6.111 Final Project Writeup

Team 11: Acoustic Levitator

David Li (davidli), William Luo (wqcluo)

System Overview

Acoustic levitators use acoustic radiation pressure to levitate objects. We sought to construct an acoustic levitator that used ultrasonic transducers to levitate small objects controlled by the FPGA and a camera. Our final project involved developing a system that levitates a styrofoam bead at a height controlled by the height of a control ball. We implemented two modes, one where the control ball is a virtual ball shown on the screen and the height is controlled by buttons on the FPGA and one where the control ball is another styrofoam ball held by hand in real life. Any changes in the position of the control styrofoam ball (held by hand) are mirrored by changes in the height of the ball being levitated. The setup we envisioned is shown in Figure 1 below, and the final product is shown in Figure 2.

Since we wanted to use the height of a control ball to control the height of our levitated ball, we need a way to read out the heights of both balls. To do so, we use a camera to image the positions of both balls. As shown in figure 2, this is in a controlled setup with a black backdrop and controlled lighting from a set of blue LEDs, which lets us tune the environment and helps to reduce variability. The pixels outputted by the camera are converted from RGB to HSV, then HSV filtered on our FPGA. We use a streaming algorithm to calculate the ball positions once per frame, which we can then use to measure the difference in height between the two balls. This step also calculates the size of the bounding box on the display.

The existing phases of the output signals and the current positions are used in a feedback control module to control the final signal sent to a pair of arrayed ultrasonic transducers generated by the FPGA. The feedback control modules ensure the height of the levitated ball matches that of the control ball, or that it adjusts in response to button presses in virtual mode. We also display the camera input, a bounding box for the ball(s) we detect, and optionally shade in the pixels that pass the HSV filter (which can be turned on with a switch). This lets us see if the FPGA is filtering correctly.

Overall, the project was a success and the goals we achieved are listed below. Our video is here.

Goals

- Baseline:
  - Be able to drive an ultrasonic transducer using the FPGA to levitate a styrofoam ball.
  - Be able to detect styrofoam balls using HSV filtering on camera data.
  - Be able to display the camera input as well as the filtered positions of the styrofoam balls.
- Expected:
  - Unite the camera input control and the acoustic levitation hardware into a complete system where the height of the styrofoam ball being levitated is controlled by the height of a control styrofoam ball detected by the camera.
- Stretch:
  - Implement an alternative form of input control, using the switches on the FPGA development board or a potentiometer to set the position of the ball, with an animated ball displayed on the screen at the corresponding height.
  - Display bounding boxes with variable sizes depending on how close the balls are. The radius is computed from the total count of pixels passing the HSV filter.

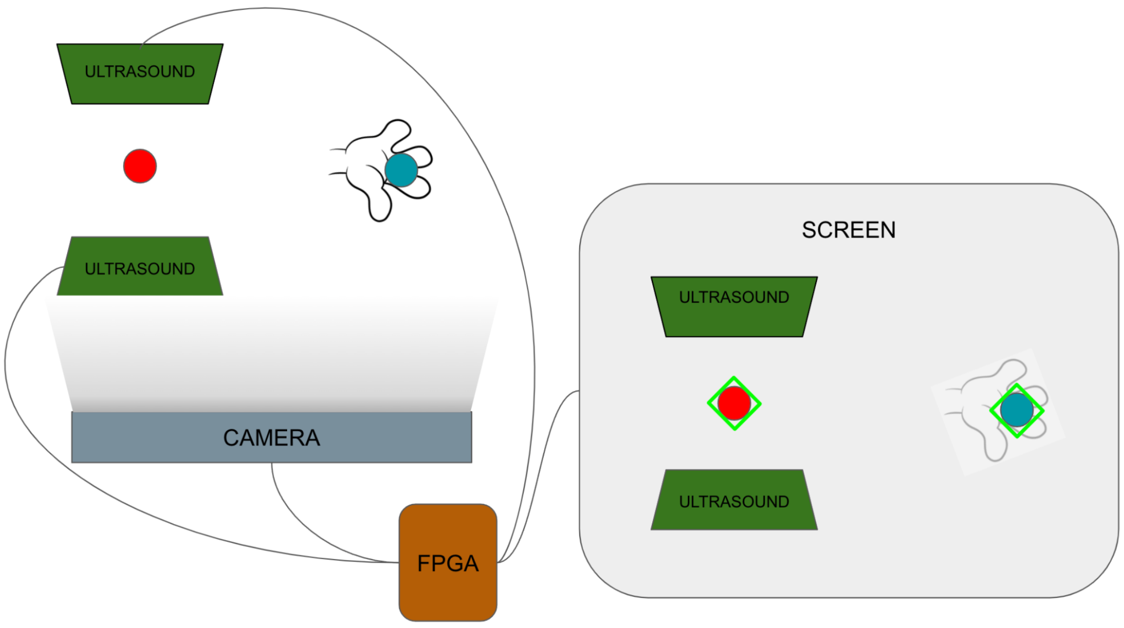

Figure 1: Idealized Setup

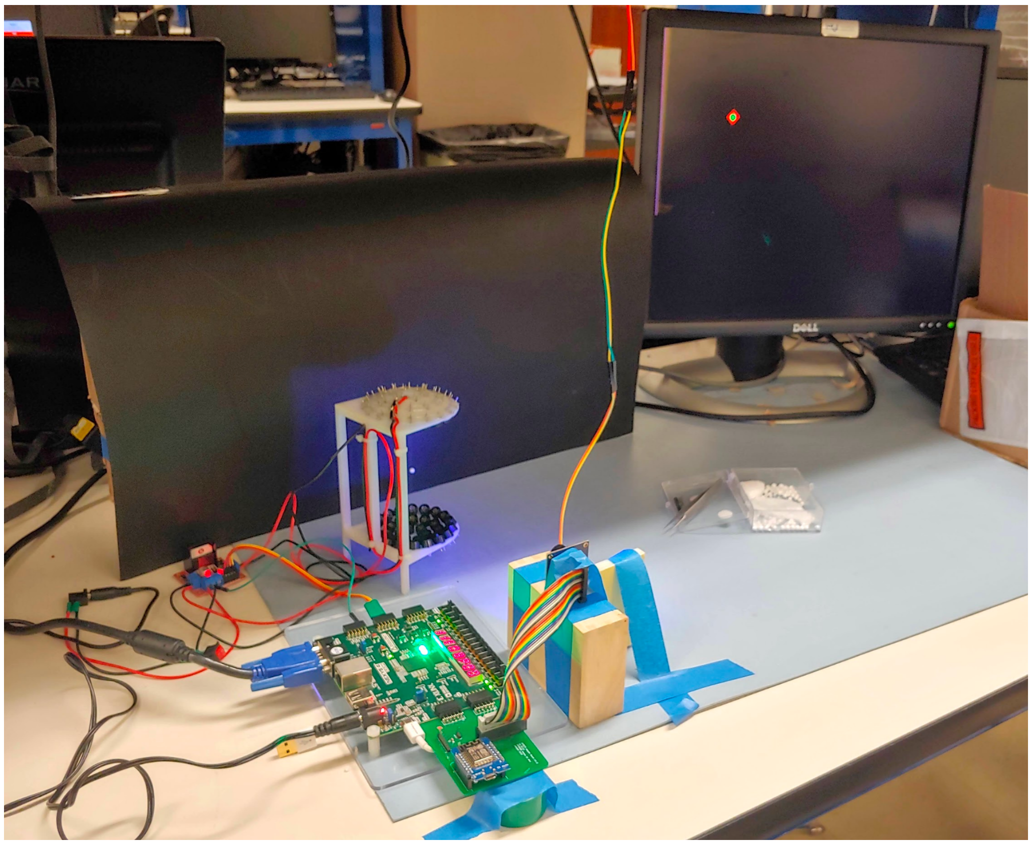

Figure 2: Final Setup

Block Diagram

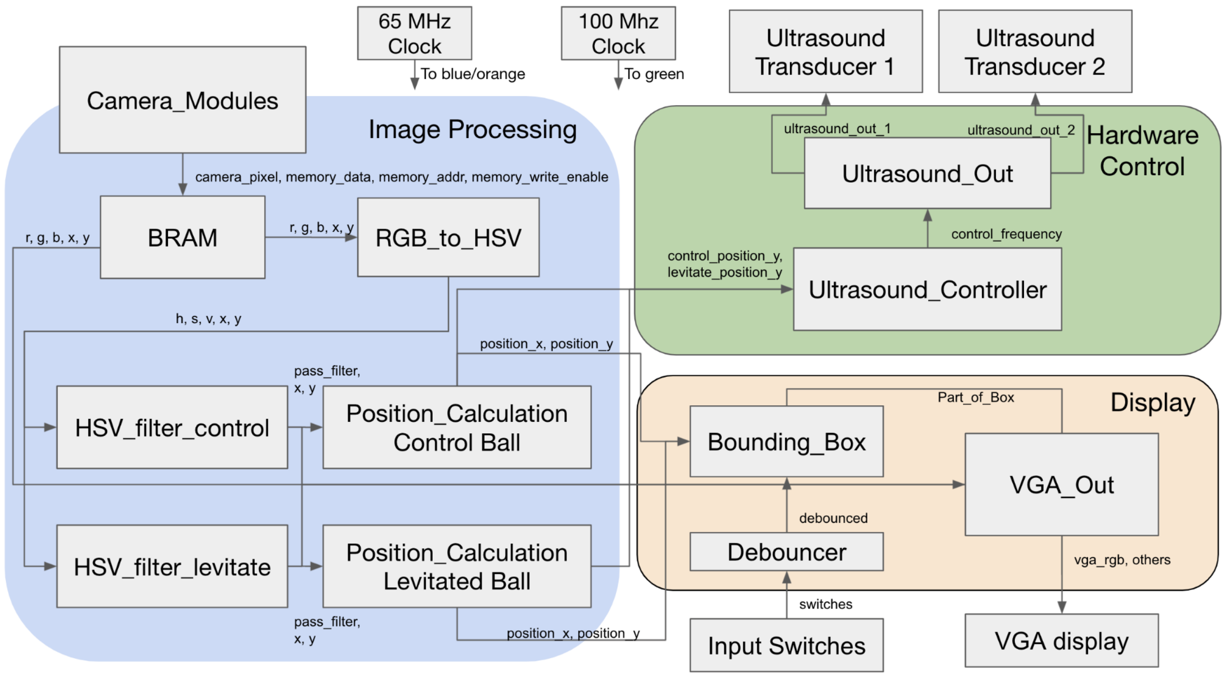

Figure 3: Conceptual Block Diagram of our System

Main Modules and High-Level Planning

As shown in the block diagram above, we split the project into three main modules: image procession (vision), display, and hardware control. We tried to develop the three modules separately before integrating them into the final system.

For the vision module, this meant developing all of the modules and running test benches to ensure accurate and rapid calculation. For the display module, we tried to display bounding boxes and check that the UI was working correctly. For hardware, we used an oscilloscope to make sure the signals we were generating were correct, and that they adjusted in response to controlled inputs. These involved creating an intermediate project/bitstream file that only allowed for HSV filtering and display of the resulting filtered image and another intermediate milestone of the ultrasound controller working without input from the camera.

After each was working, we sought to merge them. After moving the system between locations, we noticed that the filters and other parameters were very location-specific. We sought to make our set-up reproducible and adjustable by providing a standardized black backdrop, creating a project/bitstream that allowed for easy tuning of HSV filter parameters, and incorporating a DC-voltage supply-driven set of blue LEDs which we could tune to adjust the brightness. This greatly facilitated our final integration.

Image Processing (Computer Vision) Modules

These modules perform the image processing part of the project, converting a camera input from the hardware module into ball positions and indicating if a ball was found at a given pixel. The figure below demonstrates what the camera sees from the setup:

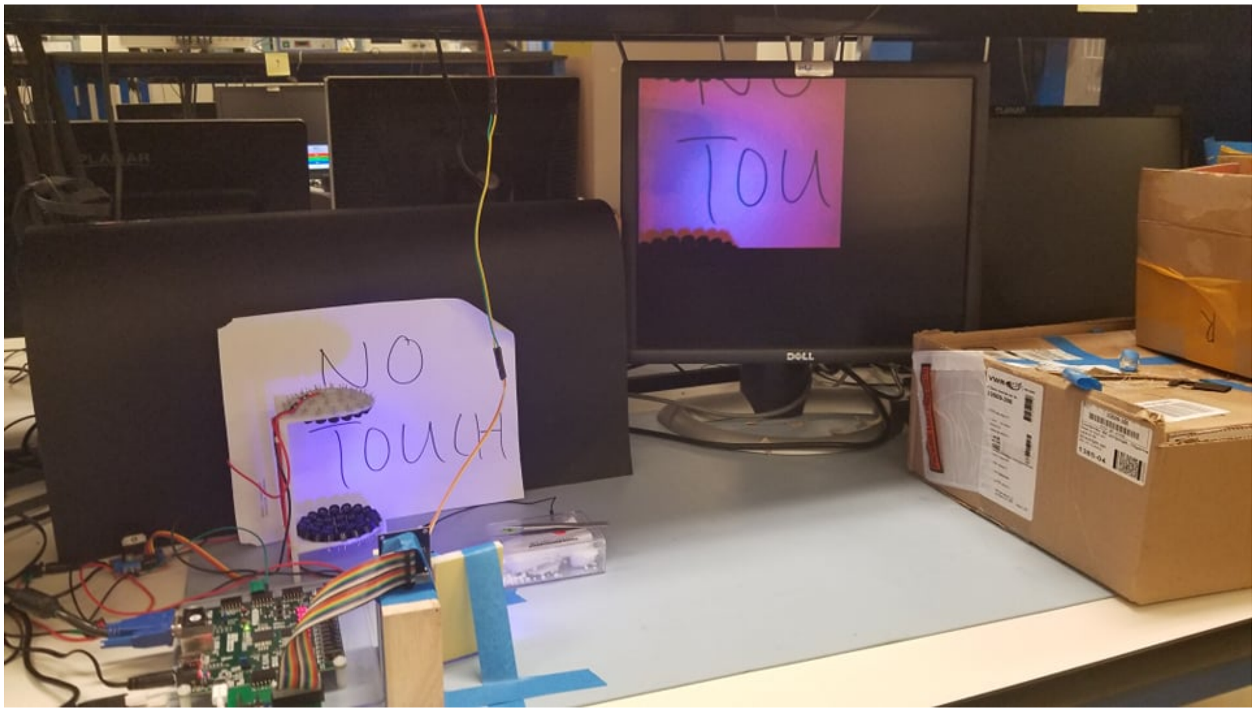

Figure 4: Camera’s perspective

After buffering the camera data through a BRAM to cross the clock domain, the vision module pulls out the data and begins to send it through a pipelined RGB to HSV module, which converts a (pixel_x, pixel_y, R, G, B) tuple into (pixel_x, pixel_y, H, S, V). The output of the pipelined module is fed into an HSV filter, which detects if the pixel location reflects a pixel of a ball. We only allow detection of the levitated ball in the left half of the screen and detection of the control ball in the right half of the screen.

