data_in,
    output logic signed [2:0][11:0] vertex_out
    );
    parameter ROTATION_LATENCY = 0;
    parameter TOTAL_LATENCY = 0;

    // Store I/O from CORDIC IP (rotates x, y coordinates by an angle)
    logic signed [1:0][15:0] cartesian_data;
    logic signed[15:0] phase_data; // signed pi radians 0x1FF = pi, 0xE00 = -pi
    logic signed [1:0][15:0] data_out;
    logic valid_in;

    // count number of clock cycles since new data
    logic [7:0] count;

    model_rotation rotation (
      .aclk(clk_in),                                // input wire aclk
      .s_axis_phase_tvalid(valid_in),          // input wire s_axis_phase_tvalid
      .s_axis_phase_tdata(phase_data),            // input wire [15 : 0] s_axis_phase_tdata
      .s_axis_cartesian_tvalid(valid_in),  // input wire s_axis_cartesian_tvalid
      .s_axis_cartesian_tdata(cartesian_data),    // input wire [31 : 0] s_axis_cartesian_tdata
      .m_axis_dout_tvalid(),            // output wire m_axis_dout_tvalid
      .m_axis_dout_tdata(data_out)              // output wire [31 : 0] m_axis_dout_tdata
    );
```

```
    always_ff @(posedge clk_in) begin
        if (rst_in || new_data_in) begin
            count <= 0;
            // When we receive new data, add model translation and save in vertex_out
            vertex_out[2] <= $signed(vertex_in[2]) + $signed(model_translation[2]);
            vertex_out[1] <= $signed(vertex_in[1]) + $signed(model_translation[1]);
            vertex_out[0] <= $signed(vertex_in[0]) + $signed(model_translation[0]);

        end else begin
            count <= count < (TOTAL_LATENCY - 1) ? count + 1 : count;

            // Inputs
            cartesian_data[1] <= (count == 0) ? vertex_out[1] :
                (count == ROTATION_LATENCY + 2) ? vertex_out[0] :
                (count == 2 * ROTATION_LATENCY + 4) ? vertex_out[2] : cartesian_data[1];
            cartesian_data[0] <= (count == 0) ? vertex_out[0] :
                (count == ROTATION_LATENCY + 2) ? vertex_out[2] :
                (count == 2 * ROTATION_LATENCY + 4) ? vertex_out[1] : cartesian_data[0];
            phase_data <= (count == 0) ? ($signed(rpy[2])>>>2) :
                (count == ROTATION_LATENCY + 2) ? ($signed(rpy[1])>>>2) :
                (count == 2 * ROTATION_LATENCY + 4) ? ($signed(rpy[0])>>>2) : phase_data;
            valid_in <= (count == 0) || (count == ROTATION_LATENCY + 2) || (count == 2 *
ROTATION_LATENCY + 4);

            // Outputs
            vertex_out[2] <= (count == 2 * ROTATION_LATENCY + 3) ? data_out[0][11:0] :
                (count == 3 * ROTATION_LATENCY + 5) ? data_out[1][11:0] :
                (count == 3 * ROTATION_LATENCY + 6) ? vertex_out[2] + world_translation[2] :
                vertex_out[2];
            vertex_out[1] <= (count == ROTATION_LATENCY + 1) ? data_out[1][11:0] :
                (count == 3 * ROTATION_LATENCY + 5) ? data_out[0][11:0] :
                (count == 3 * ROTATION_LATENCY + 6) ? vertex_out[1] + world_translation[1] :
                vertex_out[1];
            vertex_out[0] <= (count == ROTATION_LATENCY + 1) ? data_out[0][11:0] :
                (count == 2 * ROTATION_LATENCY + 3) ? data_out[1][11:0] :
                (count == 3 * ROTATION_LATENCY + 6) ? vertex_out[0] + world_translation[0] :
                vertex_out[0];
        end
    end

endmodule
```

rasterize.sv

```
`timescale 1ns / 1ps

module rasterize(
    input clk_in,
    input rst_in,
    // Inputs from shader / projection
    input [11:0] rgb_in,
    input signed [8:0][11:0] vertices,
    // FSM Signals
    input new_data,
    output logic finished,
    // Framebuffer interaction
    input signed [11:0] z_read,
    output logic write_ram,
    output logic [11:0] x_write,
    output logic [11:0] y_write,
    output logic [11:0] x_read,
    output logic [11:0] y_read,
    output logic [11:0] rgb_write,
    output logic signed [11:0] z_write
);

    parameter DIVISION_LATENCY = 30;
    parameter SCREEN_WIDTH = 0; // Gets overriden by graphics subsystem
    parameter SCREEN_HEIGHT = 0;


    // Current position and max / min coordinates for triangle
    logic [11:0] x_cur;
    logic [11:0] y_cur;
    logic signed [11:0] x_min, x_max, y_min, y_max;

    // z interpolation and triangle checking
    logic signed [23:0] area_total, area1, area2, area3; // First clock cycle
    logic signed [31:0] numerator; // Second clock cycle
    logic signed [23:0] denominator; // Second clock cycle
    logic in_triangle = 0; // Second clock cycle
    logic in_triangle_lag; // 32nd clock cycle

    // State
    logic busy;
    logic busy_lag;
    logic last_busy;
```

```
    // Calculate min / max x and y (combinational)
    // Always returns value within screen dimensions
    get_min #(.ABSOLUTE_MIN(0), .ABSOLUTE_MAX(SCREEN_WIDTH - 1))
        get_min_x(.val1(vertices[8]), .val2(vertices[5]), .val3(vertices[2]), .min(x_min));
    get_min #(.ABSOLUTE_MIN(0), .ABSOLUTE_MAX(SCREEN_HEIGHT - 1))
        get_min_y(.val1(vertices[7]), .val2(vertices[4]), .val3(vertices[1]), .min(y_min));
    get_max #(.ABSOLUTE_MIN(0), .ABSOLUTE_MAX(SCREEN_WIDTH - 1))
        get_max_x(.val1(vertices[8]), .val2(vertices[5]), .val3(vertices[2]), .max(x_max));
    get_max #(.ABSOLUTE_MIN(0), .ABSOLUTE_MAX(SCREEN_HEIGHT - 1))
        get_max_y(.val1(vertices[7]), .val2(vertices[4]), .val3(vertices[1]), .max(y_max));

    // Calculate Areas for z interpolation + triangle checking
    get_area get_area_total(.clk_in(clk_in), .x1(vertices[8]), .y1(vertices[7]), .x2(vertices[5]),
.y2(vertices[4]),
        .x3(vertices[2]), .y3(vertices[1]), .area(area_total));
    get_area get_area1(.clk_in(clk_in), .x1(x_cur), .y1(y_cur), .x2(vertices[5]), .y2(vertices[4]),
        .x3(vertices[2]), .y3(vertices[1]), .area(area1));
    get_area get_area2(.clk_in(clk_in), .x1(vertices[8]), .y1(vertices[7]), .x2(x_cur), .y2(y_cur),
        .x3(vertices[2]), .y3(vertices[1]), .area(area2));
    get_area get_area3(.clk_in(clk_in), .x1(vertices[8]), .y1(vertices[7]), .x2(vertices[5]),
.y2(vertices[4]),
        .x3(x_cur), .y3(y_cur), .area(area3));


    // Lag of 2 + DIVISION_LATENCY
    logic [55:0] divider_out;
    z_interp_divider my_div (
      .aclk(clk_in),                                // input wire aclk
      .s_axis_divisor_tvalid(1'b1),    // input wire s_axis_divisor_tvalid
      .s_axis_divisor_tdata(denominator),      // input wire [23 : 0] s_axis_divisor_tdata
      .s_axis_dividend_tvalid(1'b1),  // input wire s_axis_dividend_tvalid
      .s_axis_dividend_tdata(numerator),    // input wire [31 : 0] s_axis_dividend_tdata
      .m_axis_dout_tvalid(),          // output wire m_axis_dout_tvalid
      .m_axis_dout_tdata(divider_out)            // output wire [55 : 0] m_axis_dout_tdata
    );
    assign z_write = divider_out[35:24];


    // Delay a bunch of signals
    pipeline #(.N_BITS(12), .N_REGISTERS(DIVISION_LATENCY)) pipeline_x(
        .clk_in(clk_in),
        .rst_in(rst_in),
        .data_in(x_cur),
        .data_out(x_read));
```

```
pipeline #(.N_BITS(12), .N_REGISTERS(DIVISION_LATENCY)) pipeline_y(
    .clk_in(clk_in),
    .rst_in(rst_in),
    .data_in(y_cur),
    .data_out(y_read));
pipeline #(.N_BITS(1), .N_REGISTERS(DIVISION_LATENCY)) pipeline_intri(
    .clk_in(clk_in),
    .rst_in(rst_in),
    .data_in(in_triangle),
    .data_out(in_triangle_lag));
pipeline #(.N_BITS(1), .N_REGISTERS(DIVISION_LATENCY + 2)) pipeline_busy(
    .clk_in(clk_in),
    .rst_in(rst_in),
    .data_in(busy),
    .data_out(busy_lag));
 pipeline #(.N_BITS(12), .N_REGISTERS(2)) pipeline_x_write(
    .clk_in(clk_in),
    .rst_in(rst_in),
    .data_in(x_read),
    .data_out(x_write));
pipeline #(.N_BITS(12), .N_REGISTERS(2)) pipeline_y_write(
    .clk_in(clk_in),
    .rst_in(rst_in),
    .data_in(y_read),
    .data_out(y_write));
pipeline #(.N_BITS(12), .N_REGISTERS(2 + DIVISION_LATENCY)) pipeline_rgb(
    .clk_in(clk_in),
    .rst_in(rst_in),
    .data_in(rgb_in),
    .data_out(rgb_write));

assign write_ram = (z_write > z_read) && in_triangle_lag && busy_lag;
assign finished = ~busy && last_busy;
always_ff @(posedge clk_in) begin
    if (rst_in) begin
        busy <= 0;
        x_cur <= 0;
        y_cur <= 0;
    end else if (~busy) begin
        busy <= new_data;
        x_cur <= x_min;
        y_cur <= y_min;
    end else begin
        x_cur <= ((x_cur == x_max) ? x_min : (x_cur + 1));
```

