6.111 Introductory Digital Systems Laboratory

Final Project Report: AirPong

Sabina Chen | David Mueller | Bret Heaslet

12 December 2018

Contents

1 Abstract **3**

2 Problem Definition **3**
- 2.1 Rules of the Game . . . 3
- 2.2 Technical Challenges . . . 4

3 Design Overview **4**

4 Visualization - Sabina **5**
- 4.1 Overview . . . 5
- 4.2 Black and White Video to Color Conversion . . . 5
- 4.3 Selecting HSV Values for Color Detection . . . 6
- 4.4 Detecting Color . . . 9
- 4.5 Debugging Tools, Challenges, and Implementation Notes . . . 11
  - 4.5.1 Switches for Multiple Objects . . . 11
  - 4.5.2 Divider Delays . . . 12
  - 4.5.3 HEX Display Clocks . . . 13
  - 4.5.4 Decreasing the Overall Compile Time . . . 13
- 4.6 Detailed Block Diagram of Visualization Block . . . 13

5 Controls/Pong Game - David **14**
- 5.1 Overview . . . 14
- 5.2 controls . . . 14
  - 5.2.1 Inputs . . . 15
  - 5.2.2 Outputs . . . 15
- 5.3 feedback_controller . . . 15

5.3.1 Inputs . . . 16
5.3.2 Outputs . . . 16
5.4 estimator . . . 17
5.4.1 Inputs . . . 17
5.4.2 Outputs . . . 17
5.5 pong_game . . . 17
5.5.1 Inputs . . . 18
5.5.2 Outputs . . . 18

6 Communications - Bret **18**
6.1 Overview . . . 18
6.2 comms_command_center . . . 19
6.3 data_parser . . . 20
6.4 transmit_to_drone . . . 22
6.5 rx_from_drone . . . 22

7 Lessons Learned **23**

8 Conclusion **23**

9 Acknowledgements **24**

10 Appendix **24**
10.1 Visualization . . . 24
10.1.1 Visualization: crosshair Module . . . 24
10.1.2 Visualization: debounce Module . . . 25
10.1.3 Visualization: delayN Module . . . 26
10.1.4 Visualization: detect_color Module . . . 26
10.1.5 Visualization: display_16hex Module . . . 31
10.1.6 Visualization: display_hsv_value Module . . . 36
10.1.7 Visualization: low_pass_filter Module . . . 36
10.1.8 Visualization: move_crosshair Module . . . 37
10.1.9 Visualization: ntsc2zbt Module . . . 39
10.1.10 Visualization: ramclock Module . . . 42
10.1.11 Visualization: rgb2hsv Module . . . 44
10.1.12 Visualization: threshold_hsv Module . . . 47
10.1.13 Visualization: update_threshold Module . . . 49
10.1.14 Visualization: video_decoder Module . . . 51
10.1.15 Visualization: vram_display Module . . . 72
10.1.16 Visualization: xvga Module . . . 73
10.1.17 Visualization: ycrcb2rgb Module . . . 75
10.1.18 Visualization: zbt_6111 Module . . . 76
10.1.19 Visualization: zbt_6111_sample Module . . . 78
10.2 Communications . . . 98
10.2.1 Communications: data_parser Module . . . 98

10.2.2 Communications: comms_command_center Module 101
10.2.3 Communications: transmit_to_drone Module 103
10.2.4 Communications: rx_from_drone Module 105
10.3 Controls 107
10.3.1 Controls: pong_game Module 107
10.3.2 Controls: legacy_controls Module 111
10.3.3 Controls: controls Module 112
10.3.4 Controls: estimator Module 115
10.3.5 Controls: feedback_controller Module 117
10.3.6 Controls: blit_stiff_rect Module 121
10.3.7 Controls: lines_intersect Module 122
10.3.8 Controls: rects_overlap Module 123
10.3.9 Controls: border_collider Module 126
10.3.10 Controls: fsms Module 128
10.3.11 Controls: l2p_fsm Module 130

1 Abstract

What if you could move the game of pong into the real world? Pong is an arcade game that simulates table tennis. Players compete against each other by controlling in-game paddles on the left or right side of the screen to hit a ball back and forth. Points are earned when a player fails to return the ball to the other.

In our real-world version of Pong, AirPong, a drone is the ball and the people are the paddles. A camera that hangs from the ceiling uses computer vision to track the real-time locations of the drone and players within a defined playing field. The physical positions of the players represent the locations of the paddles, and the physical position of the drone represents the location of the ball. We used an FPGA to process camera and sensor data, create game logic, and define controls to send commands to the drone in 3D space. While the game is in motion, a visualization of the real world pong game appears on the VGA monitor.

2 Problem Definition

2.1 Rules of the Game

The rules of pong are simple: Two players bounce a puck back and forth using paddles within a defined game area usually in the shape of a rectangle. A point is scored when a player fails to bounce the puck back. The game then restarts with the puck in the middle and continues until a certain score is reached. The puck travels in a constant velocity in a straight line until it encounters a boundary or a paddle. After an encounter, the puck will bounce symmetrically away from its point of contact.

2.2 Technical Challenges

In order to make our game possible, we must address key technical problems. Using computer vision, the drone and the paddles would have to be located, tracked, and distinguished from the rest of the surroundings. Once the position of the drone is known, a proper control scheme commands the drone to move with respect to the logic of the game. This is done using a constant altitude and straight line speed. Lastly, control commands and the camera feed would have to synced up with a robust communication scheme.

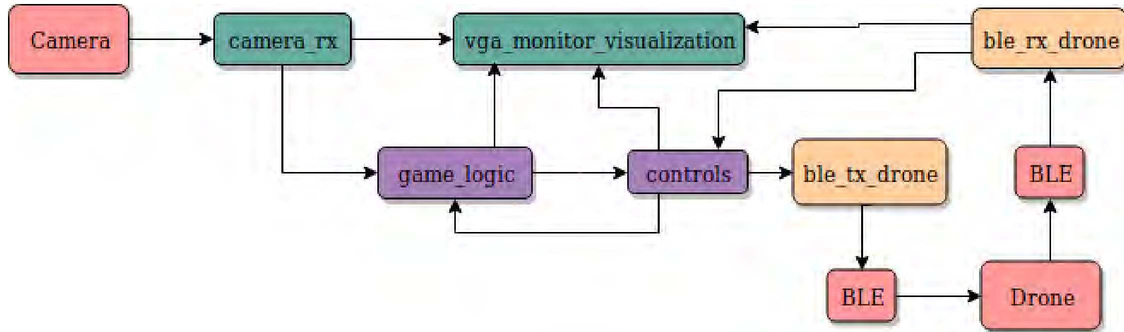


Figure 1: AirPong Initial Block Diagram

3 Design Overview

Our system is comprised of three main components: visualization and motion tracking of the ball and paddles, control of the drone via FPGA estimators, and communication with the drone using a wireless device. Visualization provides pixel coordinates of the tracked objects to the control, which in turn calculates the velocity and direction of the drone based on the current ball and paddle positions. The velocity is then sent to communications which deciphers and packetizes the appropriate commands to the drone. Sensor feedback as well as visual updates from the camera enable us to keep track of the real-world location of the drone in real-time. The main module used for integration was **zbt_6111_sample**.

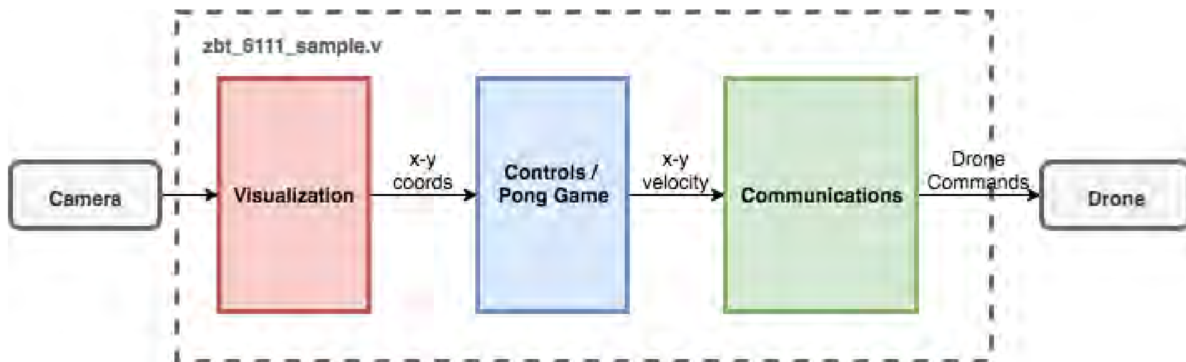


Figure 2: Modularized Block Diagram

4 Visualization - Sabina

4.1 Overview

Visualization decodes and processes data from NTSC camera and outputs the coordinate information needed to the Controls block.

First, NTSC camera data is received, decoded, and displayed as black and white video footage on the vga monitor. The footage is then converted into rgb color, and hsv values are used to track objects within frame. Users are able to select the colors they wish to track by tuning the hsv thresholds before the game starts. Taking in the movements of the objects, the relevant pixels for each color are identified and averaged to produce the center X and Y coordinates of the center of each object. The center coordinates of each object are then passed to the Controls block to provide visual feedback of the paddles and drones for the Pong game. Although the camera footage provided is 725 x 505, the tracking area of the camera was reduced to 604 x 416 in order to eliminate noise from boundary edges. Visualization is directly integrated within **zbt_6111_sample.v**

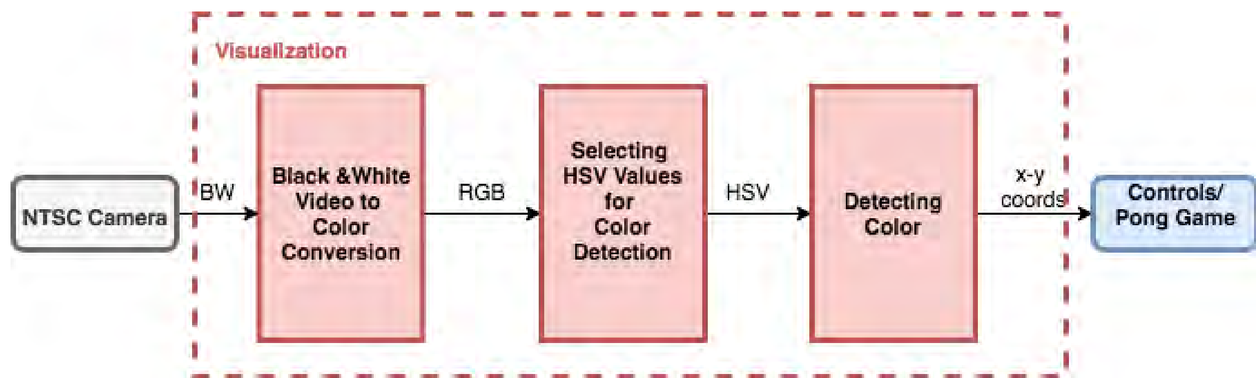


Figure 3: Visualization Block Diagram

One of the more difficult parts in getting Visualization and Object Detection working was just trying to understand how all the separate modules fit together. Therefore, throughout the rest of the report for Visualization, I will try to describe each of the separate modules and data pipeline in order, in hopes that it will make it easier for posterity to understand.

Sections 4.2 to 4.4 will describe the entire implementation pipeline to get from a black and white video image to detecting object centers and overlaying the game of pong onto the display. Section 4.5 will describe the debugging tools, challenges, and notes that were crucial in getting visualization implemented. Section 4.6 will show a final detailed block diagram for visualization with all the parts integrated.

4.2 Black and White Video to Color Conversion

Modules: zbt_6111_sample, zbt_6111, adv7185init, ntsc_decode, ycrcb2rgb, ntsc_to_zbt, vram_display (all staff provided code with modifications by Sabina)

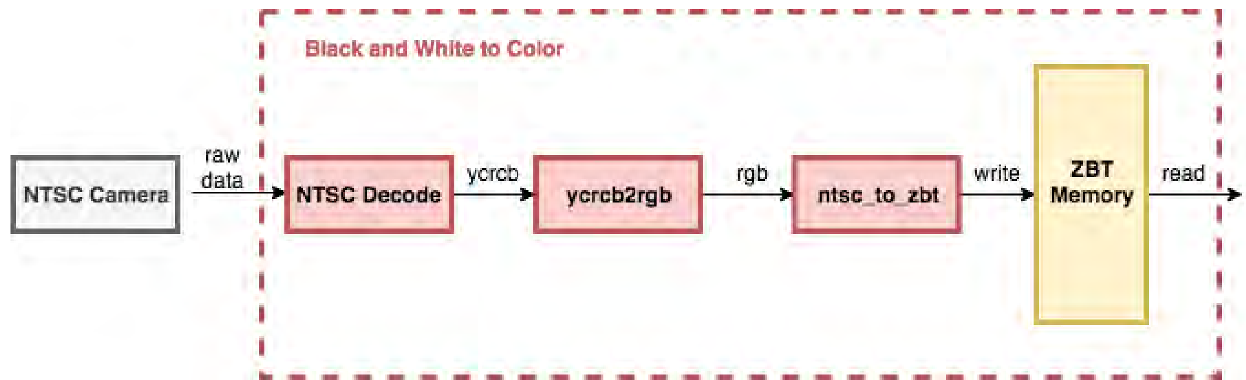


Figure 4: Block Diagram for BW to Color

In order to track the pucks and paddles in the game, we hung a NTSC camera from the ceiling to get a top-down view of the game. The staff provided initial verilog that enabled us to view black and white video footage on the VGA monitor. Our first job was figuring out how to convert the black and white images into color.

Firstly, **ntsc_decode** module is responsible for taking in NTSC signal data from the camera, and decoding it. **ycrcb2rgb** then converts ycrcb pixels into RGB, and passes the rgb pixel data into ZBT RAM on the FPGA board. ZBT RAM can only store two pixels per address (18 bits), therefore the RGB data can be saved as 6 bits each of R, G, and B, instead of the usual 8 bits each (24 bits). The **ntsc_to_zbt** module is responsible for storing the NTSC camera data. Therefore, in order to convert the black and white footage to color, data structures that deal with pixel data within the ntsc_to_zbt file needed to be modified from storing 8 bits to 18 bits. Saving the 18-bit data out of the 24-bit RGB resolution enabled us to display the video footage in color. After saving the RGB pixel data into ZBT RAM, **vram_display** reads from ZBT in order to output the camera image.

In summary, to convert the black and white video image to color, the dataflow is pipelined as follows: **zbt_6111** enables wiring and communications between the labkit and ZBT RAM, **adv7185init** converts raw NTSC output into a usable form, **ntsc_decode** parses the control signals from adv7185init, **ycrcb2rgb** takes the updated control signals and changes them to rgb format, and **ntsc_to_zbt** writes ntsc rgb data into zbt video memory, returning ntsc addresses and data, which is then read and accessed by **vram_display** to generate pixel values from ZBT memory. Furthermore, data structures within the **ntsc_to_zbt** module that deals with saving pixel data were converted from 8 bits to 18 bits.

4.3 Selecting HSV Values for Color Detection

Modules: zbt_6111_sample, vram_display, rgb2hsv, move_crosshair, display_hsv_value

After saving the RGB data into ZBT RAM, we extract the camera pixel data to select the desired HSV values via user input.

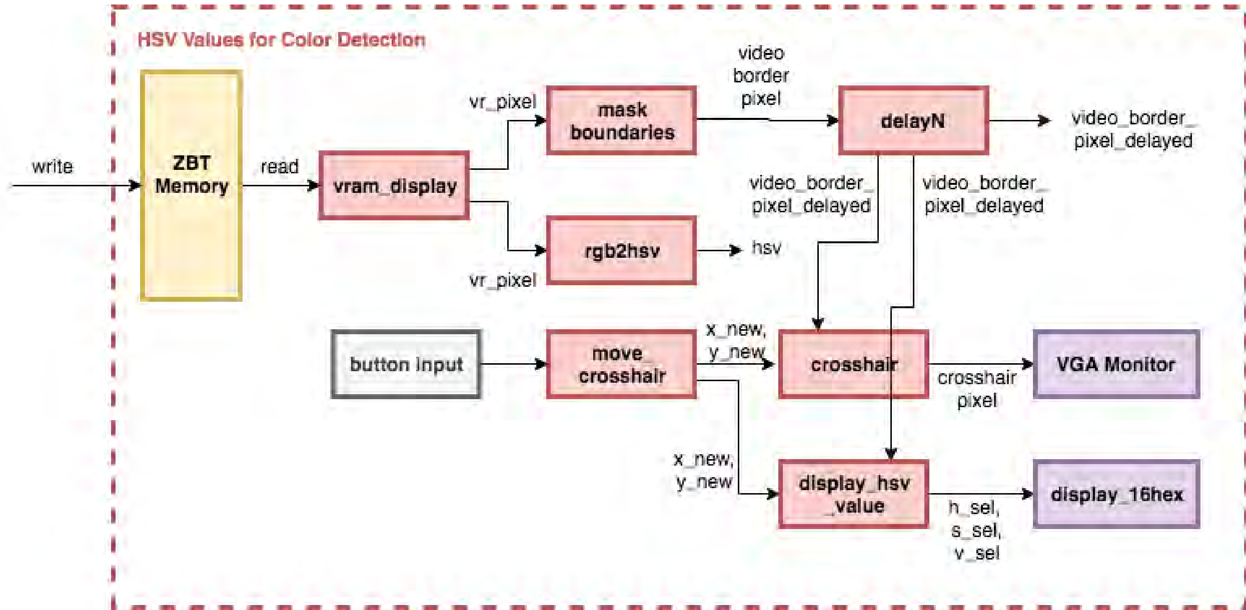


Figure 5: Block Diagram for Delecting HSV Values for Color Detection

Firstly, **vram_display** reads from ZBT RAM and outputs the camera image as vr_pixel[23:0]. The vr_pixel signal is then modified so that the borders of the camera image are black, becoming the output signal video_border_pixel[23:0]. Although the vga monitor display is 1024 x 768, the original video image is around 725 x 505. Because we were using a green screen that was not large enough to cover the entire video image, video_border_pixel also decreased the display or gameplay area to about 648 x 416 to remove the boundaries of the green screen. This will later be useful for color detection as it removes boundary noise.

After vram_display extracts color pixel data, **rgb2hsv** converts the RGB data into HSV. We chose to use HSV for color detection as opposed to RGB because HSV gives more accurate color estimation, even during the often changing/poor lighting conditions within the lab. HSV, or Hue-Saturation-Value, is broken into three parts: Hue, which defines the color, Saturation, which represents the color under various shades of dimensions, and Value, which resembles mixing the color with varying amounts of black or white paint. Individual HSV values range from 0 to 255.

Because the NTSC camera was hung from the ceiling, the lighting conditions tended to change drastically throughout the day. Even though we used HSV for color detection, it was still difficult to key in on the same HSV value during different times of the day. Therefore, rather than hard coding the HSV color, we designed a system that allowed users to manually select the pixel location at which they would like the hsv values and thresholds to be set, making it more convenient and user-friendly to use at the start of each game. Figure 6 below shows our camera setup.

The **move_crosshair** module allows users to manually move crosshairs to specific pixel

Figure 6: NTSC Camera Hanging from the Ceiling

locations. This module takes in user button presses (ie. button up, down, left, right) to update the respective objects crosshair locations, and saves the locations of the new x and y coordinates for each crosshair into registers: x_new[10:0], and y_new[9:0].

After the move_crosshair module provides new x and y locations, the **crosshair** module takes in video_border_pixel[23:0], and overlays a crosshair over the video at the desired location, outputting crosshair_pixel[23:0]. This crosshair module is solely a visualization module that is used for debugging HSV color detection when users are initially trying to key onto specific colors. Because we had three objects to keep track of, we used three crosshairs of three different colors. The puck had a magenta crosshair, paddle 1 had green, and paddle 2 had blue. The user also had the option of using switch[3] to toggle between showing the manually defined location of the crosshair or having the crosshair simply follow the center locations of detected objects (Figure 7).

Working in parallel to crosshair, **display_hsv_value** takes in video_border_pixel[23:0], the updated x_new[10:0] and y_new[9:0] coordinates from the move_crosshair module, and the newly converted hsv pixels from the rgb2hsv module, to detect the hsv value at the selected coordinate. The detected hsv is saved into 8-bit registers: h_sel[7:0], v_sel[7:0], and s_sel[7:0], to be referenced or modified later for color detection. For more debugging purposes, the detected h_sel[7:0], v_sel[7:0], and s_sel[7:0] can be shown on the hex display by sending the concatenated data to the **display_16hex** module.

In summary, users are able to select the hsv values they would like the system to detect via the following pipeline: **vram_display** reads from zbt and outputs camera image vr_pixel[23:0].

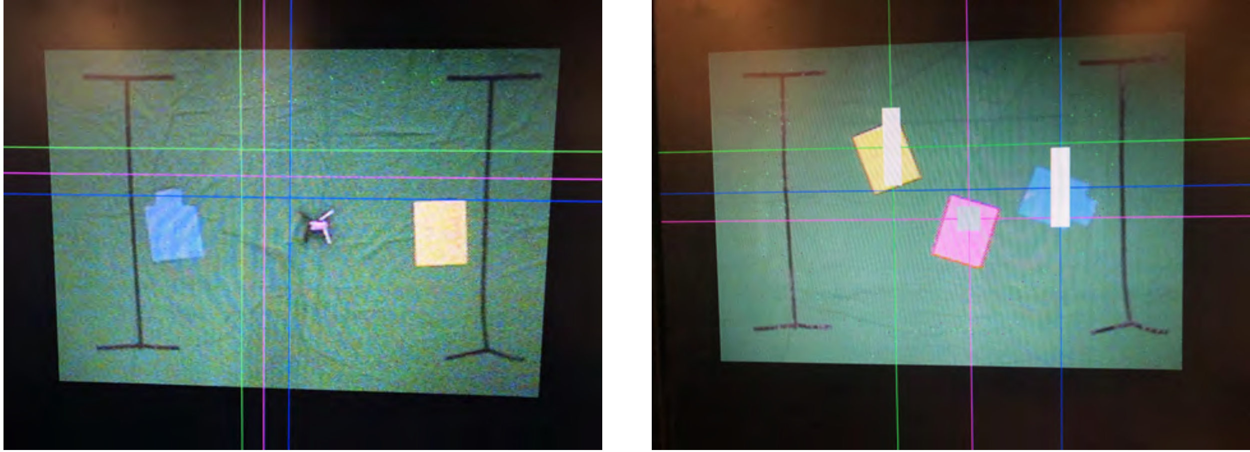


Figure 7: Movable Crosshairs vs Centered Crosshairs

The boundaries of vr_pixel[23:0] are then blacked out to define the game boundaries, creating the output signal video_border_pixel[23:0]. Simultaneously, the **rgb2hsv** module uses vr_pixel[23:0] to convert values into hsv format. The **move_crosshair** and **crosshair** modules enables users to visually navigate through the vga display space using crosshairs to select the pixel location and color in which they would like the system to detect, saving new crosshair locations into x_new[10:0] and y_new[9:0], updating the vga display via crosshair_pixel[23:0]. And finally **display_hsv_value** uses the new crosshair locations to save the hsv values at the specified coordinates into h_sel[7:0], s_sel[7:0], and v_sel[7:0], displaying the new saved hsv values using **display_16hex**.

At this point, the system has finally stored the hsv values it will be detecting into registers, and is ready to start detecting colors (yay!).

4.4 Detecting Color

Modules: zbt_6111_sample, threshold_hsv, detect_color, low_pass_filter

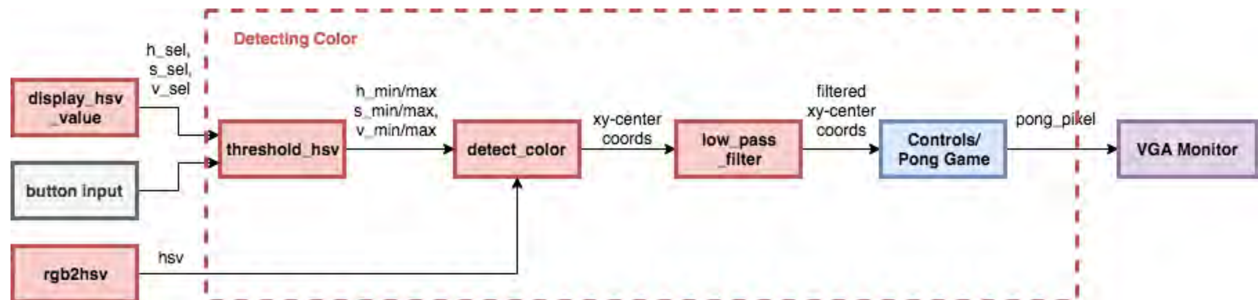


Figure 8: Block Diagram for Detecting Color

Using the saved HSV values from display_hsv_value (ie. h_sel[7:0], s_sel[7:0], v_sel[7:0]), the **threshold_hsv** module sets the new HSV minimum and maximum threshold values for color

detection when the user presses the button enter. The initial values are set by subtracting and adding a pre-defined DELTA value from h, s, and v, thereby creating the threshold values h_min[7:0], h_max[7:0], s_min[7:0], s_max[7:0], v_min[7:0], and m_max[7:0]. These values are saved into registers which can then be used as the threshold values for color detection. Users also have the ability to modify the threshold values by using pre-defined buttons on the labkit to increase or decrease each of the threshold variables individually (Figure 9).

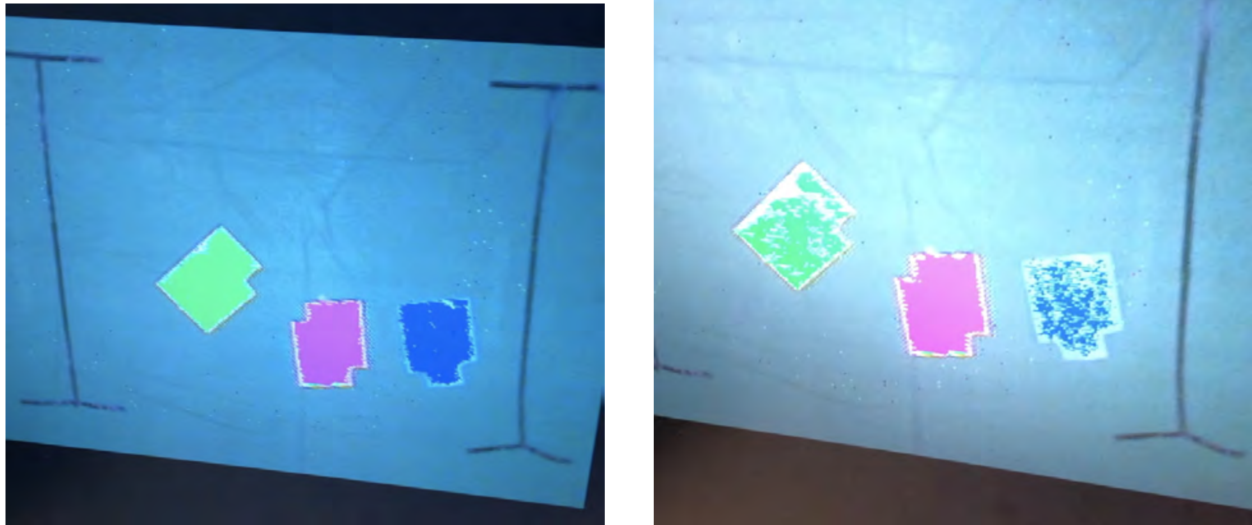


Figure 9: Before vs. After Adjusting HSV Threshold Values

The **detect_color** module takes in the selected h_min/max, s_min/max, and v_min/max thresholds and for each color that one wishes to detect, and looks for the associated pixel coordinates with that color. If the hcount and vcount are within bounds of the video display, the module scans the video image for pixels that lie within the HSV color thresholds, changing identified pixel values to another color for easy debugging later. detect_color takes in video_border_pixel[23:0] and outputs color_pixel[23:0] with the detected pixels in a different color. Furthermore, the module keeps count of how many pixels were detected for each frame, totals the X and Y coordinates, and later determines the center X and Y coordinates using Coregen dividers.

To prevent the x and y center coordinates from being too noisy since the coordinates will be responsible for controlling the drone (jumpy drone != good), we used the **low_pass_filter** to smooth out the last 16 center xy coordinates. This definitely helped smooth out the movements from one timestep to the next, and enabled the provided xy coordinates to actually be usable to the Pong Controls block.