All of this information is piped into a weighted_average module, which effectively computes the average x and y coordinates of the two balls. Further, we also compute the approximate radius of the balls by using the number of pixels that pass the HSV filter. Combining this information, we use the display modules (discussed later) to draw a diamond around the locations of the two detected balls, which are dynamically resized based on the radius of the balls.

The figure on the next page demonstrates the HSV filter picking up on the balls’ locations and using the display modules to draw a bounded box around them. We also provide some of the sub-modules of this main module and detailed information about each.

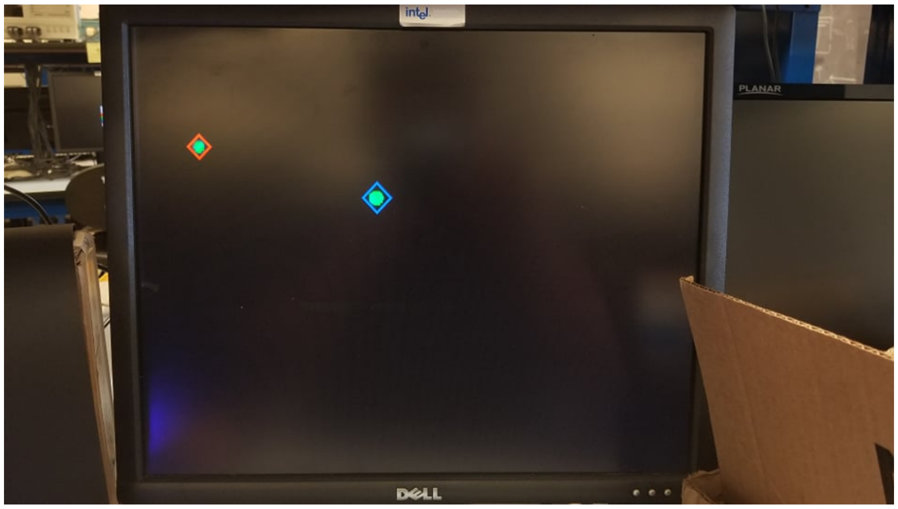
Figure 5: Vision and Display modules at work

rgb2hsv [William]

Inputs: (r, g, b, x, y) for a given pixel, clock
Outputs: (h, s, v, x, y) for a given pixel
Purpose: One of our most complicated modules. Performs the math for converting RGB values for a given pixel into HSV values. This was tested by feeding in RGB values in simulation and making sure that they are converted correctly to HSV. It computes this by performing the calculations stated in this article. We wanted to be able to perform these computations as quickly as possible (we achieved a 6 cycle delay), so we pipelined the computation as follows:

Cycle 1: buffer the inputs
Cycle 2: compute C_{max} and C_{min} (but not scaled by 1/255, since that doesn't actually change the results of any of the divisions – we separately compute V_{max})
Cycle 3: compute $\Delta = C_{max} - C_{min}$
Cycle 4: set the divisors/dividends for the division (note: for the hue calculation, we needed to deal with the wrap-around the 360 degree circle as an edge case)
Cycle 5: buffering the state variables to wait for division to complete
Cycle 6: compute the H, S, V values by adding the circle offset to the results

After we had finished implementing, we realized we could easily have got it o a lower number of clock cycles, but 6 sufficed. The rgb2hsv provided on the final project page is very slow and takes many clock cycles.

limited_divider_60, limited_divider_100 [William]

Inputs: clock, dividend, divisor,
Outputs: result
Purpose: Stores dividend/divisor scaled by the relevant factor of 60 or 100. limited_divider_60 was used to compute the hue and limited_divider_100 was used to compute the saturation and value. Basically a big look-up table.

hsv_filter [William]

Inputs: (h, s, v, x, y, for a given pixel), clock
Outputs: (h, s, v, x, y, pass_filter (binary)) for the same pixel
Parameters: lower and upper bounds on each of hue, saturation, and value
Purpose: This was a simple module that detects the pixels of the screen that pass our HSV filter for the color of our balls. We tuned the filters empirically using a top_level script that set the lower and upper bounds using a switch input.

weighted_average [William/David]

Inputs: clock, reset, start, done, pos, passed_hsv_filter
Outputs: ready_out, avg_out, avg_radius position_x, position_y, calculated
Purpose: A relatively high difficulty module that calculates a rolling sum of the number of pixels passing the HSV filter, a rolling sum of the positions along a given axis. The average is computed once per frame, computing center pixel positions for the ball and the radius of the ball from the total count. This can be cast as an FSM with four states: *resetting, waiting, summing,* and *dividing*, where the frame averages and radius are computed when ready_out is declared. Uses divider and sqrt_max_155.

divider [William]

Inputs: reset, clock, start, dividend, divisor
Output: ready, quotient, fractional
Purpose: Does division to obtain average x or average y, informs of completion with ready_out.

sqrt_max_155 [William/David]

Inputs: x
Output: result_out
Parameter: POS_BITS (desired bit size of output)
Purpose: Approximate square root lookup on x combinatorially to compute ball radius from a big chain of if/else statements, with a minimum of 10 and a max of 155 pixels. Ranges pre-computed in Python.

Display Modules

The display modules work closely with the vision modules to display the setup, as well as the positions of the balls. The display modules are comparatively simpler than the vision modules, as they are responsible mostly for just taking in the positions of the two balls, and then displaying bounding boxes plus or minus shading for pixels that pass the HSV filter, as seen above in figure 5.We also implemented a "virtual control" option, where the user can control the height of the levitated ball via displaying a virtual target ball on the XVGA and using the up/down switches on the FPGA to control the height, as shown below.

The display modules are also responsible for displaying information on the 7-segment display, such as ball positions or the current phases of the output. What exact information is being displayed is controlled by sw[15:14]. The RGB LEDs on the FPGA also flash in response to either the relative ball positions or to button presses, depending on the mode. A detailed description of what the displays and LEDs show is in the UI specification. We describe a few relevant sub-modules below.

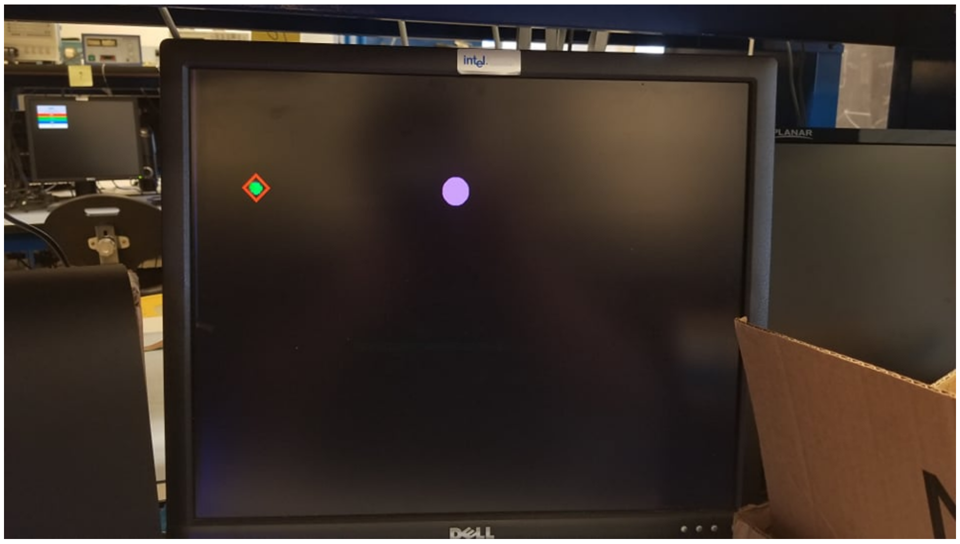
Figure 6: The levitated ball in virtual control mode with the virtual ball on the right

sync_delay [David]

Inputs: clock, reset, hsync, vsync, blank, ball_bounded, target_bounded, virtual_bounded, cam
Outputs: phsync, pvsync, pvblank, ball_bounded, target_bounded, virtual_bounded, cam
Purpose: Uses regs to delay the hsync, vsync, blank, ball_bounded, target_bounded, virtual_bounded, and camera inputs the right number of clock cycles so that everything lines up with passed_hsv_filter when displayed. hsync, vsync, and blank were shifted 13 cycles and the camera and bounding boxes were shifted 6 fewer cycles (7) as the average calculation takes 6 cycles. We counted the delays from our original design, but also tested it empirically using a piece of paper to make sure the edge of pixels lined up.

within_bounding_box [David/William]

Inputs: hcount, vcount, h_center, v_center, radius,
Outputs: bounded (Boolean), within_circle (Boolean)
Purpose: Declares whether a given pixel is part of a bounding box, which is pixels within a certain Manhattan distance of (h_center, v_center) for bounded and within a certain Euclidean distance radius of (h_center, v_center) for within_circle. We used a 5-pixel border to the bounding box and the interior radius was set once per frame by the vision module or constant at 10 pixels depending on the switches. We tested this using a standard testbench and also validated it on a real display.

abs [William]

Inputs: a, b
Outputs:c
Purpose: Computes the absolute value difference c of a and b combinatorially, to facilitate Manhatten distance calculation in within_bounding_box.

User Interface Specification
These are specs for how switches and display units are used.

- 3 general modes of operation:
    - Default/Bootup - no control input, so we can place a ball into place
    - Physical Ball Control - uses a ball on the right half of screen as control
    - Virtual Ball Control - uses up and down button as control, shows a virtual ball
- Switch assignment:
    - Sw[0] = Crosshairs, to make sure VGA is working.
    - Sw[1] = Default mode (camera only) (0), Control mode (1)
    - Sw[2] = Physical ball control (0), Virtual Ball Control (1), shows bounding boxes
    - Sw[3] = Show pixels passing filter, ball bounding box if [1], target box if [1] and [2]
    - Sw[4] = Turn resizable bounding boxes on (1) or off (0)
    - Sw[5] = Turns on controller (1 = on)
    - Sw[15:14] = Controls what is on 7-segment display
- 7-segment Display (as a function of Sw[15:14]):
        - 00 = Phase1 and phase2
        - 01 = Ball 1 x, y position
        - 10 = Ball 2 (target) x,y position
        - 11 = Ball 1 y and Ball 2 y positions
- RGB Leds:
    - Virtual Ball Control:
        - Up = Red,
        - Reset = Green,
        - Down = Blue
    - Physical Ball Control Mode:
        - Red: Levitated ball under control ball
        - Green: Levitated ball above control ball
        - Blue: Levitated ball within threshold (2 pixels) of control ball
    - Default:
        - No RGB

Hardware Control Modules

We constructed an open-source acoustic levitator known as the TinyLev, following the paper and the accompanying instructable. We use the same overall layout as the original system, controlling the ultrasonic transducers using an H-bridge powered by a 12V DC wall adaptor. The FPGA then drives the H-bridge through PMOD port ja for a peak-to-peak voltage of 24 volts for the ultrasonic transducers.

We powered the FPGA in wall mode using a 5V DC adaptor, as we noticed that fluctuations through our laptop power supply were influencing both the FPGA and the stability of the acoustic levitator. PMOD ja[0] and ja[1] constitute output 1, with ja[0] = ~ja[1] for the forward/backward command of the H-bridge. Similarly, ja[0] and ja[1] constitute output 1, with ja[0] = ~ja[1].