```
        y_cur <= (x_cur == x_max) ? (y_cur + 1) : y_cur;
        busy <= ~(x_cur == x_max && y_cur == y_max);
      end
      numerator <= area1 * $signed(vertices[6]) +
            area2 * $signed(vertices[3]) +
            area3 * $signed(vertices[0]);
      denominator <= area_total;
      in_triangle <= (area1 >= 0 && area2 >= 0 && area3 >= 0) || (area1 <= 0 && area2 <= 0
&& area3 <= 0);
      last_busy <= busy;
   end
endmodule

module get_max(
   input signed [11:0] val1,
   input signed [11:0] val2,
   input signed [11:0] val3,
   output logic signed [11:0] max
);
   parameter ABSOLUTE_MIN = 12'sb0;
   parameter ABSOLUTE_MAX = 12'sd1000;
   always_comb begin
      max = val1;
      if ($signed(val2) > $signed(max)) max = val2;
      if ($signed(val3) > $signed(max)) max = val3;
      if ($signed(ABSOLUTE_MAX) < $signed(max)) max = ABSOLUTE_MAX;
      if ($signed(ABSOLUTE_MIN) > $signed(max)) max = ABSOLUTE_MIN;


   end
endmodule

module get_min(
   input signed [11:0] val1,
   input signed [11:0] val2,
   input signed [11:0] val3,
   output logic signed [11:0] min
);
   parameter ABSOLUTE_MIN = 12'sb0;
   parameter ABSOLUTE_MAX = 12'sd1000;
   always_comb begin
      min = val1;
      if ($signed(val2) < $signed(min)) min = val2;
      if ($signed(val3) < $signed(min)) min = val3;
```

```
        if ($signed(ABSOLUTE_MAX) < $signed(min)) min = ABSOLUTE_MAX;
        if ($signed(ABSOLUTE_MIN) > $signed(min)) min = ABSOLUTE_MIN;

    end
endmodule

module get_area(
    input clk_in,
    input signed [11:0] x1,
    input signed [11:0] y1,
    input signed [11:0] x2,
    input signed [11:0] y2,
    input signed [11:0] x3,
    input signed [11:0] y3,
    output logic signed [23:0] area);

    logic signed [11:0] v1x;
    logic signed [11:0] v1y;
    logic signed [11:0] v2x;
    logic signed [11:0] v2y;
    assign v1x = x2 - x1;
    assign v1y = y2 - y1;
    assign v2x = x3 - x1;
    assign v2y = y3 - y1;
    always_ff @(posedge clk_in) begin
        area <= v1x * v2y - v2x * v1y;
    end
endmodule
```

projection.sv

```
`timescale 1ns / 1ps

module projection(
    input clk_in,
    input rst_in,
    input signed [8:0][11:0] vertices_in,
    input signed [2:0][11:0] user_in,
    input new_data_in,
    output logic signed [8:0][11:0] vertices_out,
    output logic finished_out
);
    // Project each vertex separately
    parameter DIVIDER_LATENCY = 21;
```

```
    project_vertex vertex1(
        .clk_in(clk_in), .rst_in(rst_in),
        .vertex_in(vertices_in[8:6]),
        .user_in(user_in),
        .new_data_in(new_data_in),
        .vertex_out(vertices_out[8:6])
    );
    project_vertex vertex2(
        .clk_in(clk_in), .rst_in(rst_in),
        .vertex_in(vertices_in[5:3]),
        .user_in(user_in),
        .new_data_in(new_data_in),
        .vertex_out(vertices_out[5:3])
    );
    project_vertex vertex3(
        .clk_in(clk_in), .rst_in(rst_in),
        .vertex_in(vertices_in[2:0]),
        .user_in(user_in),
        .new_data_in(new_data_in),
        .vertex_out(vertices_out[2:0])
    );

    pipeline #(.N_BITS(1), .N_REGISTERS(3 + DIVIDER_LATENCY)) pipeline_finished (
        .clk_in(clk_in), .rst_in(rst_in),
        .data_in(new_data_in),
        .data_out(finished_out)
    );

endmodule

module project_vertex(
    input clk_in,
    input rst_in,
    input signed [2:0][11:0] vertex_in,
    input signed [2:0][11:0] user_in,
    input new_data_in,
    output logic signed [2:0][11:0] vertex_out
);

    parameter SCREEN_WIDTH = 400;
    parameter SCREEN_HEIGHT = 400;

    // Registers to store inputs
    logic signed [11:0] vx, vy, vz, ux, uy, uz;
```

```
// Divider inputs / outputs
logic signed [23:0] numerator_x, numerator_y;
logic signed [15:0] denominator;
logic signed [23:0] divider_out_x, divider_out_y;
logic divider_valid_in, divider_x_valid_out, divider_y_valid_out;

pipeline #(.N_BITS(1), .N_REGISTERS(2)) pipeline_divider_valid(
    .clk_in(clk_in), .rst_in(rst_in),
    .data_in(new_data_in), .data_out(divider_valid_in));

projection_divider divider_x (
  .aclk(clk_in),                                // input wire aclk
  .s_axis_divisor_tvalid(divider_valid_in),    // input wire s_axis_divisor_tvalid
  .s_axis_divisor_tready(ready1),    // output wire s_axis_divisor_tready
  .s_axis_divisor_tdata(denominator),      // input wire [15 : 0] s_axis_divisor_tdata
  .s_axis_dividend_tvalid(divider_valid_in),  // input wire s_axis_dividend_tvalid
  .s_axis_dividend_tready(ready2),  // output wire s_axis_dividend_tready
  .s_axis_dividend_tdata(numerator_x),    // input wire [23 : 0] s_axis_dividend_tdata
  .m_axis_dout_tvalid(divider_x_valid_out),          // output wire m_axis_dout_tvalid
  .m_axis_dout_tdata(divider_out_x)            // output wire [23 : 0] m_axis_dout_tdata
);

projection_divider divider_y (
  .aclk(clk_in),                                // input wire aclk
  .s_axis_divisor_tvalid(divider_valid_in),    // input wire s_axis_divisor_tvalid
  .s_axis_divisor_tready(ready3),    // output wire s_axis_divisor_tready
  .s_axis_divisor_tdata(denominator),      // input wire [15 : 0] s_axis_divisor_tdata
  .s_axis_dividend_tvalid(divider_valid_in),  // input wire s_axis_dividend_tvalid
  .s_axis_dividend_tready(ready4),  // output wire s_axis_dividend_tready
  .s_axis_dividend_tdata(numerator_y),    // input wire [23 : 0] s_axis_dividend_tdata
  .m_axis_dout_tvalid(divider_y_valid_out),          // output wire m_axis_dout_tvalid
  .m_axis_dout_tdata(divider_out_y)            // output wire [23 : 0] m_axis_dout_tdata
);

always_ff @(posedge clk_in) begin
    if (rst_in) begin
        vx <= 0;
        vy <= 0;
        vz <= 0;
        ux <= 0;
        uy <= 0;
        uz <= 0;
        vertex_out <= 0;
```

```
        end else if (new_data_in) begin
            // Save inputs in local register on new_data_in
            vx <= $signed(vertex_in[2]);
            vy <= $signed(vertex_in[1]);
            vz <= $signed(vertex_in[0]);
            ux <= $signed(user_in[2]);
            uy <= $signed(user_in[1]);
            uz <= $signed(user_in[0]);
        end else begin
            // vertex_out_x = (vz * ux - uz * vx) / (vz - uz)
            // vertex_out_y = (vz * uy - uz * vy) / (vz - uz)
            // vertex_out_z = vz
            numerator_x <= vz * ux - uz * vx;
            numerator_y <= vz * uy - uz * vy;
            denominator <= vz - uz;
            vertex_out[2] <= divider_x_valid_out ? (divider_out_x + SCREEN_WIDTH / 2) :
vertex_out[2];
            vertex_out[1] <= divider_y_valid_out ? (divider_out_y + SCREEN_HEIGHT / 2) :
vertex_out[1];
            vertex_out[0] <= vz;

        end
    end
endmodule
```

graphics_fsm.sv

```
`timescale 1ns / 1ps

module graphics_fsm(
    input clk_in,
    input rst_in,
    input finish_shader,
    input finish_rasterize,
    input finish_projection,
    input finish_transform,
    input data_available_triangle_source,
    input next_frame,
    output logic next_triangle,
    output logic new_data_projection,
    output logic new_data_rasterize,
    output logic new_data_transform
);
```

```
    // Has finished signals (set to high when module finishes and set to low when module
receives new data)
    logic has_finished_rasterize;
    logic has_finished_projection;
    logic has_finished_shader;
    logic has_finished_transform;

    // Data availble signals (none for shader since always identical to shader)
    logic new_data;
    logic data_available_projection;
    logic data_available_rasterize;
    logic data_available_transform;

    // Next triangle once all modules have finished
    assign next_triangle = has_finished_rasterize && has_finished_projection &&
has_finished_shader && has_finished_transform;
    // New data signal sent to a module three clock cyles after next_triangle or new frame if the
modulue has data available
    assign new_data_projection = new_data && data_available_projection;
    assign new_data_rasterize = new_data && data_available_rasterize;
    assign new_data_transform = new_data && data_available_transform;

    pipeline #(.N_BITS(1), .N_REGISTERS(3)) pipeline_next_tri(
        .clk_in(clk_in), .rst_in(rst_in),
        .data_in(next_triangle || next_frame), .data_out(new_data)
    );


    always_ff @(posedge clk_in) begin
        if (rst_in) begin
                has_finished_transform <= 1;
            has_finished_rasterize <= 1;
            has_finished_projection <= 1;
            has_finished_shader <= 1;
            data_available_transform <= 0;
            data_available_rasterize <= 0;
            data_available_projection <= 0;
        end else if (next_frame) begin
            // On new frame data only available for first module (transform)
            data_available_transform <= 1;
            has_finished_transform <= 0;

            data_available_rasterize <= 0;
            has_finished_rasterize <= 1;
```

```
            data_available_projection <= 0;
            has_finished_projection <= 1;
            has_finished_shader <= 1;

        end else if (next_triangle) begin
            // On next triangle, data avaible for a given module if was available for the previous
module in the previous trianlge
            data_available_transform <= data_available_triangle_source;
            has_finished_transform <= ~data_available_triangle_source;

            data_available_projection <= data_available_transform;
            has_finished_projection <= ~data_available_transform;
            has_finished_shader <= ~data_available_transform;

            data_available_rasterize <= data_available_projection;
            has_finished_rasterize <= ~data_available_projection;
            has_finished_shader <= ~data_available_projection;
        end else begin
            has_finished_rasterize <= has_finished_rasterize || finish_rasterize;
            has_finished_projection <= has_finished_projection || finish_projection;
            has_finished_shader <= has_finished_shader || finish_shader;
            has_finished_transform <= has_finished_transform || finish_transform;
        end


    end

endmodule
```

framebuffer_manager.sv

```
`timescale 1ns / 1ps
// Takes two clock cycles to read
module frame_buffer_manager(
    input clk_in,
    input rst_in,
    input next_frame,
    // Inactive
    input write_inactive_frame,
    input [11:0] x_read_inactive_frame,
    input [11:0] y_read_inactive_frame,
    input [11:0] x_write_inactive_frame,
    input [11:0] y_write_inactive_frame,
```