The **pong_game** module then takes the filtered xy coordinates and overlays paddle and puck sprites onto the display. The module takes in video_border_pixel[23:0] and outputs pong_pixel[23:0] with the pong sprite overlay (Figure 10).

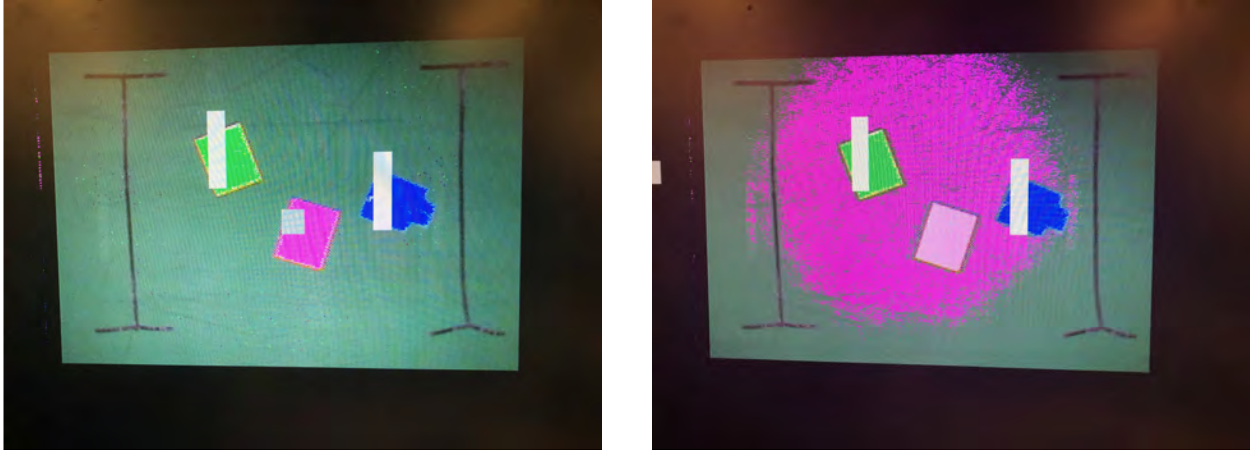


Figure 10: Chroma keying to objects vs keying to the background. Includes pong overlay

In summary, **threshold_hsv** creates HSV color threshold values from selected HSV values and enables the user to adjust the thresholds as needed. **detect_color** uses these HSV threshold values to detect pixels that fall within that threshold, calculating the center of mass of the detected object pixels. A **low_pass_filter** module then takes the center xy coordinates and smooths out the resulting output to remove the jumpiness of the resulting moving xy coordinate output. The **pong_game** module overlays the paddle and puck onto the center of detected objects, enabling users to be able to see the game in action on the monitor while playing.

Finally, the resulting xy center coordinate values of the detected objects are wired to the Controls module, which integrates it into the Pong game to be used as a paddle control and drone/puck feedback system.

4.5 Debugging Tools, Challenges, and Implementation Notes

4.5.1 Switches for Multiple Objects

In order to keep track of three objects at once, multiple registers needed to be created to save HSV threshold data for the three different objects. Each object was tagged with a different color to track, therefore switches and buttons were used to enable users to toggle between the different vga output settings the user wished to see. Below is a list of switches and buttons used, accompanied by a short description of what each switch and button did. All user inputs enabled the user to threshold hsv colors with different settings. "1" means switch up, and "0" is switch down:

- switch[0] - 1: display hsv thresholds, 0: xy drone velocities on hex display
- switch[1] - 1: disable thresholding, 0: enable thresholding
- switch[2] - 1: show chroma keying, 0: show crosshairs
- switch[3] - 1: movable crosshair, 0: center-following crosshair

- switch[5:4] - 0: puck, 1: paddle1, 2: paddle2
- switch[7:6] - 0: h, 1: s, 2: v
- button_enter - set new thresholds
- button3 - decrease hsv_min
- button2 - increase hsv_min
- button1 - decrease hsv_max
- button0 - increase hsv_max

4.5.2 Divider Delays

Due to the Coregen dividers used in both rgb2hsv and detect_color, the hcount and vcounts being used to detect hsv values were delayed, thereby causing the chroma keying and object detection to be shifted on the vga monitor. To remedy this, we delayed one of the early pixels, video_border_pixel, using the staff provided **delayN** module, and used then used video_border_pixel_delayed as input to the rest of the affected vga modules. This basically shifts the pixel count in such a way that it is re-aligned with the shift delayed caused by the dividers used above.

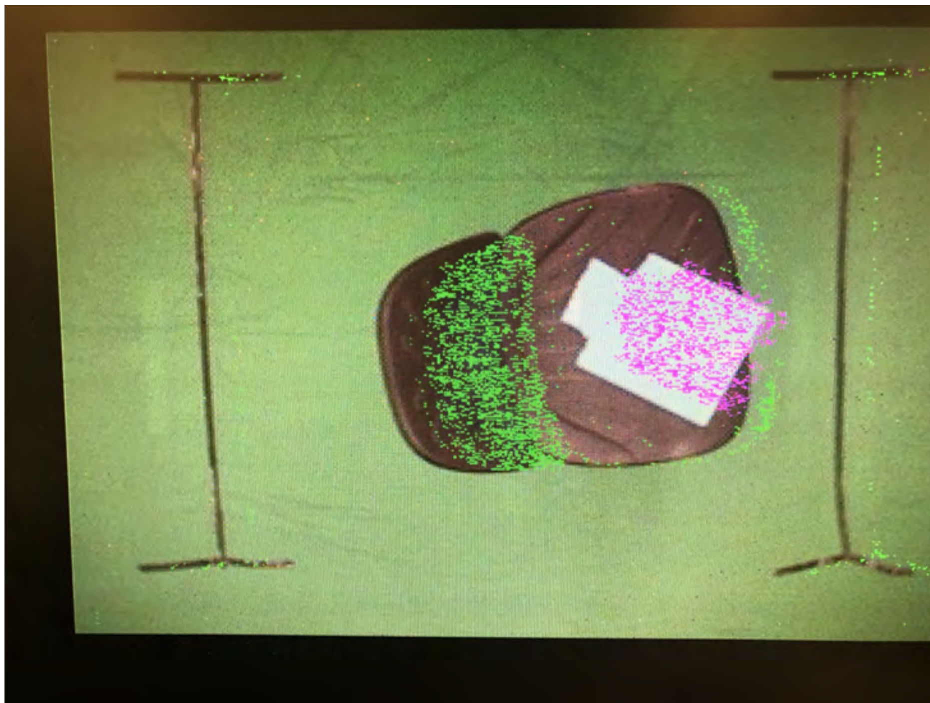


Figure 11: Shifted chroma keying due to divider delay

4.5.3 HEX Display Clocks

One thing that was really confusing was that sometimes the HEX Display Clocks would work, and sometimes it wouldn't, even when I didn't change my code between uploads to the labkit. The first change we made was changing the **display_16hex** module from combinational to sequential (thank you Driss TA). The second change we made came from the realization that the display_16hex module ran on the clock_27mhzk, however we were providing it with the main clk, which is way to fast for the hex display to process. Changing the input from clk to clock_27mhz immediately solved the problem (thanks David!). This was a very important bug for us to fix because much of the visualization code relied on using the HEX display for debugging to ensure that the correct pixel and HSV threshold values were being keyed and used.

4.5.4 Decreasing the Overall Compile Time

One major obstacle for me when implementing visualization was the LONG compile time. My compile tiles could range from 5 to 20 minutes, and at its longest, more than one hour (I stopped the process after I got tired of waiting). The long compile time made it very hard and time-consuming to debug the visualization modules, especially since we were so crunched for time at the end. We remedied this issue by commenting out many of the unused ports instantiated at the beginning of **zbt_6111_sample**. This made compile time SO MUCH faster because the FPGA didn't have to spend time looking for and connecting it didn't need. Futhermore, changing many of my sequential logic statements in my modules (ie. if-else statements) into combinational, also significantly helped save compile time. If possible, try to use combinational logic instead of sequential! Using these two optimizations, I decreased the compile time from around 20 minutes to 5-10 minutes!

4.6 Detailed Block Diagram of Visualization Block

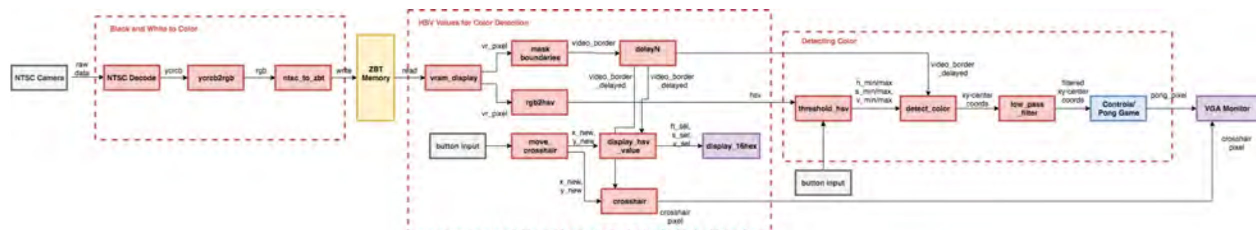


Figure 12: Huge apologies for the tiny diagram. You may have to zoom in alot on pdf to see the individual pieces more clearly.

5 Controls/Pong Game - David

5.1 Overview

The goal of the controls section was to implement and control the drone with logic of the pong game.

At a fundamental level, the velocity of the the pong puck can be relayed from the pong module to the drone with little intervention. Only a constant gain is needed to convert the relatively small integer values into a normalized velocity suitable for sending to the drone via WiFi.

The more advanced implementation of the control scheme uses a pair of state estimators in conjunction with a pair of proportional-derivative feedback controllers to directly control the roll and pitch of the drone.

Although it would be much easier to implement a controller without the inclusion of an estimator, the drone's does not send its sensor information over WiFi nearly fast enough for direct feedback control. As a result, the estimator is nessessary.

5.2 controls

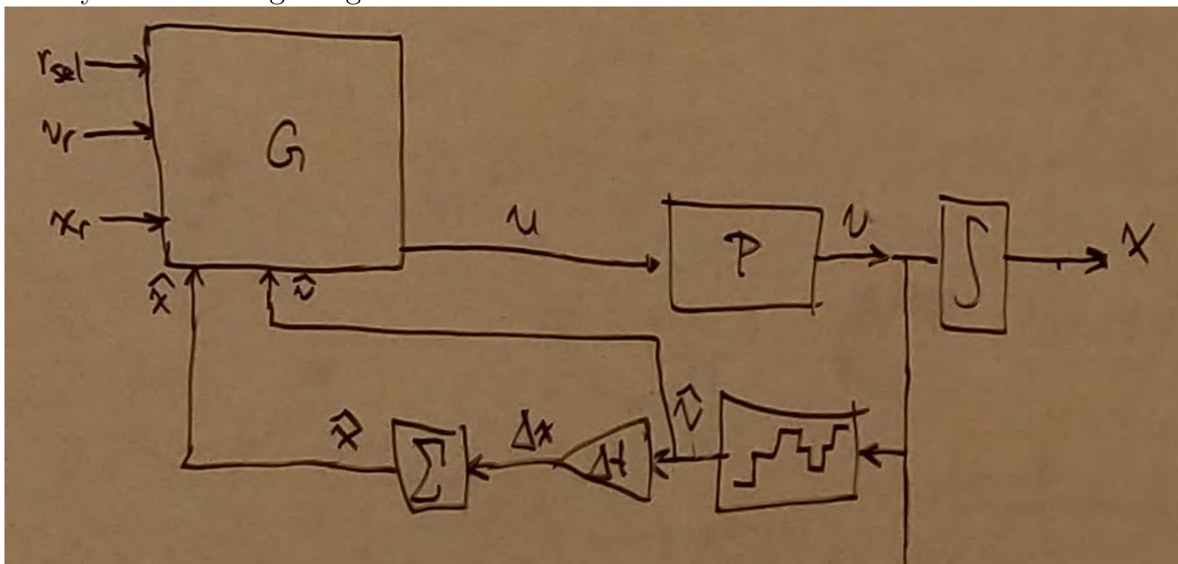
Creates and connects a pair of PD feedback controllers and corresponding state estimators.

The current implementation supports open-loop position control, or closed-loop velocity control.

NOTE: this version of the 'controls' module was not able to be tested in the fully integrated AirPong system. Thus, this module is NOT the one used for that integrated system. For that version, see the 'legacy_controls' module.

This control scheme is heavily reliant on proper operation of the estimator. Since the drone only sends sensor updates at 2Hz, the estimator is needed to propagate drone state forward so the feedback controllers can make more frequency updates than 2Hz.

Without the estimators, the drone *cannot* be made stable, as the feedback controllers essentially become bang-bang controllers.



5.2.1 Inputs

- clk: boolean 64.5 MHz clock
- reset: boolean Reset signal to clear all accumulated state, as well as intermediate calculations from various pipelines
- reference_x, reference_y: 16 bits as Q7.8 signed fixed point Reference position to track. Scaling: 256 units $\mapsto$ 1 meter
- reference_vx, reference_vy: 8 bits as Q1.6 signed fixed point Reference velocity to track. Scaling: 128 units $\mapsto$ 2 meters/sec
- sensor_updated: boolean 1 if the drone has sent updated sensor information.
- airborne: boolean 1 if the drone is currently in the air. Commands inhibited if 0
- in_flight: boolean 1 if the drone is commanded to be in the air. Commands inhibited if 0
- stable: boolean 1 if the drone is stable/suitable for controlling. Commands inhibited if 0

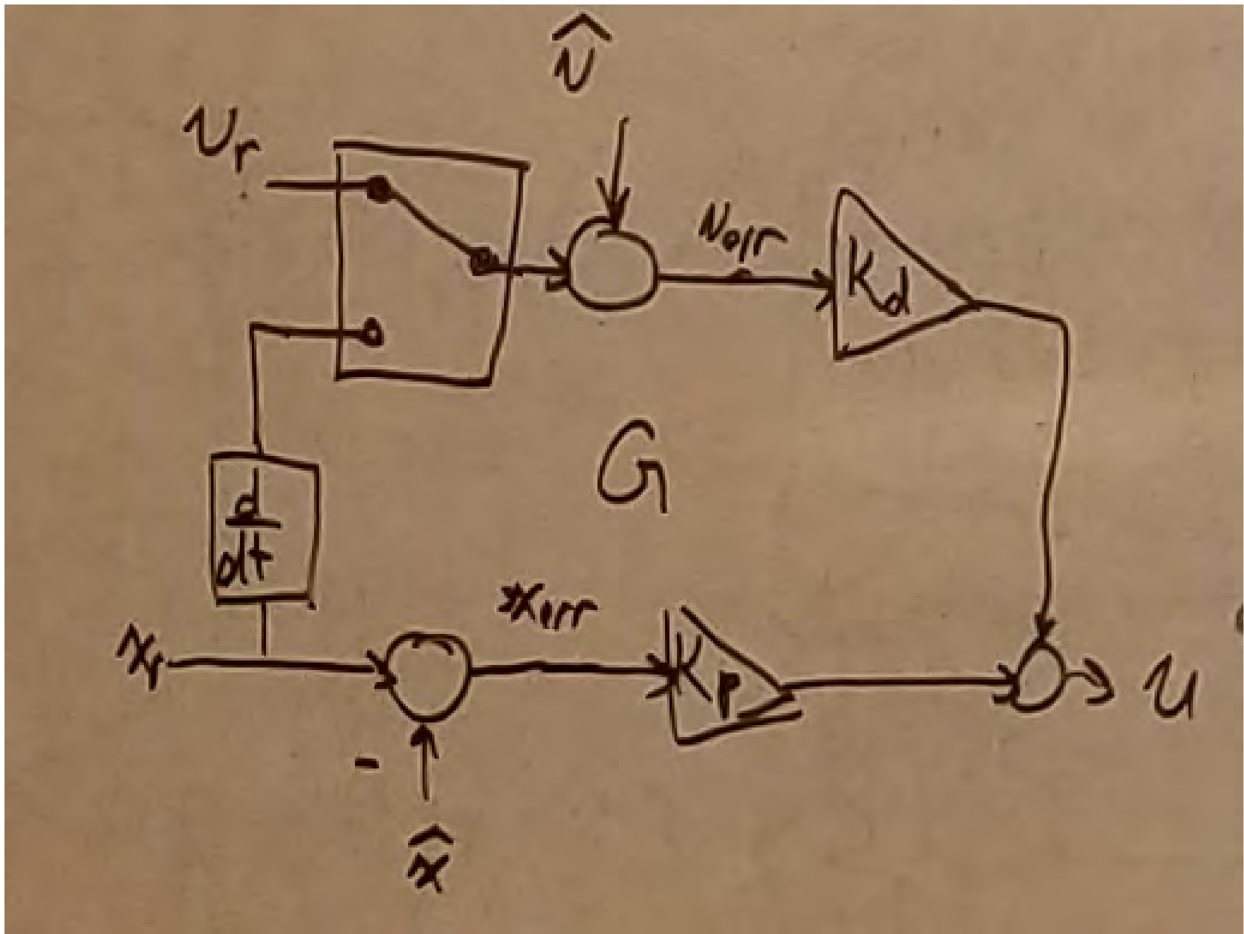
5.2.2 Outputs

- cmd_vx, cmd_vy: 8 bit signed integer Output commanded pitch and roll angles, respectively. Both values are in the range [-100, 100] Scaling: 128 units $\mapsto$ 45 degrees

5.3 feedback_controller

Implementation of a Proportional-Derivative feedback controller.

This module updates at 65MHz, while the drone's actuators can only be commanded at at 10Hz. As a result, latency from pipelining is not a concern.



5.3.1 Inputs

- clk: boolean 64.5 MHz clock
- reset: boolean Reset signal to clear all intermediate calculations from pipelines
- reference_p: 16 bits as Q7.8 signed fixed point Reference position to track Scaling: 256 units $\mapsto$ 1 meter
- reference_v: 8 bits as Q1.6 signed fixed point Reference velocity to track. Scaling: 128 units $\mapsto$ 2 meters/sec
- p_estimate: 16 bits as Q7.8 signed fixed point Estimated position Scaling: 256 units $\mapsto$ 1 meter
- v_estimate: 8 bits as Q1.6 signed fixed point Estimated velocity Scaling: 128 units $\mapsto$ 2 meters/sec

5.3.2 Outputs

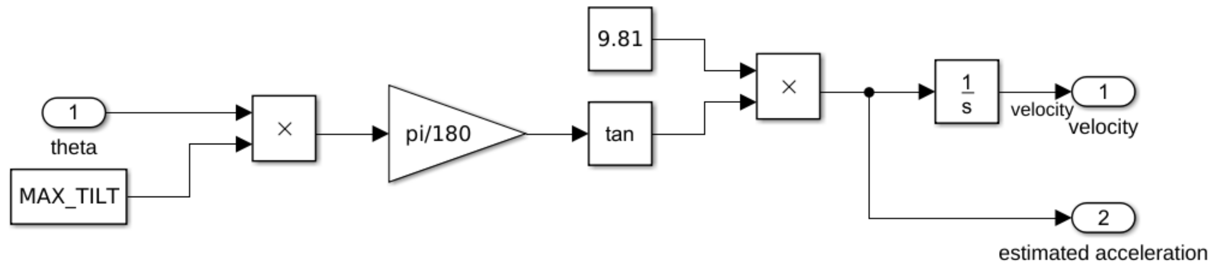
- command: 8 bit signed integer Commanded normalized angle, in the range [-100, 100] Scaling: 128 units $\mapsto$ 45 degrees

5.4 estimator

Position and velocity state estimator.

This module updates at roughly 65MHz, while the drone's actuators can only respond at 10Hz. As a result, latency from pipelining is not a concern.

Currently uses a small-angle approximation in lieu of the tangent function for the sake of simplicity and expediency.



CAUTION

Numerous design parameters were chosen specifically to simplify the math involved here, e.g. MAX_TILT. As a result, this module will be very brittle to changes in argument lengths, ranges, design parameters, and clock speeds.

5.4.1 Inputs

- clk: boolean 64.5 MHz clock
- reset: boolean Reset signal to clear all intermediate calculations from pipelines
- cmd_angle: 8 bit signed value Most recent commanded angle, in the range [-128, 127] Scaling: 128 units $\mapsto$ 45 degrees
- sensed_velocity: 8 bit signed value Most recent quantized sensor reading from the drone. Scaling: 128 units $\mapsto$ 2 meters/sec
- sensor_updated: boolean 1 if the drone has sent updated sensor information. This is used to update the velocity integrator with a reference truth value.

5.4.2 Outputs

- position_est: 16 bits as Q7.8 signed fixed point Estimated position Scaling: 256 units $\mapsto$ 1 meter
- velocity_est: 8 bits as Q1.6 signed fixed point Estimated velocity Scaling: 128 units $\mapsto$ 2 meters/sec

5.5 pong_game

Primary means of controlling and maintaining the logic of the pong game itself. Most of the logic is re-used from lab 3, but numerous changes were made to adapt it to lab 3.

5.5.1 Inputs

- reset
- enable
- pspeed
- hcount
- vcount
- hsync
- vsync
- blank
- base_pixel
- puck_x, paddle_1_x, paddle_2_x
- puck_y, paddle_1_y, paddle_2_y

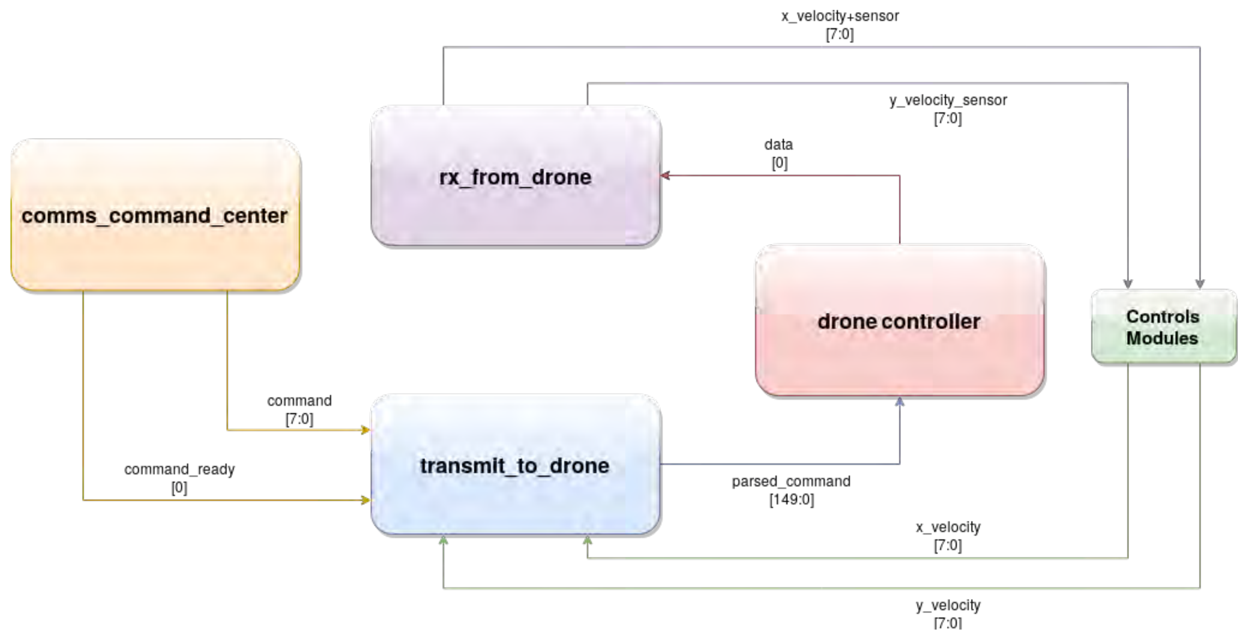
5.5.2 Outputs

- led
- pixel
- velocity_x, velocity_y
- puck_state
- halt

6 Communications - Bret

6.1 Overview

The goal of the communications section was to establish a connection with the drone, send commands to the drone, receive sensor data from the drone, and maintain this connection as necessary. This was all accomplished and worked quite well. The following diagram gives an overview of the communications modules and their interactions. One module, data_parser does not appear in the diagram, but is nested within both the rx_from_drone module and the transmit_to_drone module.



6.2 comms_command_center

The command center module acts as its name suggests: it is the main FSM for the communications to and from the drone and acts as the liaison between the drone and the other pieces of this puzzle. From a technical perspective, it is a simple state machine that performs as follows (all states are 4-bits in this section):
IDLE:

While in the 'IDLE' state, monitors the status of the 1-bit switch 0 (conveniently given the variable name 'start' within the module) on the labkit. As soon as 'start' has a value of '1' this FSM initiates a connection with the drone by assigning a specific value to the 3-bit 'command' variable, ensures that the serial transmitter module (transmit_to_drone module to be discussed later in this section) is informed that a command is ready by assigning the value '1' to the 1-bit variable 'command_ready', and changes the state to 'WAITING_FOR_COMMAND'. In case the reset button is pressed while the drone is 'IN_FLIGHT', the 'IDLE' state checks the 'airborne' variable, if holds the value '1' then the state is immediately changed to 'IN_FLIGHT' and command_ready is set to a value of '1'.

WAITING_FOR_COMMAND:

In this state, the FSM monitors the status of the 1-bit switch 1 (given the variable name 'takeoff_land') on the labkit and assigns a '0' to the 1-bit variable 'command_ready' to ensure the read buffer is not overloaded on the drone controller (in this case: a PC running a python script). Again, as soon as the switch is switched on (or to 1) the FSM assigns a specified value (indicating a takeoff command) to the 'command' variable, assigns the value '1' to the variable 'command_ready, and then changes the state to 'TAKING_OFF'.

TAKING_OFF:

This state is used as a hold over until the 'takeoff' command is sent. It sets the 1-bit 'command_ready' variable to 1'b0, again to ensure that the read buffer is not overloaded on the controller. Once the 'takeoff' command has been sent to the drone, the FSM sets the 1-bit flag 'airborne' to a value of '1', changes the state to 'IN_FLIGHT' and assigns a specified value to 'command' to command the drone to accept x and y velocities as inputs.

IN_FLIGHT:

The 'IN_FLIGHT' state ensures that the 'command_ready' variable remains at the value of '1' so that the drone constantly received updated x and y velocities. This state also monitors the 'takeoff_land' variable, if the variable is set back to '0' then the state is changed to 'LANDING' and a specified value is set to the 'command' variable alerting the drone that it needs to land.

LANDING:

Once in the 'LANDING' state, the FSM waits for the 'land' command to be sent to the drone and then changes states back to 'WAITING_FOR_COMMAND' and changes the 1-bit 'airborne' variable back to a value of '0'.

This module also contains a reset feature. Upon press of the reset button (mapped to 'enter' on the labkit), the state is set to 'IDLE' and the value of 'command_ready' is set to '0'.

6.3 data_parser

The data_parser module is used to take locally defined commands and translate them into full UDP network packets to be send to the drone. It is also used to receive and parse sensor updates from the drone.