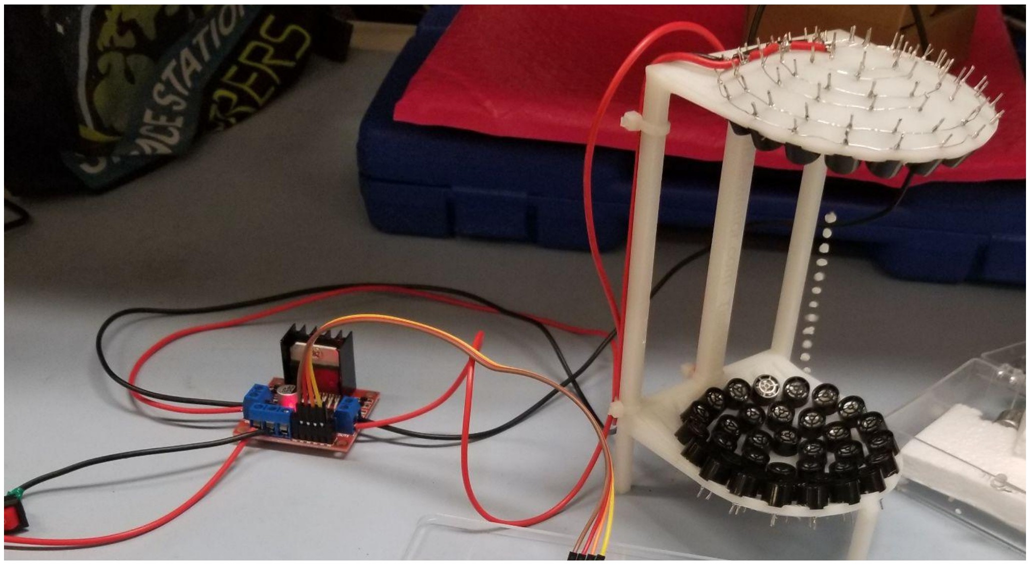
Figure 7: Our constructed acoustic levitation system

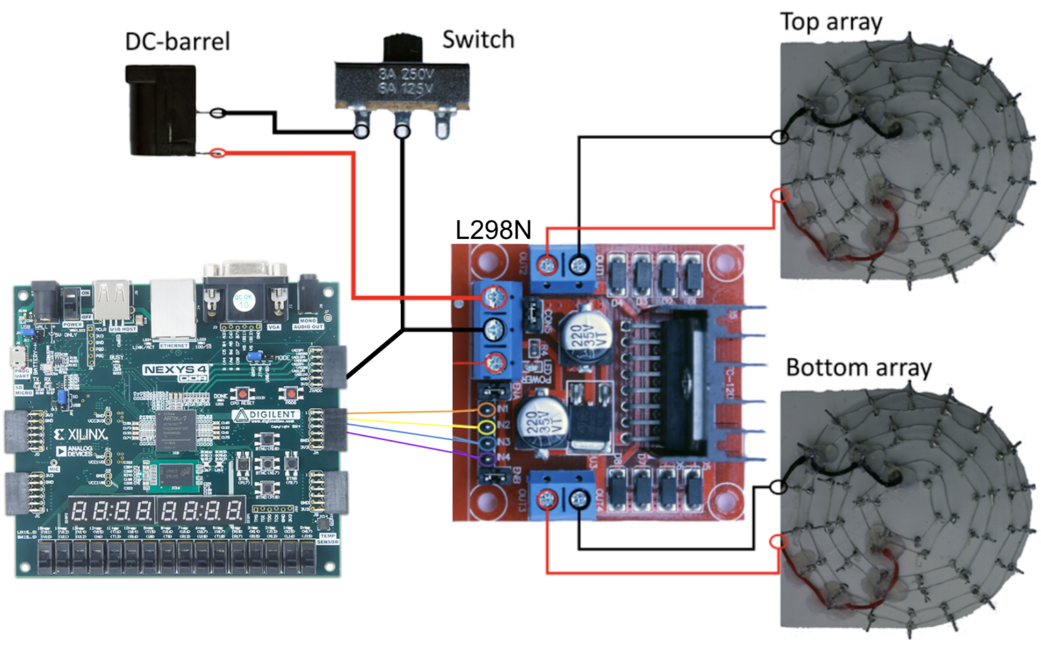

Figure 8: Wiring diagram of our acoustic levitation system

We experimented with various control schemes and tried to understand the upper limits at which the balls can move and stay stable. 40 kHz is a switch from 0 to 1 or 1 to 0 every 1250 clock cycles using the 100 mHz clock, and the upper limits of our button presses for an average styrofoam “slime” balls was a phase shift of 250 clock cycles. For smaller balls, we were able to use even larger phase shifts, up to 500 at a time. We tried a proportional control scheme, clipping the maximum input if the distance exceeded a threshold, and also a proportional-integral control scheme (the integral part was really not necessary). For stability purposes, the final control scheme implemented has three phases of control outputs, dependent on the position error. Here are a few of the sub-modules in this module.

rising_edge [David]

Inputs: debounced_buttons
Outputs: rising_edge_buttons
Purpose: Rising edge filters the buttons for a 1-clock cycle pulse.

camera_read and camera control [David/William]

Inputs: Clock
Outputs: processed_pixels, frame_done_out
Purpose: Most of the code comes from https://jodalyst.com/6111/camera_info/, but we swapped the OV7670 camera over to ports jc and jd. Returns camera data driven by a clock from the FPGA. Uses a BRAM to store 8-bit pixel values. We programmed the camera microcontroller with a modified set of weights to more accurately reflect real life when shown on a display.

ultrasound_out [David]

Inputs: reset, clock, phase_1, phase_2
Outputs: output1, output2
Purpose: Takes in phase_1, phase_2, and the clock and generates two output waveforms to drive the ultrasound transducers, one shifted by phase_1 and one shifted by phase_2 relative to a ground truth 40 kHz waveform. Directly tested on the oscilloscope first, and then with the levitator to make sure the output was what we desired.

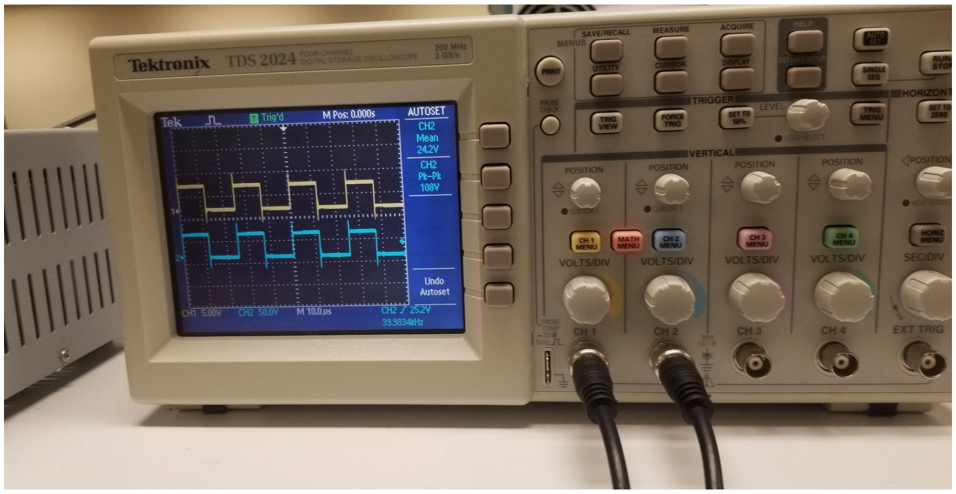

Figure 9: Our 40 kHz square waves on an oscilloscope

ultrasound_controller [David]

Inputs: reset_in, clock_in, up_in, down_in, start_in, sw_in, ball_y_in, target_y_in
Outputs: phase1_out, phase2_out
Purpose: This is the module that controls the phases to the position error of the balls or buttons. If the switches are in the right mode, the controller will respond to levitate_position_y - control_position_y or the buttons. This was tested empirically. Control schemes are detailed above.

Lessons Learned and Advice for Future Projects

Overall, we learned a lot from this project, how to integrate systems, how to work with multiple dividers and clock signals, and how to drive an acoustic levitator. If we had to restart, we would have included a motion path mode where the ball would follow a defined reprogrammable path. We would have also measured the frequency response of the system and also tried to extend the control scheme. The most helpful lesson for future groups may be to develop an easy HSV filter tuner to help deal with different lighting conditions at the start. HSV filters and the camera we used were very sensitive to the lighting conditions. The tunable blue LED from a DC power supply provided an easy way to ensure that everything was working. Also, connect everything to a common ground and make sure that you pipeline things correctly.

Code

top_level.sv

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

//////////////////////////////////////////////////////////////////////////////////
// David Li and William Luo
// 6.111 Fall 2021 Final Project Team 11
// Acoustic Levitator
// Top level file that integrates all modules
//////////////////////////////////////////////////////////////////////////////////

module top_level(
   //Inputs
   input wire clk_100mhz,
   input wire [15:0] sw,
   input wire btnc, btnu, btnd,
   input wire [7:0] jc, //pixel data from camera
   input wire [2:0] jd, //other data from camera (including clock return)
   //Outputs
   output logic [3:0] ja, //drives ultrasound transducers
   output logic jdclk, //clock FPGA drives the camera with,
   output logic [3:0] vga_r,
   output logic [3:0] vga_b,
   output logic [3:0] vga_g,
   output logic vga_hs,
   output logic vga_vs,
   output logic led16_b, led16_g, led16_r,
   output logic led17_b, led17_g, led17_r,
   output logic [15:0] led,
   output logic ca, cb, cc, cd, ce, cf, cg, dp,  // segments a-g, dp
   output logic [7:0] an    // Display location 0-7
   );

  /*
  Setup
  */

   // Create clocks for VGA and Ultrasound
   logic clk_65mhz;
   logic also_100mhz;
   clk_wiz_0 clkdivider(.clk_in1(clk_100mhz),.clk_out1(also_100mhz), .clk_out2(clk_65mhz));
```

```
// Handle all button inputs to the system by debouncing and rising edge
wire reset, up, rising_up, down, rising_down;
debounce db_reset(.reset_in(reset),.clock_in(also_100mhz),.noisy_in(btnc),.clean_out(reset));
debounce db_up(.reset_in(reset),.clock_in(also_100mhz),.noisy_in(btnu),.clean_out(up));
rising_edge rise_up(.reset_in(reset), .clock_in(also_100mhz), .clean_in(up), .rising_out(rising_up));
debounce db_down(.reset_in(reset),.clock_in(also_100mhz),.noisy_in(btnd),.clean_out(down));
rising_edge rise_down(.reset_in(reset), .clock_in(also_100mhz), .clean_in(down), .rising_out(rising_down));

// Instantiate things used for VGA output
wire [10:0] hcount;    // pixel on current line
wire [9:0] vcount;     // line number
wire hsync, vsync, blank;
reg [11:0] rgb;
xvga xvga1(.vclock_in(clk_65mhz),.hcount_out(hcount),.vcount_out(vcount),
         .hsync_out(hsync),.vsync_out(vsync),.blank_out(blank));

/*
Camera and BRAM
*/

logic xclk;
logic[1:0] xclk_count;

logic pclk_buff, pclk_in;
logic vsync_buff, vsync_in;
logic href_buff, href_in;
logic[7:0] pixel_buff, pixel_in;

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

logic [16:0] pixel_addr_in;

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

blk_mem_gen_0 camera_bram(.addra(pixel_addr_in),
                        .clka(pclk_in),
                        .dina(processed_pixels),
                        .wea(valid_pixel),
                        .addrb(pixel_addr_out),
                        .clkb(clk_65mhz),
                        .doutb(frame_buff_out));

always_ff @(posedge pclk_in)begin
    if (frame_done_out)begin
        pixel_addr_in <= 17'b0;
    end else if (valid_pixel)begin
        pixel_addr_in <= pixel_addr_in +1;
    end
end

always_ff @(posedge clk_65mhz) begin
    pclk_buff <= jd[0];
    vsync_buff <= jd[1];
    href_buff <= jd[2];
    pixel_buff <= jc;
    pclk_in <= pclk_buff;
    vsync_in <= vsync_buff;
    href_in <= href_buff;
    pixel_in <= pixel_buff;
    old_output_pixels <= output_pixels;
    xclk_count <= xclk_count + 2'b01;
    processed_pixels = {output_pixels[15:12],output_pixels[10:7],output_pixels[4:1]};
end

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

logic [16:0] pixel_addr_out;
assign cam = ((hcount<640) &&  (vcount<480)) ? frame_buff_out:12'h000;
assign pixel_addr_out = ((hcount>>1)+(vcount>>1)*32'd320);

/*
Computer Vision Pipeline
*/

localparam ZEROS = 4'b0000;
logic [8:0] hue_rgb, hue_filter;
logic [7:0] sat_rgb, val_rgb, sat_filter, val_filter;
logic [10:0] hpos_rgb, hpos_filter;
logic [9:0] vpos_rgb, vpos_filter;
logic passed_hsv_filter;

rgb2hsv rgbconv (
    .clock_in(clk_65mhz),
    .r_in({rgb[11:8], ZEROS}),
    .g_in({rgb[7:4], ZEROS}),
    .b_in({rgb[3:0], ZEROS}),
    .hor_pos_in(hcount),
    .ver_pos_in(vcount),
    .hue_out(hue_rgb),
    .sat_out(sat_rgb),
    .val_out(val_rgb),
    .hor_pos_out(hpos_rgb),
    .ver_pos_out(vpos_rgb));

localparam H_LO = 9'd160;// H used to be 200 to 260 for messenger filter
localparam H_HI = 9'd360;
localparam S_LO = 8'd30;
localparam S_HI = 8'd100;
localparam V_LO = 8'd30;
localparam V_HI = 8'd100;

hsv_filter #(.H_LO(H_LO), .H_HI(H_HI),
    .S_LO(S_LO), .S_HI(S_HI),
    .V_LO(V_LO), .V_HI(V_HI)) hsvfilter
    (
    .clock_in(clk_65mhz),
    .h_in(hue_rgb), .s_in(sat_rgb), .v_in(val_rgb),
    .hor_pos_in(hpos_rgb),
    .ver_pos_in(vpos_rgb),
    .h_out(hue_filter), .s_out(sat_filter), .v_out(val_filter),
    .hor_pos_out(hpos_filter), .ver_pos_out(vpos_filter),
    .passed_hsv_filter_out(passed_hsv_filter)
);

logic weighted_average_started, weighted_average_done, weighted_average_ready_h, weighted_average_ready_v;
logic [10:0] ball_h_avg_pos, target_h_avg_pos, ball_avg_radius, target_avg_radius;
logic [9:0] ball_v_avg_pos, target_v_avg_pos;

assign weighted_average_done = (hcount == 0) && (vcount == 0);

logic passed_hsv_filter_and_in_bounds, target_passed_hsv_and_in_bounds;
assign passed_hsv_filter_and_in_bounds = passed_hsv_filter && (hcount>=20 && hcount<=319) && (vcount>=50 && vcount<=400);
assign target_passed_hsv_and_in_bounds = passed_hsv_filter && (hcount>=320 && hcount<=639) && (vcount>=50 && vcount<=400);