```
input [11:0] rgb_write_inactive_frame,
input signed [11:0] z_write_inactive_frame,
output logic signed [11:0] z_read_inactive_frame,
// Active
input [11:0] hcount_in,
input [11:0] vcount_in,
output logic [11:0] rgb_active_frame

);
// Overriden by graphics subsystem
parameter SCREEN_WIDTH = 0;
parameter SCREEN_HEIGHT = 0;

// Dimensions of entire projection
parameter VGA_WIDTH = 640;
parameter VGA_HEIGHT = 480;

logic signed [11:0] x_active_frame;
logic signed [11:0] y_active_frame;
logic signed [11:0] last_x_active_frame;
logic signed [11:0] last_y_active_frame;
logic [17:0] address_write_active, address_write_inactive;
logic [17:0] address_read_active, address_read_inactive;
logic [23:0] data_write_active, data_write_inactive;
logic [23:0] data_read [1:0];
logic write_active_frame;
logic active_frame;
logic pixel_in_frame;

// Shift since we are not using entire screen and want to use the middle of the screen
assign x_active_frame = hcount_in - VGA_WIDTH / 2 + SCREEN_WIDTH / 2;
assign y_active_frame = vcount_in - VGA_HEIGHT / 2 + SCREEN_HEIGHT / 2;

// Pipeline by 4 since VGA clock is 4 times faster than clock
pipeline #(.N_BITS(12), .N_REGISTERS(4)) pipeline_x_active(
    .clk_in(clk_in),
    .rst_in(rst_in),
    .data_in(x_active_frame),
    .data_out(last_x_active_frame));
pipeline #(.N_BITS(12), .N_REGISTERS(4)) pipeline_y_active(
    .clk_in(clk_in),
    .rst_in(rst_in),
    .data_in(y_active_frame),
    .data_out(last_y_active_frame));
```

```
   // Two framebuffers, one of which is active and one is inactive at a time
   frame_buffer buffer0(
     .clka(clk_in),           // input wire clka
     .wea((active_frame == 0) ? write_active_frame : write_inactive_frame),             // input
wire [0 : 0] wea
     .addra((active_frame == 0) ? address_write_active : address_write_inactive),         // input
wire [16 : 0] addra
     .dina((active_frame == 0) ? data_write_active : data_write_inactive),           // input wire
[19 : 0] dina
     .clkb(clk_in),           // input wire clkb
     .addrb((active_frame == 0) ? address_read_active : address_read_inactive),         // input
wire [16 : 0] addrb
     .doutb(data_read[0])         // output wire [19 : 0] doutb
   );

   frame_buffer buffer1(
     .clka(clk_in),           // input wire clka
     .wea((active_frame == 1) ? write_active_frame : write_inactive_frame),             // input
wire [0 : 0] wea
     .addra((active_frame == 1) ? address_write_active : address_write_inactive),         // input
wire [16 : 0] addra
     .dina((active_frame == 1) ? data_write_active : data_write_inactive),           // input wire
[19 : 0] dina
     .clkb(clk_in),           // input wire clkb
     .addrb((active_frame == 1) ? address_read_active : address_read_inactive),         // input
wire [16 : 0] addrb
     .doutb(data_read[1])         // output wire [19 : 0] doutb
   );

   assign address_write_active = last_x_active_frame + last_y_active_frame *
SCREEN_WIDTH; // write based on previous hcount / vcount
   assign address_read_active = x_active_frame + y_active_frame * SCREEN_WIDTH; //
read based on hcount and vcount
   assign data_write_active = 24'h000800; // Reset after reading
   assign rgb_active_frame = pixel_in_frame ? data_read[active_frame][23:12] : 12'h000; //
Black if outside screen

   assign address_write_inactive = x_write_inactive_frame +
      y_write_inactive_frame * SCREEN_WIDTH;
   assign address_read_inactive = x_read_inactive_frame +
      y_read_inactive_frame * SCREEN_WIDTH;
   assign data_write_inactive = {rgb_write_inactive_frame, z_write_inactive_frame};
   assign z_read_inactive_frame = data_read[~active_frame][11:0];
```

```
   assign write_active_frame = (last_x_active_frame < SCREEN_WIDTH) &&
(last_x_active_frame >= 0) &&
      (last_y_active_frame >= 0) && (last_y_active_frame < SCREEN_HEIGHT);

   pipeline #(.N_BITS(1), .N_REGISTERS(2)) pipeline_pix_in_frame (
      .clk_in(clk_in), .rst_in(rst_in),
      .data_in((x_active_frame < SCREEN_WIDTH) && (x_active_frame >= 0) &&
               (y_active_frame >= 0) && (y_active_frame < SCREEN_HEIGHT)),
      .data_out(pixel_in_frame)
   );

   always_ff @(posedge clk_in) begin
      if (rst_in) begin
         active_frame <= 0;
      end else begin
         // on next frame switch which which frame is active
         active_frame <= next_frame ? ~active_frame : active_frame;
      end

   end

endmodule
```

game_logic.sv

```
`timescale 1ns / 1ps

module game_logic(
   input clk_in,
   input clk_5sec,
   input rst_in,
   input [10:0] centroid_x,
   input [9:0] centroid_y,
   input saber_detected,
   input next_frame,
   output logic signed [1:0][2:0][11:0] model_trans,
   output logic signed [1:0][2:0][11:0] rpy,
   output logic signed [1:0][2:0][11:0] world_trans,
   output logic [1:0] game_state
   );

   // Parameters for motion arc
   parameter Z_MIN = -12'sd1000;
```

```
    parameter Z_MAX = 12'sd200;
    parameter Z_MIN_SWIPE = 12'sd100;
    parameter TIME_OF_FLIGHT_FRAMES = 32'sd240; // 6 sec

    // reciprocal of time of flight so we can divide by TOF
    // 1 / TOF = TOF_M / 2^TOF_N
    parameter TOF_N = 16;
    parameter TOF_M = $signed((2**TOF_N) / TIME_OF_FLIGHT_FRAMES);

    // Thanos Arc
    logic [9:0] frame_count;
    logic signed [19:0] cur_time;
    logic signed [63:0]delta_z;
    logic signed [11:0] z_model;

    // Thanos breaking in half
    logic saber_moving;
    logic signed [11:0] separation;
    logic did_swipe_fruit; // used to be fruit, now thanos

    // Spin model
    logic [11:0] model_spin = 0;
    logic [23:0] spin_counter = 0;

    assign delta_z = ($signed(cur_time) * $signed(cur_time) * $signed(Z_MIN - Z_MAX)) >>>
(2 * TOF_N);

    detect_motion detect(
        .clk_in(clk_in),
        .rst_in(rst_in),
        .centroid_x(centroid_x),
        .centroid_y(centroid_y),
        .saber_detected(saber_detected),
        .saber_moving(saber_moving)
    );
    assign model_trans = 0;
    always_ff @(posedge clk_in) begin
        // Arc of Thanos
        cur_time <= (16'sd2 * $signed(frame_count) - TIME_OF_FLIGHT_FRAMES) *
TOF_M;
        z_model <= delta_z + Z_MAX;

        if (rst_in) begin
            frame_count <= 0;
```

```
            did_swipe_fruit <= 0;
            rpy[0] <= 0; //model 1
            rpy[1] <= 0; //model 2
            game_state <= 0;

            separation <= 0;
        end else if (next_frame && (frame_count < TIME_OF_FLIGHT_FRAMES)) begin
            frame_count <= frame_count + 10'b1;
            did_swipe_fruit <= did_swipe_fruit || ((z_model > Z_MIN_SWIPE) &&
saber_moving);
            separation <= did_swipe_fruit ? (separation + 1) : separation;
            //SET TRANSFORMATIONS
            //If in game state, set game transforms, sipe object transformations
            rpy[0][2] <= game_state==2'b11 ? (separation * 12'sh008):0;
            rpy[0][0] <= game_state==2'b11 ? (rpy[0] + 12'h00D): game_state!==2'b10 ? (rpy[0] +
model_spin):0;
            rpy[1][2] <= game_state==2'b11 ? (separation * 12'sh008):0;
            rpy[1][0] <= game_state==2'b11 ? (rpy[0] + 12'h00D): game_state!==2'b10 ? (rpy[0] +
model_spin):0;
            rpy[0][1] <= 0;
            rpy[1][1] <= 0;
            world_trans[0] <= game_state==2'b11 ? {separation, 12'h0, z_model}:0;
            world_trans[1] <= game_state==2'b11 ? {-separation, 12'h0, z_model}:0;
        end else if (next_frame) begin
            // At end of arc put thanos back together
            did_swipe_fruit <= 1'b0;
            separation <= 1'b0;
            frame_count <= 10'b0;

        end

        //Game FSM
        //Enter Game
        if (game_state==2'b10 && did_swipe_fruit) begin
            game_state <= 2'b11;
        //Start model spinning
        end else if ( (game_state==2'b00 || game_state==2'b01) && did_swipe_fruit) begin
            model_spin <= 12'd30;
        //Slow down model spinning
        end else if ( (game_state==2'b00 || game_state==2'b01) && model_spin>0 &&
spin_counter<=0) begin
            model_spin <= model_spin - 1;
            spin_counter <= 24'd10000000;
        end else if ((game_state==2'b00 || game_state==2'b01) && model_spin>0) begin
```