When parsing commands this module compares the input command, 'command_to_parse', in a case statement. Based on the 'command_to_parse', the variables contained within 'command_out' are modified forming a full data packet for the drone. These variables and possible values for these variables include:

- data_type:
  As defined by the drone's API:
  - acknowledgement = 1
  - data without acknowledgement = 2
  - low latency data = 3
  - data with acknowledgement = 4

- buffer_type:
  As defined by the drone's API:
  - send with acknowledgement = 11
  - acknowledge drone data = 127
- buffer_counter
  Each buffer requires a sequence number starting at zero and increments with each new packet sent to that buffer:
  - send_with_acknowledge_counter = 0 to 255
  - acknowledge_drone_data_counter = 0 to 255
- packet_length
  Each packet sent to the drone requires an indication of the length of the packet. We only sent two different packet lengths to the drone.
  - packet_length = 11
  - packet_length = 15
- project_name
  This variable only has one possible value for this particular use. The project name is based on the type of drone and we only used one drone.
  - MINIDRONE = 2
- class_name
  Again, we only used one of the many possible selections of the class name. Class names range from 'Change Piloting Mode' to 'Stream Video'. We only used:
  - PILOTING = 0
- command_name
  As defined in the drone's API (PCMD is the command used to control the drone's velocities directly):
  - TAKEOFF = 1
  - PCMD = 2
  - LAND = 3
- roll_value
  This value is used to control the x velocity of the drone:
  - x_velocity = -100 to 100
- pitch_value
  This value is used to control the y velocity of the drone:

  - y_velocity = -100 to 100

- yaw_value
  This value is used to control the rotation about the z axis (up and down axis in this case) we chose to maintain a value of zero:
  - yaw_value = 0
- height_off_ground
  Maintains a certain height off of the ground. We chose to maintain a height of 1 meter.
  - height_off_ground = 1

In the event that the command to be parsed is a takeoff or land command, the roll_value, pitch_value, yaw_value, and height_off_ground are all set to a 10-bits of the value '1' to indicate an idle state over serial. The only inputs when being used as a command parser are 'command_to_parse', 'x_velocity', and 'y_velocity'. There is a single output when used as a command parser: 'command_out'; this is a 150-bit packet to be sent to the drone.
When the data parser is used as an incoming sensor data parser, the 16-bit array 'sensor_data' is received from the 'rx_from_drone' module. The first byte of 'sensor_data' indicates whether the data is an x velocity or y velocity. A case statement evaluates this, and based on that will assign a new value to either the 'x_velocity_sensor' or 'y_velocity_sensor' and indicate that the sensors have updated by assigned a value of '1' to the 'sensor_update' variable.

6.4 transmit_to_drone

This module is a serializer. It currently transmits data at 1 Mbaud. It maintains a 1 MHz clock using a the variable 'baud_counter'; once that counter reaches the value '27', the 'baud_clock' is triggered for one 27MHz clock cycle. If there is a command ready to be sent and the last command was sent more than 0.05 seconds ago, then the module will load the command and shift it out 1-bit at a time.

It works as two FSMs, one for short commands of one byte and on for long commands of 150 bits. Both FSMs work the same way, as long as no data is currently being processed, it has been more than 0.05 seconds since the last command, and a command is ready, then the data is loaded, state changed to 'busy' (named 'STOP_BIT' in the code) and shifted out. Once all data bits are shifted out, the FSM returns to the 'not_busy' state (named 'START_BIT' in the code).

6.5 rx_from_drone

The rx_from_drone module receives serialized data, samples that data, and translates it into the proper x_velocity_sensor and y_velocity_sensor. Each time this sensor data is sent to this module from the drone controller, it is sent 1 bit at a time and each data frame consists of the following in order:

- 1 start bit (a '0')

- 8 data bits (this comes before each sensor velocity indicating whether it is for the x or y direction)
- 1 stop bit (a '1')
- 1 start bit (a '0')
- 8 data bits (this is the x or y velocity)
- 1 stop bit (a '1')

When this data has been receive and decoded, it is forwarded directly to the controls modules.

7 Lessons Learned

We definitely learned a lot from this project. First, and foremost, we learned that the integrating modules together is not easy, especially when those modules were all written by different people.

We also learned that hard work and time spent doesn't always equal success. I (Bret) spent well over 100 hours on this project, between figuring out the necessary communications for the drone, trying and failing many times, and writing/testing the Verilog, my time was consumed quickly.

Additionally, I (Sabina) also learned the importance of writing modular code. Writing code that can stand on its own is very important, especially when needing to debug large systems. When integrating the visualization and pong game modules, because of my initially non-modular visualization code, it was very difficult to integrate because it was hard to see where many of the failures or problems were due to the interconnectedness of the code. It was very hard to separate where the problem was. Another lesson I learned was the importance of using test benches beforehand to test code. When debugging visualization, it often became very, very time-consuming waiting for the entire program to compile. Oftentimes it would take 10-15 minutes just to get one test in. Had I used test benches and made my modules more easy to unit test, the debugging process may have been completed more quickly.

8 Conclusion

Was our project an overwhelming success? Well, no. It was, as a whole, a failure, but we did each individually reach our goals. Each of our modules worked separately, and we were able to integrate a version of the controls modules with the communications and visualization modules, and perform small maneuvers with the drone. While this was not the goal of our project, it did demonstrate that our modules were able to perform.

9 Acknowledgements

10 Appendix

10.1 Visualization

10.1.1 Visualization: crosshair Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Sabina Chen
//
// Create Date:    14:09:51 12/07/2018
// Design Name:
// Module Name:    crosshair
// Project Name:   AirPong
// Target Devices:
// Tool versions:
// Description:    Takes center location coordinates,
//                 and draws crosshairs that intersect at that location
//
//                 Inputs: xy coordinates
//                 Outputs: updated pixel vga output w/ crosshairs
                            intersecting at the provided xy coordinates
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module crosshair(
    input clk,
    input [10:0] hcount,
    input [9:0] vcount,
    input center_sel,       // select between movable crosshair, to
                            // crosshair following center of object

    input [10:0] x_new_puck,    // manual moving
    input [9:0] y_new_puck,
    input [10:0] x_new_paddle1,
    input [9:0] y_new_paddle1,
    input [10:0] x_new_paddle2,
    input [9:0] y_new_paddle2,

    input [10:0] x_center_puck, // calculated coordinates by detect_color,
                                // center of objects
    input [9:0] y_center_puck,
    input [10:0] x_center_paddle1,
    input [9:0] y_center_paddle1,
```

```
    input [10:0] x_center_paddle2,
    input [9:0] y_center_paddle2,

    input [23:0] pixel,
    output [23:0] crosshair_pixel
    );

    parameter MAGENTA = {8'd255,8'd0,8'd255};
    parameter GREEN = {8'd0,8'd255,8'd0};
    parameter BLUE = {8'd0,8'd0,8'd255};

    wire adjustable_puck, adjustable_paddle1, adjustable_paddle2;
    wire centered_puck, centered_paddle1, centered_paddle2;
    wire no_object;

    assign adjustable_puck = (hcount==x_new_puck || vcount==y_new_puck)
                             && center_sel;
    assign adjustable_paddle1 = (hcount==x_new_paddle1 ||
                             vcount==y_new_paddle1) && center_sel;
    assign adjustable_paddle2 = (hcount==x_new_paddle2 ||
                             vcount==y_new_paddle2) && center_sel;

    assign centered_puck = (hcount==x_center_puck ||
                         vcount==y_center_puck) && ~center_sel;
    assign centered_paddle1 = (hcount==x_center_paddle1 ||
                         vcount==y_center_paddle1) && ~center_sel;
    assign centered_paddle2 = (hcount==x_center_paddle2 ||
                         vcount==y_center_paddle2) && ~center_sel;

    assign no_object = ~(adjustable_puck || adjustable_paddle1 ||
                         adjustable_paddle2) && ~(centered_puck ||
                         centered_paddle1 || centered_paddle2);

    // draw crosshairs that align with the center of objects
    assign crosshair_pixel = ((adjustable_puck || centered_puck)
                             ? MAGENTA : 24'b0) |
                             ((adjustable_paddle1 || centered_paddle1)
                             ? GREEN : 24'b0) |
                             ((adjustable_paddle2 || centered_paddle2)
                             ? BLUE : 24'b0) |
                             ((no_object) ? pixel : 24'b0);

endmodule
```

10.1.2 Visualization: debounce Module

```
///////////////////////////////////////////////////////////////////////////////
//
// Pushbutton Debounce Module
//
///////////////////////////////////////////////////////////////////////////////

module debounce (reset, clk, noisy, clean);
```

```
   input reset, clk, noisy;
   output clean;

   parameter NDELAY = 650000;
   parameter NBITS = 20;

   reg [NBITS-1:0] count;
   reg xnew, clean;

   always @(posedge clk)
     if (reset) begin xnew <= noisy; clean <= noisy; count <= 0; end
     else if (noisy != xnew) begin xnew <= noisy; count <= 0; end
     else if (count == NDELAY) clean <= xnew;
     else count <= count+1;

endmodule
```

10.1.3 Visualization: delayN Module

```
///////////////////////////////////////////////////////////////////////////////
// parameterized delay line - provided by staff
//                            modified to take inputs of varying sizes

module delayN #(parameter NDELAY=22, parameter LENGTH = 10)(clk,in,out);
   input clk;
   input [LENGTH-1:0] in;
   output [LENGTH-1:0] out;

   reg [LENGTH-1:0] shiftreg [NDELAY:0];
   reg [4:0] i;

   always @(posedge clk) begin
        shiftreg[0] <= in;
        for(i=1; i<NDELAY+1; i=i+1) begin
            shiftreg[i] <= shiftreg[i-1];
        end
    end

    assign out = shiftreg[NDELAY];

endmodule
```

10.1.4 Visualization: detect_color Module

```
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Sabina chen
//
// Create Date: 18:13:29 12/06/2018
// Design Name:
// Module Name: detect_color
// Project Name: AirPong
// Target Devices:
// Tool versions:
```

```
// Description: Given HSV boundaries for each object, determine which pixels lie within
//              the HSV boundaries and color that pixel. At the same time, keep a counter
//              on the number of pixels that fall within bounds to calculate the average
//              location of the pixels.
//
//              Input: HSV boundaries
//              Outputs: 1) Updated vga pixel output with colored pixels representing detec
//                       2) xy pixel locations of the center of mass of detected color pixe
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module detect_color
    (input clk,
    input [7:0] H1_MIN_PUCK, H1_MAX_PUCK, H2_MIN_PUCK, H2_MAX_PUCK,
            S_MIN_PUCK, S_MAX_PUCK, V_MIN_PUCK, V_MAX_PUCK, // SABINA10
    input [7:0] H1_MIN_PADDLE1, H1_MAX_PADDLE1, H2_MIN_PADDLE1,
            H2_MAX_PADDLE1, S_MIN_PADDLE1, S_MAX_PADDLE1, V_MIN_PADDLE1,
            V_MAX_PADDLE1, // SABINA10
    input [7:0] H1_MIN_PADDLE2, H1_MAX_PADDLE2, H2_MIN_PADDLE2,
            H2_MAX_PADDLE2, S_MIN_PADDLE2, S_MAX_PADDLE2, V_MIN_PADDLE2,
            V_MAX_PADDLE2, // SABINA10
    input [23:0] hsv,
   input [10:0] h_count, vid_border_left, vid_border_right,
   input [9:0] v_count, vid_border_up, vid_border_down,
   input [23:0] pixel,
   output reg [23:0] color_pixel,
    output reg [24:0] x_total_puck, y_total_puck,
    output reg [24:0] x_total_paddle1, y_total_paddle1,
    output reg [24:0] x_total_paddle2, y_total_paddle2,
   output [10:0] x_center_puck_filtered, x_center_paddle1_filtered,
                 x_center_paddle2_filtered,
   output [9:0] y_center_puck_filtered, y_center_paddle1_filtered,
                 y_center_paddle2_filtered
   );

   wire [7:0] h, s, v;
   assign h = hsv[23:16];
   assign s = hsv[15:8];
   assign v = hsv[7:0];

   reg [24:0] x_counter_puck = 0;
   reg [24:0] y_counter_puck = 0;
   reg [24:0] x_numerator_puck = 0;
   reg [24:0] x_denominator_puck = 0;
   reg [24:0] y_numerator_puck = 0;
   reg [24:0] y_denominator_puck = 0;

    reg [24:0] x_counter_paddle1 = 0;
```

```
reg [24:0] y_counter_paddle1 = 0;
reg [24:0] x_numerator_paddle1 = 0;
reg [24:0] x_denominator_paddle1 = 0;
reg [24:0] y_numerator_paddle1 = 0;
reg [24:0] y_denominator_paddle1 = 0;

 reg [24:0] x_counter_paddle2 = 0;
reg [24:0] y_counter_paddle2 = 0;
reg [24:0] x_numerator_paddle2 = 0;
reg [24:0] x_denominator_paddle2 = 0;
reg [24:0] y_numerator_paddle2 = 0;
reg [24:0] y_denominator_paddle2 = 0;

// color the pixels that lie within hsv bounds
always @(posedge clk) begin
   // reset counters at the beginning of each new frame
   if(h_count==11'd0 && v_count==10'd0)begin
      color_pixel <= pixel;
      // puck
      x_total_puck <= 0;
      y_total_puck <= 0;
      x_counter_puck <= 0;
      y_counter_puck <= 0;
         // paddle1
         x_total_paddle1 <= 0;
      y_total_paddle1 <= 0;
      x_counter_paddle1 <= 0;
      y_counter_paddle1 <= 0;
         // paddle2
         x_total_paddle2 <= 0;
      y_total_paddle2 <= 0;
      x_counter_paddle2 <= 0;
      y_counter_paddle2 <= 0;
   end

   // if in frame, detect color
   else if(h_count>=vid_border_left && h_count<=vid_border_right &&
            v_count>=vid_border_up && v_count<=vid_border_down) begin
         // puck
      if( ((h>H1_MIN_PUCK && h<=H1_MAX_PUCK) || h>H2_MIN_PUCK &&
            h<=H2_MAX_PUCK) && (s>S_MIN_PUCK && s<=S_MAX_PUCK) &&
            (v>V_MIN_PUCK && v<=V_MAX_PUCK)) begin
         color_pixel <= {8'd255,8'd0,8'd255}; // shade magenta
         x_total_puck <= x_total_puck + h_count;
         y_total_puck <= y_total_puck + v_count;
         x_counter_puck <= x_counter_puck + 1;
         y_counter_puck <= y_counter_puck + 1;
      end
         else begin
         color_pixel <= pixel;
         x_total_puck <= x_total_puck;
         y_total_puck <= y_total_puck;
         x_counter_puck <= x_counter_puck;
         y_counter_puck <= y_counter_puck;
```

```
        x_numerator_puck <= x_numerator_puck;
        y_numerator_puck <= y_numerator_puck;
        x_denominator_puck <= x_denominator_puck;
        y_denominator_puck <= y_denominator_puck;
        end
        // paddle1
     if( ((h>H1_MIN_PADDLE1 && h<=H1_MAX_PADDLE1) || h>H2_MIN_PADDLE1
           && h<=H2_MAX_PADDLE1) && (s>S_MIN_PADDLE1 && s<=S_MAX_PADDLE1)
           && (v>V_MIN_PADDLE1 && v<=V_MAX_PADDLE1)) begin
        color_pixel <= {8'd0,8'd255,8'd0}; // shade green
        x_total_paddle1 <= x_total_paddle1 + h_count;
        y_total_paddle1 <= y_total_paddle1 + v_count;
        x_counter_paddle1 <= x_counter_paddle1 + 1;
        y_counter_paddle1 <= y_counter_paddle1 + 1;
     end
        else begin
            x_total_paddle1 <= x_total_paddle1;
        y_total_paddle1 <= y_total_paddle1;
        x_counter_paddle1 <= x_counter_paddle1;
        y_counter_paddle1 <= y_counter_paddle1;
        x_numerator_paddle1 <= x_numerator_paddle1;
        y_numerator_paddle1 <= y_numerator_paddle1;
        x_denominator_paddle1 <= x_denominator_paddle1;
        y_denominator_paddle1 <= y_denominator_paddle1;
        end
        // paddle2
     if( ((h>H1_MIN_PADDLE2 && h<=H1_MAX_PADDLE2) || h>H2_MIN_PADDLE2
           && h<=H2_MAX_PADDLE2) && (s>S_MIN_PADDLE2 && s<=S_MAX_PADDLE2)
           && (v>V_MIN_PADDLE2 && v<=V_MAX_PADDLE2)) begin
        color_pixel <= {8'd0,8'd0,8'd255}; // shade blue
        x_total_paddle2 <= x_total_paddle2 + h_count;
        y_total_paddle2 <= y_total_paddle2 + v_count;
        x_counter_paddle2 <= x_counter_paddle2 + 1;
        y_counter_paddle2 <= y_counter_paddle2 + 1;
     end
        else begin
            // paddle2
            x_total_paddle2 <= x_total_paddle2;
        y_total_paddle2 <= y_total_paddle2;
        x_counter_paddle2 <= x_counter_paddle2;
        y_counter_paddle2 <= y_counter_paddle2;
        x_numerator_paddle2 <= x_numerator_paddle2;
        y_numerator_paddle2 <= y_numerator_paddle2;
        x_denominator_paddle2 <= x_denominator_paddle2;
        y_denominator_paddle2 <= y_denominator_paddle2;
     end
  end

  // start divider immediately after past video boundary, if enough
  // detected pixels then find xy-center
  else if (h_count == 600 && v_count == 800) begin
        //puck
     x_numerator_puck <= (x_counter_puck > 50) ? x_total_puck : 0;
     y_numerator_puck <= (y_counter_puck > 50) ? y_total_puck : 0;
```

```
      x_denominator_puck <= (x_counter_puck==0) ? 1 : x_counter_puck;
      y_denominator_puck <= (y_counter_puck==0) ? 1 : y_counter_puck;
         //paddle1
         x_numerator_paddle1 <= (x_counter_paddle1 > 50) ?
                               x_total_paddle1 : 0;
      y_numerator_paddle1 <= (y_counter_paddle1 > 50) ?
                               y_total_paddle1 : 0;
      x_denominator_paddle1 <= (x_counter_paddle1==0) ? 1 :
                                  x_counter_paddle1;
      y_denominator_paddle1 <= (y_counter_paddle1==0) ? 1 :
                                  y_counter_paddle1;
         //paddle2
         x_numerator_paddle2 <= (x_counter_paddle2 > 50) ?
                                  x_total_paddle2 : 0;
      y_numerator_paddle2 <= (y_counter_paddle2 > 50) ?
                                  y_total_paddle2 : 0;
      x_denominator_paddle2 <= (x_counter_paddle2==0) ? 1 :
                                  x_counter_paddle2;
      y_denominator_paddle2 <= (y_counter_paddle2==0) ? 1 :
                                  y_counter_paddle2;
   end

   // if not in frame, do nothing
     else begin
         color_pixel <= pixel;
     end
end

// divide xy_total / xy_counter to get average h/vcount location
 // puck
wire [24:0] x_quotient_puck;
wire [24:0] x_remainder_puck;
 wire [24:0] y_quotient_puck;
wire [24:0] y_remainder_puck;
wire x_ready_puck, y_ready_puck;
 // paddle1
wire [24:0] x_quotient_paddle1;
wire [24:0] x_remainder_paddle1;
 wire [24:0] y_quotient_paddle1;
wire [24:0] y_remainder_paddle1;
wire x_ready_paddle1, y_ready_paddle1;
 // paddle2
wire [24:0] x_quotient_paddle2;
wire [24:0] x_remainder_paddle2;
 wire [24:0] y_quotient_paddle2;
wire [24:0] y_remainder_paddle2;
wire x_ready_paddle2, y_ready_paddle2;

 // puck
 avgdivider x_div_puck(.clk(clk), .dividend(x_numerator_puck),
   .divisor(x_denominator_puck), .quotient(x_quotient_puck),
   .fractional(x_remainder_puck), .rfd(x_ready_puck));
 avgdivider y_div_puck(.clk(clk), .dividend(y_numerator_puck),
   .divisor(y_denominator_puck), .quotient(y_quotient_puck),
```

```
      .fractional(y_remainder_puck), .rfd(y_ready_puck));
    // paddle1
    avgdivider x_div_paddle1(.clk(clk), .dividend(x_numerator_paddle1),
      .divisor(x_denominator_paddle1), .quotient(x_quotient_paddle1),
      .fractional(x_remainder_paddle1), .rfd(x_ready_paddle1));
    avgdivider y_div_paddle1(.clk(clk), .dividend(y_numerator_paddle1),
      .divisor(y_denominator_paddle1), .quotient(y_quotient_paddle1),
      .fractional(y_remainder_paddle1), .rfd(y_ready_paddle1));
    // paddle2
    avgdivider x_div_paddle2(.clk(clk), .dividend(x_numerator_paddle2),
      .divisor(x_denominator_paddle2), .quotient(x_quotient_paddle2),
      .fractional(x_remainder_paddle2), .rfd(x_ready_paddle2));
    avgdivider y_div_paddle2(.clk(clk), .dividend(y_numerator_paddle2),
      .divisor(y_denominator_paddle2), .quotient(y_quotient_paddle2),
      .fractional(y_remainder_paddle2), .rfd(y_ready_paddle2));

    wire [10:0] x_center_puck, x_center_paddle1, x_center_paddle2;
    wire [9:0] y_center_puck, y_center_paddle1, y_center_paddle2;

    // puck
   assign x_center_puck = x_ready_puck ? x_quotient_puck[10:0]
                             : x_center_puck;
   assign y_center_puck = y_ready_puck ? y_quotient_puck[9:0]
                             : y_center_puck;
    // paddle1
   assign x_center_paddle1 = x_ready_paddle1 ? x_quotient_paddle1[10:0]
                             : x_center_paddle1;
   assign y_center_paddle1 = y_ready_paddle1 ? y_quotient_paddle1[9:0]
                             : y_center_paddle1;
    // paddle2
   assign x_center_paddle2 = x_ready_paddle2 ? x_quotient_paddle2[10:0]
                             : x_center_paddle2;
   assign y_center_paddle2 = y_ready_paddle2 ? y_quotient_paddle2[9:0]
                             : y_center_paddle2;

   // pass coordinates theough a low pass filter to smooth out jumpiness
    low_pass_filter x_center_puck_coord(.clk(clk),.input_data(x_center_puck),
                           .post_filter_data(x_center_puck_filtered));
    low_pass_filter y_center_puck_coord(.clk(clk),.input_data(y_center_puck),
                           .post_filter_data(y_center_puck_filtered));
    low_pass_filter x_center_paddle1_coord(.clk(clk),.input_data(x_center_paddle1),
                           .post_filter_data(x_center_paddle1_filtered));
    low_pass_filter y_center_paddle1_coord(.clk(clk),.input_data(y_center_paddle1),
                           .post_filter_data(y_center_paddle1_filtered));
    low_pass_filter x_center_paddle2_coord(.clk(clk),.input_data(x_center_paddle2),
                           .post_filter_data(x_center_paddle2_filtered));
    low_pass_filter y_center_paddle2_coord(.clk(clk),.input_data(y_center_paddle2),
                           .post_filter_data(y_center_paddle2_filtered));

endmodule
```

10.1.5 Visualization: display_16hex Module

```
///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Hex display driver
//
//
// File:   display_16hex.v
// Date:   24-Sep-05
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
// This module drives the labkit hex displays and shows the value of
// 8 bytes (16 hex digits) on the displays.
//
// 24-Sep-05 Ike: updated to use new reset-once state machine, remove clear
// 02-Nov-05 Ike: updated to make it completely synchronous
//
// Inputs:
//
//   reset       - active high
//   clock_27mhz - the synchronous clock
//   data        - 64 bits; each 4 bits gives a hex digit
//
// Outputs:
//
//    disp_*     - display lines used in the 6.111 labkit (rev 003 & 004)
//
///////////////////////////////////////////////////////////////////////////////

module display_16hex (reset, clock_27mhz, data_in,
        disp_blank, disp_clock, disp_rs, disp_ce_b,
        disp_reset_b, disp_data_out);

   input reset, clock_27mhz;    // clock and reset (active high reset)
   input [63:0] data_in;        // 16 hex nibbles to display

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

   reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

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

   reg [5:0] count;
   reg [7:0] reset_count;
//   reg         old_clock;
   wire      dreset;
   wire      clock = (count<27) ? 0 : 1;
```

```
   always @(posedge clock_27mhz)
     begin
    count <= reset ? 0 : (count==53 ? 0 : count+1);
    reset_count <= reset ? 100 : ((reset_count==0) ? 0 : reset_count-1);
//  old_clock <= clock;
     end

   assign dreset = (reset_count != 0);
   assign disp_clock = ~clock;
   wire   clock_tick = ((count==27) ? 1 : 0);
//   wire   clock_tick = clock & ~old_clock;

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

   reg [7:0] state;     // FSM state
   reg [9:0] dot_index;     // index to current dot being clocked out
   reg [31:0] control;      // control register
   reg [3:0] char_index;    // index of current character
   reg [39:0] dots;     // dots for a single digit
   reg [3:0] nibble;        // hex nibble of current character
   reg [63:0] data;

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

   always @(posedge clock_27mhz)
     if (clock_tick)
      begin
      if (dreset)
        begin
           state <= 0;
           dot_index <= 0;
           control <= 32'h7F7F7F7F;
        end
      else
        casex (state)
          8'h00:
        begin
           // Reset displays
           disp_data_out <= 1'b0;
           disp_rs <= 1'b0; // dot register
           disp_ce_b <= 1'b1;
           disp_reset_b <= 1'b0;
           dot_index <= 0;
           state <= state+1;
        end

          8'h01:
        begin
           // End reset
```

```
      disp_reset_b <= 1'b1;
      state <= state+1;
   end

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

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

     8'h04:
   begin
      // Setup the control register
      disp_rs <= 1'b1; // Select the control register
      disp_ce_b <= 1'b0;
      disp_data_out <= control[31];
      control <= {control[30:0], 1'b0};    // shift left
      if (dot_index == 0)
        state <= state+1;
      else
        dot_index <= dot_index-1;
   end

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

     8'h06:
   begin
      // Load the user's dot data into the dot reg, char by char
      disp_rs <= 1'b0;         // Select the dot register
      disp_ce_b <= 1'b0;
      disp_data_out <= dots[dot_index]; // dot data from msb
      if (dot_index == 0)
            if (char_index == 0)
```

```
                  state <= 5;          // all done, latch data
            else
              begin
             char_index <= char_index - 1; // goto next char
             data <= data_in;
             dot_index <= 39;
              end
          else
            dot_index <= dot_index-1;  // else loop thru all dots
       end

       endcase // casex(state)
      end

//always @ (data or char_index)
 always @(posedge clock_27mhz) begin
  case (char_index)
    4'h0:        nibble <= data[3:0];
    4'h1:        nibble <= data[7:4];
    4'h2:        nibble <= data[11:8];
    4'h3:        nibble <= data[15:12];
    4'h4:        nibble <= data[19:16];
    4'h5:        nibble <= data[23:20];
    4'h6:        nibble <= data[27:24];
    4'h7:        nibble <= data[31:28];
    4'h8:        nibble <= data[35:32];
    4'h9:        nibble <= data[39:36];
    4'hA:        nibble <= data[43:40];
    4'hB:        nibble <= data[47:44];
    4'hC:        nibble <= data[51:48];
    4'hD:        nibble <= data[55:52];
    4'hE:        nibble <= data[59:56];
    4'hF:        nibble <= data[63:60];
  endcase

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

```
    end

endmodule
```

10.1.6 Visualization: display_hsv_value Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Sabina Chen
//
// Create Date:    14:48:58 12/07/2018
// Design Name:
// Module Name:    display_hsv_value
// Project Name:   AirPong
// Target Devices:
// Tool versions:
// Description:    Given a xy pixel coordinate location, output the hsv value for the selec
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module display_hsv_value(
   input clk,
   input [10:0] hcount,
   input [9:0] vcount,
   input [10:0] x_coord,
   input [9:0] y_coord,
   input [23:0] pixel,
   input [23:0] hsv,
   output reg [7:0] h_sel,
   output reg [7:0] s_sel,
   output reg [7:0] v_sel
   );

   // output the hsv value at the selected xy coordinate
    always @(posedge clk) begin
        if(hcount==x_coord && vcount==y_coord) begin
            h_sel <= hsv[23:16];
            s_sel <= hsv[15:8];
            v_sel <= hsv[7:0];
        end
    end

endmodule
```

10.1.7 Visualization: low_pass_filter Module

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