// for the levitated ball:
weighted_average #(.POS_BITS(11)) wa_h (
    .clock_in(clk_65mhz), .reset_in(reset),
    .start_in(weighted_average_started), .done_in(weighted_average_done),
    .pos_in(hpos_filter),        // index on the horizontal axis
    .passed_hsv_filter_in(passed_hsv_filter_and_in_bounds),          // whether or not this pixel passed the filter
    .ready_out(weighted_average_ready_h),
    .avg_out(ball_h_avg_pos),      // the weighted average
    .avg_radius(ball_avg_radius)
);

weighted_average #(.POS_BITS(10)) wa_v (
    .clock_in(clk_65mhz), .reset_in(reset),
    .start_in(weighted_average_started), .done_in(weighted_average_done),
    .pos_in(vpos_filter),        // index on the horizontal axis
    .passed_hsv_filter_in(passed_hsv_filter_and_in_bounds),          // whether or not this pixel passed the filter
    .ready_out(weighted_average_ready_v),
    .avg_out(ball_v_avg_pos),      // the weighted average
    .avg_radius()
);

// for the target ball:
weighted_average #(.POS_BITS(11)) target_wa_h (
    .clock_in(clk_65mhz), .reset_in(reset),
    .start_in(weighted_average_started), .done_in(weighted_average_done),
    .pos_in(hpos_filter),        // index on the horizontal axis
    .passed_hsv_filter_in(target_passed_hsv_and_in_bounds),          // whether or not this pixel passed the filter
    .ready_out(), // william comment: we don't need this output because it should be the same for both of the balls!
    .avg_out(target_h_avg_pos),      // the weighted average
    .avg_radius(target_avg_radius)
);

weighted_average #(.POS_BITS(10)) target_wa_v (
    .clock_in(clk_65mhz), .reset_in(reset),
    .start_in(weighted_average_started), .done_in(weighted_average_done),
    .pos_in(vpos_filter),        // index on the horizontal axis
    .passed_hsv_filter_in(target_passed_hsv_and_in_bounds),          // whether or not this pixel passed the filter
    .ready_out(), // william comment: we don't need this output because it should be the same for both of the balls!
    .avg_out(target_v_avg_pos),      // the weighted average
    .avg_radius()
);

/*
Outputs
```

```
*/

// Ultrasound Stuff
logic rising_average_started;
rising_edge rise_average(.reset_in(reset), .clock_in(also_100mhz), .clean_in(weighted_average_started), .rising_out(rising_average_started));

logic [10:0] phase1, phase2;
ultrasound_controller main_control(.reset_in(reset), .clock_in(also_100mhz), .sw_in(sw),
                                   .up_in(rising_up), .down_in(rising_down), .start_in(rising_average_started),
                                   .target_y_in(target_v_avg_pos_wbb), .ball_y_in(ball_v_avg_pos_wbb),
                                   .phase1_out(phase1), .phase2_out(phase2));
ultrasound_out (.reset_in(reset), .clock_in(also_100mhz), .phase1(phase1), .phase2(phase2), .output1(ja[2]), .output2(ja[0]));
assign ja[1] = ~ja[0];
assign ja[3] = ~ja[2];


// Logic for computing what is being shown on VGA

logic ball_bounded, target_bounded, virtual_bounded, within_virtual_circle;
logic [10:0] ball_h_avg_pos_wbb, target_h_avg_pos_wbb;
logic [9:0] ball_v_avg_pos_wbb, target_v_avg_pos_wbb;
logic [10:0] ball_avg_radius_wbb, target_avg_radius_wbb;

always_ff @(posedge clk_65mhz) begin
  if (reset || weighted_average_ready_h) begin
      weighted_average_started <= 1;
      ball_h_avg_pos_wbb <= ball_h_avg_pos;
      ball_v_avg_pos_wbb <= ball_v_avg_pos;
      target_h_avg_pos_wbb <= target_h_avg_pos;
      target_v_avg_pos_wbb <= target_v_avg_pos;
      if (sw[4] == 1) begin
          ball_avg_radius_wbb <= 11'd15;//(ball_avg_radius>>2 + ball_avg_radius >> 1 + ball_avg_radius_wbb>>2); //IIR Smoothing
          target_avg_radius_wbb <= 11'd15; //(target_avg_radius>>2 + target_avg_radius >> 1 + target_avg_radius_wbb>>2);
      end else begin
          ball_avg_radius_wbb <= ball_avg_radius;//(ball_avg_radius>>2 + ball_avg_radius >> 1 + ball_avg_radius_wbb>>2); //IIR
Smoothing
          target_avg_radius_wbb <= target_avg_radius; //(target_avg_radius>>2 + target_avg_radius >> 1 + target_avg_radius_wbb>>2);
      end
  end else if (weighted_average_started) begin
      weighted_average_started <= 0;
  end
end

within_bounding_box wbb (.hcount_in(hcount), .vcount_in(vcount),
    .h_center_in(ball_h_avg_pos_wbb), .v_center_in(ball_v_avg_pos_wbb),
    .radius_in(ball_avg_radius_wbb), .bounded_out(ball_bounded),
    .within_circle_out());

within_bounding_box target_wbb (.hcount_in(hcount), .vcount_in(vcount),
    .h_center_in(target_h_avg_pos_wbb), .v_center_in(target_v_avg_pos_wbb),
    .radius_in(target_avg_radius_wbb), .bounded_out(target_bounded),
    .within_circle_out());

// for the virtual ball
within_bounding_box virtual_wbb (.hcount_in(hcount), .vcount_in(vcount),
    .h_center_in(11'd480), .v_center_in(ball_v_avg_pos_wbb),
    .radius_in(11'd35), .bounded_out(virtual_bounded),
    .within_circle_out(within_virtual_circle));

// VGA Handling
logic phsync,pvsync,pblank,ball_bound,target_bound,virtual_bound;
logic [11:0] pcam;
sync_delay main_delay(.vclock_in(clk_65mhz),.reset_in(reset),
            .hsync_in(hsync),.vsync_in(vsync),.blank_in(blank), .cam_in(cam),
            .phsync_out(phsync),.pvsync_out(pvsync),.pblank_out(pblank), .cam_out(pcam),
            .ball_bounded_in(ball_bounded), .target_bounded_in(target_bounded), .virtual_bounded_in(within_virtual_circle),
            .ball_bounded_out(ball_bound), .target_bounded_out(target_bound), .virtual_bounded_out(virtual_bound)
            );

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

logic [11:0] rgb2;
logic b, hs, vs;
always_ff @(posedge clk_65mhz) begin
  if (sw[0] == 1'b1) begin
     // 1 pixel outline of visible area (white)
     hs <= hsync;
     vs <= vsync;
     b <= blank;
     rgb <= {12{border}};
     rgb2 <= {12{border}};
  end else begin
     hs <= phsync;
     vs <= pvsync;
     b <= pblank;
     rgb <= cam;
     if (sw[1] == 1'b1 && sw[2] == 1'b0 && sw[3] == 1'b0) begin
        // physical control mode
        rgb2 <= ball_bound ? 12'hF00 : (target_bound? 12'h00F : pcam);
     end else if (sw[1] == 1'b1 && sw[2] == 1'b1 && sw[3] == 1'b0) begin
        // virtual control mode
        rgb2 <= ball_bound ? 12'hF00 : (virtual_bound? 12'h80F : pcam);
     end else if (sw[1] == 1'b1 && sw[2] == 1'b0 && sw[3] == 1'b1) begin
        // show pixels passing HSV filter mode, physical control
        rgb2 <= passed_hsv_filter ? 12'h0F0 : (ball_bound ? 12'hF00 : (target_bound? 12'h00F : pcam));
     end else if (sw[1] == 1'b1 && sw[2] == 1'b1 && sw[3] == 1'b1) begin
        // show pixels passing HSV filter mode, virtual control
        rgb2 <= passed_hsv_filter ? 12'h0F0 : (ball_bound ? 12'hF00 : (virtual_bound? 12'h80F: pcam));
     end else begin
        rgb2 <= pcam;
     end
  end
end

always_comb begin
 vga_r = ~b ? rgb2[11:8]: 0;
 vga_g = ~b ? rgb2[7:4] : 0;
 vga_b = ~b ? rgb2[3:0] : 0;
 vga_hs = ~hs;
 vga_vs = ~vs;
end

// Code for the 7-segment display, LEDs, and RGB LEDs.
logic [31:0] data;      //  instantiate 7-segment display; display (8) 4-bit hex
logic [6:0] segments;
assign {cg, cf, ce, cd, cc, cb, ca} = segments[6:0];
display_8hex display(.clk_in(clk_65mhz),.data_in(data), .seg_out(segments), .strobe_out(an));
assign  dp = 1'b1;  // turn off the period
```

```
  always_comb begin
    if (sw[15:14] == 2'b00) begin
      // controller phases
      data[31:16] = phase1;
      data[15:0] = phase2;
    end else if (sw[15:14] == 2'b01) begin
      // controlled ball position
      data[31:16] = ball_h_avg_pos_wbb; // 4 hex are horizontal
      data[15:0] = ball_v_avg_pos_wbb; // 4 hex are vertical
    end else if (sw[15:14] == 2'b01) begin
      // target ball position
      data[31:16] = target_h_avg_pos_wbb; // 4 hex are horizontal
      data[15:0] = target_v_avg_pos_wbb; // 4 hex are vertical
    end else begin
      // target and controlled ball vertical position
      data[31:16] = ball_v_avg_pos_wbb; // 4 hex are controlled vertical
      data[15:0] = target_v_avg_pos_wbb; // 4 hex are target vertical
    end
  end

  always_comb begin
    // Code for led16/17rgb
    // In physical control mode reflects feedback control status
    // In virtual control mode reflects pushbutton status
    led16_r = (sw[1] == 1'b1 && sw[2] == 1'b0)?(target_v_avg_pos_wbb+2<ball_v_avg_pos_wbb):((sw[1] == 1'b1 && sw[2] ==
1'b1)?up:1'b0);
    led17_r = (sw[1] == 1'b1 && sw[2] == 1'b0)?(target_v_avg_pos_wbb+2<ball_v_avg_pos_wbb):((sw[1] == 1'b1 && sw[2] ==
1'b1)?up:1'b0);
    led16_g = (sw[1] == 1'b1 && sw[2] == 1'b0)?(target_v_avg_pos_wbb>ball_v_avg_pos_wbb+2):((sw[1] == 1'b1 && sw[2] ==
1'b1)?reset:1'b0);
    led17_g = (sw[1] == 1'b1 && sw[2] == 1'b0)?(target_v_avg_pos_wbb>ball_v_avg_pos_wbb+2):((sw[1] == 1'b1 && sw[2] ==
1'b1)?reset:1'b0);
    led16_b = (sw[1] == 1'b1 && sw[2] ==
1'b0)?~((target_v_avg_pos_wbb+2<ball_v_avg_pos_wbb)|(target_v_avg_pos_wbb>ball_v_avg_pos_wbb+2)):((sw[1] == 1'b1 && sw[2] ==
1'b1)?down:1'b0);
    led17_b = (sw[1] == 1'b1 && sw[2] ==
1'b0)?~((target_v_avg_pos_wbb+2<ball_v_avg_pos_wbb)|(target_v_avg_pos_wbb>ball_v_avg_pos_wbb+2)):((sw[1] == 1'b1 && sw[2] ==
1'b1)?down:1'b0);
  end