```
        spin_counter <= spin_counter -1;
    end
    //Switch between Models
    if (game_state!=2'b11 && clk_5sec) begin
        game_state <= game_state==2'b10 ? 2'b00:(game_state+1);
    end
  end




endmodule
```

detect_motion.sv

```
`timescale 1ns / 1ps

// Detects motion of hand (previously a saber)
module detect_motion(
  input clk_in,
  input rst_in,
  input [10:0] centroid_x,
  input [9:0] centroid_y,
  input saber_detected,
  output logic saber_moving
  );

  parameter CYCLES_PER_SAMPLE = 20000000; // 200 ms, how frequently we sample
centroid
  parameter PIXEL_THRESHOLD = 12'd25; // About 1 foot at chest height, how far hand
must move in 200 ms to count as motion


  logic [31:0] count;

  // Keep track of last saber detected / centroid values
  logic last_saber_detected;
  logic [10:0] last_centroid_x;
  logic [9:0] last_centroid_y;

  // Calculate distance moved since last time
  logic [10:0] delta_centroid_x;
  logic [9:0] delta_centroid_y;
  logic [19:0] dist_sq;
```

```
    // Pipelined signals (since calculating distance takes multiple clock cycles)
    logic [10:0] lag_centroid_x;
    logic [9:0] lag_centroid_y;
    logic lag_saber_detected;

    pipeline #(.N_BITS(11), .N_REGISTERS(2)) pipeline_x(
        .clk_in(clk_in), .rst_in(rst_in),
        .data_in(centroid_x), .data_out(lag_centroid_x)
    );
    pipeline #(.N_BITS(10), .N_REGISTERS(2)) pipeline_y(
        .clk_in(clk_in), .rst_in(rst_in),
        .data_in(centroid_y), .data_out(lag_centroid_y)
    );
    pipeline #(.N_BITS(1), .N_REGISTERS(2)) pipeline_detected(
        .clk_in(clk_in), .rst_in(rst_in),
        .data_in(saber_detected), .data_out(lag_saber_detected)
    );

    always_ff @(posedge clk_in) begin
        if (rst_in) begin
            count <= 32'b0;
            last_saber_detected <= 1'b0;
            last_centroid_x <= 11'b0;
            last_centroid_y <= 10'b0;
            saber_moving <= 1'b0;
        end else if (count == CYCLES_PER_SAMPLE - 3) begin
            // On third to last  clock cycles calculate displacement
            count <= count + 32'b1;
            delta_centroid_x <= (centroid_x > last_centroid_x) ? (centroid_x - last_centroid_x) :
(last_centroid_x - centroid_x);
            delta_centroid_y <= (centroid_y > last_centroid_y) ? (centroid_y - last_centroid_y) :
(last_centroid_y - centroid_y);
        end else if (count == CYCLES_PER_SAMPLE - 2) begin
            // On second to last clock cycle calculate distance squared
            count <= count + 32'b1;
            dist_sq <= (delta_centroid_x * delta_centroid_x + delta_centroid_y *
delta_centroid_y);
        end else if (count == CYCLES_PER_SAMPLE - 1) begin
            // On last clock cycle calculate whether it moved enough
            count <= 32'b0;
            last_saber_detected <= lag_saber_detected;
            last_centroid_x <= lag_centroid_x;
            last_centroid_y <= lag_centroid_y;
```

```
        saber_moving <= lag_saber_detected && last_saber_detected && (dist_sq >
PIXEL_THRESHOLD * PIXEL_THRESHOLD);
      end else begin
        count <= count + 32'b1;
      end
   end
endmodule
```

display_height.sv

```
`timescale 1ns / 1ps

// Much unused code here, now pretty much just sets height to a constant
// Previously was used to allow user to adjust height using switches on Nexys DDR4 and
display height on hex display
module display_height(
    input clk_in,
    input rst_in,
    input [7:0] sw,
    output logic signed [11:0] height,
   output reg [6:0] seg_out,     // seven segment display output
   output logic dp,
    output reg [7:0] strobe_out);   // digit strobe



   logic [3:0] height_ft, height_inches_ones;
   logic [4:0] height_inches;
   assign height_ft = (sw[5:1] > 24) ? 4'd6 : (sw[5:1] > 12) ? 4'd5 : 4'd4;
   assign height_inches_ones = (height_inches[4:1] > 9) ? (height_inches[4:1] - 10) :
height_inches[4:1];
   assign height_inches = sw[5:0] - 24 * (height_ft - 4);
   parameter TABLE_HEIGHT_INCHES = 26;
   parameter MIN_USER_HEIGHT_INCHES = 48;
   parameter PIXELS_PER_INCH = 640 / 28;


   localparam bits = 13;

    reg [bits:0] counter = 0;  // clear on power up

    wire [6:0] segments[15:0]; // 16 7 bit memorys
    assign segments[0]  = 7'b100_0000;  // inverted logic
    assign segments[1]  = 7'b111_1001;  // gfedcba
```

```
   assign segments[2]  = 7'b010_0100;
   assign segments[3]  = 7'b011_0000;
   assign segments[4]  = 7'b001_1001;
   assign segments[5]  = 7'b001_0010;
   assign segments[6]  = 7'b000_0010;
   assign segments[7]  = 7'b111_1000;
   assign segments[8]  = 7'b000_0000;
   assign segments[9]  = 7'b001_1000;
   assign segments[10] = 7'b000_1000;
   assign segments[11] = 7'b000_0011;
   assign segments[12] = 7'b010_0111;
   assign segments[13] = 7'b010_0001;
   assign segments[14] = 7'b000_0110;
   assign segments[15] = 7'b000_1110;

   always_ff @(posedge clk_in) begin
     // Here I am using a counter and select 3 bits which provides
     // a reasonable refresh rate starting the left most digit
     // and moving left.
       height <= ((sw[5:0] * PIXELS_PER_INCH) >> 1) + PIXELS_PER_INCH *
(MIN_USER_HEIGHT_INCHES - TABLE_HEIGHT_INCHES);

     counter <= counter + 1;
     case (counter[bits:bits-2])
        3'b000: begin  // use the MSB 4 bits
              seg_out <= 7'b111_1111; // Unused
              strobe_out <= 8'b0111_1111 ;
              dp <= 1;
             end

        3'b001: begin
             seg_out <= 7'b111_1111; // Unused
             strobe_out <= 8'b1011_1111 ;
             dp <= 1;
              end

        3'b010: begin
               seg_out <= segments[height_ft];
               strobe_out <= 8'b1101_1111 ;
               dp <= 1;
              end
        3'b011: begin
              seg_out <= 8'b101_1111;
              strobe_out <= 8'b1110_1111;
```

```
            dp <= 1;
           end
       3'b100: begin
            seg_out <= height_inches[4:1] > 9 ? segments[1] : 7'b111_1111;
            strobe_out <= 8'b1111_0111;
            dp <= 1;

           end

       3'b101: begin
            seg_out <= segments[height_inches_ones];
            strobe_out <= 8'b1111_1011;
            dp <= 0;
           end

       3'b110: begin
             seg_out <= height_inches[0] ? segments[5] : segments[0];
             strobe_out <= 8'b1111_1101;
             dp <= 1;
           end
       3'b111: begin
            seg_out <= 7'b1011101;
            strobe_out <= 8'b1111_1110;
            dp <= 1;
           end

     endcase
    end

endmodule
```

5sec_clock.sv

```
`timescale 1ns / 1ps

module five_sec_clk(
input clk_in,
input rst_in,
output logic clk_5sec
   );
parameter TIMER = 40'd500000000;
logic [39:0] counter = 0;

always_ff @(posedge clk_in) begin
```

```
    if (rst_in) begin
        counter = 32'd0;
        clk_5sec <= 0;
    end else if (counter >= TIMER) begin
        counter <= 0;
        clk_5sec <= 1;
    end else begin
        counter <= counter+1;
        clk_5sec <= 0;
    end
end
endmodule
```

cv2render.sv

```
module cv2render(
    input [10:0] blob_x,
    input [9:0] blob_y,
    input next_frame,
    input signed [11:0] user_z,
    input clk_in,
    output logic signed [2:0][11:0] user = 0
    );
parameter cam_x = 320;
parameter cam_y = 240;
//fov_ratio = M/2^N
//(proj_x*measure_cam)/(cam_x*measure_proj)
parameter fov_ratio_m = 12'sd9;
parameter fov_ratio_n = 1;
//Weight of filter
parameter fir_alpha = 16'sh2; // 4 bits
parameter fir_n = 4;

logic [10:0] blob_x_synced;
logic [9:0] blob_y_synced;

pipeline #(.N_BITS(11), .N_REGISTERS(3)) pipelinex (
    .clk_in(clk_in), .rst_in(1'b0),
    .data_in(blob_x), .data_out(blob_x_synced));
pipeline #(.N_BITS(10), .N_REGISTERS(2)) pipeliney (
    .clk_in(clk_in), .rst_in(1'b0),
    .data_in(blob_y), .data_out(blob_y_synced));
//Shift blob location origin to center
logic signed [11:0] blob_x_shifted;
```