```
////////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    21:43:38 12/10/2018
// Design Name:
// Module Name:    low_pass_filter
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////////
module low_pass_filter(
    input clk,
    input [10:0] input_data,
    output reg [10:0] post_filter_data);

    reg [10:0] lpf_gain, previous_output_data;

    always @(posedge clk) begin
        if (input_data < previous_output_data)
        begin
            lpf_gain = (previous_output_data - input_data)>>4;
            post_filter_data = previous_output_data - lpf_gain;
        end
        else
        begin
            lpf_gain = (input_data - previous_output_data)>>4;
            post_filter_data = lpf_gain + previous_output_data;
        end
        previous_output_data = post_filter_data;
    end

endmodule
```

10.1.8 Visualization: move_crosshair Module

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    21:09:20 12/08/2018
// Design Name:
// Module Name:    move_crosshair
// Project Name:
// Target Devices:
```

```
// Tool versions:
// Description:    Given user button inputs, move the crosshairs in the
                    selected direction.
//
//                 Inputs: user button presses
//                 Outputs: updated xy coordinate locations for the
                    crosshairs
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module move_crosshair(
    input clk, // vsync
    input enable,
    input btn_up_debounced,
    input btn_down_debounced,
    input btn_left_debounced,
    input btn_right_debounced,
    output reg [10:0] x_new = 11'd400,
    output reg [9:0] y_new = 10'd300
    );

    parameter DELTA = 2;
    parameter X_MIN = 0;
    parameter X_MAX = 1023;
    parameter Y_MIN = 0;
    parameter Y_MAX = 767;

    // determine if the new coordinate at the next timestep overshoots
    // the video boundaries
    wire overshoot_left, overshoot_right, overshoot_up, overshoot_down;
    assign overshoot_left = btn_left_debounced && (x_new > X_MIN + DELTA)
                            && enable;
    assign overshoot_right = btn_right_debounced && (x_new < X_MAX - DELTA)
                            && enable;
    assign overshoot_up = btn_up_debounced && (y_new > Y_MIN + DELTA)
                            && enable;
    assign overshoot_down = btn_down_debounced && (y_new < Y_MAX - DELTA)
                            && enable;

    // if the next move does not overshoot the video boundaries,
    // update the coordinate location
    always @(negedge clk) begin
        case({overshoot_left, overshoot_right})
            2'b00: x_new = x_new;
            2'b01: x_new = x_new + DELTA;
            2'b10: x_new = x_new - DELTA;
            2'b11: x_new = x_new;
        endcase
        case({overshoot_up, overshoot_down})
```

```
            2'b00: y_new = y_new;
            2'b01: y_new = y_new + DELTA;
            2'b10: y_new = y_new - DELTA;
            2'b11: y_new = y_new;
        endcase
    end

endmodule
```

10.1.9 Visualization: ntsc2zbt Module

```
//
// File:   ntsc2zbt.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Example for MIT 6.111 labkit showing how to prepare NTSC data
// (from Javier's decoder) to be loaded into the ZBT RAM for video
// display.
//
// The ZBT memory is 36 bits wide; we only use 32 bits of this, to
// store 4 bytes of black-and-white intensity data from the NTSC
// video input.
//
// Bug fix: Jonathan P. Mailoa <jpmailoa@mit.edu>
// Date   : 11-May-09  // gph mod 11/3/2011
//
//
// Bug due to memory management will be fixed. It happens because
// the memory addressing protocol is off between ntsc2zbt.v and
// vram_display.v. There are 2 solutions:
// -. Fix the memory addressing in this module (neat addressing protocol)
//    and do memory forecast in vram_display module.
// -. Do nothing in this module and do memory forecast in vram_display
//    module (different forecast count) while cutting off reading from
//    address(0,0,0).
//
// Bug in this module causes 4 pixel on the rightmost side of the camera
// to be stored in the address that belongs to the leftmost side of the
// screen.
//
// In this example, the second method is used. NOTICE will be provided
// on the crucial source of the bug.
//
/////////////////////////////////////////////////////////////////////////////
// Prepare data and address values to fill ZBT memory with NTSC data

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

   input      clk;   // system clock
   input      vclk;  // video clock from camera
   input [2:0]    fvh;
   input      dv;
   input [17:0]      din; // Sabina - 7->18 bits for bw->color
```

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

parameter     COL_START = 10'd30; // Sabina - changed starting drawing location
 //parameter      COL_START = 10'd800;
parameter     ROW_START = 10'd30;

// here put the luminance data from the ntsc decoder into the ram
// this is for 1024 * 788 XGA display

reg [9:0]     col = 0;
reg [9:0]     row = 0;
reg [17:0]    vdata = 0; // Sabina - 7->18 bits for bw->color
reg       vwe;
reg       old_dv;
reg       old_frame; // frames are even / odd interlaced
reg       even_odd;  // decode interlaced frame to this wire

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

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

 if (!fvh[2])
   begin
      col <= fvh[0] ? COL_START :
         (!fvh[2] && !fvh[1] && dv && (col < 1024)) ? col + 1 : col;
      row <= fvh[1] ? ROW_START :
         (!fvh[2] && fvh[0] && (row < 768)) ? row + 1 : row;
      vdata <= (dv && !fvh[2]) ? din : vdata;
   end
  end

// synchronize with system clock

reg [9:0] x[1:0],y[1:0];
reg [17:0] data[1:0];        // Sabina - 7->18 bits for bw->color
reg       we[1:0];
reg       eo[1:0];

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

```
  end

// edge detection on write enable signal

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

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

reg [35:0] mydata;
always @(posedge clk)
  if (we_edge)
    mydata <= { mydata[17:0], data[1] }; // // Sabina - 23->18 bits for bw->color

// NOTICE : Here we have put 4 pixel delay on mydata. For example, when:
// (x[1], y[1]) = (60, 80) and eo[1] = 0, then:
// mydata[31:0] = ( pixel(56,160), pixel(57,160), pixel(58,160), pixel(59,160) )
// This is the root of the original addressing bug.


// NOTICE : Notice that we have decided to store mydata, which
//          contains pixel(56,160) to pixel(59,160) in address
//          (0, 160 (10 bits), 60 >> 2 = 15 (8 bits)).
//
//          This protocol is dangerous, because it means
//          pixel(0,0) to pixel(3,0) is NOT stored in address
//          (0, 0 (10 bits), 0 (8 bits)) but is rather stored
//          in address (0, 0 (10 bits), 4 >> 2 = 1 (8 bits)). This
//          calculation ignores COL_START & ROW_START.
//
//          4 pixels from the right side of the camera input will
//          be stored in address corresponding to x = 0.
//
//          To fix, delay col & row by 4 clock cycles.
//          Delay other signals as well.

reg [39:0] x_delay;
reg [39:0] y_delay;
reg [3:0] we_delay;
reg [3:0] eo_delay;

always @ (posedge clk)
begin
  x_delay <= {x_delay[29:0], x[1]};
  y_delay <= {y_delay[29:0], y[1]};
  we_delay <= {we_delay[2:0], we[1]};
  eo_delay <= {eo_delay[2:0], eo[1]};
end

// compute address to store data in
wire [8:0] y_addr = y_delay[38:30];
 wire [9:0] x_addr = x_delay[39:30];
```

```
  //wire [18:0] myaddr = {1'b0, y_addr[8:0], eo_delay[3], x_addr[9:2]};
  wire [18:0] myaddr = {y_addr[8:0], eo_delay[3], x_addr[9:1]};
   // Sabina - shift left

  // Now address (0,0,0) contains pixel data(0,0) etc.


  // alternate (256x192) image data and address
  wire [31:0] mydata2 = {data[1],data[1],data[1],data[1]};
  wire [18:0] myaddr2 = {1'b0, y_addr[8:0], eo_delay[3], x_addr[7:0]};

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

  reg [18:0] ntsc_addr;
  reg [35:0] ntsc_data;
  //wire ntsc_we = sw ? we_edge : (we_edge & (x_delay[31:30]==2'b00));
    wire ntsc_we = sw ? we_edge : (we_edge & (x_delay[30]==0));
     // Sabina - 7->18 bits for bw->color

  always @(posedge clk)
    if ( ntsc_we )
      begin
     ntsc_addr <= sw ? myaddr2 : myaddr;   // normal and expanded modes
     ntsc_data <= sw ? {4'b0,mydata2} : mydata; // Sabina - remove 4'b0
      end

endmodule // ntsc_to_zbt
```

10.1.10 Visualization: ramclock Module

```
////////////////////////////////////////////////////////////////////////////
// ramclock module

///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- ZBT RAM clock generation
//
//
// Created: April 27, 2004
// Author: Nathan Ickes
//
///////////////////////////////////////////////////////////////////////////////
//
// This module generates deskewed clocks for driving the ZBT SRAMs and FPGA
// registers. A special feedback trace on the labkit PCB (which is length
// matched to the RAM traces) is used to adjust the RAM clock phase so that
// rising clock edges reach the RAMs at exactly the same time as rising clock
// edges reach the registers in the FPGA.
//
// The RAM clock signals are driven by DDR output buffers, which further
// ensures that the clock-to-pad delay is the same for the RAM clocks as it is
// for any other registered RAM signal.
//
// When the FPGA is configured, the DCMs are enabled before the chip-level I/O
```

```
// drivers are released from tristate. It is therefore necessary to
// artificially hold the DCMs in reset for a few cycles after configuration.
// This is done using a 16-bit shift register. When the DCMs have locked, the
// <lock> output of this mnodule will go high. Until the DCMs are locked, the
// ouput clock timings are not guaranteed, so any logic driven by the
// <fpga_clock> should probably be held inreset until <locked> is high.
//
///////////////////////////////////////////////////////////////////////////////

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

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

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

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

   //To force ISE to compile the ramclock, this line has to be removed.
   //IBUFG ref_buf (.O(ref_clk), .I(ref_clock));

    assign ref_clk = ref_clock;

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

   DCM int_dcm (.CLKFB(fpga_clock),
        .CLKIN(ref_clk),
        .RST(dcm_reset),
        .CLK0(fpga_clk),
        .LOCKED(lock1));
   // synthesis attribute DLL_FREQUENCY_MODE of int_dcm is "LOW"
   // synthesis attribute DUTY_CYCLE_CORRECTION of int_dcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of int_dcm is "FALSE"
   // synthesis attribute DFS_FREQUENCY_MODE of int_dcm is "LOW"
   // synthesis attribute CLK_FEEDBACK of int_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of int_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of int_dcm is 0

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

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

   DCM ext_dcm (.CLKFB(fb_clk),
            .CLKIN(ref_clk),
            .RST(dcm_reset),
            .CLK0(ram_clk),
            .LOCKED(lock2));
   // synthesis attribute DLL_FREQUENCY_MODE of ext_dcm is "LOW"
```

```
   // synthesis attribute DUTY_CYCLE_CORRECTION of ext_dcm is "TRUE"
   // synthesis attribute STARTUP_WAIT of ext_dcm is "FALSE"
   // synthesis attribute DFS_FREQUENCY_MODE of ext_dcm is "LOW"
   // synthesis attribute CLK_FEEDBACK of ext_dcm  is "1X"
   // synthesis attribute CLKOUT_PHASE_SHIFT of ext_dcm is "NONE"
   // synthesis attribute PHASE_SHIFT of ext_dcm is 0

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


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

   assign locked = lock1 && lock2;

endmodule // ramclock
```

10.1.11 Visualization: rgb2hsv Module

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Kevin Zheng Class of 2012
//           Dept of Electrical Engineering &  Computer Science
//
// Create Date:    18:45:01 11/10/2010
// Design Name:
// Module Name:    rgb2hsv
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////////
module rgb2hsv(clock, reset, r, g, b, h, s, v);
        input wire clock;
        input wire reset;
        input wire [7:0] r;
        input wire [7:0] g;
        input wire [7:0] b;
        output reg [7:0] h;
        output reg [7:0] s;
```

```
    output reg [7:0] v;
    reg [7:0] my_r_delay1, my_g_delay1, my_b_delay1;
    reg [7:0] my_r_delay2, my_g_delay2, my_b_delay2;
    reg [7:0] my_r, my_g, my_b;
    reg [7:0] min, max, delta;
    reg [15:0] s_top;
    reg [15:0] s_bottom;
    reg [15:0] h_top;
    reg [15:0] h_bottom;
    wire [15:0] s_quotient;
    wire [15:0] s_remainder;
    wire s_rfd;
    wire [15:0] h_quotient;
    wire [15:0] h_remainder;
    wire h_rfd;
    reg [7:0] v_delay [19:0];
    reg [18:0] h_negative;
    reg [15:0] h_add [18:0];
    reg [4:0] i;
    // Clocks 4-18: perform all the divisions
    //the s_divider (16/16) has delay 18
    //the hue_div (16/16) has delay 18

    hsvdivider hue_divider1(
    .clk(clock),
    .dividend(s_top),
    .divisor(s_bottom),
    .quotient(s_quotient),
        // note: the "fractional" output was originally named "remainder" in this
    // file -- it seems coregen will name this output "fractional" even if
    // you didn't select the remainder type as fractional.
    .fractional(s_remainder),
    .rfd(s_rfd)
    );
    hsvdivider hue_divider2(
    .clk(clock),
    .dividend(h_top),
    .divisor(h_bottom),
    .quotient(h_quotient),
    .fractional(h_remainder),
    .rfd(h_rfd)
    );
    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
endmodule
```

10.1.12 Visualization: threshold_hsv Module

```
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Sabina chen
//
// Create Date: 21:26:30 12/07/2018
// Design Name:
// Module Name:     threshold_hsv
// Project Name:    AirPong
// Target Devices:
// Tool versions:
// Description:    Allows users to adjust the threshold of hsv values via user button press
//
//                  Inputs: 1) Current HSV threshold values for each object
//                          2) User button presses
//                  Output: Updated HSV threshold values for the selected object
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module threshold_hsv #(parameter INITIAL_THRESHOLD = 25, SPEED = 3)(
    input clk, reset,
    input [1:0] obj_sel,          // puck, paddle1, paddle2
    input [7:0] h_sel_puck, s_sel_puck, v_sel_puck,
    input [7:0] h_sel_paddle1, s_sel_paddle1, v_sel_paddle1,
    input [7:0] h_sel_paddle2, s_sel_paddle2, v_sel_paddle2,
    input btn_enter,              // calibrate on current center value
    input btn_min_decrease,       // button3:     min decrease
    input btn_min_increase,       // button2:     min increase
```

```
    input btn_max_decrease,     // button1:     max decrease
    input btn_max_increase,     // button0:     max increase
    input enable_threshold,     // switch[7]:   enable thresholding
    input [1:0] hsv_sel,        // switch[5:4]: 0:h1, 1:h2, 2:s, 3:v
    input [1:0] H, S, V, PUCK, PADDLE1, PADDLE2,
    output [7:0] h1_min_puck, h1_max_puck, h2_min_puck, h2_max_puck,
                s_min_puck, s_max_puck, v_min_puck, v_max_puck,
    output [7:0] h1_min_paddle1, h1_max_paddle1, h2_min_paddle1,
                h2_max_paddle1, s_min_paddle1, s_max_paddle1,
                v_min_paddle1, v_max_paddle1,
    output [7:0] h1_min_paddle2, h1_max_paddle2, h2_min_paddle2,
                h2_max_paddle2, s_min_paddle2, s_max_paddle2,
                v_min_paddle2, v_max_paddle2
    );

    // check for button pulse (ie. has the button changed from one state
    // to another -> indicates button press)
    wire btn_enter_pulsed, btn_min_decr_pulsed, btn_min_incr_pulsed,
         btn_max_decr_pulsed, btn_max_incr_pulsed;
    l2p_fsm pulse_enter(.clk(clk),.reset(reset),.level(btn_enter),
            .out(btn_enter_pulsed));
    l2p_fsm pulse_min_decr(.clk(clk),.reset(reset),
            .level(btn_min_decrease),.out(btn_min_decr_pulsed));
    l2p_fsm pulse_min_incr(.clk(clk),.reset(reset),
            .level(btn_min_increase),.out(btn_min_incr_pulsed));
    l2p_fsm pulse_max_decr(.clk(clk),.reset(reset),
            .level(btn_max_decrease),.out(btn_max_decr_pulsed));
    l2p_fsm pulse_max_incr(.clk(clk),.reset(reset),
            .level(btn_max_increase),.out(btn_max_incr_pulsed));

    //puck
    update_threshold #(.INITIAL_THRESHOLD(INITIAL_THRESHOLD), .SPEED(SPEED))
        threshold_puck(
            .clk(clk),.obj_sel(obj_sel),
            .update_value(obj_sel==PUCK),
            .enable_threshold(enable_threshold),
            .btn_enter_pulsed(btn_enter_pulsed),
            .btn_min_incr_pulsed(btn_min_incr_pulsed),
            .btn_min_decr_pulsed(btn_min_decr_pulsed),
            .btn_max_incr_pulsed(btn_max_incr_pulsed),
            .btn_max_decr_pulsed(btn_max_decr_pulsed),
            .H(H), .S(S), .V(V), .PUCK(PUCK),
            .PADDLE1(PADDLE1), .PADDLE2(PADDLE2),
            .h_sel(h_sel_puck), .s_sel(s_sel_puck),
            .v_sel(v_sel_puck),
            .h1_min(h1_min_puck), .h1_max(h1_max_puck),
            .h2_min(h2_min_puck), .h2_max(h2_max_puck),
            .s_min(s_min_puck), .s_max(s_max_puck),
            .v_min(v_min_puck), .v_max(v_max_puck)
        );

    //paddle1
    update_threshold #(.INITIAL_THRESHOLD(INITIAL_THRESHOLD), .SPEED(SPEED))
        threshold_paddle1(
```

```
            .clk(clk),.obj_sel(obj_sel),
            .update_value(obj_sel==PADDLE1),
            .enable_threshold(enable_threshold),
            .btn_enter_pulsed(btn_enter_pulsed),
            .btn_min_incr_pulsed(btn_min_incr_pulsed),
            .btn_min_decr_pulsed(btn_min_decr_pulsed),
            .btn_max_incr_pulsed(btn_max_incr_pulsed),
            .btn_max_decr_pulsed(btn_max_decr_pulsed),
            .H(H), .S(S), .V(V), .PUCK(PUCK),
            .PADDLE1(PADDLE1), .PADDLE2(PADDLE2),
            .h_sel(h_sel_paddle1), .s_sel(s_sel_paddle1),
            .v_sel(v_sel_paddle1),
            .h1_min(h1_min_paddle1), .h1_max(h1_max_paddle1),
            .h2_min(h2_min_paddle1), .h2_max(h2_max_paddle1),
            .s_min(s_min_paddle1), .s_max(s_max_paddle1),
            .v_min(v_min_paddle1), .v_max(v_max_paddle1)
        );

    //paddle2
    update_threshold #(.INITIAL_THRESHOLD(INITIAL_THRESHOLD), .SPEED(SPEED))
        threshold_paddle2(
            .clk(clk),.obj_sel(obj_sel),
            .update_value(obj_sel==PADDLE2),
            .enable_threshold(enable_threshold),
            .btn_enter_pulsed(btn_enter_pulsed),
            .btn_min_incr_pulsed(btn_min_incr_pulsed),
            .btn_min_decr_pulsed(btn_min_decr_pulsed),
            .btn_max_incr_pulsed(btn_max_incr_pulsed),
            .btn_max_decr_pulsed(btn_max_decr_pulsed),
            .H(H), .S(S), .V(V), .PUCK(PUCK),
            .PADDLE1(PADDLE1), .PADDLE2(PADDLE2),
            .h_sel(h_sel_paddle2), .s_sel(s_sel_paddle2),
            .v_sel(v_sel_paddle2),
            .h1_min(h1_min_paddle2), .h1_max(h1_max_paddle2),
            .h2_min(h2_min_paddle2), .h2_max(h2_max_paddle2),
            .s_min(s_min_paddle2), .s_max(s_max_paddle2),
            .v_min(v_min_paddle2), .v_max(v_max_paddle2)
        );

endmodule
```

10.1.13 Visualization: update_threshold Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:        Sabina Chen
//
// Create Date:    13:18:18 12/10/2018
// Design Name:
// Module Name:    update_threshold
// Project Name:   AirPong
// Target Devices:
// Tool versions:
```

```
// Description:     Update saved hsv theshold value sfor selected objecs based on user butt
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module update_threshold #(parameter INITIAL_THRESHOLD = 25, SPEED = 3) (
        input clk,
        input [1:0] obj_sel,
        input update_value, enable_threshold,
        input btn_enter_pulsed, btn_min_incr_pulsed, btn_min_decr_pulsed, btn_max_incr_puls
        input [1:0] H, S, V, PUCK, PADDLE1, PADDLE2,
        input [7:0] h_sel, v_sel, s_sel,
        output reg [7:0] h1_min=0, h1_max=0, h2_min=0, h2_max=0, s_min=0, s_max=0, v_min=0,
    );

    // initialize and update thresholds based on initial crosshair center locations, and th
    always @(posedge clk) begin
        // set initial thresholds based on hsv center values, minimum 0, max is 255
        if(enable_threshold && update_value && btn_enter_pulsed) begin
            h1_min <= (h_sel-INITIAL_THRESHOLD<0) ? 8'd0 : (h_sel-INITIAL_THRESHOLD);
            h1_max <= (h_sel+INITIAL_THRESHOLD>255) ? 8'd255 : (h_sel+INITIAL_THRESHOLD);
            h2_min <= (h_sel-INITIAL_THRESHOLD<0) ? 8'd0 : (h_sel-INITIAL_THRESHOLD);
            h2_max <= (h_sel+INITIAL_THRESHOLD>255) ? 8'd255 : (h_sel+INITIAL_THRESHOLD);
            s_min <= (s_sel-INITIAL_THRESHOLD<0) ? 8'd0 : (s_sel-INITIAL_THRESHOLD);
            s_max <= (s_sel+INITIAL_THRESHOLD>255) ? 8'd255 : (s_sel+INITIAL_THRESHOLD);
            v_min <= (v_sel-INITIAL_THRESHOLD<0) ? 8'd0 : (v_sel-INITIAL_THRESHOLD);
            v_max <= (v_sel+INITIAL_THRESHOLD>255) ? 8'd255 : (v_sel+INITIAL_THRESHOLD);
        end

        // manually update hsv threholds via use button presses
        // HUE
        if (enable_threshold && obj_sel==H && update_value && btn_min_incr_pulsed) begin
            h1_min <= (h1_min+SPEED>255) ? 8'd255 : (h1_min+SPEED);
            h2_min <= (h2_min+SPEED>255) ? 8'd255 : (h2_min+SPEED);
        end
        if (enable_threshold && obj_sel==H && update_value && btn_min_decr_pulsed) begin
            h1_min <= (h1_min-SPEED<0) ? 8'd0 : (h1_min-SPEED);
            h2_min <= (h2_min-SPEED<0) ? 8'd0 : (h2_min-SPEED);
        end
        if (enable_threshold && obj_sel==H && update_value && btn_max_incr_pulsed) begin
            h1_max <= (h1_max+SPEED>255) ? 8'd255 : (h1_max+SPEED);
            h2_max <= (h2_max+SPEED>255) ? 8'd255 : (h2_max+SPEED);
        end
        if (enable_threshold && obj_sel==H && update_value && btn_max_decr_pulsed) begin
            h1_max <= (h1_max-SPEED<0) ? 8'd0 : (h1_max-SPEED);
            h2_max <= (h2_max-SPEED<0) ? 8'd0 : (h2_max-SPEED);
        end
        // SATURATION
        if (enable_threshold && obj_sel==S && update_value && btn_min_incr_pulsed) begin
            s_min <= (s_min+SPEED>255) ? 8'd255 : (s_min+SPEED);
```

```
        end
        if (enable_threshold && obj_sel==S && update_value && btn_min_decr_pulsed) begin
            s_min <= (s_min-SPEED<0) ? 8'd0 : (s_min-SPEED);
        end
        if (enable_threshold && obj_sel==S && update_value && btn_max_incr_pulsed) begin
            s_max <= (s_max+SPEED>255) ? 8'd255 : (s_max+SPEED);
        end
        if (enable_threshold && obj_sel==S && update_value && btn_max_decr_pulsed) begin
            s_max <= (s_max-SPEED<0) ? 8'd0 : (s_max-SPEED);
        end
        // VALUE
        if (enable_threshold && obj_sel==V && update_value && btn_min_incr_pulsed) begin
            v_min <= (v_min+SPEED>255) ? 8'd255 : (v_min+SPEED);
        end
        if (enable_threshold && obj_sel==V && update_value && btn_min_decr_pulsed) begin
            v_min <= (v_min-SPEED<0) ? 8'd0 : (v_min-SPEED);
        end
        if (enable_threshold && obj_sel==V && update_value && btn_max_incr_pulsed) begin
            v_max <= (v_max+SPEED>255) ? 8'd255 : (v_max+SPEED);
        end
        if (enable_threshold && obj_sel==V && update_value && btn_max_decr_pulsed) begin
            v_max <= (v_max-SPEED<0) ? 8'd0 : (v_max-SPEED);
        end
    end

endmodule
```

10.1.14 Visualization: video_decoder Module

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

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

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

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

   // clk - line-locked clock (in this case, LLC1 which runs at 27Mhz)
```

```
// reset - system reset
// tv_in_ycrcb - 10-bit input from chip. should map to pins [19:10]
// ycrcb - 24 bit luminance and chrominance (8 bits each)
// f - field: 1 indicates an even field, 0 an odd field
// v - vertical sync: 1 means vertical sync
// h - horizontal sync: 1 means horizontal sync

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

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




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

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

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

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

```
 wire [4:0] state;
assign   state = current_state;

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

   end
 else
   begin
      // these states don't do much except allow us to know where we are in the stream.
      // whenever the synchronization code is seen, go back to the sync_state before
      // transitioning to the new state
      case (current_state)
        SYNC_1: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_2 : SYNC_1;
        SYNC_2: current_state <= (tv_in_ycrcb == 10'h000) ? SYNC_3 : SYNC_1;
        SYNC_3: current_state <= (tv_in_ycrcb == 10'h200) ? SAV_f1_cb0 :
                 (tv_in_ycrcb == 10'h274) ? EAV_f1 :
                 (tv_in_ycrcb == 10'h2ac) ? SAV_VBI_f1 :
                 (tv_in_ycrcb == 10'h2d8) ? EAV_VBI_f1 :
                 (tv_in_ycrcb == 10'h31c) ? SAV_f2_cb0 :
                 (tv_in_ycrcb == 10'h368) ? EAV_f2 :
                 (tv_in_ycrcb == 10'h3b0) ? SAV_VBI_f2 :
                 (tv_in_ycrcb == 10'h3c4) ? EAV_VBI_f2 : SYNC_1;

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

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

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

      endcase
   end
  end // always @ (posedge clk)

// implement our decoding mechanism

wire y_enable;
wire cr_enable;
```

```
   wire cb_enable;

   // if y is coming in, enable the register
   // likewise for cr and cb
   assign y_enable = (current_state == SAV_f1_y0) ||
                     (current_state == SAV_f1_y1) ||
                     (current_state == SAV_f2_y0) ||
                     (current_state == SAV_f2_y1);
   assign cr_enable = (current_state == SAV_f1_cr1) ||
                      (current_state == SAV_f2_cr1);
   assign cb_enable = (current_state == SAV_f1_cb0) ||
                      (current_state == SAV_f2_cb0);

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

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

   reg    f = 0;

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

endmodule



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

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

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

```
  // 2: Autodetect: NTSC or PAL (N), w/o pedestal
  // 3: Autodetect: NTSC or PAL (N), w/pedestal
  // 4: NTSC w/o pedestal
  // 5: NTSC w/pedestal
  // 6: NTSC 4.43 w/o pedestal
  // 7: NTSC 4.43 w/pedestal
  // 8: PAL BGHID w/o pedestal
  // 9: PAL N w/pedestal
  // A: PAL M w/o pedestal
  // B: PAL M w/pedestal
  // C: PAL combination N
  // D: PAL combination N w/pedestal
  // E-F: [Not valid]