  // LEDs correspond to switch inputs
  assign led = sw;
endmodule
`default_nettype wire
```

vision_helpers.sv

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

// Computes if the pixel passes the filter
module hsv_filter
  #(
    parameter H_LO = 0,
    parameter H_HI = 50,
    parameter S_LO = 0,
    parameter S_HI = 20,
    parameter V_LO = 0,
    parameter V_HI = 20
  ) (
    input wire clock_in,
    input wire [8:0] h_in,
    input wire [7:0] s_in, v_in,
    input wire [10:0] hor_pos_in,
    input wire [9:0] ver_pos_in,
    output logic [8:0] h_out,
    output logic [7:0] s_out, v_out,
    output logic [10:0] hor_pos_out,
    output logic [9:0] ver_pos_out,
    output logic passed_hsv_filter_out
  );

  always @(posedge clock_in) begin
    h_out <= h_in;
    s_out <= s_in;
    v_out <= v_in;
    hor_pos_out <= hor_pos_in;
    ver_pos_out <= ver_pos_in;

      if (h_in >= H_LO && h_in <= H_HI && s_in >= S_LO && s_in <= S_HI && v_in >= V_LO && v_in <= V_HI) begin
        passed_hsv_filter_out <= 1;
      end else begin
        passed_hsv_filter_out <=0;
      end
  end

endmodule

module rgb2hsv (
  input wire clock_in,
  input wire [7:0] r_in,
  input wire [7:0] g_in,
  input wire [7:0] b_in,
  input wire [10:0] hor_pos_in,
  input wire [9:0] ver_pos_in,
  output logic [8:0] hue_out,
  output logic [7:0] sat_out,
  output logic [7:0] val_out,
  output logic [10:0] hor_pos_out,
  output logic [9:0] ver_pos_out
  );

  logic [7:0] hue_dividend, hue_divisor;
  logic [13:0] hue_result;

  logic [7:0] sat_dividend, sat_divisor;
  logic [14:0] sat_result;

  logic [7:0] val_dividend;
  logic [14:0] val_result;

  limited_divider_60 hue_calculation  (.clock_in(clock_in), .dividend_in(hue_dividend), .divisor_in(hue_divisor), .result_out(hue_result));
  limited_divider_100 sat_calculation (.clock_in(clock_in), .dividend_in(sat_dividend), .divisor_in(sat_divisor), .result_out(sat_result));
  limited_divider_100 val_calculation (.clock_in(clock_in), .dividend_in(val_dividend), .divisor_in(8'd255), .result_out(val_result));

  logic [7:0] cmin, cmax, delta;

  logic [7:0] my_r_delay1, my_g_delay1, my_b_delay1;
  logic [7:0] my_r_delay2, my_g_delay2, my_b_delay2;
  logic [7:0] my_r_delay3, my_g_delay3, my_b_delay3;

  logic [7:0] cmax_delay3;
  logic negate_output, negate_output_delay_5;
  logic [8:0] hue_to_add, hue_to_add_delay_5;

  parameter DELAY = 6;

  logic [10:0] hor_pos_delay [DELAY-2 : 0];
  logic [9:0] ver_pos_delay [DELAY-2 : 0];

  genvar i;
  generate
    for (i = 1; i <= DELAY - 2; i = i + 1) begin
      always @(posedge clock_in) begin
        hor_pos_delay[i] <= hor_pos_delay[i-1];
        ver_pos_delay[i] <= ver_pos_delay[i-1];
      end
    end
  endgenerate

  always @(posedge clock_in) begin

    // clock cycle 1:
    {my_r_delay1, my_g_delay1, my_b_delay1} <= {r_in, g_in, b_in};
    hor_pos_delay[0] <= hor_pos_in;
    ver_pos_delay[0] <= ver_pos_in;

    // clock cycle 2 (compute the cmax/cmin):
    {my_r_delay2, my_g_delay2, my_b_delay2} <= {my_r_delay1, my_g_delay1, my_b_delay1};

    if ((my_r_delay1 >= my_g_delay1) && (my_r_delay1 >= my_b_delay1)) begin // Cmax == R
      cmax <= my_r_delay1;
    end else if ((my_g_delay1 >= my_r_delay1) && (my_g_delay1 >= my_b_delay1)) begin // Cmax == G
      cmax <= my_g_delay1;
    end else begin  // Cmax == B
      cmax <= my_b_delay1;
    end

    if ((my_r_delay1 <= my_g_delay1) && (my_r_delay1 <= my_b_delay1)) begin // Cmin == R
      cmin <= my_r_delay1;
    end else if ((my_g_delay1 <= my_r_delay1) && (my_g_delay1 <= my_b_delay1)) begin // Cmin == G
      cmin <= my_g_delay1;
    end else begin // Cmin == b
      cmin <= my_b_delay1;
    end

    // clock cycle 3 (compute delta):
```

```
{my_r_delay3, my_g_delay3, my_b_delay3} <= {my_r_delay2, my_g_delay2, my_b_delay2};
delta <= cmax - cmin;
cmax_delay3 <= cmax;

// clock cycle 4 (divisions, set the numerators/denominators):

// compute 100 * (delta / cmax)
sat_dividend <= delta;
sat_divisor <= cmax_delay3;

// compute 100 * cmax
val_dividend <= cmax_delay3;

// hue computation
hue_divisor <= delta; // set divisor

// set the dividend
if (cmax_delay3 == my_r_delay3) begin              // if cmax equal r then compute h = (60 * ((g - b) / delta) + 360) % 360

   if (my_g_delay3 >= my_b_delay3) begin
      hue_dividend <= my_g_delay3 - my_b_delay3;  // compute g - b
      negate_output <= 0;
      hue_to_add <= 0;                            // 0 because it will be on the positive side of the circle
   end else begin
      hue_dividend <= my_b_delay3 - my_g_delay3;  // compute b - g (will be negative though, so it will go around the circle)
      negate_output <= 1;
      hue_to_add <= 360;
   end

end else if (cmax_delay3 == my_g_delay3) begin      // if cmax equal g then compute h = (60 * ((b - r) / delta) + 120) % 360

   hue_to_add <= 120;

   if (my_b_delay3 >= my_r_delay3) begin
      hue_dividend <= my_b_delay3 - my_r_delay3;  // compute b - r
      negate_output <= 0;
   end else begin
      hue_dividend <= my_r_delay3 - my_b_delay3;  // compute r - b and negate it at the end
      negate_output <= 1;
   end

end else begin          // if cmax equals b then compute h = (60 * ((r - g) / diff) + 240) % 360

   hue_to_add <= 240;

   if (my_r_delay3 >= my_r_delay3) begin
      hue_dividend <= my_r_delay3 - my_g_delay3;  // compute r - g
      negate_output <= 0;
   end else begin
      hue_dividend <= my_g_delay3 - my_r_delay3;  // compute g - r and negate it at the end
      negate_output <= 1;
   end

end

// clock cycle 5 (wait for division)

hue_to_add_delay_5 <= hue_to_add;
negate_output_delay_5 <= negate_output;

// clock cycle 6 (compute the outputs)
```

```
        sat_out <= sat_result;
        val_out <= val_result;

        if (negate_output_delay_5) begin
            hue_out <= hue_to_add_delay_5 - hue_result;
        end else begin
            hue_out <= hue_to_add_delay_5 + hue_result;
        end

        hor_pos_out <= hor_pos_delay[DELAY - 2];
        ver_pos_out <= ver_pos_delay[DELAY - 2];

    end

endmodule


// Computes the approximate value of 60 * (dividend_in / divisor_in)
module limited_divider_60 (
    input wire clock_in,
    input wire [7:0] dividend_in,
    input wire [7:0] divisor_in,
    output logic [13:0] result_out
    );

    always @(posedge clock_in) begin
        case (divisor_in)
            8'd1: result_out <= ((61440 * dividend_in) >> 10);
            8'd2: result_out <= ((30720 * dividend_in) >> 10);
            8'd3: result_out <= ((20480 * dividend_in) >> 10);
            8'd4: result_out <= ((15360 * dividend_in) >> 10);
            8'd5: result_out <= ((12288 * dividend_in) >> 10);
            8'd6: result_out <= ((10240 * dividend_in) >> 10);
            8'd7: result_out <= ((8777 * dividend_in) >> 10);
            8'd8: result_out <= ((7680 * dividend_in) >> 10);
            8'd9: result_out <= ((6827 * dividend_in) >> 10);
            8'd10: result_out <= ((6144 * dividend_in) >> 10);
            8'd11: result_out <= ((5585 * dividend_in) >> 10);
            8'd12: result_out <= ((5120 * dividend_in) >> 10);
            8'd13: result_out <= ((4726 * dividend_in) >> 10);
            8'd14: result_out <= ((4389 * dividend_in) >> 10);
            8'd15: result_out <= ((4096 * dividend_in) >> 10);
            8'd16: result_out <= ((3840 * dividend_in) >> 10);
            8'd17: result_out <= ((3614 * dividend_in) >> 10);
            8'd18: result_out <= ((3413 * dividend_in) >> 10);
            8'd19: result_out <= ((3234 * dividend_in) >> 10);
            8'd20: result_out <= ((3072 * dividend_in) >> 10);
            8'd21: result_out <= ((2926 * dividend_in) >> 10);
            8'd22: result_out <= ((2793 * dividend_in) >> 10);
            8'd23: result_out <= ((2671 * dividend_in) >> 10);
            8'd24: result_out <= ((2560 * dividend_in) >> 10);
            8'd25: result_out <= ((2458 * dividend_in) >> 10);
            8'd26: result_out <= ((2363 * dividend_in) >> 10);
            8'd27: result_out <= ((2276 * dividend_in) >> 10);
            8'd28: result_out <= ((2194 * dividend_in) >> 10);
            8'd29: result_out <= ((2119 * dividend_in) >> 10);
            8'd30: result_out <= ((2048 * dividend_in) >> 10);
            8'd31: result_out <= ((1982 * dividend_in) >> 10);
            8'd32: result_out <= ((1920 * dividend_in) >> 10);
            8'd33: result_out <= ((1862 * dividend_in) >> 10);
```

```
8'd34: result_out <= ((1807 * dividend_in) >> 10);
8'd35: result_out <= ((1755 * dividend_in) >> 10);
8'd36: result_out <= ((1707 * dividend_in) >> 10);
8'd37: result_out <= ((1661 * dividend_in) >> 10);
8'd38: result_out <= ((1617 * dividend_in) >> 10);
8'd39: result_out <= ((1575 * dividend_in) >> 10);
8'd40: result_out <= ((1536 * dividend_in) >> 10);
8'd41: result_out <= ((1499 * dividend_in) >> 10);
8'd42: result_out <= ((1463 * dividend_in) >> 10);
8'd43: result_out <= ((1429 * dividend_in) >> 10);
8'd44: result_out <= ((1396 * dividend_in) >> 10);
8'd45: result_out <= ((1365 * dividend_in) >> 10);
8'd46: result_out <= ((1336 * dividend_in) >> 10);
8'd47: result_out <= ((1307 * dividend_in) >> 10);
8'd48: result_out <= ((1280 * dividend_in) >> 10);
8'd49: result_out <= ((1254 * dividend_in) >> 10);
8'd50: result_out <= ((1229 * dividend_in) >> 10);
8'd51: result_out <= ((1205 * dividend_in) >> 10);
8'd52: result_out <= ((1182 * dividend_in) >> 10);
8'd53: result_out <= ((1159 * dividend_in) >> 10);
8'd54: result_out <= ((1138 * dividend_in) >> 10);
8'd55: result_out <= ((1117 * dividend_in) >> 10);
8'd56: result_out <= ((1097 * dividend_in) >> 10);
8'd57: result_out <= ((1078 * dividend_in) >> 10);
8'd58: result_out <= ((1059 * dividend_in) >> 10);
8'd59: result_out <= ((1041 * dividend_in) >> 10);
8'd60: result_out <= ((1024 * dividend_in) >> 10);
8'd61: result_out <= ((1007 * dividend_in) >> 10);
8'd62: result_out <= ((991 * dividend_in) >> 10);
8'd63: result_out <= ((975 * dividend_in) >> 10);
8'd64: result_out <= ((960 * dividend_in) >> 10);
8'd65: result_out <= ((945 * dividend_in) >> 10);
8'd66: result_out <= ((931 * dividend_in) >> 10);
8'd67: result_out <= ((917 * dividend_in) >> 10);
8'd68: result_out <= ((904 * dividend_in) >> 10);
8'd69: result_out <= ((890 * dividend_in) >> 10);
8'd70: result_out <= ((878 * dividend_in) >> 10);
8'd71: result_out <= ((865 * dividend_in) >> 10);
8'd72: result_out <= ((853 * dividend_in) >> 10);
8'd73: result_out <= ((842 * dividend_in) >> 10);
8'd74: result_out <= ((830 * dividend_in) >> 10);
8'd75: result_out <= ((819 * dividend_in) >> 10);
8'd76: result_out <= ((808 * dividend_in) >> 10);
8'd77: result_out <= ((798 * dividend_in) >> 10);
8'd78: result_out <= ((788 * dividend_in) >> 10);
8'd79: result_out <= ((778 * dividend_in) >> 10);
8'd80: result_out <= ((768 * dividend_in) >> 10);
8'd81: result_out <= ((759 * dividend_in) >> 10);
8'd82: result_out <= ((749 * dividend_in) >> 10);
8'd83: result_out <= ((740 * dividend_in) >> 10);
8'd84: result_out <= ((731 * dividend_in) >> 10);
8'd85: result_out <= ((723 * dividend_in) >> 10);
8'd86: result_out <= ((714 * dividend_in) >> 10);
8'd87: result_out <= ((706 * dividend_in) >> 10);
8'd88: result_out <= ((698 * dividend_in) >> 10);
8'd89: result_out <= ((690 * dividend_in) >> 10);
8'd90: result_out <= ((683 * dividend_in) >> 10);
8'd91: result_out <= ((675 * dividend_in) >> 10);
8'd92: result_out <= ((668 * dividend_in) >> 10);
8'd93: result_out <= ((661 * dividend_in) >> 10);
8'd94: result_out <= ((654 * dividend_in) >> 10);
```

```
8'd95: result_out <= ((647 * dividend_in) >> 10);
8'd96: result_out <= ((640 * dividend_in) >> 10);
8'd97: result_out <= ((633 * dividend_in) >> 10);
8'd98: result_out <= ((627 * dividend_in) >> 10);
8'd99: result_out <= ((621 * dividend_in) >> 10);
8'd100: result_out <= ((614 * dividend_in) >> 10);
8'd101: result_out <= ((608 * dividend_in) >> 10);
8'd102: result_out <= ((602 * dividend_in) >> 10);
8'd103: result_out <= ((597 * dividend_in) >> 10);
8'd104: result_out <= ((591 * dividend_in) >> 10);
8'd105: result_out <= ((585 * dividend_in) >> 10);
8'd106: result_out <= ((580 * dividend_in) >> 10);
8'd107: result_out <= ((574 * dividend_in) >> 10);
8'd108: result_out <= ((569 * dividend_in) >> 10);
8'd109: result_out <= ((564 * dividend_in) >> 10);
8'd110: result_out <= ((559 * dividend_in) >> 10);