```
logic signed [11:0] blob_y_shifted;
assign blob_x_shifted = $signed(blob_x_synced) - $signed(cam_x/2);
assign blob_y_shifted = $signed(blob_y_synced) - $signed(cam_y/2);
//Fill Buffer with X, Y, Z
logic signed [2:0] [11:0] buffer = 0;
logic signed [2:0] [11:0] diff = 0;
logic signed [2:0] [11:0] post_fir = 0;


always_ff @(posedge clk_in) begin
   //Scale user position into graphics reference frame
   buffer [2] <= (fov_ratio_m * blob_x_shifted) >>> fov_ratio_n;
   buffer [1] <= (fov_ratio_m * blob_y_shifted) >>> fov_ratio_n;
   //Apply Filter
   diff[2] <= (($signed(buffer[2]) - $signed(user[2])) * fir_alpha) >>> fir_n;
   diff[1] <= (($signed(buffer[1]) - $signed(user[1])) * fir_alpha) >>> fir_n;
   post_fir[2] <= $signed(diff[2]) + $signed(user[2]);
   post_fir[1] <= $signed(diff[1]) + $signed(user[1]);
   post_fir[0] <= user_z;
   if (next_frame) begin
      user <= post_fir;
   end
end
endmodule
```

xvga.sv

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
// 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,
//           input vclock_enable, // used if input clock frequency does not match desired
frequency (can still maintain one clock domain)
           input rst_in,
           output reg [11:0] hcount_out = 0,    // pixel number on current line
           output reg [11:0] vcount_out = 0,     // line number
           output reg vsync_out = 0, hsync_out = 0,
           output reg blank_out = 0);

   parameter DISPLAY_WIDTH  = 640;      // display width
   parameter DISPLAY_HEIGHT = 480;       // number of lines

   parameter  H_FP = 16;                // horizontal front porch
   parameter  H_SYNC_PULSE = 96;        // horizontal sync
   parameter  H_BP = 48;                // horizontal back porch

   parameter  V_FP = 11;                 // vertical front porch
   parameter  V_SYNC_PULSE = 2;          // vertical sync
   parameter  V_BP = 31;                 // vertical back porch

   // horizontal: 1344 pixels total
   // display 1024 pixels per line
   reg hblank = 0,vblank = 0;
   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
        if (rst_in) begin
            hcount_out <= 0;
            hblank <= 0;
            hsync_out <= 1;

            vcount_out <= 0;
            vblank <= 0;
            vsync_out <= 1;
            blank_out <= 0;
        end else 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
    end

endmodule
```

pipeline.sv

```
//used to delay registers if some adjacent data needs to pass through a module with latency
`timescale 1ns / 1ps

module pipeline(
    input clk_in,
    input rst_in,
    input [(N_BITS-1):0] data_in,
    output logic [(N_BITS-1):0] data_out);
    parameter N_BITS = 1;
    parameter N_REGISTERS = 2; // If just one register do manually
```

```
    logic [(N_BITS * N_REGISTERS - 1):0] buffer ;
    always_ff @(posedge clk_in) begin
        if (rst_in) begin
            buffer <= 0;
        end else begin
            buffer <= {data_in, buffer[(N_BITS * N_REGISTERS - 1):N_BITS]};
        end

    end
    assign data_out = buffer[(N_BITS - 1):0];

endmodule
```