`define ADV7185_REGISTER_0 {`INPUT_MODE, `INPUT_SELECT}

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

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

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

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

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

```
  // 5: Composite = -1.25dB, s-video = -3.0dB
  // 6: Composite = -1.75dB, s-video = -8.0dB
  // 7: Composite = -3.0dB,  s-video = -8.0dB
`define CORING                                          2'h0
  // 0: No coring
  // 1: Truncate if Y < black+8
  // 2: Truncate if Y < black+16
  // 3: Truncate if Y < black+32

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

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

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

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

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

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

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

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

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


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

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

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

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

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

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

`define INPUT_CONTRAST_ADJUST                          8'h80

`define ADV7185_REGISTER_8 {`INPUT_CONTRAST_ADJUST}

////////////////////////////////////////////////////////////////////////////////
// Register 9
////////////////////////////////////////////////////////////////////////////////

`define INPUT_SATURATION_ADJUST                        8'h8C

`define ADV7185_REGISTER_9 {`INPUT_SATURATION_ADJUST}
```

```
////////////////////////////////////////////////////////////////////////////////
// Register A
////////////////////////////////////////////////////////////////////////////////

`define INPUT_BRIGHTNESS_ADJUST                        8'h00

`define ADV7185_REGISTER_A {`INPUT_BRIGHTNESS_ADJUST}

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

`define INPUT_HUE_ADJUST                               8'h00

`define ADV7185_REGISTER_B {`INPUT_HUE_ADJUST}

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

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

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

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

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

`define ADV7185_REGISTER_D {`DEFAULT_CB_VALUE, `DEFAULT_CR_VALUE}

////////////////////////////////////////////////////////////////////////////////
// Register E
////////////////////////////////////////////////////////////////////////////////

`define TEMPORAL_DECIMATION_ENABLE               1'b0
  // 0: Disable
  // 1: Enable
`define TEMPORAL_DECIMATION_CONTROL              2'h0
  // 0: Supress frames, start with even field
  // 1: Supress frames, start with odd field
```

```
  // 2: Supress even fields only
  // 3: Supress odd fields only
`define TEMPORAL_DECIMATION_RATE                4'h0
  // 0-F: Number of fields/frames to skip

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

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

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

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

///////////////////////////////////////////////////////////////////////////////
// Register 33
///////////////////////////////////////////////////////////////////////////////

`define PEAK_WHITE_UPDATE                       1'b1
  // 0: Update gain once per line
  // 1: Update gain once per field
`define AVERAGE_BIRIGHTNESS_LINES               1'b1
  // 0: Use lines 33 to 310
  // 1: Use lines 33 to 270
`define MAXIMUM_IRE                             3'h0
  // 0: PAL: 133, NTSC: 122
  // 1: PAL: 125, NTSC: 115
  // 2: PAL: 120, NTSC: 110
```

```
  // 3: PAL: 115, NTSC: 105
  // 4: PAL: 110, NTSC: 100
  // 5: PAL: 105, NTSC: 100
  // 6-7: PAL: 100, NTSC: 100
`define COLOR_KILL                              1'b1
  // 0: Disable color kill
  // 1: Enable color kill

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

`define ADV7185_REGISTER_10 8'h00
`define ADV7185_REGISTER_11 8'h00
`define ADV7185_REGISTER_12 8'h00
`define ADV7185_REGISTER_13 8'h45
`define ADV7185_REGISTER_14 8'h18
`define ADV7185_REGISTER_15 8'h60
`define ADV7185_REGISTER_16 8'h00
`define ADV7185_REGISTER_17 8'h01
`define ADV7185_REGISTER_18 8'h00
`define ADV7185_REGISTER_19 8'h10
`define ADV7185_REGISTER_1A 8'h10
`define ADV7185_REGISTER_1B 8'hF0
`define ADV7185_REGISTER_1C 8'h16
`define ADV7185_REGISTER_1D 8'h01
`define ADV7185_REGISTER_1E 8'h00
`define ADV7185_REGISTER_1F 8'h3D
`define ADV7185_REGISTER_20 8'hD0
`define ADV7185_REGISTER_21 8'h09
`define ADV7185_REGISTER_22 8'h8C
`define ADV7185_REGISTER_23 8'hE2
`define ADV7185_REGISTER_24 8'h1F
`define ADV7185_REGISTER_25 8'h07
`define ADV7185_REGISTER_26 8'hC2
`define ADV7185_REGISTER_27 8'h58
`define ADV7185_REGISTER_28 8'h3C
`define ADV7185_REGISTER_29 8'h00
`define ADV7185_REGISTER_2A 8'h00
`define ADV7185_REGISTER_2B 8'hA0
`define ADV7185_REGISTER_2C 8'hCE
`define ADV7185_REGISTER_2D 8'hF0
`define ADV7185_REGISTER_2E 8'h00
`define ADV7185_REGISTER_2F 8'hF0
`define ADV7185_REGISTER_30 8'h00
`define ADV7185_REGISTER_31 8'h70
`define ADV7185_REGISTER_32 8'h00
`define ADV7185_REGISTER_34 8'h0F
`define ADV7185_REGISTER_35 8'h01
`define ADV7185_REGISTER_36 8'h00
`define ADV7185_REGISTER_37 8'h00
`define ADV7185_REGISTER_38 8'h00
`define ADV7185_REGISTER_39 8'h00
`define ADV7185_REGISTER_3A 8'h00
`define ADV7185_REGISTER_3B 8'h00
```

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

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


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

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

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

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

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

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

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

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

assign reset_slow = reset_count != 0;

//
// I2C driver
//

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

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

//
// State machine
//

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

always @(posedge clock_slow)
   if (reset_slow)
 begin
    state <= 0;
    load <= 0;
    tv_in_reset_b <= 0;
    old_source <= 0;
 end
   else
 case (state)
   8'h00:
     begin
        // Assert reset
        load <= 1'b0;
        tv_in_reset_b <= 1'b0;
        if (!ack)
      state <= state+1;
```

```
      end
    8'h01:
      state <= state+1;
    8'h02:
      begin
         // Release reset
         tv_in_reset_b <= 1'b1;
         state <= state+1;
              end
    8'h03:
      begin
         // Send ADV7185 address
         data <= 8'h8A;
         load <= 1'b1;
         if (ack)
       state <= state+1;
      end
    8'h04:
      begin
         // Send subaddress of first register
         data <= 8'h00;
         if (ack)
       state <= state+1;
      end
    8'h05:
      begin
         // Write to register 0
         data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
         if (ack)
       state <= state+1;
      end
    8'h06:
      begin
         // Write to register 1
         data <= `ADV7185_REGISTER_1;
         if (ack)
       state <= state+1;
      end
    8'h07:
      begin
         // Write to register 2
         data <= `ADV7185_REGISTER_2;
         if (ack)
       state <= state+1;
      end
    8'h08:
      begin
         // Write to register 3
         data <= `ADV7185_REGISTER_3;
         if (ack)
       state <= state+1;
      end
    8'h09:
      begin
```

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

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

```
8'h19:
  begin
     load <= 1'b0;
     if (idle)
   state <= state+1;
  end

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

```
          if (ack) state <= state+1;
       end
       8'h22: begin
          // Send subaddress of register 0
          data <= 8'h00;
          if (ack) state <= state+1;
       end
       8'h23: begin
          // Write to register 0
          data <= `ADV7185_REGISTER_0 | {5'h00, {3{source}}};
          if (ack) state <= state+1;
       end
       8'h24: begin
          // Wait for I2C transmitter to finish
          load <= 1'b0;
          if (idle) state <= 8'h20;
       end
        endcase

endmodule

// i2c module for use with the ADV7185

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

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

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

   reg [7:0] state;

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

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

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

```
            state <= state+1;
        end
      8'h0A:
        begin
            scl <= 1'b0;
            state <= state+1;
        end
      8'h0B:
        begin
            sdai <= ldata[5];
            state <= state+1;
        end
      8'h0C:
        begin
            scl <= 1'b1;
            state <= state+1;
        end
      8'h0D:
        begin
            state <= state+1;
        end
      8'h0E:
        begin
            scl <= 1'b0;
            state <= state+1;
        end
      8'h0F:
        begin
            sdai <= ldata[4];
            state <= state+1;
        end
      8'h10:
        begin
            scl <= 1'b1;
            state <= state+1;
        end
      8'h11:
        begin
            state <= state+1;
        end
      8'h12:
        begin
            scl <= 1'b0;
            state <= state+1;
        end
      8'h13:
        begin
            sdai <= ldata[3];
            state <= state+1;
        end
      8'h14:
        begin
            scl <= 1'b1;
            state <= state+1;
```

```
      end
    8'h15:
      begin
         state <= state+1;
      end
    8'h16:
      begin
         scl <= 1'b0;
         state <= state+1;
      end
    8'h17:
      begin
         sdai <= ldata[2];
         state <= state+1;
      end
    8'h18:
      begin
         scl <= 1'b1;
         state <= state+1;
      end
    8'h19:
      begin
         state <= state+1;
      end
    8'h1A:
      begin
         scl <= 1'b0;
         state <= state+1;
      end
    8'h1B:
      begin
         sdai <= ldata[1];
         state <= state+1;
      end
    8'h1C:
      begin
         scl <= 1'b1;
         state <= state+1;
      end
    8'h1D:
      begin
         state <= state+1;
      end
    8'h1E:
      begin
         scl <= 1'b0;
         state <= state+1;
      end
    8'h1F:
      begin
         sdai <= ldata[0];
         state <= state+1;
      end
    8'h20:
```

```
    begin
       scl <= 1'b1;
       state <= state+1;
    end
  8'h21:
    begin
       state <= state+1;
    end
  8'h22:
    begin
       scl <= 1'b0;
       state <= state+1;
    end
  8'h23: // Acknowledge bit
    begin
       state <= state+1;
    end
  8'h24:
    begin
       scl <= 1'b1;
       state <= state+1;
    end
  8'h25:
    begin
       state <= state+1;
    end
  8'h26:
    begin
       scl <= 1'b0;
       if (load)
     begin
        ldata <= data;
        ack <= 1'b1;
        state <= 3;
     end
       else
     state <= state+1;
    end
  8'h27:
    begin
       sdai <= 1'b0;
       state <= state+1;
    end
  8'h28:
    begin
       scl <= 1'b1;
       state <= state+1;
    end
  8'h29:
    begin
       sdai <= 1'b1;
       state <= 0;
    end
    endcase
```

```
endmodule
```

10.1.15 Visualization: vram_display Module

```
///////////////////////////////////////////////////////////////////////////////
// generate display pixels from reading the ZBT ram
// note that the ZBT ram has 2 cycles of read (and write) latency
//
// We take care of that by latching the data at an appropriate time.
//
// Note that the ZBT stores 36 bits per word; we use only 32 bits here,
// decoded into four bytes of pixel data.
//
// Bug due to memory management will be fixed. The bug happens because
// memory is called based on current hcount & vcount, which will actually
// shows up 2 cycle in the future. Not to mention that these incoming data
// are latched for 2 cycles before they are used. Also remember that the
// ntsc2zbt's addressing protocol has been fixed.

// The original bug:
// -. At (hcount, vcount) = (100, 201) data at memory address(0,100,49)
//    arrives at vram_read_data, latch it to vr_data_latched.
// -. At (hcount, vcount) = (100, 203) data at memory address(0,100,49)
//    is latched to last_vr_data to be used for display.
// -. Remember that memory address(0,100,49) contains camera data
//    pixel(100,192) - pixel(100,195).
// -. At (hcount, vcount) = (100, 204) camera pixel data(100,192) is shown.
// -. At (hcount, vcount) = (100, 205) camera pixel data(100,193) is shown.
// -. At (hcount, vcount) = (100, 206) camera pixel data(100,194) is shown.
// -. At (hcount, vcount) = (100, 207) camera pixel data(100,195) is shown.
//
// Unfortunately this means that at (hcount == 0) to (hcount == 11) data from
// the right side of the camera is shown instead (including possible sync signals).

// To fix this, two corrections has been made:
// -. Fix addressing protocol in ntsc_to_zbt module.
// -. Forecast hcount & vcount 8 clock cycles ahead and use that
//    instead to call data from ZBT.


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

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

   //forecast hcount & vcount 8 clock cycles ahead to get data from ZBT
   wire [10:0] hcount_f = (hcount >= 1048) ? (hcount - 1048) : (hcount + 8);
   wire [9:0] vcount_f = (hcount >= 1048) ? ((vcount == 805) ? 0 : vcount + 1) : vcount;
```

```
   //wire [18:0]     vram_addr = {1'b0, vcount_f, hcount_f[9:2]};
    wire [18:0]      vram_addr = {vcount_f, hcount_f[9:1]}; // SABINA

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

   always @(posedge clk)
     //last_vr_data <= (hc4==2'd3) ? vr_data_latched : last_vr_data;
      last_vr_data <= (hc4==1'b1) ? vr_data_latched : last_vr_data; // SABINA

   always @(posedge clk)
     //vr_data_latched <= (hc4==2'd1) ? vram_read_data : vr_data_latched;
      vr_data_latched <= (hc4==1'b0) ? vram_read_data : vr_data_latched; // SABINA

   always @(*)      // each 36-bit word from RAM is decoded to 4 bytes
     case (hc4)
       //2'd3: vr_pixel = last_vr_data[7:0];
       //2'd2: vr_pixel = last_vr_data[7+8:0+8];
       //2'd1: vr_pixel = last_vr_data[7+16:0+16];
       //2'd0: vr_pixel = last_vr_data[7+24:0+24];
         1: vr_pixel = last_vr_data[17:0]; // SABINA
         2: vr_pixel = last_vr_data[35:18]; // SABINA
     endcase


endmodule // vram_display
```

10.1.16 Visualization: xvga Module

```
////////////////////////////////////////////////////////////////////////////////
// xvga: Generate XVGA display signals (1024 x 768 @ 60Hz)

module xvga(vclock,hcount,vcount,hsync,vsync,blank);
   input vclock;
   output [10:0] hcount;
   output [9:0] vcount;
   output   vsync;
   output   hsync;
   output   blank;

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

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

```
   assign    hreset = (hcount == 1343);

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

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

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

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

/*
///////////////////////////////////////////////////////////////////////////////
// xvga: Generate XVGA display signals (800 x 600 @ 60Hz)

module xvga(vclock,hcount,vcount,hsync,vsync,blank);
   input vclock;
   output [10:0] hcount;
   output [9:0] vcount;
   output   vsync;
   output   hsync;
   output   blank;

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

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

   // vertical: 628 lines total
   // display 600 lines
   wire      vsyncon,vsyncoff,vreset,vblankon;
```

```
   assign    vblankon = hreset & (vcount == 599);
   assign    vsyncon = hreset & (vcount == 600);
   assign    vsyncoff = hreset & (vcount == 604);
   assign    vreset = hreset & (vcount == 627);

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

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

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

10.1.17 Visualization: ycrcb2rgb Module

```
/*****************************************************************************
 **
 ** Module: ycrcb2rgb
 **
 ** Generic Equations:
 *****************************************************************************/

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

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

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

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

//registering constants
always @ (posedge clk)
begin
 const1 = 10'b 0100101010; //1.164 = 01.00101010
 const2 = 10'b 0110011000; //1.596 = 01.10011000
 const3 = 10'b 0011010000; //0.813 = 00.11010000
 const4 = 10'b 0001100100; //0.392 = 00.01100100
 const5 = 10'b 1000000100; //2.017 = 10.00000100
end

always @ (posedge clk or posedge rst)
```

```
   if (rst)
      begin
      Y_reg <= 0; Cr_reg <= 0; Cb_reg <= 0;
      end
   else
      begin
      Y_reg <= Y; Cr_reg <= Cr; Cb_reg <= Cb;
      end

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

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

/* limit output to 0 - 4095, <0 equals o and >4095 equals 4095 */
assign R = (R_int[20]) ? 0 : (R_int[19:18] == 2'b0) ? R_int[17:10] : 8'b11111111;
assign G = (G_int[20]) ? 0 : (G_int[19:18] == 2'b0) ? G_int[17:10] : 8'b11111111;
assign B = (B_int[20]) ? 0 : (B_int[19:18] == 2'b0) ? B_int[17:10] : 8'b11111111;

endmodule
```

10.1.18 Visualization: zbt_6111 Module

```
//
// File:   zbt_6111.v
// Date:   27-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Simple ZBT driver for the MIT 6.111 labkit, which does not hide the
// pipeline delays of the ZBT from the user.  The ZBT memories have
// two cycle latencies on read and write, and also need extra-long data hold
// times around the clock positive edge to work reliably.
//
```

```
///////////////////////////////////////////////////////////////////////////////
// Ike's simple ZBT RAM driver for the MIT 6.111 labkit
//
// Data for writes can be presented and clocked in immediately; the actual
// writing to RAM will happen two cycles later.
//
// Read requests are processed immediately, but the read data is not available
// until two cycles after the intial request.
//
// A clock enable signal is provided; it enables the RAM clock when high.

module zbt_6111(clk, cen, we, addr, write_data, read_data,
          ram_clk, ram_we_b, ram_address, ram_data, ram_cen_b);

   input clk;           // system clock
   input cen;           // clock enable for gating ZBT cycles
   input we;            // write enable (active HIGH)
   input [18:0] addr;       // memory address
   input [35:0] write_data; // data to write
   output [35:0] read_data; // data read from memory
   output    ram_clk;   // physical line to ram clock
   output    ram_we_b;  // physical line to ram we_b
   output [18:0] ram_address;   // physical line to ram address
   inout [35:0]  ram_data;  // physical line to ram data
   output    ram_cen_b; // physical line to ram clock enable

   // clock enable (should be synchronous and one cycle high at a time)
   wire      ram_cen_b = ~cen;

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

   reg [1:0]   we_delay;

   always @(posedge clk)
     we_delay <= cen ? {we_delay[0],we} : we_delay;

   // create two-stage pipeline for write data

   reg [35:0]  write_data_old1;
   reg [35:0]  write_data_old2;
   always @(posedge clk)
     if (cen)
       {write_data_old2, write_data_old1} <= {write_data_old1, write_data};

   // wire to ZBT RAM signals

   assign      ram_we_b = ~we;
   assign      ram_clk = 1'b0;  // gph 2011-Nov-10
                                  // set to zero as place holder

//   assign      ram_clk = ~clk;     // RAM is not happy with our data hold
```

```
                                    // times if its clk edges equal FPGA's
                                    // so we clock it on the falling edges
                                    // and thus let data stabilize longer
   assign       ram_address = addr;

   assign       ram_data = we_delay[1] ? write_data_old2 : {36{1'bZ}};
   assign       read_data = ram_data;

endmodule // zbt_6111
```

10.1.19 Visualization: zbt_6111_sample Module

```
`default_nettype none
//
// File:   zbt_6111_sample.v
// Date:   26-Nov-05
// Author: I. Chuang <ichuang@mit.edu>
//
// Sample code for the MIT 6.111 labkit demonstrating use of the ZBT
// memories for video display.  Video input from the NTSC digitizer is
// displayed within an XGA 1024x768 window.  One ZBT memory (ram0) is used
// as the video frame buffer, with 8 bits used per pixel (black & white).
//
// Since the ZBT is read once for every four pixels, this frees up time for
// data to be stored to the ZBT during other pixel times.  The NTSC decoder
// runs at 27 MHz, whereas the XGA runs at 65 MHz, so we synchronize
// signals between the two (see ntsc2zbt.v) and let the NTSC data be
// stored to ZBT memory whenever it is available, during cycles when
// pixel reads are not being performed.
//
// We use a very simple ZBT interface, which does not involve any clock
// generation or hiding of the pipelining.  See zbt_6111.v for more info.
//
// Bug fix: Jonathan P. Mailoa <jpmailoa@mit.edu>
// Date   : 11-May-09
//
// Use ramclock module to deskew clocks;  GPH
// To change display from 1024*787 to 800*600, use clock_40mhz and change
// accordingly. Verilog ntsc2zbt.v will also need changes to change resolution.
//
// Date   : 10-Nov-11

///////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
///////////////////////////////////////////////////////////////////////////////
//
```

```
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
///////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2011-Nov-10: Changed resolution to 1024 * 768.
//                   Added back ramclok to deskew RAM clock
//
// 2009-May-11: Fixed memory management bug by 8 clock cycle forecast.
//              Changed resolution to  800 * 600.
//              Reduced clock speed to 40MHz.
//              Disconnected zbt_6111's ram_clk signal.
//              Added ramclock to control RAM.
//              Added notes about ram1 default values.
//              Commented out clock_feedback_out assignment.
//              Removed delayN modules because ZBT's latency has no more effect.
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
//              "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//              actually populated on the boards. (The boards support up to
//              256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//              value. (Previous versions of this file declared this port to
//              be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
```

```
//              actually populated on the boards. (The boards support up to
//              72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
///////////////////////////////////////////////////////////////////////////////

module zbt_6111_sample(
             beep, audio_reset_b,
            ac97_sdata_out, ac97_sdata_in, ac97_synch,
           ac97_bit_clock,

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

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

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

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

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

           clock_feedback_out, clock_feedback_in,

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

           rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

           //mouse_clock, mouse_data, keyboard_clock, keyboard_data,

           clock_27mhz,
           //clock1, clock2,

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

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

           switch,

           led,

           //user1, user2, user3, user4,
```

```
          user3,

         //daughtercard,

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

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

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

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

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

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

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

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

input  clock_feedback_in;
output clock_feedback_out;

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

output rs232_txd, rs232_rts;
input  rs232_rxd, rs232_cts;

//input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;
```

```
input  clock_27mhz;
 //clock1, clock2;

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

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

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

//inout [43:0] daughtercard;

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

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

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

// Audio Input and Output
assign beep= 1'b0;
assign audio_reset_b = 1'b0;
assign ac97_synch = 1'b0;
assign ac97_sdata_out = 1'b0;
// ac97_sdata_in is an input

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

// Video Input
//assign tv_in_i2c_clock = 1'b0;
```

```
assign tv_in_fifo_read = 1'b1;
assign tv_in_fifo_clock = 1'b0;
assign tv_in_iso = 1'b1;
//assign tv_in_reset_b = 1'b0;
assign tv_in_clock = clock_27mhz;//1'b0;
//assign tv_in_i2c_data = 1'bZ;

 // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
// tv_in_aef, tv_in_hff, and tv_in_aff are inputs

// SRAMs

 /* change lines below to enable ZBT RAM bank0 */

 /*
assign ram0_data = 36'hZ;
assign ram0_address = 19'h0;
assign ram0_clk = 1'b0;
assign ram0_we_b = 1'b1;
assign ram0_cen_b = 1'b0;    // clock enable
 */

 /* enable RAM pins */

assign ram0_ce_b = 1'b0;
assign ram0_oe_b = 1'b0;
assign ram0_adv_ld = 1'b0;
assign ram0_bwe_b = 4'h0;

 /**********/

//assign ram1_data = 36'hZ;
//assign ram1_address = 19'h0;
//assign ram1_adv_ld = 1'b0;
//assign ram1_clk = 1'b0;

//These values has to be set to 0 like ram0 if ram1 is used.
//assign ram1_cen_b = 1'b1;
//assign ram1_ce_b = 1'b1;
//assign ram1_oe_b = 1'b1;
//assign ram1_we_b = 1'b1;
//assign ram1_bwe_b = 4'hF;

// clock_feedback_out will be assigned by ramclock
// assign clock_feedback_out = 1'b0;  //2011-Nov-10
// clock_feedback_in is an input

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

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

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

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

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

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

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

// Daughtercard Connectors
//assign daughtercard = 44'hZ;

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

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

```
// -----------------------------------------------------------------------------
// 65 MHz clock creation
// -----------------------------------------------------------------------------

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

// -----------------------------------------------------------------------------
// Ram clock creation
// -----------------------------------------------------------------------------

wire clk; // Primary logic clock
 wire locked;
 //assign clock_feedback_out = 0; // gph 2011-Nov-10

ramclock rc(.ref_clock(clock_65mhz),
        .fpga_clock(clk),
        .ram0_clock(ram0_clk),
        .ram1_clock(/*ram1_clk*/),   //uncomment if ram1 is used
        .clock_feedback_in(clock_feedback_in),
        .clock_feedback_out(clock_feedback_out),
        .locked(locked));

// -----------------------------------------------------------------------------
// Power-on reset generation
// -----------------------------------------------------------------------------

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

 // -----------------------------------------------------------------------------
// Set Parameters
// -----------------------------------------------------------------------------

/////////////////
// VID_COORD
/////////////////

// camera frame image 725 x 505
parameter  VID_BORDER_UP =    10'd126;       // initial numbers
parameter  VID_BORDER_DOWN =  10'd542;
parameter  VID_BORDER_LEFT =  11'd82;
parameter  VID_BORDER_RIGHT = 11'd730;
```

```verilog
// define selectable objects
parameter PUCK = 0;
parameter PADDLE1 = 1;
parameter PADDLE2 = 2;

// define selectables hsv
parameter H = 0;
parameter S = 1;
parameter V = 2;

// define switches and buttons
wire CROSSHAIR_CHROMA_SEL, CENTER_SEL, DISP_XY_HSV, ENABLE_THRESH;
wire [1:0] OBJ_SEL, HSV_SEL;

assign CROSSHAIR_CHROMA_SEL   = switch[2];
assign CENTER_SEL             = switch[3];
assign DISP_XY_HSV            = switch[0];
assign OBJ_SEL                = switch[5:4];
assign HSV_SEL                = switch[7:6];
assign ENABLE_THRESH          = ~switch[1];

////////////////
// PONG
////////////////

wire [1:0] DISPLAY_KILL;
wire [3:0] PSPEED;

assign DISPLAY_KILL = switch[1:0];
assign PSPEED = switch[7:4];

// ---------------------------------------------------------------------------
// Debouncing
// ---------------------------------------------------------------------------

// ENTER button is user reset
wire reset,user_reset;
assign reset = user_reset | power_on_reset;
debounce db1(.reset(power_on_reset), .clk(clk), .noisy(~button_enter), .clean(user_reset

wire btn_up_debounced,btn_down_debounced, btn_left_debounced, btn_right_debounced;
debounce db3(.reset(reset),.clk(clk),.noisy(~button_up),.clean(btn_up_debounced));
debounce db4(.reset(reset),.clk(clk),.noisy(~button_down),.clean(btn_down_debounced));
debounce db5(.reset(reset),.clk(clk),.noisy(~button_left),.clean(btn_left_debounced));
debounce db6(.reset(reset),.clk(clk),.noisy(~button_right),.clean(btn_right_debounced))

wire btn_enter_debounced,btn_0_debounced,btn_1_debounced,btn_2_debounced,btn_3_debounced
debounce db7(.reset(reset),.clk(clk),.noisy(~button_enter),.clean(btn_enter_debounced))
debounce db8(.reset(reset),.clk(clk),.noisy(~button0),.clean(btn_0_debounced));
debounce db9(.reset(reset),.clk(clk),.noisy(~button1),.clean(btn_1_debounced));
debounce db10(.reset(reset),.clk(clk),.noisy(~button2),.clean(btn_2_debounced));
debounce db11(.reset(reset),.clk(clk),.noisy(~button3),.clean(btn_3_debounced));
```

```
// ---------------------------------------------------------------------------
// Hex dot matrix displays
// ---------------------------------------------------------------------------

// display module for debugging

reg [63:0] dispdata;

display_16hex hexdisp1(reset, clock_27mhz, dispdata,
           disp_blank, disp_clock, disp_rs, disp_ce_b,
           disp_reset_b, disp_data_out);

// generate basic XVGA video signals
wire [10:0] hcount;
wire [9:0]  vcount;
wire hsync,vsync,blank;
xvga xvga1(.vclock(clk),.hcount(hcount),.vcount(vcount),
            .hsync(hsync),.vsync(vsync),.blank(blank));

// ---------------------------------------------------------------------------
// ZBT Wiring - wire labkit directly to zbt
// ---------------------------------------------------------------------------

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

wire ram0_clk_not_used;
zbt_6111 zbt1(clk, 1'b1, vram_we, vram_addr,
        vram_write_data, vram_read_data,
        ram0_clk_not_used,   //to get good timing, don't connect ram_clk to zbt_6111
        ram0_we_b, ram0_address, ram0_data, ram0_cen_b);

// ---------------------------------------------------------------------------
// vram_display - read from ZBT memory
// ---------------------------------------------------------------------------

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

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

// ---------------------------------------------------------------------------
// adv7185 - take raw ntsc output and converts to usable form
// ---------------------------------------------------------------------------

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

```
            .tv_in_i2c_data(tv_in_i2c_data));

// ------------------------------------------------------------------------------
// ntsc_decode - parse control signals
// ------------------------------------------------------------------------------

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

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

// ------------------------------------------------------------------------------
// Ycrcb2rgb - takes updated control signals and changes to rgb format
// ------------------------------------------------------------------------------

 wire [7:0] red, green, blue;
 wire [17:0] rgb;

 YCrCb2RGB color_conv( red, green, blue, tv_in_line_clock1, reset,
                           ycrcb[29:20], ycrcb[19:10],ycrcb[9:0] );

 assign rgb = {red[7:2],green[7:2],blue[7:2]};

 // ------------------------------------------------------------------------------
// ntsc_to_zbt - saves rgb data to zbt, and returns ntsc address + data
// ------------------------------------------------------------------------------