8'd111: result_out <= ((554 * dividend_in) >> 10);
8'd112: result_out <= ((549 * dividend_in) >> 10);
8'd113: result_out <= ((544 * dividend_in) >> 10);
8'd114: result_out <= ((539 * dividend_in) >> 10);
8'd115: result_out <= ((534 * dividend_in) >> 10);
8'd116: result_out <= ((530 * dividend_in) >> 10);
8'd117: result_out <= ((525 * dividend_in) >> 10);
8'd118: result_out <= ((521 * dividend_in) >> 10);
8'd119: result_out <= ((516 * dividend_in) >> 10);
8'd120: result_out <= ((512 * dividend_in) >> 10);
8'd121: result_out <= ((508 * dividend_in) >> 10);
8'd122: result_out <= ((504 * dividend_in) >> 10);
8'd123: result_out <= ((500 * dividend_in) >> 10);
8'd124: result_out <= ((495 * dividend_in) >> 10);
8'd125: result_out <= ((492 * dividend_in) >> 10);
8'd126: result_out <= ((488 * dividend_in) >> 10);
8'd127: result_out <= ((484 * dividend_in) >> 10);
8'd128: result_out <= ((480 * dividend_in) >> 10);
8'd129: result_out <= ((476 * dividend_in) >> 10);
8'd130: result_out <= ((473 * dividend_in) >> 10);
8'd131: result_out <= ((469 * dividend_in) >> 10);
8'd132: result_out <= ((465 * dividend_in) >> 10);
8'd133: result_out <= ((462 * dividend_in) >> 10);
8'd134: result_out <= ((459 * dividend_in) >> 10);
8'd135: result_out <= ((455 * dividend_in) >> 10);
8'd136: result_out <= ((452 * dividend_in) >> 10);
8'd137: result_out <= ((448 * dividend_in) >> 10);
8'd138: result_out <= ((445 * dividend_in) >> 10);
8'd139: result_out <= ((442 * dividend_in) >> 10);
8'd140: result_out <= ((439 * dividend_in) >> 10);
8'd141: result_out <= ((436 * dividend_in) >> 10);
8'd142: result_out <= ((433 * dividend_in) >> 10);
8'd143: result_out <= ((430 * dividend_in) >> 10);
8'd144: result_out <= ((427 * dividend_in) >> 10);
8'd145: result_out <= ((424 * dividend_in) >> 10);
8'd146: result_out <= ((421 * dividend_in) >> 10);
8'd147: result_out <= ((418 * dividend_in) >> 10);
8'd148: result_out <= ((415 * dividend_in) >> 10);
8'd149: result_out <= ((412 * dividend_in) >> 10);
8'd150: result_out <= ((410 * dividend_in) >> 10);
8'd151: result_out <= ((407 * dividend_in) >> 10);
8'd152: result_out <= ((404 * dividend_in) >> 10);
8'd153: result_out <= ((402 * dividend_in) >> 10);
8'd154: result_out <= ((399 * dividend_in) >> 10);
8'd155: result_out <= ((396 * dividend_in) >> 10);
```

```
8'd156: result_out <= ((394 * dividend_in) >> 10);
8'd157: result_out <= ((391 * dividend_in) >> 10);
8'd158: result_out <= ((389 * dividend_in) >> 10);
8'd159: result_out <= ((386 * dividend_in) >> 10);
8'd160: result_out <= ((384 * dividend_in) >> 10);
8'd161: result_out <= ((382 * dividend_in) >> 10);
8'd162: result_out <= ((379 * dividend_in) >> 10);
8'd163: result_out <= ((377 * dividend_in) >> 10);
8'd164: result_out <= ((375 * dividend_in) >> 10);
8'd165: result_out <= ((372 * dividend_in) >> 10);
8'd166: result_out <= ((370 * dividend_in) >> 10);
8'd167: result_out <= ((368 * dividend_in) >> 10);
8'd168: result_out <= ((366 * dividend_in) >> 10);
8'd169: result_out <= ((364 * dividend_in) >> 10);
8'd170: result_out <= ((361 * dividend_in) >> 10);
8'd171: result_out <= ((359 * dividend_in) >> 10);
8'd172: result_out <= ((357 * dividend_in) >> 10);
8'd173: result_out <= ((355 * dividend_in) >> 10);
8'd174: result_out <= ((353 * dividend_in) >> 10);
8'd175: result_out <= ((351 * dividend_in) >> 10);
8'd176: result_out <= ((349 * dividend_in) >> 10);
8'd177: result_out <= ((347 * dividend_in) >> 10);
8'd178: result_out <= ((345 * dividend_in) >> 10);
8'd179: result_out <= ((343 * dividend_in) >> 10);
8'd180: result_out <= ((341 * dividend_in) >> 10);
8'd181: result_out <= ((339 * dividend_in) >> 10);
8'd182: result_out <= ((338 * dividend_in) >> 10);
8'd183: result_out <= ((336 * dividend_in) >> 10);
8'd184: result_out <= ((334 * dividend_in) >> 10);
8'd185: result_out <= ((332 * dividend_in) >> 10);
8'd186: result_out <= ((330 * dividend_in) >> 10);
8'd187: result_out <= ((329 * dividend_in) >> 10);
8'd188: result_out <= ((327 * dividend_in) >> 10);
8'd189: result_out <= ((325 * dividend_in) >> 10);
8'd190: result_out <= ((323 * dividend_in) >> 10);
8'd191: result_out <= ((322 * dividend_in) >> 10);
8'd192: result_out <= ((320 * dividend_in) >> 10);
8'd193: result_out <= ((318 * dividend_in) >> 10);
8'd194: result_out <= ((317 * dividend_in) >> 10);
8'd195: result_out <= ((315 * dividend_in) >> 10);
8'd196: result_out <= ((313 * dividend_in) >> 10);
8'd197: result_out <= ((312 * dividend_in) >> 10);
8'd198: result_out <= ((310 * dividend_in) >> 10);
8'd199: result_out <= ((309 * dividend_in) >> 10);
8'd200: result_out <= ((307 * dividend_in) >> 10);
8'd201: result_out <= ((306 * dividend_in) >> 10);
8'd202: result_out <= ((304 * dividend_in) >> 10);
8'd203: result_out <= ((303 * dividend_in) >> 10);
8'd204: result_out <= ((301 * dividend_in) >> 10);
8'd205: result_out <= ((300 * dividend_in) >> 10);
8'd206: result_out <= ((298 * dividend_in) >> 10);
8'd207: result_out <= ((297 * dividend_in) >> 10);
8'd208: result_out <= ((295 * dividend_in) >> 10);
8'd209: result_out <= ((294 * dividend_in) >> 10);
8'd210: result_out <= ((293 * dividend_in) >> 10);
8'd211: result_out <= ((291 * dividend_in) >> 10);
8'd212: result_out <= ((290 * dividend_in) >> 10);
8'd213: result_out <= ((288 * dividend_in) >> 10);
8'd214: result_out <= ((287 * dividend_in) >> 10);
8'd215: result_out <= ((286 * dividend_in) >> 10);
8'd216: result_out <= ((284 * dividend_in) >> 10);
```

```
            8'd217: result_out <= ((283 * dividend_in) >> 10);
            8'd218: result_out <= ((282 * dividend_in) >> 10);
            8'd219: result_out <= ((281 * dividend_in) >> 10);
            8'd220: result_out <= ((279 * dividend_in) >> 10);
            8'd221: result_out <= ((278 * dividend_in) >> 10);
            8'd222: result_out <= ((277 * dividend_in) >> 10);
            8'd223: result_out <= ((276 * dividend_in) >> 10);
            8'd224: result_out <= ((274 * dividend_in) >> 10);
            8'd225: result_out <= ((273 * dividend_in) >> 10);
            8'd226: result_out <= ((272 * dividend_in) >> 10);
            8'd227: result_out <= ((271 * dividend_in) >> 10);
            8'd228: result_out <= ((269 * dividend_in) >> 10);
            8'd229: result_out <= ((268 * dividend_in) >> 10);
            8'd230: result_out <= ((267 * dividend_in) >> 10);
            8'd231: result_out <= ((266 * dividend_in) >> 10);
            8'd232: result_out <= ((265 * dividend_in) >> 10);
            8'd233: result_out <= ((264 * dividend_in) >> 10);
            8'd234: result_out <= ((263 * dividend_in) >> 10);
            8'd235: result_out <= ((261 * dividend_in) >> 10);
            8'd236: result_out <= ((260 * dividend_in) >> 10);
            8'd237: result_out <= ((259 * dividend_in) >> 10);
            8'd238: result_out <= ((258 * dividend_in) >> 10);
            8'd239: result_out <= ((257 * dividend_in) >> 10);
            8'd240: result_out <= ((256 * dividend_in) >> 10);
            8'd241: result_out <= ((255 * dividend_in) >> 10);
            8'd242: result_out <= ((254 * dividend_in) >> 10);
            8'd243: result_out <= ((253 * dividend_in) >> 10);
            8'd244: result_out <= ((252 * dividend_in) >> 10);
            8'd245: result_out <= ((251 * dividend_in) >> 10);
            8'd246: result_out <= ((250 * dividend_in) >> 10);
            8'd247: result_out <= ((249 * dividend_in) >> 10);
            8'd248: result_out <= ((248 * dividend_in) >> 10);
            8'd249: result_out <= ((247 * dividend_in) >> 10);
            8'd250: result_out <= ((246 * dividend_in) >> 10);
            8'd251: result_out <= ((245 * dividend_in) >> 10);
            8'd252: result_out <= ((244 * dividend_in) >> 10);
            8'd253: result_out <= ((243 * dividend_in) >> 10);
            8'd254: result_out <= ((242 * dividend_in) >> 10);
            8'd255: result_out <= ((241 * dividend_in) >> 10);
            default: result_out <= 0;
        endcase
    end
endmodule

// Compute the approximate value of 100 * (dividend_in / divisor_in).
module limited_divider_100 (
    input wire clock_in,
    input wire [7:0] dividend_in,
    input wire [7:0] divisor_in,
    output logic [14:0] result_out
    );

    always @(posedge clock_in) begin
        case (divisor_in)
            8'd1: result_out <= ((102400 * dividend_in) >> 10);
            8'd2: result_out <= ((51200 * dividend_in) >> 10);
            8'd3: result_out <= ((34133 * dividend_in) >> 10);
            8'd4: result_out <= ((25600 * dividend_in) >> 10);
            8'd5: result_out <= ((20480 * dividend_in) >> 10);
            8'd6: result_out <= ((17067 * dividend_in) >> 10);
            8'd7: result_out <= ((14629 * dividend_in) >> 10);
```

```
8'd8: result_out <= ((12800 * dividend_in) >> 10);
8'd9: result_out <= ((11378 * dividend_in) >> 10);
8'd10: result_out <= ((10240 * dividend_in) >> 10);
8'd11: result_out <= ((9309 * dividend_in) >> 10);
8'd12: result_out <= ((8533 * dividend_in) >> 10);
8'd13: result_out <= ((7877 * dividend_in) >> 10);
8'd14: result_out <= ((7314 * dividend_in) >> 10);
8'd15: result_out <= ((6827 * dividend_in) >> 10);
8'd16: result_out <= ((6400 * dividend_in) >> 10);
8'd17: result_out <= ((6024 * dividend_in) >> 10);
8'd18: result_out <= ((5689 * dividend_in) >> 10);
8'd19: result_out <= ((5389 * dividend_in) >> 10);
8'd20: result_out <= ((5120 * dividend_in) >> 10);
8'd21: result_out <= ((4876 * dividend_in) >> 10);
8'd22: result_out <= ((4655 * dividend_in) >> 10);
8'd23: result_out <= ((4452 * dividend_in) >> 10);
8'd24: result_out <= ((4267 * dividend_in) >> 10);
8'd25: result_out <= ((4096 * dividend_in) >> 10);
8'd26: result_out <= ((3938 * dividend_in) >> 10);
8'd27: result_out <= ((3793 * dividend_in) >> 10);
8'd28: result_out <= ((3657 * dividend_in) >> 10);
8'd29: result_out <= ((3531 * dividend_in) >> 10);
8'd30: result_out <= ((3413 * dividend_in) >> 10);
8'd31: result_out <= ((3303 * dividend_in) >> 10);
8'd32: result_out <= ((3200 * dividend_in) >> 10);
8'd33: result_out <= ((3103 * dividend_in) >> 10);
8'd34: result_out <= ((3012 * dividend_in) >> 10);
8'd35: result_out <= ((2926 * dividend_in) >> 10);
8'd36: result_out <= ((2844 * dividend_in) >> 10);
8'd37: result_out <= ((2768 * dividend_in) >> 10);
8'd38: result_out <= ((2695 * dividend_in) >> 10);
8'd39: result_out <= ((2626 * dividend_in) >> 10);
8'd40: result_out <= ((2560 * dividend_in) >> 10);
8'd41: result_out <= ((2498 * dividend_in) >> 10);
8'd42: result_out <= ((2438 * dividend_in) >> 10);
8'd43: result_out <= ((2381 * dividend_in) >> 10);
8'd44: result_out <= ((2327 * dividend_in) >> 10);
8'd45: result_out <= ((2276 * dividend_in) >> 10);
8'd46: result_out <= ((2226 * dividend_in) >> 10);
8'd47: result_out <= ((2179 * dividend_in) >> 10);
8'd48: result_out <= ((2133 * dividend_in) >> 10);
8'd49: result_out <= ((2090 * dividend_in) >> 10);
8'd50: result_out <= ((2048 * dividend_in) >> 10);
8'd51: result_out <= ((2008 * dividend_in) >> 10);
8'd52: result_out <= ((1969 * dividend_in) >> 10);
8'd53: result_out <= ((1932 * dividend_in) >> 10);
8'd54: result_out <= ((1896 * dividend_in) >> 10);
8'd55: result_out <= ((1862 * dividend_in) >> 10);
8'd56: result_out <= ((1829 * dividend_in) >> 10);
8'd57: result_out <= ((1796 * dividend_in) >> 10);
8'd58: result_out <= ((1766 * dividend_in) >> 10);
8'd59: result_out <= ((1736 * dividend_in) >> 10);
8'd60: result_out <= ((1707 * dividend_in) >> 10);
8'd61: result_out <= ((1679 * dividend_in) >> 10);
8'd62: result_out <= ((1652 * dividend_in) >> 10);
8'd63: result_out <= ((1625 * dividend_in) >> 10);
8'd64: result_out <= ((1600 * dividend_in) >> 10);
8'd65: result_out <= ((1575 * dividend_in) >> 10);
8'd66: result_out <= ((1552 * dividend_in) >> 10);
8'd67: result_out <= ((1528 * dividend_in) >> 10);
8'd68: result_out <= ((1506 * dividend_in) >> 10);
```

```
8'd69: result_out <= ((1484 * dividend_in) >> 10);
8'd70: result_out <= ((1463 * dividend_in) >> 10);
8'd71: result_out <= ((1442 * dividend_in) >> 10);
8'd72: result_out <= ((1422 * dividend_in) >> 10);
8'd73: result_out <= ((1403 * dividend_in) >> 10);
8'd74: result_out <= ((1384 * dividend_in) >> 10);
8'd75: result_out <= ((1365 * dividend_in) >> 10);
8'd76: result_out <= ((1347 * dividend_in) >> 10);
8'd77: result_out <= ((1330 * dividend_in) >> 10);
8'd78: result_out <= ((1313 * dividend_in) >> 10);