Nexys-Video-Master.xdc

```
### This file is a general .xdc for the Nexys Video Rev. A
### To use it in a project:
### - uncomment the lines corresponding to used pins
### - rename the used ports (in each line, after get_ports) according to the top level signal
names in the project


## Clock Signal
set_property -dict { PACKAGE_PIN R4    IOSTANDARD LVCMOS33 } [get_ports { clk }];
#IO_L13P_T2_MRCC_34 Sch=sysclk
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

## FMC Transceiver clocks (Must be set to value provided by Mezzanine card, currently set to
156.25 MHz)
## Note: This clock is attached to a MGTREFCLK pin
#set_property -dict { PACKAGE_PIN E6 } [get_ports { GTP_CLK_N }];
#set_property -dict { PACKAGE_PIN F6 } [get_ports { GTP_CLK_P }];
#create_clock -add -name gtpclk0_pin -period 6.400 -waveform {0 3.200} [get_ports
{GTP_CLK_P}];
#set_property -dict { PACKAGE_PIN E10 } [get_ports { FMC_MGT_CLK_N }];
#set_property -dict { PACKAGE_PIN F10 } [get_ports { FMC_MGT_CLK_P }];
#create_clock -add -name mgtclk1_pin -period 6.400 -waveform {0 3.200} [get_ports
{FMC_MGT_CLK_P}];


## LEDs
#set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS25 } [get_ports {
led[0] }]; #IO_L15P_T2_DQS_13 Sch=led[0]
```

```
#set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS25 } [get_ports {
led[1] }]; #IO_L15N_T2_DQS_13 Sch=led[1]
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS25 } [get_ports {
led[2] }]; #IO_L17P_T2_13 Sch=led[2]
#set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS25 } [get_ports {
led[3] }]; #IO_L17N_T2_13 Sch=led[3]
#set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS25 } [get_ports {
led[4] }]; #IO_L14N_T2_SRCC_13 Sch=led[4]
#set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS25 } [get_ports {
led[5] }]; #IO_L16N_T2_13 Sch=led[5]
#set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS25 } [get_ports {
led[6] }]; #IO_L16P_T2_13 Sch=led[6]
#set_property -dict { PACKAGE_PIN Y13   IOSTANDARD LVCMOS25 } [get_ports {
led[7] }]; #IO_L5P_T0_13 Sch=led[7]


## Buttons
#set_property -dict { PACKAGE_PIN B22 IOSTANDARD LVCMOS12 } [get_ports { btnc
}]; #IO_L20N_T3_16 Sch=btnc
#set_property -dict { PACKAGE_PIN D22 IOSTANDARD LVCMOS12 } [get_ports { btnd
}]; #IO_L22N_T3_16 Sch=btnd
#set_property -dict { PACKAGE_PIN C22 IOSTANDARD LVCMOS12 } [get_ports { btnl
}]; #IO_L20P_T3_16 Sch=btnl
#set_property -dict { PACKAGE_PIN D14 IOSTANDARD LVCMOS12 } [get_ports { btnr
}]; #IO_L6P_T0_16 Sch=btnr
#set_property -dict { PACKAGE_PIN F15 IOSTANDARD LVCMOS12 } [get_ports { btnu
}]; #IO_0_16 Sch=btnu
#set_property -dict { PACKAGE_PIN G4  IOSTANDARD LVCMOS15 } [get_ports {
cpu_resetn }]; #IO_L12N_T1_MRCC_35 Sch=cpu_resetn


## Switches
set_property -dict { PACKAGE_PIN E22  IOSTANDARD LVCMOS12 } [get_ports { sw[0]
}]; #IO_L22P_T3_16 Sch=sw[0]
set_property -dict { PACKAGE_PIN F21  IOSTANDARD LVCMOS12 } [get_ports { sw[1]
}]; #IO_25_16 Sch=sw[1]
set_property -dict { PACKAGE_PIN G21  IOSTANDARD LVCMOS12 } [get_ports { sw[2]
}]; #IO_L24P_T3_16 Sch=sw[2]
set_property -dict { PACKAGE_PIN G22  IOSTANDARD LVCMOS12 } [get_ports { sw[3]
}]; #IO_L24N_T3_16 Sch=sw[3]
set_property -dict { PACKAGE_PIN H17  IOSTANDARD LVCMOS12 } [get_ports { sw[4]
}]; #IO_L6P_T0_15 Sch=sw[4]
set_property -dict { PACKAGE_PIN J16  IOSTANDARD LVCMOS12 } [get_ports { sw[5]
}]; #IO_0_15 Sch=sw[5]
```

```
set_property -dict { PACKAGE_PIN K13  IOSTANDARD LVCMOS12 } [get_ports { sw[6] }]; #IO_L19P_T3_A22_15 Sch=sw[6]
set_property -dict { PACKAGE_PIN M17  IOSTANDARD LVCMOS12 } [get_ports { sw[7] }]; #IO_25_15 Sch=sw[7]


## OLED Display
#set_property -dict { PACKAGE_PIN W22   IOSTANDARD LVCMOS33 } [get_ports { oled_dc }]; #IO_L7N_T1_D10_14 Sch=oled_dc
#set_property -dict { PACKAGE_PIN U21   IOSTANDARD LVCMOS33 } [get_ports { oled_res }]; #IO_L4N_T0_D05_14 Sch=oled_res
#set_property -dict { PACKAGE_PIN W21   IOSTANDARD LVCMOS33 } [get_ports { oled_sclk }]; #IO_L7P_T1_D09_14 Sch=oled_sclk
#set_property -dict { PACKAGE_PIN Y22   IOSTANDARD LVCMOS33 } [get_ports { oled_sdin }]; #IO_L9N_T1_DQS_D13_14 Sch=oled_sdin
#set_property -dict { PACKAGE_PIN P20   IOSTANDARD LVCMOS33 } [get_ports { oled_vbat }]; #IO_0_14 Sch=oled_vbat
#set_property -dict { PACKAGE_PIN V22   IOSTANDARD LVCMOS33 } [get_ports { oled_vdd }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=oled_vdd


## HDMI in
#set_property -dict { PACKAGE_PIN AA5   IOSTANDARD LVCMOS33 } [get_ports { hdmi_rx_cec }]; #IO_L10P_T1_34 Sch=hdmi_rx_cec
#set_property -dict { PACKAGE_PIN W4    IOSTANDARD TMDS_33  } [get_ports { hdmi_rx_clk_n }]; #IO_L12N_T1_MRCC_34 Sch=hdmi_rx_clk_n
#set_property -dict { PACKAGE_PIN V4    IOSTANDARD TMDS_33  } [get_ports { hdmi_rx_clk_p }]; #IO_L12P_T1_MRCC_34 Sch=hdmi_rx_clk_p
#set_property -dict { PACKAGE_PIN AB12  IOSTANDARD LVCMOS25 } [get_ports { hdmi_rx_hpa }]; #IO_L7N_T1_13 Sch=hdmi_rx_hpa
#set_property -dict { PACKAGE_PIN Y4    IOSTANDARD LVCMOS33 } [get_ports { hdmi_rx_scl }]; #IO_L11P_T1_SRCC_34 Sch=hdmi_rx_scl
#set_property -dict { PACKAGE_PIN AB5   IOSTANDARD LVCMOS33 } [get_ports { hdmi_rx_sda }]; #IO_L10N_T1_34 Sch=hdmi_rx_sda
#set_property -dict { PACKAGE_PIN R3    IOSTANDARD LVCMOS33 } [get_ports { hdmi_rx_txen }]; #IO_L3P_T0_DQS_34 Sch=hdmi_rx_txen
#set_property -dict { PACKAGE_PIN AA3   IOSTANDARD TMDS_33  } [get_ports { hdmi_rx_n[0] }]; #IO_L9N_T1_DQS_34 Sch=hdmi_rx_n[0]
#set_property -dict { PACKAGE_PIN Y3    IOSTANDARD TMDS_33  } [get_ports { hdmi_rx_p[0] }]; #IO_L9P_T1_DQS_34 Sch=hdmi_rx_p[0]
#set_property -dict { PACKAGE_PIN Y2    IOSTANDARD TMDS_33  } [get_ports { hdmi_rx_n[1] }]; #IO_L4N_T0_34 Sch=hdmi_rx_n[1]
#set_property -dict { PACKAGE_PIN W2    IOSTANDARD TMDS_33  } [get_ports { hdmi_rx_p[1] }]; #IO_L4P_T0_34 Sch=hdmi_rx_p[1]
```

```
#set_property -dict { PACKAGE_PIN V2    IOSTANDARD TMDS_33  } [get_ports {
hdmi_rx_n[2] }]; #IO_L2N_T0_34 Sch=hdmi_rx_n[2]
#set_property -dict { PACKAGE_PIN U2    IOSTANDARD TMDS_33  } [get_ports {
hdmi_rx_p[2] }]; #IO_L2P_T0_34 Sch=hdmi_rx_p[2]


## HDMI out
#set_property -dict { PACKAGE_PIN AA4   IOSTANDARD LVCMOS33 } [get_ports {
hdmi_tx_cec }]; #IO_L11N_T1_SRCC_34 Sch=hdmi_tx_cec
set_property -dict { PACKAGE_PIN U1    IOSTANDARD TMDS_33  } [get_ports {
hdmi_tx_clk_n }]; #IO_L1N_T0_34 Sch=hdmi_tx_clk_n
set_property -dict { PACKAGE_PIN T1    IOSTANDARD TMDS_33  } [get_ports {
hdmi_tx_clk_p }]; #IO_L1P_T0_34 Sch=hdmi_tx_clk_p
#set_property -dict { PACKAGE_PIN AB13  IOSTANDARD LVCMOS25 } [get_ports {
hdmi_tx_hpd }]; #IO_L3N_T0_DQS_13 Sch=hdmi_tx_hpd
#set_property -dict { PACKAGE_PIN U3    IOSTANDARD LVCMOS33 } [get_ports {
hdmi_tx_rscl }]; #IO_L6P_T0_34 Sch=hdmi_tx_rscl
#set_property -dict { PACKAGE_PIN V3    IOSTANDARD LVCMOS33 } [get_ports {
hdmi_tx_rsda }]; #IO_L6N_T0_VREF_34 Sch=hdmi_tx_rsda
set_property -dict { PACKAGE_PIN Y1    IOSTANDARD TMDS_33  } [get_ports {
hdmi_tx_n[0] }]; #IO_L5N_T0_34 Sch=hdmi_tx_n[0]
set_property -dict { PACKAGE_PIN W1    IOSTANDARD TMDS_33  } [get_ports {
hdmi_tx_p[0] }]; #IO_L5P_T0_34 Sch=hdmi_tx_p[0]
set_property -dict { PACKAGE_PIN AB1   IOSTANDARD TMDS_33  } [get_ports {
hdmi_tx_n[1] }]; #IO_L7N_T1_34 Sch=hdmi_tx_n[1]
set_property -dict { PACKAGE_PIN AA1   IOSTANDARD TMDS_33  } [get_ports {
hdmi_tx_p[1] }]; #IO_L7P_T1_34 Sch=hdmi_tx_p[1]
set_property -dict { PACKAGE_PIN AB2   IOSTANDARD TMDS_33  } [get_ports {
hdmi_tx_n[2] }]; #IO_L8N_T1_34 Sch=hdmi_tx_n[2]
set_property -dict { PACKAGE_PIN AB3   IOSTANDARD TMDS_33  } [get_ports {
hdmi_tx_p[2] }]; #IO_L8P_T1_34 Sch=hdmi_tx_p[2]


## Display Port
#set_property -dict { PACKAGE_PIN AB10  IOSTANDARD TMDS_33  } [get_ports {
dp_tx_aux_n }]; #IO_L8N_T1_13 Sch=dp_tx_aux_n
#set_property -dict { PACKAGE_PIN AA11  IOSTANDARD TMDS_33  } [get_ports {
dp_tx_aux_n }]; #IO_L9N_T1_DQS_13 Sch=dp_tx_aux_n
#set_property -dict { PACKAGE_PIN AA9   IOSTANDARD TMDS_33  } [get_ports {
dp_tx_aux_p }]; #IO_L8P_T1_13 Sch=dp_tx_aux_p
#set_property -dict { PACKAGE_PIN AA10  IOSTANDARD TMDS_33  } [get_ports {
dp_tx_aux_p }]; #IO_L9P_T1_DQS_13 Sch=dp_tx_aux_p
#set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports {
dp_tx_hpd }]; #IO_25_14 Sch=dp_tx_hpd
```

```
## Audio Codec
#set_property -dict { PACKAGE_PIN T4    IOSTANDARD LVCMOS33 } [get_ports {
ac_adc_sdata }]; #IO_L13N_T2_MRCC_34 Sch=ac_adc_sdata
#set_property -dict { PACKAGE_PIN T5    IOSTANDARD LVCMOS33 } [get_ports {
ac_bclk }]; #IO_L14P_T2_SRCC_34 Sch=ac_bclk
#set_property -dict { PACKAGE_PIN W6    IOSTANDARD LVCMOS33 } [get_ports {
ac_dac_sdata }]; #IO_L15P_T2_DQS_34 Sch=ac_dac_sdata
#set_property -dict { PACKAGE_PIN U5    IOSTANDARD LVCMOS33 } [get_ports {
ac_lrclk }]; #IO_L14N_T2_SRCC_34 Sch=ac_lrclk
#set_property -dict { PACKAGE_PIN U6    IOSTANDARD LVCMOS33 } [get_ports {
ac_mclk }]; #IO_L16P_T2_34 Sch=ac_mclk


## Pmod header JA
#set_property -dict { PACKAGE_PIN AB22  IOSTANDARD LVCMOS33 } [get_ports {
ja[0] }]; #IO_L10N_T1_D15_14 Sch=ja[1]
#set_property -dict { PACKAGE_PIN AB21  IOSTANDARD LVCMOS33 } [get_ports {
ja[1] }]; #IO_L10P_T1_D14_14 Sch=ja[2]
#set_property -dict { PACKAGE_PIN AB20  IOSTANDARD LVCMOS33 } [get_ports {
ja[2] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=ja[3]
#set_property -dict { PACKAGE_PIN AB18  IOSTANDARD LVCMOS33 } [get_ports {
ja[3] }]; #IO_L17N_T2_A13_D29_14 Sch=ja[4]
#set_property -dict { PACKAGE_PIN Y21   IOSTANDARD LVCMOS33 } [get_ports { ja[4]
}]; #IO_L9P_T1_DQS_14 Sch=ja[7]
#set_property -dict { PACKAGE_PIN AA21  IOSTANDARD LVCMOS33 } [get_ports {
ja[5] }]; #IO_L8N_T1_D12_14 Sch=ja[8]
#set_property -dict { PACKAGE_PIN AA20  IOSTANDARD LVCMOS33 } [get_ports {