// code to write NTSC data to video memory

wire [18:0] ntsc_addr;
wire [35:0] ntsc_data;
wire        ntsc_we;
 ntsc_to_zbt n2z (clk, tv_in_line_clock1, fvh, dv, rgb,
         ntsc_addr, ntsc_data, ntsc_we, 1'b0);

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

wire [18:0]  vram_addr2 = count[0+18:0];
 wire [35:0]     vpat = ( 1'b0 ? {4{count[3+3:3],4'b0}} :
               {4{count[3+4:4],4'b0}} );
               // Sabina - removed to use switch[6]

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

//wire   sw_ntsc = ~switch[7]; // Sabina - removed to use switch[7]
wire sw_ntsc = 1'b1;
wire     my_we = sw_ntsc ? (hcount[0]==1'b1) : blank;
               // Sabina - removed to use switch[8]
```

```
wire [18:0]  write_addr = sw_ntsc ? ntsc_addr : vram_addr2;
wire [35:0]  write_data = sw_ntsc ? ntsc_data : vpat;

assign   vram_addr = my_we ? write_addr : vram_addr1;
assign   vram_we = my_we;
assign   vram_write_data = write_data;

// ---------------------------------------------------------------------------
// rgb2hsv - rgb->hsv
// ---------------------------------------------------------------------------

// clk (input) and vr_pixel (output) data input received from vram_display
 wire [7:0] vr_red, vr_green, vr_blue;
 assign vr_red = {vr_pixel[17:12], 2'b0};
 assign vr_green = {vr_pixel[11:6], 2'b0};
 assign vr_blue = {vr_pixel[5:0], 2'b0};

wire [23:0] hsv;
rgb2hsv rgb_hsv_convert(.clock(clk), .reset(reset), .r(vr_red),
                        .g(vr_green), .b(vr_blue),
                                 .h(hsv[23:16]), .s(hsv[15:8]), .v(hsv[7:0]));
                        //outputs hsv values

 // ---------------------------------------------------------------------------
// Delay - add hcount vcount delays due to rgb2hsv divider delay
// ---------------------------------------------------------------------------

 wire [10:0] hcount_delayed;
 wire [9:0] vcount_delayed;
 delayN #(22,11) delay_h(clk, hcount, hcount_delayed);
 delayN #(22,10) delay_v(clk, vcount, vcount_delayed);

// ---------------------------------------------------------------------------
// video_border_pixel - create moving border (takes in vr_pixel,
//                      outputs video_border_pixel)
// ---------------------------------------------------------------------------

wire in_video_boundary;
assign in_video_boundary = (hcount >= VID_BORDER_LEFT &&
                              hcount < VID_BORDER_RIGHT) &&
                                         (vcount >= VID_BORDER_UP &&
                              vcount < VID_BORDER_DOWN);

 wire [23:0] video_border_pixel;
 assign video_border_pixel = (in_video_boundary)
                        ? {vr_red, vr_green, vr_blue} : 24'd0;

 wire [23:0] video_border_pixel_delayed;
 delayN #(22,24) delay_border(clk, video_border_pixel,
                              video_border_pixel_delayed);

 // ---------------------------------------------------------------------------
// move & display crosshair - debug/visualization module
//              (takes in video_border_pixel, outputs crosshair_pixel)
```

```
// ---------------------------------------------------------------------------

// debugging module - can take in any xy-coord and move the center
// of the crosshair to that location

 // set puck, paddle1, or paddle2
 wire [10:0] x_new_puck, x_new_paddle1, x_new_paddle2;
wire [9:0] y_new_puck, y_new_paddle1, y_new_paddle2;

 wire enable_puck, enable_paddle1, enable_paddle2;
 assign enable_puck = (OBJ_SEL == PUCK);
 assign enable_paddle1 = (OBJ_SEL == PADDLE1);
 assign enable_paddle2 = (OBJ_SEL == PADDLE2);

 //puck
 move_crosshair move_crosshair_puck(
                         .clk(vsync),
                         .enable(enable_puck),
                         .btn_up_debounced(btn_up_debounced),
                         .btn_down_debounced(btn_down_debounced),
                         .btn_left_debounced(btn_left_debounced),
                         .btn_right_debounced(btn_right_debounced),
                         .x_new(x_new_puck),
                         .y_new(y_new_puck) );
 //paddle1
 move_crosshair move_crosshair_paddle1(
                         .clk(vsync),
                         .enable(enable_paddle1),
                         .btn_up_debounced(btn_up_debounced),
                         .btn_down_debounced(btn_down_debounced),
                         .btn_left_debounced(btn_left_debounced),
                         .btn_right_debounced(btn_right_debounced),
                         .x_new(x_new_paddle1),
                         .y_new(y_new_paddle1) );
 //paddle2
 move_crosshair move_crosshair_paddle2(
                         .clk(vsync),
                         .enable(enable_paddle2),
                         .btn_up_debounced(btn_up_debounced),
                         .btn_down_debounced(btn_down_debounced),
                         .btn_left_debounced(btn_left_debounced),
                         .btn_right_debounced(btn_right_debounced),
                         .x_new(x_new_paddle2),
                         .y_new(y_new_paddle2) );

 wire [23:0] crosshair_pixel;
 wire [10:0] x_center_puck;
wire [9:0] y_center_puck;
 wire [10:0] x_center_paddle1;
wire [9:0] y_center_paddle1;
 wire [10:0] x_center_paddle2;
wire [9:0] y_center_paddle2;

 //crosshair - puck:magenta, paddle1:green, paddle2:blue
```

```
 crosshair display_crosshair(
     .clk(clk), .hcount(hcount), .vcount(vcount), .center_sel(CENTER_SEL),

     .x_new_puck(x_new_puck), .y_new_puck(y_new_puck),
     .x_new_paddle1(x_new_paddle1), .y_new_paddle1(y_new_paddle1),
     .x_new_paddle2(x_new_paddle2), .y_new_paddle2(y_new_paddle2),

     .x_center_puck(x_center_puck), .y_center_puck(y_center_puck),
     .x_center_paddle1(x_center_paddle1), .y_center_paddle1(y_center_paddle1),
     .x_center_paddle2(x_center_paddle2), .y_center_paddle2(y_center_paddle2),

     .pixel(video_border_pixel), .crosshair_pixel(crosshair_pixel)
 );

 // -------------------------------------------------------------------------
// display_hsv_value - takes x_new/y_new coords provided by crosshair,
//                        and uses video_border_pixel to detect hsv at
//                        selected coord.
 //                        displays hsv value on hex display. used for
//                        debugging detect_color
// -------------------------------------------------------------------------

// debugging module

 wire [7:0] h_sel_puck, s_sel_puck, v_sel_puck;
 wire [7:0] h_sel_paddle1, s_sel_paddle1, v_sel_paddle1;
 wire [7:0] h_sel_paddle2, s_sel_paddle2, v_sel_paddle2;
 display_hsv_value view_hsv_puck(.clk(clk), .hcount(hcount),
   .vcount(vcount), .x_coord(x_new_puck), .y_coord(y_new_puck),
     .pixel(video_border_pixel), .hsv(hsv), .h_sel(h_sel_puck),
   .s_sel(s_sel_puck), .v_sel(v_sel_puck));
 display_hsv_value view_hsv_paddle1(.clk(clk), .hcount(hcount),
   .vcount(vcount), .x_coord(x_new_paddle1), .y_coord(y_new_paddle1),
     .pixel(video_border_pixel), .hsv(hsv), .h_sel(h_sel_paddle1),
   .s_sel(s_sel_paddle1), .v_sel(v_sel_paddle1));
 display_hsv_value view_hsv_paddle2(.clk(clk), .hcount(hcount),
   .vcount(vcount), .x_coord(x_new_paddle2), .y_coord(y_new_paddle2),
     .pixel(video_border_pixel), .hsv(hsv), .h_sel(h_sel_paddle2),
   .s_sel(s_sel_paddle2), .v_sel(v_sel_paddle2));

// -------------------------------------------------------------------------
// threshold_hsv - takes in current hsv + user button interactions
//                    -> spits out new hsv thresholds, which is used
//                      by detect_color (ie.color_pixel)
// -------------------------------------------------------------------------

// debugging module - can input approximate hue we want to test
//    (red overlaps around so need to test two, but other colors just one)

// outputs created thresholds from selected hsv center value
wire [7:0] h1_min_puck, h1_max_puck, h2_min_puck, h2_max_puck,
           s_min_puck, s_max_puck, v_min_puck, v_max_puck;
 wire [7:0] h1_min_paddle1, h1_max_paddle1, h2_min_paddle1,
           h2_max_paddle1, s_min_paddle1, s_max_paddle1,
```

```
            v_min_paddle1, v_max_paddle1;
wire [7:0] h1_min_paddle2, h1_max_paddle2, h2_min_paddle2,
            h2_max_paddle2, s_min_paddle2, s_max_paddle2,
               v_min_paddle2, v_max_paddle2;

threshold_hsv #(.INITIAL_THRESHOLD(25),.SPEED(3)) update_threshold (
    .clk(clk), .reset(reset),
    .obj_sel(OBJ_SEL),

    .h_sel_puck(h_sel_puck), .s_sel_puck(s_sel_puck), .v_sel_puck(v_sel_puck),
    .h_sel_paddle1(h_sel_paddle1), .s_sel_paddle1(s_sel_paddle1),
                .v_sel_paddle1(v_sel_paddle1),
    .h_sel_paddle2(h_sel_paddle2), .s_sel_paddle2(s_sel_paddle2),
                .v_sel_paddle2(v_sel_paddle2),

    .btn_enter(btn_enter_debounced), .btn_min_decrease(btn_3_debounced),
                .btn_min_increase(btn_2_debounced),
    .btn_max_decrease(btn_1_debounced), .btn_max_increase(btn_0_debounced),
    .hsv_sel(HSV_SEL), .enable_threshold(ENABLE_THRESH),
    .H(H), .S(S), .V(V), .PUCK(PUCK), .PADDLE1(PADDLE1), .PADDLE2(PADDLE2),

    .h1_min_puck(h1_min_puck), .h1_max_puck(h1_max_puck),
     .h2_min_puck(h2_min_puck), .h2_max_puck(h2_max_puck),
    .s_min_puck(s_min_puck), .s_max_puck(s_max_puck),
     .v_min_puck(v_min_puck), .v_max_puck(v_max_puck),
    .h1_min_paddle1(h1_min_paddle1), .h1_max_paddle1(h1_max_paddle1),
     .h2_min_paddle1(h2_min_paddle1), .h2_max_paddle1(h2_max_paddle1),
    .s_min_paddle1(s_min_paddle1), .s_max_paddle1(s_max_paddle1),
     .v_min_paddle1(v_min_paddle1), .v_max_paddle1(v_max_paddle1),
    .h1_min_paddle2(h1_min_paddle2), .h1_max_paddle2(h1_max_paddle2),
     .h2_min_paddle2(h2_min_paddle2), .h2_max_paddle2(h2_max_paddle2),
    .s_min_paddle2(s_min_paddle2), .s_max_paddle2(s_max_paddle2),
     .v_min_paddle2(v_min_paddle2), .v_max_paddle2(v_max_paddle2)
);

// ----------------------------------------------------------------------------
// color_pixel - detect red and replace pixels with black/magenta
//                (takes in video_border_pixel
//                outputs color_pixel + x_center/y_center of object)
// ----------------------------------------------------------------------------

// color detection, can set hsv thresholds manually through here
// after we've tested/gotten the numbers

// detect color based on specific hsv thresholds, and output color_pixel
// with detected colors blackened for debugging
wire [23:0] color_pixel;
 wire [24:0] x_total_puck, y_total_puck;
 wire [24:0] x_total_paddle1, y_total_paddle1;
 wire [24:0] x_total_paddle2, y_total_paddle2;

 detect_color //#(.H1_MIN(h1_min), .H1_MAX(h1_max), .H2_MIN(h2_min),
      .H2_MAX(h2_max), .SMIN(s_min), .SMAX(s_max), .VMIN(v_min), .VMAX(v_max))
   detect_objects(
```

```
            .clk(clk),

            .H1_MIN_PUCK(h1_min_puck), .H1_MAX_PUCK(h1_max_puck),
        .H2_MIN_PUCK(h2_min_puck), .H2_MAX_PUCK(h2_max_puck),
            .S_MIN_PUCK(s_min_puck), .S_MAX_PUCK(s_max_puck),
        .V_MIN_PUCK(v_min_puck), .V_MAX_PUCK(v_max_puck),
            .H1_MIN_PADDLE1(h1_min_paddle1), .H1_MAX_PADDLE1(h1_max_paddle1),
        .H2_MIN_PADDLE1(h2_min_paddle1), .H2_MAX_PADDLE1(h2_max_paddle1),
            .S_MIN_PADDLE1(s_min_paddle1), .S_MAX_PADDLE1(s_max_paddle1),
        .V_MIN_PADDLE1(v_min_paddle1), .V_MAX_PADDLE1(v_max_paddle1),
            .H1_MIN_PADDLE2(h1_min_paddle2), .H1_MAX_PADDLE2(h1_max_paddle2),
        .H2_MIN_PADDLE2(h2_min_paddle2), .H2_MAX_PADDLE2(h2_max_paddle2),
            .S_MIN_PADDLE2(s_min_paddle2), .S_MAX_PADDLE2(s_max_paddle2),
        .V_MIN_PADDLE2(v_min_paddle2), .V_MAX_PADDLE2(v_max_paddle2),

            .hsv(hsv), .h_count(hcount), .v_count(vcount),
            .vid_border_left(VID_BORDER_LEFT),
        .vid_border_right(VID_BORDER_RIGHT),
        .vid_border_up(VID_BORDER_UP),
            .vid_border_down(VID_BORDER_DOWN),
        .pixel(video_border_pixel_delayed),
        .color_pixel(color_pixel),

            .x_total_puck(x_total_puck),
        .y_total_puck(y_total_puck),
            .x_total_paddle1(x_total_paddle1),
        .y_total_paddle1(y_total_paddle1),
            .x_total_paddle2(x_total_paddle2),
        .y_total_paddle2(y_total_paddle2),

            .x_center_puck_filtered(x_center_puck),
        .y_center_puck_filtered(y_center_puck),
            .x_center_paddle1_filtered(x_center_paddle1),
        .y_center_paddle1_filtered(y_center_paddle1),
            .x_center_paddle2_filtered(x_center_paddle2),
        .y_center_paddle2_filtered(y_center_paddle2)
    );

// ---------------------------------------------------------------------------
// Adjust xy-centers to game space
// ---------------------------------------------------------------------------

wire [10:0] adj_x_puck, adj_x_paddle1, adj_x_paddle2;
wire [9:0] adj_y_puck, adj_y_paddle1, adj_y_paddle2;

assign adj_x_puck = x_center_puck - VID_BORDER_LEFT;
assign adj_y_puck = y_center_puck - VID_BORDER_UP;
assign adj_x_paddle1 = x_center_paddle1 - VID_BORDER_LEFT;
assign adj_y_paddle1 = y_center_paddle1 - VID_BORDER_UP;
assign adj_x_paddle2 = x_center_paddle2 - VID_BORDER_LEFT;
assign adj_y_paddle2 = y_center_paddle2 - VID_BORDER_UP;

 wire [23:0] color_pixel;
 wire [11:0] x_center_puck;
```

```
 wire [10:0] y_center_puck;
 wire [11:0] x_center_paddle1;
 wire [10:0] y_center_paddle1;
 wire [11:0] x_center_paddle2;
 wire [10:0] y_center_paddle2;

 wire [7:0] h1_min_puck, h1_max_puck, h2_min_puck, h2_max_puck,
                  s_min_puck, s_max_puck, v_min_puck, v_max_puck;
 wire [7:0] h1_min_paddle1, h1_max_paddle1, h2_min_paddle1,
               h2_max_paddle1, s_min_paddle1, s_max_paddle1,
               v_min_paddle1, v_max_paddle1;
 wire [7:0] h1_min_paddle2, h1_max_paddle2, h2_min_paddle2,
            h2_max_paddle2, s_min_paddle2, s_max_paddle2,
            v_min_paddle2, v_max_paddle2;

 wire [7:0] h_sel_puck, s_sel_puck, v_sel_puck;
 wire [7:0] h_sel_paddle1, s_sel_paddle1, v_sel_paddle1;
 wire [7:0] h_sel_paddle2, s_sel_paddle2, v_sel_paddle2;

 wire [10:0] adj_x_puck, adj_x_paddle1, adj_x_paddle2;
wire [9:0] adj_y_puck, adj_y_paddle1, adj_y_paddle2;

 wire [23:0] crosshair_pixel;

 assign tv_in_i2c_clock = 1'b0;
assign tv_in_reset_b = 1'b0;
//assign tv_in_i2c_data = 1'bZ;

// ----------------------------------------------------------------------------
// Pong logic
// ----------------------------------------------------------------------------

// feed XVGA signals to user's pong game
wire [23:0] pong_pixel;
wire [23:0] cb_pixel;
wire [7:0]  velocity_x, velocity_y;

wire        halt; // 1 to stop flight due to game conditions
wire        kill; // 1 to stop flight due to other conditions

 wire [23:0] video_coord_pixel;
 assign video_coord_pixel = (CROSSHAIR_CHROMA_SEL) ?
                                  color_pixel : crosshair_pixel;

pong_game pg(
    .vclock(clk), .reset(reset),
    .enable(~kill),
    .left_up(btn_up_debounced), .left_down(btn_down_debounced),
    .right_up(btn_1_debounced), .right_down(btn_0_debounced),
    .pspeed(PSPEED),
    .hcount(hcount), .vcount(vcount),
    .hsync(hsync),.vsync(vsync),.blank(blank),

    .base_pixel(video_coord_pixel),
```

```
        .puck_x(x_center_puck), .puck_y(y_center_puck),
          .led(led[3:0]),
        .paddle_1_x(x_center_paddle1), .paddle_1_y(y_center_paddle1),
        .paddle_2_x(x_center_paddle2), .paddle_2_y(y_center_paddle2),

        .pixel(pong_pixel),
        .velocity_x(velocity_x), .velocity_y(velocity_y),
        .halt(halt)
    );

    // -----------------------------------------------------------------------------
    // Estimator
    // -----------------------------------------------------------------------------
     wire    airborne; // 1 if drone is currently in the air
     wire    stable;   // 1 if drone is stable enough to control

//
//    estimator estimator(
//          .imu(),
//          .barometer(),
//          .ultrasound(),
//          .magnetometer(),
//          .optical_x(),
//          .optical_y(),
//          .data_ready(),
//          .airborne(airborne),
//          .stable(stable)
//    );

    // -----------------------------------------------------------------------------
    // Controls
    // -----------------------------------------------------------------------------
    //  switch[1:0] selects which video generator to use:
    //  00: user's pong game, safety ON (kill is true)
    //  01: 1 pixel outline of active video area (adjust screen controls)
    //  10: user's pong game, safety OFF (kill is false)
    //  11: user's pong game, safety ON (kill is true)

    wire        display_game; // 1 to display the game on VGA
    wire        in_flight; // 1 if drone is commanded to be flying
     wire [7:0] x_velocity,y_velocity;

    assign display_game = DISPLAY_KILL != 2'b01;
    assign kill = DISPLAY_KILL != 2'b10;
    assign in_flight = ~(halt || kill) && display_game;
     assign airborne = 1;
     assign stable = 1;


    controls controls(
             .reference_vx(velocity_x),
             .reference_vy(velocity_y),
             .airborne(airborne),
             .in_flight(in_flight),
```

```
          .stable(stable),
          .cmd_vx(x_velocity),
          .cmd_vy(y_velocity)
);

// ----------------------------------------------------------------------------
// Communications
// ----------------------------------------------------------------------------

 wire [7:0] data_in,commanded;
 wire done_send,command_ready;
 wire [3:0] state_bret;

 comms_command_center command(.start(switch[3]),.clock(clock_27mhz),
                          .reset(reset),
 .sensor_data(data_in), .connected(), .in_the_air(),
              .takeoff_land(switch[2]),.done_sending(done_send),
 .up(), .right(), .down(), .left(), .done_sending_sens(),
 .command(commanded),.roll_and_pitch(),.state(state_bret),
                    .command_ready(command_ready));

 transmit_to_drone tx(.clock(clock_27mhz),.command_ready(command_ready),
 .command(commanded),.x_velocity(x_velocity),.y_velocity(y_velocity),
 .tx_out(user3[0]),.done_sending(done_send));

 rx_from_drone rx(.clock_27mhz(clock_27mhz),.data(user3[1]),.data_receive(data_in));


// ----------------------------------------------------------------------------
// Display select - pong vs video coord
// ----------------------------------------------------------------------------

// select output pixel data

wire [63:0] disp_hsv_puck, disp_hsv_paddle1, disp_hsv_paddle2;
wire [63:0] disp_sel_puck, disp_sel_paddle1, disp_sel_paddle2;
wire [63:0] disp_coord_puck, disp_coord_paddle1, disp_coord_paddle2;
 wire [63:0] disp_velocity_puck;

assign disp_hsv_puck = {h1_max_puck,h1_min_puck, h2_max_puck,h2_min_puck,
                     s_max_puck,s_min_puck, v_max_puck,v_min_puck};
assign disp_hsv_paddle1 = {h1_max_paddle1,h1_min_paddle1,
                     h2_max_paddle1,h2_min_paddle1, s_max_paddle1,
                     s_min_paddle1, v_max_paddle1,v_min_paddle1};
assign disp_hsv_paddle2 = {h1_max_paddle2,h1_min_paddle2, h2_max_paddle2,
                     h2_min_paddle2, s_max_paddle2,s_min_paddle2,
                     v_max_paddle2,v_min_paddle2};

assign disp_sel_puck = {16'b0, 8'b0,h_sel_puck, 8'b0,s_sel_puck,
                        8'b0,v_sel_puck};
assign disp_sel_paddle1 = {16'b0, 8'b0,h_sel_paddle1,
                        8'b0,s_sel_paddle1, 8'b0,v_sel_paddle1};
assign disp_sel_paddle2 = {16'b0, 8'b0,h_sel_paddle2,
                        8'b0,s_sel_paddle2, 8'b0,v_sel_paddle2};
```

```
  assign disp_coord_puck = {5'b0,x_center_puck, 6'b0,y_center_puck,
                        5'b0,adj_x_puck, 6'b0,adj_y_puck};
  assign disp_coord_paddle1 = {5'b0,x_center_paddle1, 6'b0,y_center_paddle1,
                        5'b0,adj_x_paddle1, 6'b0,adj_y_paddle1};
  assign disp_coord_paddle2 = {5'b0,x_center_paddle2, 6'b0,y_center_paddle2,
                        5'b0,adj_x_paddle2, 6'b0,adj_y_paddle2};

   assign disp_velocity_puck = {8'b0,x_velocity, 8'b0,y_velocity,
                        8'b0,velocity_x, 8'b0,velocity_y};

   reg [23:0] pixel;
  reg b,hs,vs;

  always @(posedge clk) begin
     b <= blank;
     hs <= hsync;
     vs <= vsync;

     case (OBJ_SEL)
        PUCK: dispdata = (DISP_XY_HSV) ? disp_hsv_puck : disp_velocity_puck;
        PADDLE1: dispdata = (DISP_XY_HSV) ? disp_hsv_paddle1: disp_velocity_puck;
        PADDLE2: dispdata = (DISP_XY_HSV) ? disp_hsv_paddle2 : disp_velocity_puck;
        //PUCK: dispdata = (DISP_XY_HSV) ? disp_hsv_puck : disp_coord_puck;
        //PADDLE1: dispdata = (DISP_XY_HSV) ? disp_hsv_paddle1: disp_coord_paddle1;
        //PADDLE2: dispdata = (DISP_XY_HSV) ? disp_hsv_paddle2 : disp_coord_paddle2;
     endcase

     pixel <= (display_game) ? pong_pixel : video_coord_pixel;

  end

  // --------------------------------------------------------------------------
  // final monitor visualization - currently used for debugging color detection,
  //                                will later have pong game overlay
  // --------------------------------------------------------------------------

  // VGA Output.  In order to meet the setup and hold times of the
  // AD7125, we send it ~clk.
   assign vga_out_red = pixel[23:16];
  assign vga_out_green = pixel[15:8];
  assign vga_out_blue = pixel[7:0];

  assign vga_out_sync_b = 1'b1;    // not used
  assign vga_out_pixel_clock = ~clk;
  assign vga_out_blank_b = ~b;
  assign vga_out_hsync = hs;
  assign vga_out_vsync = vs;

  // debugging
  assign led[7:4] = ~state_bret[3:0];


endmodule
```

10.2 Communications

10.2.1 Communications: data_parser Module

```
//Bret Heaslet
//This module converts locally used parameters in verilog
//into readable packets sent to drone.
module data_parser #(parameter
    //command states used throughout all modules
    IDLE = 3'b000,
    SEND_TAKEOFF_COMMAND = 3'b111,
    SEND_PCMD_COMMAND = 3'b110,
    SEND_LAND_COMMAND = 3'b100,
    //Start/Stop/IDLE_HIGH_BYTE for serial comms
    STOP_BIT = 1'b0,
    START_BIT = 1'b1,
    IDLE_HIGH_BYTE = 8'b1111_1111,
    //Data type as defined by drones api
    ACKNOLEDGEMENT = 8'd1,
    DATA_NO_ACKNOWLEDGE = 8'd2,
    LOW_LATENCY_DATA = 8'd3,
    DATA_WITH_ACKNOWLEDGE = 8'd4,
    //Buffer type as defined in drone api
    SEND_WITH_ACKNOWLEDGE = 8'd11,
    ACKNOWLEDGE_DRONE_DATA = 8'd127,
    //Each packet is required to have trailing zeros and
    //zeros padding when commands are not a full
    //11 bytes
    PADDING_ZEROS = 8'd0,
    //Command names/numbers as specified in drone api
    TAKEOFF = 8'd1,
    PCMD = 8'd2,//This is the command used to change the velocity of the drone
    LAND = 8'd3,
    //Class names/numbers as specified in drone api
    PILOTING = 8'd0,
    //Project name/number as specified in drone api
    MINIDRONE = 8'd2,
    //Sensor update parameters
    UPDATE_X_VELOCITY = 8'd19,
    UPDATE_Y_VELOCITY = 8'd20
    )
    (
    input clk,
    input [4:0] command_to_parse,
    input [15:0] sensor_data,
    output reg sensor_update,
    output reg [7:0] x_velocity_sensor, y_velocity_sensor,
    output reg [149:0] command_out);

    //Each buffer requires each packet to be numbered;
    //counts get recycled after 255...
    reg [7:0]   send_with_acknowledge_counter = 0,
                acknowledge_drone_data_counter = 0,
                data_type, buffer_type, buffer_counter, project_name,
```

```
            class_name, command_name, packet_length, which_sensor,
            sensor_velocity;

reg [9:0]   roll_value, pitch_value, yaw_value, height_off_ground;

always @(posedge clk)
begin
    //Compiled packet to send out over serial;
    //the case statement below changes variables within this packet
    //as directed by the comms_command_module
    command_out <= {height_off_ground,yaw_value,pitch_value,roll_value,
                    START_BIT,STOP_BIT,PADDING_ZEROS,START_BIT,STOP_BIT,
                    command_name,START_BIT,STOP_BIT,class_name,START_BIT,
                    STOP_BIT,project_name,START_BIT,STOP_BIT,
                    PADDING_ZEROS,START_BIT,STOP_BIT,PADDING_ZEROS,
                    START_BIT,STOP_BIT,PADDING_ZEROS,START_BIT,STOP_BIT,
                    STOP_BIT,buffer_counter,START_BIT,STOP_BIT,
                    buffer_type,START_BIT,STOP_BIT,data_type,START_BIT};
    //parses local command into drone command
    //It is simply assigning values based on the drones api
    case(command_to_parse)

        IDLE: command_out <= {150{1'b1}};

        SEND_TAKEOFF_COMMAND:
        begin
            data_type <= DATA_WITH_ACKNOWLEDGE;
            buffer_type <= SEND_WITH_ACKNOWLEDGE;
            buffer_counter <= send_with_acknowledge_counter;
            packet_length <= 8'd11;
            project_name <= MINIDRONE;
            class_name <= PILOTING;
            command_name <= TAKEOFF;
            roll_value <= 10'b1111_1111_11;
            pitch_value <= 10'b1111_1111_11;
            yaw_value <= 10'b1111_1111_11;
            height_off_ground <= 10'b1111_1111_11;
            send_with_acknowledge_counter <=
                send_with_acknowledge_counter + 1;
        end