8'd79: result_out <= ((1296 * dividend_in) >> 10);
8'd80: result_out <= ((1280 * dividend_in) >> 10);
8'd81: result_out <= ((1264 * dividend_in) >> 10);
8'd82: result_out <= ((1249 * dividend_in) >> 10);
8'd83: result_out <= ((1234 * dividend_in) >> 10);
8'd84: result_out <= ((1219 * dividend_in) >> 10);
8'd85: result_out <= ((1205 * dividend_in) >> 10);
8'd86: result_out <= ((1191 * dividend_in) >> 10);
8'd87: result_out <= ((1177 * dividend_in) >> 10);
8'd88: result_out <= ((1164 * dividend_in) >> 10);
8'd89: result_out <= ((1151 * dividend_in) >> 10);
8'd90: result_out <= ((1138 * dividend_in) >> 10);
8'd91: result_out <= ((1125 * dividend_in) >> 10);
8'd92: result_out <= ((1113 * dividend_in) >> 10);
8'd93: result_out <= ((1101 * dividend_in) >> 10);
8'd94: result_out <= ((1089 * dividend_in) >> 10);
8'd95: result_out <= ((1078 * dividend_in) >> 10);
8'd96: result_out <= ((1067 * dividend_in) >> 10);
8'd97: result_out <= ((1056 * dividend_in) >> 10);
8'd98: result_out <= ((1045 * dividend_in) >> 10);
8'd99: result_out <= ((1034 * dividend_in) >> 10);
8'd100: result_out <= ((1024 * dividend_in) >> 10);
8'd101: result_out <= ((1014 * dividend_in) >> 10);
8'd102: result_out <= ((1004 * dividend_in) >> 10);
8'd103: result_out <= ((994 * dividend_in) >> 10);
8'd104: result_out <= ((985 * dividend_in) >> 10);
8'd105: result_out <= ((975 * dividend_in) >> 10);
8'd106: result_out <= ((966 * dividend_in) >> 10);
8'd107: result_out <= ((957 * dividend_in) >> 10);
8'd108: result_out <= ((948 * dividend_in) >> 10);
8'd109: result_out <= ((939 * dividend_in) >> 10);
8'd110: result_out <= ((931 * dividend_in) >> 10);
8'd111: result_out <= ((923 * dividend_in) >> 10);
8'd112: result_out <= ((914 * dividend_in) >> 10);
8'd113: result_out <= ((906 * dividend_in) >> 10);
8'd114: result_out <= ((898 * dividend_in) >> 10);
8'd115: result_out <= ((890 * dividend_in) >> 10);
8'd116: result_out <= ((883 * dividend_in) >> 10);
8'd117: result_out <= ((875 * dividend_in) >> 10);
8'd118: result_out <= ((868 * dividend_in) >> 10);
8'd119: result_out <= ((861 * dividend_in) >> 10);
8'd120: result_out <= ((853 * dividend_in) >> 10);
8'd121: result_out <= ((846 * dividend_in) >> 10);
8'd122: result_out <= ((839 * dividend_in) >> 10);
8'd123: result_out <= ((833 * dividend_in) >> 10);
8'd124: result_out <= ((826 * dividend_in) >> 10);
8'd125: result_out <= ((819 * dividend_in) >> 10);
8'd126: result_out <= ((813 * dividend_in) >> 10);
8'd127: result_out <= ((806 * dividend_in) >> 10);
8'd128: result_out <= ((800 * dividend_in) >> 10);
8'd129: result_out <= ((794 * dividend_in) >> 10);
```

```
8'd130: result_out <= ((788 * dividend_in) >> 10);
8'd131: result_out <= ((782 * dividend_in) >> 10);
8'd132: result_out <= ((776 * dividend_in) >> 10);
8'd133: result_out <= ((770 * dividend_in) >> 10);
8'd134: result_out <= ((764 * dividend_in) >> 10);
8'd135: result_out <= ((759 * dividend_in) >> 10);
8'd136: result_out <= ((753 * dividend_in) >> 10);
8'd137: result_out <= ((747 * dividend_in) >> 10);
8'd138: result_out <= ((742 * dividend_in) >> 10);
8'd139: result_out <= ((737 * dividend_in) >> 10);
8'd140: result_out <= ((731 * dividend_in) >> 10);
8'd141: result_out <= ((726 * dividend_in) >> 10);
8'd142: result_out <= ((721 * dividend_in) >> 10);
8'd143: result_out <= ((716 * dividend_in) >> 10);
8'd144: result_out <= ((711 * dividend_in) >> 10);
8'd145: result_out <= ((706 * dividend_in) >> 10);
8'd146: result_out <= ((701 * dividend_in) >> 10);
8'd147: result_out <= ((697 * dividend_in) >> 10);
8'd148: result_out <= ((692 * dividend_in) >> 10);
8'd149: result_out <= ((687 * dividend_in) >> 10);
8'd150: result_out <= ((683 * dividend_in) >> 10);
8'd151: result_out <= ((678 * dividend_in) >> 10);
8'd152: result_out <= ((674 * dividend_in) >> 10);
8'd153: result_out <= ((669 * dividend_in) >> 10);
8'd154: result_out <= ((665 * dividend_in) >> 10);
8'd155: result_out <= ((661 * dividend_in) >> 10);
8'd156: result_out <= ((656 * dividend_in) >> 10);
8'd157: result_out <= ((652 * dividend_in) >> 10);
8'd158: result_out <= ((648 * dividend_in) >> 10);
8'd159: result_out <= ((644 * dividend_in) >> 10);
8'd160: result_out <= ((640 * dividend_in) >> 10);
8'd161: result_out <= ((636 * dividend_in) >> 10);
8'd162: result_out <= ((632 * dividend_in) >> 10);
8'd163: result_out <= ((628 * dividend_in) >> 10);
8'd164: result_out <= ((624 * dividend_in) >> 10);
8'd165: result_out <= ((621 * dividend_in) >> 10);
8'd166: result_out <= ((617 * dividend_in) >> 10);
8'd167: result_out <= ((613 * dividend_in) >> 10);
8'd168: result_out <= ((610 * dividend_in) >> 10);
8'd169: result_out <= ((606 * dividend_in) >> 10);
8'd170: result_out <= ((602 * dividend_in) >> 10);
8'd171: result_out <= ((599 * dividend_in) >> 10);
8'd172: result_out <= ((595 * dividend_in) >> 10);
8'd173: result_out <= ((592 * dividend_in) >> 10);
8'd174: result_out <= ((589 * dividend_in) >> 10);
8'd175: result_out <= ((585 * dividend_in) >> 10);
8'd176: result_out <= ((582 * dividend_in) >> 10);
8'd177: result_out <= ((579 * dividend_in) >> 10);
8'd178: result_out <= ((575 * dividend_in) >> 10);
8'd179: result_out <= ((572 * dividend_in) >> 10);
8'd180: result_out <= ((569 * dividend_in) >> 10);
8'd181: result_out <= ((566 * dividend_in) >> 10);
8'd182: result_out <= ((563 * dividend_in) >> 10);
8'd183: result_out <= ((560 * dividend_in) >> 10);
8'd184: result_out <= ((557 * dividend_in) >> 10);
8'd185: result_out <= ((554 * dividend_in) >> 10);
8'd186: result_out <= ((551 * dividend_in) >> 10);
8'd187: result_out <= ((548 * dividend_in) >> 10);
8'd188: result_out <= ((545 * dividend_in) >> 10);
8'd189: result_out <= ((542 * dividend_in) >> 10);
8'd190: result_out <= ((539 * dividend_in) >> 10);
```

```
8'd191: result_out <= ((536 * dividend_in) >> 10);
8'd192: result_out <= ((533 * dividend_in) >> 10);
8'd193: result_out <= ((531 * dividend_in) >> 10);
8'd194: result_out <= ((528 * dividend_in) >> 10);
8'd195: result_out <= ((525 * dividend_in) >> 10);
8'd196: result_out <= ((522 * dividend_in) >> 10);
8'd197: result_out <= ((520 * dividend_in) >> 10);
8'd198: result_out <= ((517 * dividend_in) >> 10);
8'd199: result_out <= ((515 * dividend_in) >> 10);
8'd200: result_out <= ((512 * dividend_in) >> 10);
8'd201: result_out <= ((509 * dividend_in) >> 10);
8'd202: result_out <= ((507 * dividend_in) >> 10);
8'd203: result_out <= ((504 * dividend_in) >> 10);
8'd204: result_out <= ((502 * dividend_in) >> 10);
8'd205: result_out <= ((500 * dividend_in) >> 10);
8'd206: result_out <= ((497 * dividend_in) >> 10);
8'd207: result_out <= ((495 * dividend_in) >> 10);
8'd208: result_out <= ((492 * dividend_in) >> 10);
8'd209: result_out <= ((490 * dividend_in) >> 10);
8'd210: result_out <= ((488 * dividend_in) >> 10);
8'd211: result_out <= ((485 * dividend_in) >> 10);
8'd212: result_out <= ((483 * dividend_in) >> 10);
8'd213: result_out <= ((481 * dividend_in) >> 10);
8'd214: result_out <= ((479 * dividend_in) >> 10);
8'd215: result_out <= ((476 * dividend_in) >> 10);
8'd216: result_out <= ((474 * dividend_in) >> 10);
8'd217: result_out <= ((472 * dividend_in) >> 10);
8'd218: result_out <= ((470 * dividend_in) >> 10);
8'd219: result_out <= ((468 * dividend_in) >> 10);
8'd220: result_out <= ((465 * dividend_in) >> 10);
8'd221: result_out <= ((463 * dividend_in) >> 10);
8'd222: result_out <= ((461 * dividend_in) >> 10);
8'd223: result_out <= ((459 * dividend_in) >> 10);
8'd224: result_out <= ((457 * dividend_in) >> 10);
8'd225: result_out <= ((455 * dividend_in) >> 10);
8'd226: result_out <= ((453 * dividend_in) >> 10);
8'd227: result_out <= ((451 * dividend_in) >> 10);
8'd228: result_out <= ((449 * dividend_in) >> 10);
8'd229: result_out <= ((447 * dividend_in) >> 10);
8'd230: result_out <= ((445 * dividend_in) >> 10);
8'd231: result_out <= ((443 * dividend_in) >> 10);
8'd232: result_out <= ((441 * dividend_in) >> 10);
8'd233: result_out <= ((439 * dividend_in) >> 10);
8'd234: result_out <= ((438 * dividend_in) >> 10);
8'd235: result_out <= ((436 * dividend_in) >> 10);
8'd236: result_out <= ((434 * dividend_in) >> 10);
8'd237: result_out <= ((432 * dividend_in) >> 10);
8'd238: result_out <= ((430 * dividend_in) >> 10);
8'd239: result_out <= ((428 * dividend_in) >> 10);
8'd240: result_out <= ((427 * dividend_in) >> 10);
8'd241: result_out <= ((425 * dividend_in) >> 10);
8'd242: result_out <= ((423 * dividend_in) >> 10);
8'd243: result_out <= ((421 * dividend_in) >> 10);
8'd244: result_out <= ((420 * dividend_in) >> 10);
8'd245: result_out <= ((418 * dividend_in) >> 10);
8'd246: result_out <= ((416 * dividend_in) >> 10);
8'd247: result_out <= ((415 * dividend_in) >> 10);
8'd248: result_out <= ((413 * dividend_in) >> 10);
8'd249: result_out <= ((411 * dividend_in) >> 10);
8'd250: result_out <= ((410 * dividend_in) >> 10);
8'd251: result_out <= ((408 * dividend_in) >> 10);
```

```
            8'd252: result_out <= ((406 * dividend_in) >> 10);
            8'd253: result_out <= ((405 * dividend_in) >> 10);
            8'd254: result_out <= ((403 * dividend_in) >> 10);
            8'd255: result_out <= ((402 * dividend_in) >> 10);
            default: result_out <= 0;
        endcase
    end
endmodule


module weighted_average #(
    // parameter LOWER = 0,
    // parameter UPPER = 512,
    parameter POS_BITS = 11) (
    input wire clock_in,
    input wire reset_in,
    input wire start_in,
    input wire done_in,
    input wire [POS_BITS-1 : 0] pos_in,         // index on the axis
    input wire passed_hsv_filter_in,            // whether or not this pixel passed the filter
    output logic ready_out,
    output logic [POS_BITS-1 : 0] avg_out,      // the weighted average
    output logic [POS_BITS-1 : 0] avg_radius
    );

    localparam DIVISION_WIDTH = 30;

    localparam WAITING   = 2'b00;
    localparam SUMMING   = 2'b01;
    localparam DIVIDING  = 2'b10;
    localparam RESETTING = 2'b11;

    logic [1:0] state;
    logic [29:0] sum;       // 20 bits to accumulate on
    logic [29:0] count;      // 10 bits to accumulate the count
    logic [29:0] old_count;       // 20 bits to accumulate on

    logic start_dividing;   // indicate to the the divider module that it should start dividing

    logic division_finished;
    logic [DIVISION_WIDTH-1 : 0] quotient;
    logic [DIVISION_WIDTH-1 : 0] fractional;

    assign avg_out = quotient;

    divider #(.WIDTH(DIVISION_WIDTH)) d (.clock_in(clock_in), .reset_in(reset_in),
        .start_in(start_dividing), .dividend_in(sum), .divisor_in(count),
        .ready_out(division_finished), .quotient_out(quotient),
        .fractional_out(fractional));

    sqrt_max_155 #(.POS_BITS(POS_BITS)) radius_calculator(.x(old_count), .result_out(avg_radius));

    always @(posedge clock_in) begin

        if (reset_in || state == RESETTING) begin

            state <= WAITING;
            sum <= 0;
            ready_out <= 0;
            count <= 0;
            start_dividing <= 0;
            old_count <= 0;
        end else if (start_in) begin

            state <= SUMMING;
            sum <= 0;
            ready_out <= 0;
            count <= 0;
            start_dividing <= 0;

        end else if (state == SUMMING) begin

            if (done_in) begin              // no more pixels -- we should start dividing
                state <= DIVIDING;
                start_dividing <= 1;
            end else if (passed_hsv_filter_in) begin
                sum <= sum + pos_in;
                count <= count + 1;
            end

        end else if (state == DIVIDING) begin
            old_count <= count;
            start_dividing <= 0;



            if (division_finished) begin    // the division completed -- raise the ready signal
                state <= RESETTING;
                ready_out <= 1;
            end

        end

    end

endmodule

module divider #(parameter WIDTH = 20) (
    input wire clock_in,
    input wire reset_in,
    input wire start_in,
    input wire [WIDTH-1 : 0] dividend_in,
    input wire [WIDTH-1 : 0] divisor_in,
    output logic ready_out,
    output logic [WIDTH-1 : 0] quotient_out,
    output logic [WIDTH-1 : 0] fractional_out
    );

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

    assign quotient_out = quotient_temp;
    assign fractional_out = dividend_copy;

    logic [WIDTH-1 : 0] zeros;
    assign zeros = 0;