ja[6] }]; #IO_L8P_T1_D11_14 Sch=ja[9]
#set_property -dict { PACKAGE_PIN AA18  IOSTANDARD LVCMOS33 } [get_ports {
ja[7] }]; #IO_L17P_T2_A14_D30_14 Sch=ja[10]


## Pmod header JB
set_property -dict { PACKAGE_PIN V9    IOSTANDARD LVCMOS33 } [get_ports { jb[0]
}]; #IO_L21P_T3_DQS_34 Sch=jb_p[1]
set_property -dict { PACKAGE_PIN V8    IOSTANDARD LVCMOS33 } [get_ports { jb[1]
}]; #IO_L21N_T3_DQS_34 Sch=jb_n[1]
set_property -dict { PACKAGE_PIN V7    IOSTANDARD LVCMOS33 } [get_ports { jb[2]
}]; #IO_L19P_T3_34 Sch=jb_p[2]
set_property -dict { PACKAGE_PIN W7    IOSTANDARD LVCMOS33 } [get_ports { jb[3]
}]; #IO_L19N_T3_VREF_34 Sch=jb_n[2]
set_property -dict { PACKAGE_PIN W9    IOSTANDARD LVCMOS33 } [get_ports { jb[4]
```

```
}]; #IO_L24P_T3_34 Sch=jb_p[3]
set_property -dict { PACKAGE_PIN Y9    IOSTANDARD LVCMOS33 } [get_ports { jb[5]
}]; #IO_L24N_T3_34 Sch=jb_n[3]
set_property -dict { PACKAGE_PIN Y8    IOSTANDARD LVCMOS33 } [get_ports { jb[6]
}]; #IO_L23P_T3_34 Sch=jb_p[4]
set_property -dict { PACKAGE_PIN Y7    IOSTANDARD LVCMOS33 } [get_ports { jb[7]
}]; #IO_L23N_T3_34 Sch=jb_n[4]


## Pmod header JC
set_property -dict { PACKAGE_PIN Y6    IOSTANDARD LVCMOS33 } [get_ports { jc[0]
}]; #IO_L18P_T2_34 Sch=jc_p[1]
set_property -dict { PACKAGE_PIN AA6   IOSTANDARD LVCMOS33 } [get_ports { jc[1]
}]; #IO_L18N_T2_34 Sch=jc_n[1]
set_property -dict { PACKAGE_PIN AA8   IOSTANDARD LVCMOS33 } [get_ports { jc[2]
}]; #IO_L22P_T3_34 Sch=jc_p[2]
set_property -dict { PACKAGE_PIN AB8   IOSTANDARD LVCMOS33 } [get_ports { jcclk
}]; #IO_L22N_T3_34 Sch=jc_n[2]
#set_property -dict { PACKAGE_PIN R6    IOSTANDARD LVCMOS33 } [get_ports { jc[4]
}]; #IO_L17P_T2_34 Sch=jc_p[3]
#set_property -dict { PACKAGE_PIN T6    IOSTANDARD LVCMOS33 } [get_ports { jc[5]
}]; #IO_L17N_T2_34 Sch=jc_n[3]
#set_property -dict { PACKAGE_PIN AB7   IOSTANDARD LVCMOS33 } [get_ports { jc[6]
}]; #IO_L20P_T3_34 Sch=jc_p[4]
#set_property -dict { PACKAGE_PIN AB6   IOSTANDARD LVCMOS33 } [get_ports { jc[7]
}]; #IO_L20N_T3_34 Sch=jc_n[4]


## XADC Header
#set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports {
xa_p[0] }]; #IO_L3P_T0_DQS_AD1P_15 Sch=xa_p[1]
#set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [get_ports {
xa_n[0] }]; #IO_L3N_T0_DQS_AD1N_15 Sch=xa_n[1]
#set_property -dict { PACKAGE_PIN H13   IOSTANDARD LVCMOS33 } [get_ports {
xa_p[1] }]; #IO_L1P_T0_AD0P_15 Sch=xa_p[2]
#set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 } [get_ports {
xa_n[1] }]; #IO_L1N_T0_AD0N_15 Sch=xa_n[2]
#set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports {
xa_p[2] }]; #IO_L2P_T0_AD8P_15 Sch=xa_p[3]
#set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 } [get_ports {
xa_n[2] }]; #IO_L2N_T0_AD8N_15 Sch=xa_n[3]
#set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports {
xa_p[3] }]; #IO_L5P_T0_AD9P_15 Sch=xa_p[4]
#set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports {
```

```
xa_n[3] }]; #IO_L5N_T0_AD9N_15 Sch=xa_n[4]


## UART
#set_property -dict { PACKAGE_PIN AA19  IOSTANDARD LVCMOS33 } [get_ports {
uart_rx_out }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=uart_rx_out
#set_property -dict { PACKAGE_PIN V18   IOSTANDARD LVCMOS33 } [get_ports {
uart_tx_in }]; #IO_L14P_T2_SRCC_14 Sch=uart_tx_in


## Ethernet
#set_property -dict { PACKAGE_PIN Y14   IOSTANDARD LVCMOS25 } [get_ports {
eth_int_b }]; #IO_L6N_T0_VREF_13 Sch=eth_int_b
#set_property -dict { PACKAGE_PIN AA16  IOSTANDARD LVCMOS25 } [get_ports {
eth_mdc }]; #IO_L1N_T0_13 Sch=eth_mdc
#set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS25 } [get_ports {
eth_mdio }]; #IO_L1P_T0_13 Sch=eth_mdio
#set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS25 } [get_ports {
eth_pme_b }]; #IO_L6P_T0_13 Sch=eth_pme_b
#set_property -dict { PACKAGE_PIN U7    IOSTANDARD LVCMOS33 } [get_ports {
eth_rst_b }]; #IO_25_34 Sch=eth_rst_b
#set_property -dict { PACKAGE_PIN V13   IOSTANDARD LVCMOS25 } [get_ports {
eth_rxck }]; #IO_L13P_T2_MRCC_13 Sch=eth_rxck
#set_property -dict { PACKAGE_PIN W10   IOSTANDARD LVCMOS25 } [get_ports {
eth_rxctl }]; #IO_L10N_T1_13 Sch=eth_rxctl
#set_property -dict { PACKAGE_PIN AB16  IOSTANDARD LVCMOS25 } [get_ports {
eth_rxd[0] }]; #IO_L2P_T0_13 Sch=eth_rxd[0]
#set_property -dict { PACKAGE_PIN AA15  IOSTANDARD LVCMOS25 } [get_ports {
eth_rxd[1] }]; #IO_L4P_T0_13 Sch=eth_rxd[1]
#set_property -dict { PACKAGE_PIN AB15  IOSTANDARD LVCMOS25 } [get_ports {
eth_rxd[2] }]; #IO_L4N_T0_13 Sch=eth_rxd[2]
#set_property -dict { PACKAGE_PIN AB11  IOSTANDARD LVCMOS25 } [get_ports {
eth_rxd[3] }]; #IO_L7P_T1_13 Sch=eth_rxd[3]
#set_property -dict { PACKAGE_PIN AA14  IOSTANDARD LVCMOS25 } [get_ports {
eth_txck }]; #IO_L5N_T0_13 Sch=eth_txck
#set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS25 } [get_ports {
eth_txctl }]; #IO_L10P_T1_13 Sch=eth_txctl
#set_property -dict { PACKAGE_PIN Y12   IOSTANDARD LVCMOS25 } [get_ports {
eth_txd[0] }]; #IO_L11N_T1_SRCC_13 Sch=eth_txd[0]
#set_property -dict { PACKAGE_PIN W12   IOSTANDARD LVCMOS25 } [get_ports {
eth_txd[1] }]; #IO_L12N_T1_MRCC_13 Sch=eth_txd[1]
#set_property -dict { PACKAGE_PIN W11   IOSTANDARD LVCMOS25 } [get_ports {
eth_txd[2] }]; #IO_L12P_T1_MRCC_13 Sch=eth_txd[2]
#set_property -dict { PACKAGE_PIN Y11   IOSTANDARD LVCMOS25 } [get_ports {
```

```
eth_txd[3] }]; #IO_L11P_T1_SRCC_13 Sch=eth_txd[3]


## Fan PWM
#set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS25 } [get_ports {
fan_pwm }]; #IO_L14P_T2_SRCC_13 Sch=fan_pwm


## DPTI/DSPI
#set_property -dict { PACKAGE_PIN Y18   IOSTANDARD LVCMOS33 } [get_ports {
prog_clko }]; #IO_L13P_T2_MRCC_14 Sch=prog_clko
#set_property -dict { PACKAGE_PIN U20   IOSTANDARD LVCMOS33 } [get_ports {
prog_d[0]}]; #IO_L11P_T1_SRCC_14 Sch=prog_d0/sck
#set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports {
prog_d[1] }]; #IO_L19P_T3_A10_D26_14 Sch=prog_d1/mosi
#set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports {
prog_d[2] }]; #IO_L22P_T3_A05_D21_14 Sch=prog_d2/miso
#set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports {
prog_d[3]}]; #IO_L18P_T2_A12_D28_14 Sch=prog_d3/ss
#set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports {
prog_d[4] }]; #IO_L24N_T3_A00_D16_14 Sch=prog_d[4]
#set_property -dict { PACKAGE_PIN P16   IOSTANDARD LVCMOS33 } [get_ports {
prog_d[5] }]; #IO_L24P_T3_A01_D17_14 Sch=prog_d[5]
#set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports {
prog_d[6] }]; #IO_L20P_T3_A08_D24_14 Sch=prog_d[6]
#set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports {
prog_d[7] }]; #IO_L23N_T3_A02_D18_14 Sch=prog_d[7]
#set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {
prog_oen }]; #IO_L16P_T2_CSI_B_14 Sch=prog_oen
#set_property -dict { PACKAGE_PIN P19   IOSTANDARD LVCMOS33 } [get_ports {
prog_rdn }]; #IO_L5P_T0_D06_14 Sch=prog_rdn
#set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports {
prog_rxen }]; #IO_L21P_T3_DQS_14 Sch=prog_rxen
#set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports {
prog_siwun }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=prog_siwun
#set_property -dict { PACKAGE_PIN R14   IOSTANDARD LVCMOS33 } [get_ports {
prog_spien }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=prog_spien
#set_property -dict { PACKAGE_PIN Y19   IOSTANDARD LVCMOS33 } [get_ports {
prog_txen }]; #IO_L13N_T2_MRCC_14 Sch=prog_txen
#set_property -dict { PACKAGE_PIN R19   IOSTANDARD LVCMOS33 } [get_ports {
prog_wrn }]; #IO_L5N_T0_D07_14 Sch=prog_wrn


## HID port
```

```
#set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33   PULLUP true } [get_ports { ps2_clk }]; #IO_L16N_T2_A15_D31_14 Sch=ps2_clk
#set_property -dict { PACKAGE_PIN N13   IOSTANDARD LVCMOS33   PULLUP true } [get_ports { ps2_data }]; #IO_L23P_T3_A03_D19_14 Sch=ps2_data


## QSPI
#set_property -dict { PACKAGE_PIN T19   IOSTANDARD LVCMOS33 } [get_ports { qspi_cs }]; #IO_L6P_T0_FCS_B_14 Sch=qspi_cs
#set_property -dict { PACKAGE_PIN P22   IOSTANDARD LVCMOS33 } [get_ports { qspi_dq[0] }]; #IO_L1P_T0_D00_MOSI_14 Sch=qspi_dq[0]
#set_property -dict { PACKAGE_PIN R22   IOSTANDARD LVCMOS33 } [get_ports { qspi_dq[1] }]; #IO_L1N_T0_D01_DIN_14 Sch=qspi_dq[1]
#set_property -dict { PACKAGE_PIN P21   IOSTANDARD LVCMOS33 } [get_ports { qspi_dq[2] }]; #IO_L2P_T0_D02_14 Sch=qspi_dq[2]
#set_property -dict { PACKAGE_PIN R21   IOSTANDARD LVCMOS33 } [get_ports { qspi_dq[3] }]; #IO_L2N_T0_D03_14 Sch=qspi_dq[3]


## SD card
#set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports { sd_cclk }]; #IO_L12P_T1_MRCC_14 Sch=sd_cclk
#set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { sd_cd }]; #IO_L20N_T3_A07_D23_14 Sch=sd_cd
#set_property -dict { PACKAGE_PIN W20   IOSTANDARD LVCMOS33 } [get_ports { sd_cmd }]; #IO_L12N_T1_MRCC_14 Sch=sd_cmd
#set_property -dict { PACKAGE_PIN V19   IOSTANDARD LVCMOS33 } [get_ports { sd_d[0] }]; #IO_L14N_T2_SRCC_14 Sch=sd_d[0]
#set_property -dict { PACKAGE_PIN T21   IOSTANDARD LVCMOS33 } [get_ports { sd_d[1] }]; #IO_L4P_T0_D04_14 Sch=sd_d[1]
#set_property -dict { PACKAGE_PIN T20   IOSTANDARD LVCMOS33 } [get_ports { sd_d[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sd_d[2]
#set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { sd_d[3] }]; #IO_L18N_T2_A11_D27_14 Sch=sd_d[3]
#set_property -dict { PACKAGE_PIN V20   IOSTANDARD LVCMOS33 } [get_ports { sd_reset }]; #IO_L11N_T1_SRCC_14 Sch=sd_reset


## I2C
#set_property -dict { PACKAGE_PIN W5    IOSTANDARD LVCMOS33 } [get_ports { scl }]; #IO_L15N_T2_DQS_34 Sch=scl
#set_property -dict { PACKAGE_PIN V5    IOSTANDARD LVCMOS33 } [get_ports { sda }]; #IO_L16N_T2_34 Sch=sda
```

```
## Voltage Adjust
#set_property -dict { PACKAGE_PIN AA13  IOSTANDARD LVCMOS25 } [get_ports {