        SEND_PCMD_COMMAND:
        begin
            data_type <= DATA_WITH_ACKNOWLEDGE;
            buffer <= SEND_WITH_ACKNOWLEDGE;
            buffer_counter <= send_with_acknowledge_counter;
            packet_length <= 8'd15;
            project_name <= MINIDRONE;
            class_name <= PILOTING;
            command_name <= PCMD;
            roll_value <= {STOP_BIT,x_velocity,START_BIT};
            pitch_value <= {STOP_BIT,y_velocity,START_BIT};
            yaw_value <= {STOP_BIT,8'd0,START_BIT}:
            height_off_ground <= {STOP_BIT,8'd1,START_BIT};
```

```
            send_with_acknowledge_counter <=
                send_with_acknowledge_counter + 1;
        end

        SEND_LAND_COMMAND:
        begin
            data_type <= DATA_WITH_ACKNOWLEDGE;
            buffer_type <= SEND_WITH_ACKNOWLEDGE;
            buffer_counter <= send_with_acknowledge_counter;
            packet_length <= 8'd11;
            project_name <= MINIDRONE;
            class_name <= PILOTING;
            command_name <= LAND;
            roll_value <= 10'b1111_1111_11;
            pitch_value <= 10'b1111_1111_11;
            yaw_value <= 10'b1111_1111_11;
            height_off_ground <= 10'b1111_1111_11;
            send_with_acknowledge_counter <=
                send_with_acknowledge_counter + 1;
        end

        default: command_out <= {150{1'b1}};

    endcase

    which_sensor <= sensor_data[15:8];
    sensor_velocity <= sensor_data[7:0];


    //This statement parses the sensor data coming from the drone
    //then updates the velocity data and sets a flag letting other
    //modules know that new data has arrived.
    case(which_sensor)

        IDLE: sensor_update <= 1'b0;

        UPDATE_X_VELOCITY:
        begin
            if (x_velocity_sensor != sensor_velocity)
            begin
                x_velocity_sensor <= sensor_velocity;
                sensor_update <= 1'b1;
            end
        end

        UPDATE_Y_VELOCITY:
        begin
            if (y_velocity_sensor != sensor_velocity)
            begin
                y_velocity_sensor <= sensor_velocity;
                sensor_update <= 1'b1;
            end
        end
```

```
            default: sensor_update <= 1'b0;

        endcase

    end
endmodule
```

10.2.2 Communications: comms_command_center Module

```
`timescale 1ns / 1ps
//Bret Heaslet
//This module is the central FSM controlling the state of the drone:
//Those states being:
//IDLE
//WAITING_FOR_COMMAND (either take off or disconnect)
//TAKING_OFF
//IN_FLIGHT (This constantly streams a command to drone along
//with the changing velocities)
//LANDING

module comms_command_center #(
    parameter
    IDLE = 4'b0000,
    WAITING_FOR_COMMAND = 4'b0011,
    TAKING_OFF = 4'b0111,
    IN_FLIGHT = 4'b1111,
    LANDING = 4'b1110,
    IDLE_COMMAND = 3'b000,
    TAKEOFF = 3'b111,
    LAND = 3'b100,
    ADJUST_ROLL_AND_PITCH = 3'b110,
    INITIATE_CONNECTION = 19
    )(
    input start, clock, reset, takeoff_land, done_sending,
    input signed [7:0] sensor_data,
    output reg [7:0] command = 8'b1111_1111,
    output reg [3:0] state = IDLE,
    output reg command_ready = 1'b0,airborne = 1'b0);

    always @(posedge clock)
    begin
        //keep track of previous reset button state
        old_reset <= reset;
        //if reset button is pressed
        //return state to 'IDLE' and
        //'command_ready' to 0.
        if (reset && ~old_reset)
        begin
            state <= IDLE;
            command_ready <= 1'b0;
        end
        case(state)

            //IDLE checks for switch to be turned on (start)
```

```
//once on, changes state to 'WAITING_FOR_COMMAND'
//and send the initiate connection command
IDLE:
begin,
    if (start)
    begin
        state <= WAITING_FOR_COMMAND;
        command <= INITIATE_CONNECTION;
        command_ready <= 1'b1;
    end else if (airborne) begin
        state <= IN_FLIGHT;
        command_ready <1'b1;
    end
end

//this state simply waits for the takeoff_land
//switch to be flipped, then sends the takeoff
//command to the drone
WAITING_FOR_COMMAND:
begin
    if (~done_sending) command_ready <= 0;
    if (takeoff_land)
    begin
        state <= TAKING_OFF;
        command <= TAKEOFF;
        command_ready <= 1'b1;
    end
end

//this state sets the airborne flag to 1
//indicating we are in the air
//and changes the state to 'IN_FLIGHT'
TAKING_OFF:
begin
    if (~done_sending) command_ready <= 1'b0;
    else
    begin
        airborne <= 1'b1;
        state <= IN_FLIGHT;
        command <= ADJUST_ROLL_AND_PITCH;
    end
end

//sends command to change velocities
//and monitors for the takeoff_land
//switch to be turned off. Once turned off
//the drone is sent the land command.
IN_FLIGHT:
begin
     command <= ADJUST_ROLL_AND_PITCH;
     command_ready <= 1'b1;
    if (~takeoff_land)
    begin
        state <= LANDING;
```

```
                command <= LAND;
                command_ready <= 1'b1;
            end
        end

        //sets the airborne flag to zero
        //indicating a land and sets the
        //state back to 'WAITING_FOR_COMMAND'
        LANDING:
        begin
            if (~done_sending) command_ready <= 1'b0;
            else
            begin
                state <= WAITING_FOR_COMMAND;
                airborne <= 1'b0;
            end
        end

        default:
        begin
            state <= IDLE;
            command_ready <= 1'b0;
        end

    endcase

end

endmodule
```

10.2.3 Communications: transmit_to_drone Module

```
`timescale 1ns / 1ps
//Bret Heaslet
//This module transmits data over serial (currently at 1Mbaud)

module transmit_to_drone #(
    parameter
    CLOCK_FACTOR = 26,
    STOP_BIT = 1'b1,
    START_BIT = 1'b0,
    INITIATE_CONNECTION = 19
    )(
    input clock,command_ready,
    input [7:0] command, x_velocity, y_velocity,
    input [150:0] parsed_command,
    output reg tx_out = 1'b1,done_sending = 1'b1
    );

    reg [9:0] data_stream_short = 10'b1111_1111_11;
    reg [8:0] baud_counter = 8'd0, bit_counter = 8'd0;
    reg baud_clock = 1'b0, busy = 1'b0;
    reg [20:0] control_frequency = 0;
```

```
always @(posedge clock)
begin
    //these two lines maintain the baud clock
    //(1 Mbaud rate)
    baud_counter <= baud_clock ? 0 : baud_counter+1;
    baud_clock <= baud_counter == CLOCK_FACTOR;
    if (baud_clock)
    begin
        //counter used to control how frequently
        //commands are sent (set at 20 Hz)
        control_frequency <= control_frequency + 1;
        //if the command input has initiate_connection
        //assigned then use the short otherwise
        //use the longer command sender
        case(command)

            INITIATE_CONNECTION:
            begin

                //if not busy, when command is ready and it's
                //been at least 0.05 seconds, then send
                //the command over serial, set done_sending
                //to 0
                case(busy)
                    START_BIT:
                    begin
                        if (command_ready && control_frequency >= 50000)
                        begin
                            data_stream_short[9:0] <=
                                {STOP_BIT,command[7:0],START_BIT};
                            busy <= STOP_BIT;
                            control_frequency <= 0;
                            done_sending <= 0;
                        end
                    end

                    //if busy then current sending a command
                    //shift out data bits until all have been
                    //sent, then indicate done_sending
                    //and switch back to not busy
                    STOP_BIT:
                    begin
                        tx_out <= data_stream_short[0];
                        data_stream_short <= data_stream_short >> 1;
                        data_stream_short[9] <= STOP_BIT;
                        bit_counter <= bit_counter + 1;
                        if (bit_counter == 9)
                        begin
                            busy <= START_BIT;
                            bit_counter <= 0;
                            tx_out <= 1'b1;
                            done_sending <= 1;
                        end
                    end
```

```
                    endcase
                end

                default:
                begin

                    //if not busy, when command is ready and it's
                    //been at least 0.05 seconds, then send
                    //the command over serial, set done_sending
                    //to 0
                    case(busy)

                        START_BIT:
                        begin
                            if (command_ready && control_frequency >= 50000)
                            begin
                                data_stream_long[149:0] <= parsed_command;
                                busy <= STOP_BIT;
                                control_frequency <= 0;
                                done_sending <= 0;
                            end
                        end

                        //if busy then current sending a command
                        //shift out data bits until all have been
                        //sent, then indicate done_sending
                        //and switch back to not busy
                        STOP_BIT:
                        begin
                            tx_out <= data_stream_long[0];
                            data_stream_long <= data_stream_long >> 1;
                            data_stream_long[149] <= STOP_BIT;
                            bit_counter <= bit_counter + 1;
                            if (bit_counter == 149)
                            begin
                                busy <= START_BIT;
                                bit_counter <= 0;
                                tx_out <= 1'b1;
                                done_sending <= 1;
                            end
                        end

                        default: busy <= START_BIT;
                    endcase
                end

            endcase
        end
    end
endmodule
```

10.2.4 Communications: rx_from_drone Module

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

```
//Bret Heaslet
//This module is receives data via serial (currently at 1Mbaud)

module rx_from_drone #(parameter
    CLOCK_FACTOR = 26,
    IDLE = 2'b00,
    CHECK_START_BIT = 2'b01,
    OPEN_FOR_DATA = 2'b10,
    UPDATE_X_VELOCITY = 8'd19,
    UPDATE_Y_VELOCITY = 8'd20
    )
    (
    input clock_27mhz, data,
    output [7:0] x_velocity_sensor,y_velocity_sensor,
    output sensor_update
    );

    reg [4:0] sample_counter = 5'd0, index = 5'd0,sample_count = 5'd0;
    reg sample_clock = 1'b0, data_reg_1 = 1'b1, data_reg_2 = 1'b1, data_old;
    reg [2:0] sampling_state = IDLE;
    reg [18:0] store_data;
    reg [15:0] data_receive;

    data_parser parse_sensor(.clk(clock_27mhz),.sensor_data(data_receive),
                            .sensor_update(sensor_update),
                            .x_velocity_sensor(x_velocity_sensor),
                            .y_velocity_sensor(y_velocity_sensor));

    //data registers to debounce signal
    always @(posedge clock_27mhz)
    begin
        data_reg_1 <= data;
        data_reg_2 <= data_reg_1;
    end

    always @(posedge clock_27mhz) begin
        //maintains count of samples
        sample_count <= sample_count + 1;
        //keep track of last data bit
        data_old <= data_reg_2;
        case(sampling_state)
            //once a start bit is received
            //look at the middle sample of that
            //same start bit, if it is still
            //a start bit, the proceed to continue
            //sampling
            IDLE:
            begin
                if(~data_reg_2 && data_old)
                begin
                    index <= 0;
                    store_data <= 0;
                    sample_count <= 0;
                    sampling_state <= CHECK_START_BIT;
```

```
                end else sampling_state <= IDLE;
            end

            CHECK_START_BIT:
            begin
                if (sample_count == 13)
                begin
                    if(~data_reg_2)
                    begin
                    sampling_state <= OPEN_FOR_DATA;
                    sample_count <= 0;
                    end else sampling_state <= IDLE;
                end else sampling_state <= CHECK_START_BIT;
            end

            //keep sampling the data until 19 bits have been received
            //(the data plus 2 stop bits and one start bit)
            OPEN_FOR_DATA:
            begin
                if (sample_count == 26)
                begin
                    store_data[index] <= data_reg_2;
                    sample_count <= 0;
                    index <= index + 1;
                end
                if (index == 18)
                begin
                    data_receive[15:0] <= {store_data[18:11],store_data[7:0];
                    sample_count <= 0;
                    sampling_state <= IDLE;
                    index <= 0;
                end else sampling_state <= OPEN_FOR_DATA;
            end

            default: sampling_state <= IDLE;

        endcase

    end
endmodule
```

10.3 Controls

10.3.1 Controls: pong_game Module

```
`timescale 1ns / 1ps
`default_nettype none
/**********************************************************************
 Engineer: David Mueller
 Module Name:    pong_game
**********************************************************************/
module pong_game #(parameter BITS_X=12, BITS_Y=11)
    (input wire           vclock, // 65MHz clock
     input wire           reset, // 1 to initialize module
```

```
    input wire           enable, // 1 to allow game to advance
    input wire           left_up, // 1 when paddle should move up
    input wire           left_down, // 1 when paddle should move down
    input wire           right_up, // 1 when right paddle should move up
    input wire           right_down, // 1 when right paddle should move down
    input wire [3:0]     pspeed, // puck speed in pixels/tick
    input wire [10:0]    hcount, // horizontal index of current pixel (0..1023)
    input wire [9:0]     vcount, // vertical index of current pixel (0..767)
    input wire           hsync, // XVGA horizontal sync signal (active low)
    input wire           vsync, // XVGA vertical sync signal (active low)
    input wire           blank, // XVGA blanking (1 means output black pixel)

    output wire [3:0]        led,
    input wire [23:0]        base_pixel, // SABINA - comment when running pong
    input wire signed [BITS_X-1:0]  puck_x, paddle_1_x, paddle_2_x,
    input wire signed [BITS_Y-1:0]  puck_y, paddle_1_y, paddle_2_y,

    output wire [23:0]       pixel, // pong game's pixel  // r=23:16, g=15:8, b=7:0
    output wire signed [7:0] velocity_x, velocity_y,
    output wire        [3:0] puck_state,
    output wire            halt
     );

   // Bus widths propogated forward to other modules
   // Additional bit allows signed values, which makes life much
   // easier if we allow negative pixel values, which is useful

   // Convenient color constants
   localparam C_BLACK = 24'h000000;
   localparam C_RED   = 24'hCC0000;
   localparam C_WHITE = 24'hFFFFFF;
   localparam C_GRAY  = 24'hCCCCCC;
   localparam C_DEEP_GREEN = 24'h118811;

   // Deathstar sprite
   localparam DS_ALPHA = 9'd128;
   localparam [BITS_X-1:0] DS_WIDTH = 32; // SABINA - change size of puck
   localparam [BITS_Y-1:0] DS_HEIGHT = 32; // SABINA - change size of puck

   // Paddle
   localparam [BITS_X-1:0] P_WIDTH = 25;
   localparam [BITS_Y-1:0] P_HEIGHT = 100;
   localparam [2:0]        P_SPEED = 4;
   localparam [BITS_X-1:0] LEFT_INIT_X = 8;
   localparam [BITS_X-1:0] RIGHT_INIT_X = 1024 - P_WIDTH - 8;
   localparam [BITS_Y-1:0] INIT_Y = 8;

   // Deathstar motion control ------
   // Position
   wire signed [BITS_X-1:0] ds_x;
   wire signed [BITS_Y-1:0] ds_y;

   // +1, 0, -1 move multiplier
   wire signed [1:0]      factor_x;
```

```
    wire signed [1:0]      factor_y;

    // Uncorrected position after next logic update
    wire signed [BITS_X-1:0] next_ds_x;
    wire signed [BITS_Y-1:0] next_ds_y;

    // Paddle motion control ------
    // Position
    wire signed [BITS_X-1:0] left_paddle_x;
    wire signed [BITS_Y-1:0] left_paddle_y;
    wire signed [BITS_X-1:0] right_paddle_x;
    wire signed [BITS_Y-1:0] right_paddle_y;

    // Uncorrected position after next logic update
//    wire signed [BITS_X-1:0] next_left_paddle_x;
//    wire signed [BITS_Y-1:0] next_left_paddle_y;
//    wire signed [BITS_X-1:0] next_right_paddle_x;
//    wire signed [BITS_Y-1:0] next_right_paddle_y;

    //wire [23:0]     base_pixel; // SABINA - comment to run standalone pong
    wire [23:0]          left_paddle_pixel;
    wire [23:0]          right_paddle_pixel;
    wire                paddle_collide_left;
    wire                paddle_collide_right;
    wire [3:0]       collisions;

    assign led [3:0] = ~({factor_x, factor_y});

    // PUCK INPUT -------------------------------------------
    assign ds_x = puck_x - (DS_WIDTH >> 1);
    assign ds_y = puck_y - (DS_HEIGHT >> 1);

    // PLAYER INPUT -------------------------------------------
    assign left_paddle_x = paddle_1_x - (P_WIDTH >> 1);
    assign left_paddle_y = paddle_1_y - (P_HEIGHT >> 1);
    assign right_paddle_x = paddle_2_x - (P_WIDTH >> 1);
    assign right_paddle_y = paddle_2_y - (P_HEIGHT >> 1);

    // RENDERING -----------------------------------------------------
    // draw colored rectangle
    blit_stiff_rect #(.WIDTH(P_WIDTH),
              .HEIGHT(P_HEIGHT),
              .COLOR(C_WHITE))
    left_paddle (
         .x(left_paddle_x),
         .y(left_paddle_y),
         .draw_x(hcount),
         .draw_y(vcount),
         .pixel_in(base_pixel),
         .pixel_out(left_paddle_pixel));

    blit_stiff_rect #(.WIDTH(P_WIDTH),
              .HEIGHT(P_HEIGHT),
              .COLOR(C_WHITE))
```

```
right_paddle (
      .x(right_paddle_x),
      .y(right_paddle_y),
      .draw_x(hcount),
      .draw_y(vcount),
      .pixel_in(left_paddle_pixel),
      .pixel_out(right_paddle_pixel));

// draw puck
blit_stiff_rect #(.WIDTH(DS_WIDTH),
          .HEIGHT(DS_HEIGHT),
          .COLOR(C_GRAY))
gray_blob (
       .x(puck_x),
         .y(puck_y),
       .draw_x(hcount),
       .draw_y(vcount),
       .pixel_in(right_paddle_pixel),
       .pixel_out(pixel));

 // COLLISION DETECTION -------------------------------------------

// check paddle collision
rects_overlap #(.BITS_X(BITS_X),
        .BITS_Y(BITS_Y))
left_paddle_collider (
          .x_1(left_paddle_x),
          .y_1(left_paddle_y),
          .w_1(P_WIDTH),
          .h_1(P_HEIGHT),
          .x_2(ds_x),
          .y_2(ds_y),
          .w_2(DS_WIDTH),
          .h_2(DS_HEIGHT),
          .result(paddle_collide_left));

rects_overlap #(.BITS_X(BITS_X),
        .BITS_Y(BITS_Y))
right_paddle_collider (
           .x_1(right_paddle_x),
           .y_1(right_paddle_y),
           .w_1(P_WIDTH),
           .h_1(P_HEIGHT),
           .x_2(ds_x),
           .y_2(ds_y),
           .w_2(DS_WIDTH),
           .h_2(DS_HEIGHT),
           .result(paddle_collide_right));

// check border collisions
border_collider #(.BITS_X(BITS_X),
          .BITS_Y(BITS_Y))
collider (
        .x(ds_x),
```

```
          .y(ds_y),
          .w(DS_WIDTH),
          .h(DS_HEIGHT),
          .collisions(collisions)
     );

    // FSM/CONTINUOUS STATE UPDATE ----------------------------------

    assign velocity_x = enable ? factor_x * $signed({1'b0,pspeed}) : 8'd0;
    assign velocity_y = enable ? factor_y * $signed({1'b0,pspeed}) : 8'd0;
    assign next_ds_x = ds_x + velocity_x;
    assign next_ds_y = ds_y + velocity_y;

    // cycle fsm
    deathstar_fsm ds_fsm(.lclock(vclock),
             .reset(reset),
             .collisions(collisions),
             .paddle(paddle_collide_left|paddle_collide_right),
             .halt(halt),
             .factor_x(factor_x),
             .factor_y(factor_y));

    // update logic at a dramatically lower speed
/*
    always @(posedge vsync) begin

    if( reset ) begin
        ds_x <= 128;
        ds_y <= 128;
    end else begin
        ds_x <= next_ds_x;
        ds_y <= next_ds_y;
      end
   end
*/
endmodule // pong_game
```

10.3.2 Controls: legacy_controls Module

```
`timescale 1ns / 1ps
`default_nettype none
/////////////////////////////////////////////////////////////////////////////////
// Engineer: David Mueller
//
// Create Date:    22:13:59 12/02/2018
// Design Name:
// Module Name:    controls
/////////////////////////////////////////////////////////////////////////////////
module controls
    (input                    clk,
     input             reset,
     input wire               position_mode,
     input wire signed [7:0]  reference_x, reference_y,
     input wire signed [7:0]  reference_vx, reference_vy,
```

```
     input wire signed [31:0] estimated_x, estimated_y,
     input wire signed [31:0] estimated_vx, estimated_vy,
     input wire           airborne,
     input wire           in_flight,
     input wire           stable,
     output wire signed [7:0] cmd_vx, cmd_vy
    );

    wire             controls_active;
    reg signed [31:0]        float_rvx, float_rvy;
    reg signed [31:0]        err_vx, err_vy;
    reg signed [7:0]         gated_vx, gated_vy;

    localparam GAIN = 5;
    localparam GAIN_SHIFT = 2;

    assign controls_active = airborne && in_flight && stable && ~reset;

    assign cmd_vx = controls_active ? reference_vx : 0;
    assign cmd_vy = controls_active ? reference_vy : 0;

//    assign cmd_vx = controls_active ? gated_vx : 0;
//    assign cmd_vy = controls_active ? gated_vy : 0;


    always @(posedge clk) begin
    if(reset) begin
        float_rvx <= 0;
        float_rvy <= 0;
        err_vx <= 0;
        err_vy <= 0;
        gated_vx <= 0;
        gated_vy <= 0;

    end else begin
        float_rvx <= reference_vx >> 4; // TODO: convert to float
        float_rvy <= reference_vy >> 4; // TODO: convert to float

        err_vx <= float_rvx - estimated_vx;
        err_vy <= float_rvy - estimated_vy;

        gated_vx <= (err_vx * GAIN) << GAIN_SHIFT; // TODO: convert from float
        gated_vy <= (err_vy * GAIN) << GAIN_SHIFT; // TODO: convert from float

    end

    end

endmodule
```

10.3.3 Controls: controls Module

```
`timescale 1ns / 1ps
`default_nettype none
```

```
/*******************************************************************************
 * Engineer: David Mueller
 * Module Name: controls
 *
 * Creates and connects a pair of PD feedback controllers and corresponding
 * state estimators.
 *
 * The current implementation supports open-loop position control, or closed-loop
 * velocity control.
 *
 * NOTE: this version of the `controls` module was not able to be tested in the
 * fully integrated AirPong system. Thus, this module is NOT the one used for that
 * integrated system. For that version, see the `legacy_controls` module.
 *
 * This control scheme is heavily reliant on proper operation of the estimator.
 * Since the drone only sends sensor updates at 2Hz, the estimator is needed to
 * propogate drone state forward so the feedback controllers can make more
 * freqency updates than 2Hz.
 *
 * Without the estimators, the drone _cannot_ be made stable, as the feedback
 * controllers essentially become bang-bang controllers.
 *
 * Inputs
 * ------
 * clk: boolean
 *     64.5 MHz clock
 *
 * reset: boolean
 *     Reset signal to clear all accumlated state, as well as intermediate
 *     calculations from various pipelines
 *
 * reference_x, reference_y: 16 bits as Q7.8 signed fixed point
 *     Reference position to track.
 *     Scaling: 256 units -> 1 meter
 *
 * reference_vx, reference_vy: 8 bits as Q1.6 signed fixed point
 *     Reference velocity to track.
 *     Scaling: 128 units -> 2 meters/sec
 *
 * sensor_updated: boolean
 *     1 if the drone has sent updated sensor information.
 *
 * airborne: boolean
 *     1 if the drone is currently in the air.
 *     Commands inhibited if 0
 *
 * in_flight: boolean
 *     1 if the drone is commanded to be in the air.
 *     Commands inhibited if 0
 *
 * stable: boolean
 *     1 if the drone is stable/suitable for controlling.
 *     Commands inhibited if 0
 *
```

```
 * Outputs
 * -------
 * cmd_vx, cmd_vy: 8 bit signed integer
 *     Output commanded pitch and roll angles, respectively.
 *     Both values are in the range [-100, 100]
 *     Scaling: 128 units -> 45 degrees
 *
*******************************************************************************/
module controls
    (input wire               clk,
     input wire           reset,
     input wire signed [7:0]  reference_x, reference_y,
     input wire signed [7:0]  reference_vx, reference_vy,
     input wire signed [7:0]  sensed_vx, sensed_vy,
     input wire           sensor_updated,
     input wire           airborne,
     input wire           in_flight,
     input wire           stable,
     output wire signed [7:0] cmd_vx, cmd_vy
    );

    wire             controls_active;
    wire signed [15:0]       upscale_rx, upscale_ry;
    wire signed [7:0]        gated_vx, gated_vy;

    wire signed [15:0]       px_estimate, py_estimate;
    wire signed [7:0]        vx_estimate, vy_estimate;

    assign controls_active = airborne && in_flight && stable && ~reset;

    assign cmd_vx = controls_active ? gated_vx : 0;
    assign cmd_vy = controls_active ? gated_vy : 0;

    assign upscale_rx = reference_vx << 8;
    assign upscale_ry = reference_vy << 8;

    feedback_controller pitch_controller
    (.clk(clk),
     .reset(reset),
     .reference_p(upscale_rx),
     .reference_v(reference_vx),
     .p_estimate(px_estimate),
     .v_estimate(vx_estimate),
     .command(gated_vx));

    feedback_controller roll_controller
    (.clk(clk),
     .reset(reset),
     .reference_p(upscale_ry),
     .reference_v(reference_vy),
     .p_estimate(py_estimate),
     .v_estimate(vy_estimate),
     .command(gated_vy));
```

```
    estimator pitch_estimator
    (.clk(clk),
     .reset(reset),
     .cmd_angle(gated_vx),
     .sensed_velocity(sensed_vx),
     .sensor_updated(sensor_updated),
     .position_est(px_estimate),
     .velocity_est(vx_estimate));

    estimator roll_estimator
    (.clk(clk),
     .reset(reset),
     .cmd_angle(gated_vy),
     .sensed_velocity(sensed_vy),
     .sensor_updated(sensor_updated),
     .position_est(py_estimate),
     .velocity_est(vy_estimate));


endmodule
```

10.3.4 Controls: estimator Module

```
`timescale 1ns / 1ps
`default_nettype none
/*****************************************************************************
 * Engineer: David Mueller
 * Module Name: estimator
 *
 * Position and velocity state estimator.
 *
 * This module updates at ~65MHz, while the drone's actuators can only respond
 * at 10Hz. As a result, latency from pipelining is not a concern.
 *
 * Currently uses a small-angle approximation in leiu of the tangent function
 * for the sake of simplicity and expediency. (i.e. I ran out of time to
 * implement it.) In future revisions,
 *
 * CAUTION
 * -------
 * Numerous design parameters were chosen specifically to simplify the math
 * involved here, e.g. MAX_TILT. As a result, this module will be very brittle
 * to changes in argument lengths, ranges, design parameters, and clock speeds.
 *
 * Inputs
 * ------
 * clk: boolean
 *     64.5 MHz clock
 *
 * reset: boolean
 *     Reset signal to clear all intermediate calculations from pipelines
 *
 * cmd_angle: 8 bit signed value
 *     Most recent commanded angle, in the range [-128, 127]
```