set_vadj[0] }]; #IO_L3P_T0_DQS_13 Sch=set_vadj[0]
#set_property -dict { PACKAGE_PIN AB17  IOSTANDARD LVCMOS25 } [get_ports {
set_vadj[1] }]; #IO_L2N_T0_13 Sch=set_vadj[1]
#set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS25 } [get_ports {
vadj_en }]; #IO_L13N_T2_MRCC_13 Sch=vadj_en


## FMC
#set_property -dict { PACKAGE_PIN H19   IOSTANDARD LVCMOS12 } [get_ports {
fmc_clk0_m2c_n }]; #IO_L12N_T1_MRCC_15 Sch=fmc_clk0_m2c_n
#set_property -dict { PACKAGE_PIN J19   IOSTANDARD LVCMOS12 } [get_ports {
fmc_clk0_m2c_p }]; #IO_L12P_T1_MRCC_15 Sch=fmc_clk0_m2c_p
#set_property -dict { PACKAGE_PIN C19   IOSTANDARD LVCMOS12 } [get_ports {
fmc_clk1_m2c_n }]; #IO_L13N_T2_MRCC_16 Sch=fmc_clk1_m2c_n
#set_property -dict { PACKAGE_PIN C18   IOSTANDARD LVCMOS12 } [get_ports {
fmc_clk1_m2c_p }]; #IO_L13P_T2_MRCC_16 Sch=fmc_clk1_m2c_p
#set_property -dict { PACKAGE_PIN K19   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la00_cc_n }]; #IO_L13N_T2_MRCC_15 Sch=fmc_la00_cc_n
#set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la00_cc_p }]; #IO_L13P_T2_MRCC_15 Sch=fmc_la00_cc_p
#set_property -dict { PACKAGE_PIN J21   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la01_cc_n }]; #IO_L11N_T1_SRCC_15 Sch=fmc_la01_cc_n
#set_property -dict { PACKAGE_PIN J20   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la01_cc_p }]; #IO_L11P_T1_SRCC_15 Sch=fmc_la01_cc_p
#set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[02] }]; #IO_L16N_T2_A27_15 Sch=fmc_la_n[02]
#set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[02] }]; #IO_L16P_T2_A28_15 Sch=fmc_la_p[02]
#set_property -dict { PACKAGE_PIN N19   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[03] }]; #IO_L17N_T2_A25_15 Sch=fmc_la_n[03]
#set_property -dict { PACKAGE_PIN N18   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[03] }]; #IO_L17P_T2_A26_15 Sch=fmc_la_p[03]
#set_property -dict { PACKAGE_PIN M20   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[04] }]; #IO_L18N_T2_A23_15 Sch=fmc_la_n[04]
#set_property -dict { PACKAGE_PIN N20   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[04] }]; #IO_L18P_T2_A24_15 Sch=fmc_la_p[04]
#set_property -dict { PACKAGE_PIN L21   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[05] }]; #IO_L10N_T1_AD11N_15 Sch=fmc_la_n[05]
#set_property -dict { PACKAGE_PIN M21   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[05] }]; #IO_L10P_T1_AD11P_15 Sch=fmc_la_p[05]
#set_property -dict { PACKAGE_PIN M22   IOSTANDARD LVCMOS12 } [get_ports {
```

```
fmc_la_n[06] }]; #IO_L15N_T2_DQS_ADV_B_15 Sch=fmc_la_n[06]
#set_property -dict { PACKAGE_PIN N22   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[06] }]; #IO_L15P_T2_DQS_15 Sch=fmc_la_p[06]
#set_property -dict { PACKAGE_PIN L13   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[07] }]; #IO_L20N_T3_A19_15 Sch=fmc_la_n[07]
#set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[07] }]; #IO_L20P_T3_A20_15 Sch=fmc_la_p[07]
#set_property -dict { PACKAGE_PIN M16   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[08] }]; #IO_L24N_T3_RS0_15 Sch=fmc_la_n[08]
#set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[08] }]; #IO_L24P_T3_RS1_15 Sch=fmc_la_p[08]
#set_property -dict { PACKAGE_PIN G20   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[09] }]; #IO_L8N_T1_AD10N_15 Sch=fmc_la_n[09]
#set_property -dict { PACKAGE_PIN H20   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[09] }]; #IO_L8P_T1_AD10P_15 Sch=fmc_la_p[09]
#set_property -dict { PACKAGE_PIN K22   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[10] }]; #IO_L9N_T1_DQS_AD3N_15 Sch=fmc_la_n[10]
#set_property -dict { PACKAGE_PIN K21   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[10] }]; #IO_L9P_T1_DQS_AD3P_15 Sch=fmc_la_p[10]
#set_property -dict { PACKAGE_PIN L15   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[11] }]; #IO_L22N_T3_A16_15 Sch=fmc_la_n[11]
#set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[11] }]; #IO_L22P_T3_A17_15 Sch=fmc_la_p[11]
#set_property -dict { PACKAGE_PIN L20   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[12] }]; #IO_L14N_T2_SRCC_15 Sch=fmc_la_n[12]
#set_property -dict { PACKAGE_PIN L19   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[12] }]; #IO_L14P_T2_SRCC_15 Sch=fmc_la_p[12]
#set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[13] }]; #IO_L21N_T3_DQS_A18_15 Sch=fmc_la_n[13]
#set_property -dict { PACKAGE_PIN K17   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[13] }]; #IO_L21P_T3_DQS_15 Sch=fmc_la_p[13]
#set_property -dict { PACKAGE_PIN H22   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[14] }]; #IO_L7N_T1_AD2N_15 Sch=fmc_la_n[14]
#set_property -dict { PACKAGE_PIN J22   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[14] }]; #IO_L7P_T1_AD2P_15 Sch=fmc_la_p[14]
#set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[15] }]; #IO_L23N_T3_FWE_B_15 Sch=fmc_la_n[15]
#set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[15] }]; #IO_L23P_T3_FOE_B_15 Sch=fmc_la_p[15]
#set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[16] }]; #IO_L4N_T0_15 Sch=fmc_la_n[16]
#set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[16] }]; #IO_L4P_T0_15 Sch=fmc_la_p[16]
#set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS12 } [get_ports {
```

```
fmc_la17_cc_n }]; #IO_L11N_T1_SRCC_16 Sch=fmc_la17_cc_n
#set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la17_cc_p }]; #IO_L11P_T1_SRCC_16 Sch=fmc_la17_cc_p
#set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la18_cc_n }]; #IO_L12N_T1_MRCC_16 Sch=fmc_la18_cc_n
#set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la18_cc_p }]; #IO_L12P_T1_MRCC_16 Sch=fmc_la18_cc_p
#set_property -dict { PACKAGE_PIN A19   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[19] }]; #IO_L17N_T2_16 Sch=fmc_la_n[19]
#set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[19] }]; #IO_L17P_T2_16 Sch=fmc_la_p[19]
#set_property -dict { PACKAGE_PIN F20   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[20] }]; #IO_L18N_T2_16 Sch=fmc_la_n[20]
#set_property -dict { PACKAGE_PIN F19   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[20] }]; #IO_L18P_T2_16 Sch=fmc_la_p[20]
#set_property -dict { PACKAGE_PIN D19   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[21] }]; #IO_L14N_T2_SRCC_16 Sch=fmc_la_n[21]
#set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[21] }]; #IO_L14P_T2_SRCC_16 Sch=fmc_la_p[21]
#set_property -dict { PACKAGE_PIN D21   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[22] }]; #IO_L23N_T3_16 Sch=fmc_la_n[22]
#set_property -dict { PACKAGE_PIN E21   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[22] }]; #IO_L23P_T3_16 Sch=fmc_la_p[22]
#set_property -dict { PACKAGE_PIN A21   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[23] }]; #IO_L21N_T3_DQS_16 Sch=fmc_la_n[23]
#set_property -dict { PACKAGE_PIN B21   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[23] }]; #IO_L21P_T3_DQS_16 Sch=fmc_la_p[23]
#set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[24] }]; #IO_L7N_T1_16 Sch=fmc_la_n[24]
#set_property -dict { PACKAGE_PIN B15   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[24] }]; #IO_L7P_T1_16 Sch=fmc_la_p[24]
#set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[25] }]; #IO_L2N_T0_16 Sch=fmc_la_n[25]
#set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[25] }]; #IO_L2P_T0_16 Sch=fmc_la_p[25]
#set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[26] }]; #IO_L15N_T2_DQS_16 Sch=fmc_la_n[26]
#set_property -dict { PACKAGE_PIN F18   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[26] }]; #IO_L15P_T2_DQS_16 Sch=fmc_la_p[26]
#set_property -dict { PACKAGE_PIN A20   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[27] }]; #IO_L16N_T2_16 Sch=fmc_la_n[27]
#set_property -dict { PACKAGE_PIN B20   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[27] }]; #IO_L16P_T2_16 Sch=fmc_la_p[27]
#set_property -dict { PACKAGE_PIN B13   IOSTANDARD LVCMOS12 } [get_ports {
```

```
fmc_la_n[28] }]; #IO_L8N_T1_16 Sch=fmc_la_n[28]
#set_property -dict { PACKAGE_PIN C13   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[28] }]; #IO_L8P_T1_16 Sch=fmc_la_p[28]
#set_property -dict { PACKAGE_PIN C15   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[29] }]; #IO_L3N_T0_DQS_16 Sch=fmc_la_n[29]
#set_property -dict { PACKAGE_PIN C14   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[29] }]; #IO_L3P_T0_DQS_16 Sch=fmc_la_p[29]
#set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[30] }]; #IO_L10N_T1_16 Sch=fmc_la_n[30]
#set_property -dict { PACKAGE_PIN A13   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[30] }]; #IO_L10P_T1_16 Sch=fmc_la_p[30]
#set_property -dict { PACKAGE_PIN E14   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[31] }]; #IO_L4N_T0_16 Sch=fmc_la_n[31]
#set_property -dict { PACKAGE_PIN E13   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[31] }]; #IO_L4P_T0_16 Sch=fmc_la_p[31]
#set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[32] }]; #IO_L9N_T1_DQS_16 Sch=fmc_la_n[32]
#set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[32] }]; #IO_L9P_T1_DQS_16 Sch=fmc_la_p[32]
#set_property -dict { PACKAGE_PIN F14   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_n[33] }]; #IO_L1N_T0_16 Sch=fmc_la_n[33]
#set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS12 } [get_ports {
fmc_la_p[33] }]; #IO_L1P_T0_16 Sch=fmc_la_p[33]


## Configuration options, can be used for all designs
set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]
```