```
*     Scaling: 128 units -> 45 degrees
*
* sensed_velocity: 8 bit signed value
*     Most recent quantized sensor reading from the drone.
*     Scaling: 128 units -> 2 meters/sec
*
* sensor_updated: boolean
*     1 if the drone has sent updated sensor information. This is used to
*     update the velocity integrator with a reference truth value.
*
* Outputs
* -------
* position_est: 16 bits as Q7.8 signed fixed point
*     Estimated position
*     Scaling: 256 units -> 1 meter
*
* velocity_est: 8 bits as Q1.6 signed fixed point
*     Estimated velocity
*     Scaling: 128 units -> 2 meters/sec
*
******************************************************************************/
module estimator #(parameter CLOCKS_PER_UPDATE = 6450000)
   (input wire               clk,
    input wire           reset,
    input wire signed [7:0]  cmd_angle, // [-128,127]
    input wire signed [7:0]  sensed_velocity, // int(m*s * 64)
    input wire           sensor_updated,
    output wire signed [7:0] velocity_est,  // Q1.6 m/s
    output wire signed [15:0] position_est  // Q7.8 m
   );

   localparam GAIN_SHIFT = 3;
   localparam G_DT = 11'b0____11_1110_1101;  // Q0.10
                                            // 9.81 * .1 = .981 ~= 1005/1024
   localparam PI =   11'b011__0010_0100;     // Q2.8
                                            // 3.14159 ~= 3 + 36/256
   localparam DT = 10'b__00_0110_0110;      // uQ0.10

   // Apply gains to convert normalized pitch and roll signals to desired angles
   // Uses scaled radians, so 30 degrees -> 1/6 scaled radian
   //
   //     signal / 128 * MAX_TILT / 180
   // ==> signal / 128 * 45 / 180
   // ==> signal >>> 9
   reg [22:0]             counter = 0;
   reg [3:0]              ticks = 0;

   wire signed [10:0]         norm_angle;  // Q1.9
   reg signed [20:0]          angle = 0;   // Q3.17

   reg signed [30:0]         delta_v = 0;  // Q3.27
   reg signed [28:0]         v_est = 0;    // Q1.27
   wire signed [7:0]         v_sensed;     // Q1.6
   wire signed [28:0]        v_next;       // Q1.27
```

```
    reg signed [38:0]         delta_p = 0;  // Q1.37
    reg signed [44:0]         p_est = 0;    // Q7.37


    // normalized angle - equal to angle / pi
    assign norm_angle = {{3{cmd_angle[7]}}, cmd_angle};

    //
    assign v_sensed = sensed_velocity;
    assign velocity_est = v_est[28:21];
    assign position_est = p_est[44:29];

    assign v_next = sensor_updated ?
            ($signed({v_sensed, 21'b0}) + v_est) >>> 2 : v_est + delta_v;

    always @(posedge clk) begin
    if(reset) begin
        angle <= 0;
        delta_v <= 0;
        delta_p <= 0;
        counter <= 0;
        ticks <= 0;
        v_est <= 0;
        p_est <= 0;

    end else begin
        angle <= norm_angle * $signed(PI);
            delta_v <= angle * $signed(G_DT);

        // TODO: implement tangent function when time allows
        // delta_v <= tan(norm_angle) * $signed(G_DT);

        delta_p <= v_next * $signed(DT);

        if( counter == CLOCKS_PER_UPDATE) begin
        counter <= 0;
        ticks <= ticks + 1;

        v_est <= v_next;
        p_est <= p_est + delta_p;
        end else begin
        counter <= counter + 1;
        end

    end

    end

endmodule
```

10.3.5 Controls: feedback_controller Module

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

```verilog
`default_nettype none
/****************************************************************************
 * Engineer: David Mueller
 * Module Name: feedback_controller
 *
 * Implementation of a Proportional-Derivative feedback controller.
 *
 * This module updates at ~65MHz, while the drone's actuators can only respond
 * at 10Hz. As a result, latency from pipelining is not a concern.
 *
 * Inputs
 * ------
 * clk: boolean
 *     64.5 MHz clock
 *
 * reset: boolean
 *     Reset signal to clear all intermediate calculations from pipelines
 *
 * reference_p: 16 bits as Q7.8 signed fixed point
 *     Reference position to track
 *     Scaling: 256 units -> 1 meter
 *
 * reference_v: 8 bits as Q1.6 signed fixed point
 *     Reference velocity to track.
 *     Scaling: 128 units -> 2 meters/sec
 *
 * p_estimate: 16 bits as Q7.8 signed fixed point
 *     Estimated position
 *     Scaling: 256 units -> 1 meter
 *
 * v_estimate: 8 bits as Q1.6 signed fixed point
 *     Estimated velocity
 *     Scaling: 128 units -> 2 meters/sec
 *
 * Outputs
 * -------
 * command: 8 bit signed integer
 *     Commanded normalized angle, in the range [-100, 100]
 *     Scaling: 128 units -> 45 degrees
 *
 ****************************************************************************/
module feedback_controller #(parameter CLOCKS_PER_UPDATE = 6450000)
    (input wire               clk,
     input wire           reset,
     input wire signed [15:0] reference_p, // Q7.8            [m]
     input wire signed [7:0]  reference_v, // Q1.6 : 128 -> 2 [m/s]
     input wire signed [15:0] p_estimate,  // Q7.8            [m]
     input wire signed [7:0]  v_estimate,  // Q1.6 : 128 -> 2 [m/s]
     output wire signed [7:0] command
    );

//    localparam CLOCKS_PER_UPDATE = 6450000;  // 64,500,000 clk / 1 sec * .1 sec / 1 tick

    localparam K_P = 1.0;
```

```
    localparam K_D = 3.0;
    localparam MASTER_GAIN = 100;

    reg signed [15:0]         p_error = 0; // Q7.8
    reg signed [7:0]          v_error = 0; // Q1.6

    reg signed [15:0]         p_term = 0; // Q7.8
    reg signed [7:0]          v_term = 0; // Q1.6

    reg signed [15:0]         command_sum; // Q7.8
    wire signed [7:0]         command_int;

    wire [1:0]            clipping;
    wire signed [7:0]         command_clipped;

    // Get the integer component
    assign command_int = command_sum[15:8];

    // Determine if any clipping needs to happen
    assign clipping[0] = command_int < -100 ? 1 : 0;
    assign clipping[1] = command_int > 100 ? 1 : 0;

    // Get a clipped version of the command
    assign command_clipped = clipping[0] ? -100 : 100;

    // Apply the clipped command if clipping is called for
    assign command = ( | clipping ) ? command_clipped : command_int;

    always @(posedge clk) begin
    if(reset) begin
        p_error <= 0;
        v_error <= 0;

        p_term <= 0;
        v_term <= 0;

        command_sum <= 0;

    end else begin
        p_error <= reference_p - p_estimate;
        v_error <= reference_v - v_estimate;

        p_term <= p_error * K_P;
        v_term <= v_error * K_D;

        command_sum <= p_term + (v_term << 2);

    end

    end
/*
    assign cmd_vx = controls_active ? gated_vx : 0;
    assign cmd_vy = controls_active ? gated_vy : 0;
```

```
    always @(posedge clk) begin
    if(reset) begin
        float_rvx <= 0;
        float_rvy <= 0;
        err_vx <= 0;
        err_vy <= 0;
        gated_vx <= 0;
        gated_vy <= 0;

    end else begin
        float_rvx <= reference_vx >> 4; // TODO: convert to float
        float_rvy <= reference_vy >> 4; // TODO: convert to float

        err_vx <= float_rvx - estimated_vx;
        err_vy <= float_rvy - estimated_vy;

        gated_vx <= (err_vx * GAIN) << GAIN_SHIFT; // TODO: convert from float
        gated_vy <= (err_vy * GAIN) << GAIN_SHIFT; // TODO: convert from float

    end

    end

    assign controls_active = airborne && in_flight && stable && ~reset;

    assign cmd_vx = controls_active ? reference_vx : 0;
    assign cmd_vy = controls_active ? reference_vy : 0;

//    assign cmd_vx = controls_active ? gated_vx : 0;
//    assign cmd_vy = controls_active ? gated_vy : 0;
*/
/*
    always @(posedge clk) begin
        if(reset) begin
            float_rvx <= 0;
            float_rvy <= 0;
            err_vx <= 0;
            err_vy <= 0;
            gated_vx <= 0;
            gated_vy <= 0;

        end else begin
            float_rvx <= reference_vx >> 4; // TODO: convert to float
            float_rvy <= reference_vy >> 4; // TODO: convert to float

            err_vx <= float_rvx - estimated_vx;
            err_vy <= float_rvy - estimated_vy;

            gated_vx <= (err_vx * GAIN) << GAIN_SHIFT; // TODO: convert from float
            gated_vy <= (err_vy * GAIN) << GAIN_SHIFT; // TODO: convert from float

        end

    end
```

```
*/
endmodule
```

10.3.6 Controls: blit_stiff_rect Module

```
`timescale 1ns / 1ps
`default_nettype none
/*******************************************************************
 Module Name:    blit_stiff_rect
 Description
 -----------
 Render a single-color rectangle via blitting method.

 Blitting depends on the "mask" concept. The mask is a 1-bit value
 which is 1 if the pixel in question falls OUTSIDE the drawing region,
 and is 0 if the pixel falls WITHIN the drawing region.

 The pixel is colored via:
 OLD & MASK | NEW & ~MASK

 Parameters
 ----------
 WIDTH, HEIGHT:  Width and height of the rectangle, respectively
 COLOR:          RGB color to use when drawing. i.e. NEW color

 Inputs
 ------
 x, y:           x,y coordinates of the rectangle's position
 draw_x, draw_y: x,y coordinates of the pixel currently being drawn
 pixel_in:       RGB color of the pixel as it currently exists at
                 draw_x, draw_y. i.e. OLD color

 Outputs
 -------
 pixel_out: RGB color of the pixel as it exists after blitting at
            draw_x, draw_y
*******************************************************************/
module blit_stiff_rect #(parameter WIDTH=64, HEIGHT=64,
             parameter COLOR=24'hFFFFFF)
    (
     input wire [11:0] x,
     input wire [10:0] y,
     input wire [10:0] draw_x,
     input wire [9:0]  draw_y,
     input wire [23:0] pixel_in,
     output wire [23:0] pixel_out
     );

    wire [11:0] left = x;
    wire [11:0] right = x + WIDTH;
    wire [10:0] top = y;
    wire [10:0] bottom = y + HEIGHT;
    wire    mask;
```

```
    // mask is 1 for transparent, i.e. do not affect the buffer
    // mask should therefore be 0 if and only if the current pixel is
    // within the bounding rectangle
    assign mask = ~(draw_x >= left && draw_x < right
            && draw_y >= top && draw_y < bottom);

    assign pixel_out = (pixel_in & {24{mask}}) | (COLOR & ~{24{mask}});

endmodule
```

10.3.7 Controls: lines_intersect Module

```
`timescale 1ns / 1ps
`default_nettype none
/**********************************************************************
 Engineer: David Mueller
 Module Name:    lines_intersect
 Description
 -----------
 Determine if a horizontal and vertical line segment intersect.

 Parameters
 ----------
 BITS_X, BITS_Y: number of bits uses for horizonal and vertical values

 Inputs
 ------
 v1: concatenated x,y coordinates of the upper point on the vertical
   line. Format: {x,y}
 v2: concatenated x,y coordinates of the lower point on the vertical
   line.
 h1: concatenated x,y coordinates of the left point on the horizontal
   line.
 h2: concatenated x,y coordinates of the right point on the horizontal
   line.

 Outputs
 -------
 result: 1 if the lines intersect. 0 otherwise.

**********************************************************************/

module lines_intersect #(parameter BITS_X=16, BITS_Y=16,
             parameter TOTAL_WIDTH=BITS_X+BITS_Y)
    (
     input wire signed [TOTAL_WIDTH-1:0] v1,
     input wire signed [TOTAL_WIDTH-1:0] v2,
     input wire signed [TOTAL_WIDTH-1:0] h1,
     input wire signed [TOTAL_WIDTH-1:0] h2,
     output wire             result
    );

    localparam X = TOTAL_WIDTH-1;
    localparam Y = BITS_Y-1;
```

```
    wire signed [BITS_X-1:0] x11, x21, x22;
    wire signed [BITS_Y-1:0] y11, y12, y21;

    assign x11 = v1[X:Y+1];
//    assign x12 = v2[X:Y+1];
    assign y11 = v1[Y:0];
    assign y12 = v2[Y:0];

    assign x21 = h1[X:Y+1];
    assign x22 = h2[X:Y+1];
    assign y21 = h1[Y:0];
//    assign y22 = h2[Y:0];

    assign result = (x11 >= x21) & (x11 <= x22) & (y11 <= y21) & (y12 >= y21);

endmodule
```

10.3.8 Controls: rects_overlap Module

```
`timescale 1ns / 1ps
`default_nettype none
/**********************************************************************
 Engineer: David Mueller

 Module Name:    rects_overlap
 Description
 -----------
 Determine if 2 rectangles overlap each other.

 Parameters
 ----------
 BITS_X, BITS_Y: number of bits uses for horizonal and vertical values

 Inputs
 ------
 x_1, y_1: x,y coordinates of 1st rectangle
 x_2, y_2: x,y coordinates of 2nd rectangle
 w_1, h_1: width and height of 1st rectangle
 w_2, h_2: width and height of 2nd rectangle

 Outputs
 -------
 result: 1 if the rectangles overlap, 0 otherwise.
**********************************************************************/
module rects_overlap #(parameter BITS_X=16, BITS_Y=16)
    (
     input wire signed [BITS_X-1:0] x_1, w_1, x_2, w_2,
     input wire signed [BITS_Y-1:0] y_1, h_1, y_2, h_2,
     output wire            result
    );

    // four corners of each rectangle
    wire signed [BITS_X+BITS_Y-1:0]   p11, p12, p13, p14;
```

```
wire signed [BITS_X+BITS_Y-1:0]   p21, p22, p23, p24;

// Wires to store intermediate collision checks
// right-top, right-bottom, left-top, left-bottom, ...
wire                  rt,rb,lt,lb,tl,tr,bl,br;

// Corner containment checks
wire                  A, B, C, D;

// top-left, top-right, bottom-left, bottom-right, respectively
assign p11 = {x_1,       y_1};
assign p12 = {x_1+w_1-1, y_1};
assign p13 = {x_1,       y_1+h_1-1};
assign p14 = {x_1+w_1-1, y_1+h_1-1};

assign p21 = {x_2,       y_2};
assign p22 = {x_2+w_2-1, y_2};
assign p23 = {x_2,       y_2+h_2-1};
assign p24 = {x_2+w_2-1, y_2+h_2-1};

// Right-top
lines_intersect
#(.BITS_X(BITS_X), .BITS_Y(BITS_Y))
rt_check
(.v1(p12), .v2(p14),
 .h1(p21), .h2(p22),
 .result(rt));

// Right-bottom
lines_intersect
#(.BITS_X(BITS_X), .BITS_Y(BITS_Y))
rb_check
(.v1(p12), .v2(p14),
 .h1(p23), .h2(p24),
 .result(rb));

// Left-top
lines_intersect
#(.BITS_X(BITS_X), .BITS_Y(BITS_Y))
lt_check
(.v1(p11), .v2(p13),
 .h1(p21), .h2(p22),
 .result(lt));

// Left-bottom
lines_intersect
#(.BITS_X(BITS_X), .BITS_Y(BITS_Y))
lb_check
(.v1(p11), .v2(p13),
 .h1(p23), .h2(p24),
 .result(lb));

// Top-right
lines_intersect
```

```
#(.BITS_X(BITS_X), .BITS_Y(BITS_Y))
tr_check
(.h1(p11), .h2(p12),
 .v1(p22), .v2(p24),
 .result(tr));

// Top-left
lines_intersect
#(.BITS_X(BITS_X), .BITS_Y(BITS_Y))
tl_check
(.h1(p11), .h2(p12),
 .v1(p21), .v2(p23),
 .result(tl));

// Bottom-right
lines_intersect
#(.BITS_X(BITS_X), .BITS_Y(BITS_Y))
br_check
(.h1(p13), .h2(p14),
 .v1(p22), .v2(p24),
 .result(br));

// Bottom-left
lines_intersect
#(.BITS_X(BITS_X), .BITS_Y(BITS_Y))
bl_check
(.h1(p13), .h2(p14),
 .v1(p21), .v2(p23),
 .result(bl));

// x,y coordinates for the 4 collision cases: A,B,C,D
wire signed [BITS_X-1:0]   Ap_x, Bp_x, Cp_x, Dp_x;
wire signed [BITS_Y-1:0]   Ap_y, Bp_y, Cp_y, Dp_y;

assign Ap_x = x_2;
assign Bp_x = x_2;
assign Cp_x = x_2 + w_2 - 1;
assign Dp_x = x_2 + w_2 - 1;

assign Ap_y = y_2;
assign Bp_y = y_2 + h_2 - 1;
assign Cp_y = y_2 + h_2 - 1;
assign Dp_y = y_2;

// Case A: rectangle 2 upper left corner in region
rect_contains
#(.BITS_X(BITS_X), .BITS_Y(BITS_Y))
A_check
(
 .r_x(x_1), .r_y(y_1), .r_w(w_1), .r_h(h_1), // Rectangle 1
 .p_x(Ap_x), .p_y(Ap_y),                     // Rectangle 2 corner
 .result(A));

// Case B: rectangle 2 lower left corner in region
```

```
    rect_contains
    #(.BITS_X(BITS_X), .BITS_Y(BITS_Y))
    B_check
    (
     .r_x(x_1), .r_y(y_1), .r_w(w_1), .r_h(h_1), // Rectangle 1
     .p_x(Bp_x), .p_y(Bp_y),                     // Rectangle 2 corner
     .result(B));

    // Case C: rectangle 2 extending off top left
    rect_contains
    #(.BITS_X(BITS_X), .BITS_Y(BITS_Y))
    C_check
    (
     .r_x(x_1), .r_y(y_1), .r_w(w_1), .r_h(h_1), // Rectangle 1
     .p_x(Cp_x), .p_y(Cp_y),                     // Rectangle 2 corner
     .result(C));

    // Case D: rectangle 2 extending off bottom left
    rect_contains
    #(.BITS_X(BITS_X), .BITS_Y(BITS_Y))
    D_check
    (
     .r_x(x_1), .r_y(y_1), .r_w(w_1), .r_h(h_1), // Rectangle 1
     .p_x(Dp_x), .p_y(Dp_y),                     // Rectangle 2 corner
     .result(D));

    assign result = tr | tl | br | bl
                  | rt | rb | lt | lb
                  |  A |  B |  C |  D;

endmodule // rects_overlap

module rect_contains #(parameter BITS_X=16, BITS_Y=16)
    (
     input wire signed [BITS_X-1:0] r_x, r_w, p_x,
     input wire signed [BITS_Y-1:0] r_y, r_h, p_y,
     output wire              result
     );

    assign result = p_x >= r_x          // pixel right of left edge - inclusively
            && p_x < r_x + r_w  // pixel left of right edge - exclusively
            && p_y >= r_y       // pixel below top edge - inclusively
            && p_y < r_y + r_h; // pixel above bottom edge - exclusively

endmodule // rect_contains
```

10.3.9 Controls: border_collider Module

```
`timescale 1ns / 1ps
`default_nettype none
/*********************************************************************
 Engineer: David Mueller
 Module Name:    border_collider
 Description
```

```
-----------
Checks whether a rectangular region circumscribes another rectangle
by checking if the smaller rectangle overlaps any of its borders:
left, right, top, or bottom.

Parameters
----------
BITS_X, BITS_Y: number of bits uses for horizonal and vertical values
SCREEN_WIDTH, SCREEN_HEIGHT: width and height of larger rectangular
  region

Inputs
------
x, y: x,y coordinates of smaller rectangle.
w, h: width and height of smaller rectangle.

Outputs
-------
collisions: 4-element array of bits to indicate left, right, top, and
  bottom border collisions, respectively. 1 indicates a collision,
  while 0 indicates no collision.
*********************************************************************/
module border_collider #(parameter BITS_X=16, BITS_Y=16,
             parameter [BITS_X-1:0] LEFT_EDGE=82,
             parameter [BITS_Y-1:0] TOP_EDGE=126,
             parameter [BITS_X-1:0] SCREEN_WIDTH=648,
             parameter [BITS_Y-1:0] SCREEN_HEIGHT=416,
             parameter [BITS_X-1:0] RIGHT_EDGE=LEFT_EDGE+SCREEN_WIDTH,
             parameter [BITS_Y-1:0] BOTTOM_EDGE=TOP_EDGE+SCREEN_HEIGHT)

    (input wire signed [BITS_X-1:0] x, w,
     input wire signed [BITS_Y-1:0] y, h,
     output wire [3:0]         collisions);

    localparam signed [BITS_X-1:0] REGION_W = 100;
    localparam signed [BITS_Y-1:0] REGION_H = 100;

    // left edge
    rects_overlap #(.BITS_X(BITS_X), .BITS_Y(BITS_Y)) left
    (.x_2(x), .y_2(y), .w_2(w), .h_2(h),
     .x_1(LEFT_EDGE-REGION_W), .y_1(TOP_EDGE),
     .w_1(REGION_W), .h_1(SCREEN_HEIGHT),
     .result(collisions[3]));

    // right edge
    rects_overlap #(.BITS_X(BITS_X), .BITS_Y(BITS_Y)) right
    (.x_2(x), .y_2(y), .w_2(w), .h_2(h),
     .x_1(RIGHT_EDGE), .y_1(TOP_EDGE),
     .w_1(REGION_W), .h_1(SCREEN_HEIGHT),
     .result(collisions[2]));

    // top edge
    rects_overlap #(.BITS_X(BITS_X), .BITS_Y(BITS_Y)) top
    (.x_2(x), .y_2(y), .w_2(w), .h_2(h),
```

```
     .x_1(LEFT_EDGE), .y_1(TOP_EDGE-REGION_H),
     .w_1(SCREEN_WIDTH), .h_1(REGION_H),
     .result(collisions[1]));

    // bottom edge
    rects_overlap #(.BITS_X(BITS_X), .BITS_Y(BITS_Y)) bottom
    (.x_2(x), .y_2(y), .w_2(w), .h_2(h),
     .x_1(LEFT_EDGE), .y_1(BOTTOM_EDGE),
     .w_1(SCREEN_WIDTH), .h_1(REGION_H),
     .result(collisions[0]));

endmodule
```

10.3.10 Controls: fsms Module

```
`timescale 1ns / 1ps
`default_nettype none
/*******************************************************************
 Engineer: David Mueller
 Module Name:    deathstar_fsm
 Description
 -----------
 FSM to control the ongoing behavior of the pong puck. Handles
 collisions with the screen border as well as left and right paddles.

 Inputs
 ------
 lclock: logic clock. Possibly distinct from the video clock.
 reset: pulse 1 to trigger a reset of the FSM
 collisions: 4-element array of bits to indicate left, right, top, and
   bottom border collisions, respectively. 1 indicates a collision,
   while 0 indicates no collision.
 paddle: 1 if a collision with either the left or right paddle
   occured, false otherwise.

 Outputs
 -------
 factor_x, factor_y: takes a value of -1, 0, or 1 for each signal. 1
   indicates that the puck should move in the positive direction along
   the corresponding axis, -1 indicates movement in the negative
   direction. 0 indicates no movement.
*******************************************************************/
module deathstar_fsm
    (input wire              lclock, reset,
     input wire [3:0]        collisions,
     input wire          paddle,
     output wire         halt,
     output reg signed [1:0] factor_x, factor_y);

    wire [1:0]           state_x, state_y;
    wire             toggle_x, toggle_y;

    assign halt = |collisions[3:2];
```

```
    l2p_fsm pulse_x(.clk(lclock),
            .reset(reset),
            .level(paddle),
            .out(toggle_x));

    l2p_fsm pulse_y(.clk(lclock),
            .reset(reset),
            .level(collisions[1] | collisions[0]),
            .out(toggle_y));

    halting_latch_fsm fsm_x (.clk(lclock), .reset(reset),
                 .toggle(toggle_x),
                 .halt(halt),
                 .resume(1'b0),
                 .out(state_x));

    halting_latch_fsm fsm_y (.clk(lclock), .reset(reset),
                 .toggle(toggle_y),
                 .halt(halt),
                 .resume(1'b0),
                 .out(state_y));

    always @(state_x or state_y) begin
        casex(state_x)
        2'b11: factor_x = 1;
        2'b10: factor_x = -1;
        2'b0x: factor_x = 0;
        default: factor_x = -2;
    endcase // casex (state_x)

    casex(state_y)
        2'b11: factor_y = 1;
        2'b10: factor_y = -1;
        2'b0x: factor_y = 0;
        default: factor_y = -2;
    endcase // casex (state_y)
    end

endmodule // deathstar_fsm

module halting_latch_fsm
    #(parameter RESET_LEVEL=1)
    (input wire       clk, reset,
     input wire       toggle, halt, resume,
     output wire [1:0] out);

    // states of interest
    localparam [1:0] S_ANY =       2'bxx;  // any state
    localparam [1:0] S_HIGH =      2'b11;  // high/on
    localparam [1:0] S_LOW =       2'b10;  // low/off
    localparam [1:0] S_HALT_X =    2'b0x;  // any halted state
    localparam [1:0] S_HALT_HIGH = 2'b01;  // halted from high
    localparam [1:0] S_HALT_LOW =  2'b00;  // halted from low
    localparam [1:0] S_RESET = {1'b1, RESET_LEVEL[0]};
```

```
    // defined transitions
    localparam [3:0] T_LATCH =  4'b0x00;   // no attempt to toggle
    localparam [3:0] T_TOGGLE = 4'b1x00;   // toggle
    localparam [3:0] T_RESUME = 4'bx100;   // resume from halt
    localparam [3:0] T_HALT =   4'bxx10;   // halt from any state
    localparam [3:0] T_ANY =    4'bxxx0;   // any non-reset
    localparam [3:0] T_RESET =  4'bxxx1;   // reset

    reg [1:0]         state = S_RESET;
    reg [1:0]         next_state = S_RESET;
    wire          toggle_pulse;

    l2p_fsm l2p(.clk(clk), .reset(reset),
        .level(toggle),
        .out(toggle_pulse));

    always @* begin
    casex({state, toggle_pulse, resume, halt, reset})
        {S_HIGH,      T_TOGGLE}: next_state = S_LOW;
        {S_LOW,       T_TOGGLE}: next_state = S_HIGH;
        {S_ANY,       T_LATCH}:  next_state = state;
        {S_HIGH,      T_HALT}:   next_state = S_HALT_HIGH;
        {S_LOW,       T_HALT}:   next_state = S_HALT_LOW;
        {S_HALT_HIGH, T_RESUME}: next_state = S_HIGH;
        {S_HALT_LOW,  T_RESUME}: next_state = S_LOW;
        {S_HALT_X,    T_ANY}:    next_state = state;
        {S_ANY,       T_RESET}:  next_state = S_RESET;
        default: begin
        next_state = S_RESET;
        $stop;
        end
    endcase
    end // always @ *

    always @(posedge clk)
    state <= next_state;

    assign out = state;

endmodule // halting_latch_fsm
```

10.3.11 Controls: l2p_fsm Module

```
`timescale 1ns / 1ps
`default_nettype none
/********************************************************************
 Engineer: David Mueller
 Module Name:    l2p_fsm
 Description
 -----------
 Level-to-pulse FSM.

 Captures the change in level of a signal as a 1-clock-width pulse.
```

```
 The pulse does not retrigger until either the level returns to its
 inactive state, or the reset is set HIGH.

 Inputs
 ------
 clk: clock
 reset: 1 causes reset
 level: level signal to monitor

 Outputs
 -------
 out: output level
**********************************************************************/
module l2p_fsm
    #(parameter ACTIVE_LEVEL='b1)
    (input wire  clk, reset,
     input wire  level,
     output wire out);

    localparam S_ANY = 1'bx;
    localparam S_ACTIVE = ACTIVE_LEVEL[0];
    localparam S_INACTIVE = ~ACTIVE_LEVEL[0];
    localparam S_RESET = S_INACTIVE;

    localparam T_ACTIVE =   1'b1;
    localparam T_INACTIVE = 1'b0;

    reg      state, next_state;
    wire     rectified;
    assign rectified = ~(level ^ ACTIVE_LEVEL[0]);

    always @* begin
    if (reset) begin
        next_state = S_RESET;
    end
    else begin
        casex(rectified)
        T_ACTIVE: next_state = S_ACTIVE;
        T_INACTIVE: next_state = S_INACTIVE;
        default: next_state = S_RESET;
        endcase // casex (rectified)
    end
    end // always @ *

    always @(posedge clk)
    state <= next_state;

    assign out = level & ~state;

endmodule
```