    logic [5:0] iteration;

    always @(posedge clock_in) begin
        if (reset_in || ready_out) begin // ready_out is turned on in the last iteration of the loop
```

```
            iteration <= 0; // iteration needs to be 0 to prevent it from running off
            ready_out <= 0;

        end else if (start_in) begin

            iteration <= WIDTH;
            ready_out <= 0;

            quotient_temp <= 0;

            dividend_copy <= {zeros, dividend_in}; // equivalent to R
            divider_copy <= {divisor_in, zeros};   // equivalent to D

        end else if (iteration > 0) begin

            if ((dividend_copy << 1) >= divider_copy) begin
                quotient_temp[iteration - 1] <= 1;
                dividend_copy <= (dividend_copy << 1) - divider_copy;
            end else begin
                quotient_temp[iteration - 1] <= 0;
                dividend_copy <= (dividend_copy << 1);
            end

            if (iteration == 1) begin
                ready_out <= 1;
            end

            iteration <= iteration - 1;
        end
    end

endmodule



module sqrt_max_155 #(
    parameter POS_BITS = 11) (
    input wire [29:0] x,
    output logic [POS_BITS-1 : 0] result_out
    );

    always_comb begin
        if (x >= 0 && x < 1) begin
            result_out = 10;
        end else if (x >= 1 && x < 4) begin
            result_out = 10;
        end else if (x >= 4 && x < 9) begin
            result_out = 10;
        end else if (x >= 9 && x < 16) begin
            result_out = 10;
        end else if (x >= 16 && x < 25) begin
            result_out = 10;
        end else if (x >= 25 && x < 36) begin
            result_out = 10;
        end else if (x >= 36 && x < 49) begin
            result_out = 10;
        end else if (x >= 49 && x < 64) begin
            result_out = 10;
        end else if (x >= 64 && x < 81) begin
            result_out = 10;
        end else if (x >= 81 && x < 100) begin
            result_out = 10;
        end else if (x >= 100 && x < 121) begin
            result_out = 11;
        end else if (x >= 121 && x < 144) begin
            result_out = 12;
        end else if (x >= 144 && x < 169) begin
            result_out = 13;
        end else if (x >= 169 && x < 196) begin
            result_out = 14;
        end else if (x >= 196 && x < 225) begin
            result_out = 15;
        end else if (x >= 225 && x < 256) begin
            result_out = 16;
        end else if (x >= 256 && x < 289) begin
            result_out = 17;
        end else if (x >= 289 && x < 324) begin
            result_out = 18;
        end else if (x >= 324 && x < 361) begin
            result_out = 19;
        end else if (x >= 361 && x < 400) begin
            result_out = 20;
        end else if (x >= 400 && x < 441) begin
            result_out = 21;
        end else if (x >= 441 && x < 484) begin
            result_out = 22;
        end else if (x >= 484 && x < 529) begin
            result_out = 23;
        end else if (x >= 529 && x < 576) begin
            result_out = 24;
        end else if (x >= 576 && x < 625) begin
            result_out = 25;
        end else if (x >= 625 && x < 676) begin
            result_out = 26;
        end else if (x >= 676 && x < 729) begin
            result_out = 27;
        end else if (x >= 729 && x < 784) begin
            result_out = 28;
        end else if (x >= 784 && x < 841) begin
            result_out = 29;
        end else if (x >= 841 && x < 900) begin
            result_out = 30;
        end else if (x >= 900 && x < 961) begin
            result_out = 31;
        end else if (x >= 961 && x < 1024) begin
            result_out = 32;
        end else if (x >= 1024 && x < 1089) begin
            result_out = 33;
        end else if (x >= 1089 && x < 1156) begin
            result_out = 34;
        end else if (x >= 1156 && x < 1225) begin
            result_out = 35;
        end else if (x >= 1225 && x < 1296) begin
            result_out = 36;
        end else if (x >= 1296 && x < 1369) begin
            result_out = 37;
        end else if (x >= 1369 && x < 1444) begin
            result_out = 38;
        end else if (x >= 1444 && x < 1521) begin
            result_out = 39;
        end else if (x >= 1521 && x < 1600) begin
            result_out = 40;
```

```
end else if (x >= 1600 && x < 1681) begin
   result_out = 41;
end else if (x >= 1681 && x < 1764) begin
   result_out = 42;
end else if (x >= 1764 && x < 1849) begin
   result_out = 43;
end else if (x >= 1849 && x < 1936) begin
   result_out = 44;
end else if (x >= 1936 && x < 2025) begin
   result_out = 45;
end else if (x >= 2025 && x < 2116) begin
   result_out = 46;
end else if (x >= 2116 && x < 2209) begin
   result_out = 47;
end else if (x >= 2209 && x < 2304) begin
   result_out = 48;
end else if (x >= 2304 && x < 2401) begin
   result_out = 49;
end else if (x >= 2401 && x < 2500) begin
   result_out = 50;
end else if (x >= 2500 && x < 2601) begin
   result_out = 51;
end else if (x >= 2601 && x < 2704) begin
   result_out = 52;
end else if (x >= 2704 && x < 2809) begin
   result_out = 53;
end else if (x >= 2809 && x < 2916) begin
   result_out = 54;
end else if (x >= 2916 && x < 3025) begin
   result_out = 55;
end else if (x >= 3025 && x < 3136) begin
   result_out = 56;
end else if (x >= 3136 && x < 3249) begin
   result_out = 57;
end else if (x >= 3249 && x < 3364) begin
   result_out = 58;
end else if (x >= 3364 && x < 3481) begin
   result_out = 59;
end else if (x >= 3481 && x < 3600) begin
   result_out = 60;
end else if (x >= 3600 && x < 3721) begin
   result_out = 61;
end else if (x >= 3721 && x < 3844) begin
   result_out = 62;
end else if (x >= 3844 && x < 3969) begin
   result_out = 63;
end else if (x >= 3969 && x < 4096) begin
   result_out = 64;
end else if (x >= 4096 && x < 4225) begin
   result_out = 65;
end else if (x >= 4225 && x < 4356) begin
   result_out = 66;
end else if (x >= 4356 && x < 4489) begin
   result_out = 67;
end else if (x >= 4489 && x < 4624) begin
   result_out = 68;
end else if (x >= 4624 && x < 4761) begin
   result_out = 69;
end else if (x >= 4761 && x < 4900) begin
   result_out = 70;
end else if (x >= 4900 && x < 5041) begin
```

```
    result_out = 71;
end else if (x >= 5041 && x < 5184) begin
    result_out = 72;
end else if (x >= 5184 && x < 5329) begin
    result_out = 73;
end else if (x >= 5329 && x < 5476) begin
    result_out = 74;
end else if (x >= 5476 && x < 5625) begin
    result_out = 75;
end else if (x >= 5625 && x < 5776) begin
    result_out = 76;
end else if (x >= 5776 && x < 5929) begin
    result_out = 77;
end else if (x >= 5929 && x < 6084) begin
    result_out = 78;
end else if (x >= 6084 && x < 6241) begin
    result_out = 79;
end else if (x >= 6241 && x < 6400) begin
    result_out = 80;
end else if (x >= 6400 && x < 6561) begin
    result_out = 81;
end else if (x >= 6561 && x < 6724) begin
    result_out = 82;
end else if (x >= 6724 && x < 6889) begin
    result_out = 83;
end else if (x >= 6889 && x < 7056) begin
    result_out = 84;
end else if (x >= 7056 && x < 7225) begin
    result_out = 85;
end else if (x >= 7225 && x < 7396) begin
    result_out = 86;
end else if (x >= 7396 && x < 7569) begin
    result_out = 87;
end else if (x >= 7569 && x < 7744) begin
    result_out = 88;
end else if (x >= 7744 && x < 7921) begin
    result_out = 89;
end else if (x >= 7921 && x < 8100) begin
    result_out = 90;
end else if (x >= 8100 && x < 8281) begin
    result_out = 91;
end else if (x >= 8281 && x < 8464) begin
    result_out = 92;
end else if (x >= 8464 && x < 8649) begin
    result_out = 93;
end else if (x >= 8649 && x < 8836) begin
    result_out = 94;
end else if (x >= 8836 && x < 9025) begin
    result_out = 95;
end else if (x >= 9025 && x < 9216) begin
    result_out = 96;
end else if (x >= 9216 && x < 9409) begin
    result_out = 97;
end else if (x >= 9409 && x < 9604) begin
    result_out = 98;
end else if (x >= 9604 && x < 9801) begin
    result_out = 99;
end else if (x >= 9801 && x < 10000) begin
    result_out = 100;
end else if (x >= 10000 && x < 10201) begin
    result_out = 101;
```

```
end else if (x >= 10201 && x < 10404) begin
    result_out = 102;
end else if (x >= 10404 && x < 10609) begin
    result_out = 103;
end else if (x >= 10609 && x < 10816) begin
    result_out = 104;
end else if (x >= 10816 && x < 11025) begin
    result_out = 105;
end else if (x >= 11025 && x < 11236) begin
    result_out = 106;
end else if (x >= 11236 && x < 11449) begin
    result_out = 107;
end else if (x >= 11449 && x < 11664) begin
    result_out = 108;
end else if (x >= 11664 && x < 11881) begin
    result_out = 109;
end else if (x >= 11881 && x < 12100) begin
    result_out = 110;
end else if (x >= 12100 && x < 12321) begin
    result_out = 111;
end else if (x >= 12321 && x < 12544) begin
    result_out = 112;
end else if (x >= 12544 && x < 12769) begin
    result_out = 113;
end else if (x >= 12769 && x < 12996) begin
    result_out = 114;
end else if (x >= 12996 && x < 13225) begin
    result_out = 115;
end else if (x >= 13225 && x < 13456) begin
    result_out = 116;
end else if (x >= 13456 && x < 13689) begin
    result_out = 117;
end else if (x >= 13689 && x < 13924) begin
    result_out = 118;
end else if (x >= 13924 && x < 14161) begin
    result_out = 119;
end else if (x >= 14161 && x < 14400) begin
    result_out = 120;
end else if (x >= 14400 && x < 14641) begin
    result_out = 121;
end else if (x >= 14641 && x < 14884) begin
    result_out = 122;
end else if (x >= 14884 && x < 15129) begin
    result_out = 123;
end else if (x >= 15129 && x < 15376) begin
    result_out = 124;
end else if (x >= 15376 && x < 15625) begin
    result_out = 125;
end else if (x >= 15625 && x < 15876) begin
    result_out = 126;
end else if (x >= 15876 && x < 16129) begin
    result_out = 127;
end else if (x >= 16129 && x < 16384) begin
    result_out = 128;
end else if (x >= 16384 && x < 16641) begin
    result_out = 129;
end else if (x >= 16641 && x < 16900) begin
    result_out = 130;
end else if (x >= 16900 && x < 17161) begin
    result_out = 131;
end else if (x >= 17161 && x < 17424) begin
```

```
            result_out = 132;
        end else if (x >= 17424 && x < 17689) begin
            result_out = 133;
        end else if (x >= 17689 && x < 17956) begin
            result_out = 134;
        end else if (x >= 17956 && x < 18225) begin
            result_out = 135;
        end else if (x >= 18225 && x < 18496) begin
            result_out = 136;
        end else if (x >= 18496 && x < 18769) begin
            result_out = 137;
        end else if (x >= 18769 && x < 19044) begin
            result_out = 138;
        end else if (x >= 19044 && x < 19321) begin
            result_out = 139;
        end else if (x >= 19321 && x < 19600) begin
            result_out = 140;
        end else if (x >= 19600 && x < 19881) begin
            result_out = 141;
        end else if (x >= 19881 && x < 20164) begin
            result_out = 142;
        end else if (x >= 20164 && x < 20449) begin
            result_out = 143;
        end else if (x >= 20449 && x < 20736) begin
            result_out = 144;
        end else if (x >= 20736 && x < 21025) begin
            result_out = 145;
        end else if (x >= 21025 && x < 21316) begin
            result_out = 146;
        end else if (x >= 21316 && x < 21609) begin
            result_out = 147;
        end else if (x >= 21609 && x < 21904) begin
            result_out = 148;
        end else if (x >= 21904 && x < 22201) begin
            result_out = 149;
        end else if (x >= 22201 && x < 22500) begin
            result_out = 150;
        end else if (x >= 22500 && x < 22801) begin
            result_out = 151;
        end else if (x >= 22801 && x < 23104) begin
            result_out = 152;
        end else if (x >= 23104 && x < 23409) begin
            result_out = 153;
        end else if (x >= 23409 && x < 23716) begin
            result_out = 154;
        end else begin
            result_out = 155;
        end
    end
endmodule


`default_nettype wire
```

display_helpers.sv

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

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

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

   parameter DISPLAY_WIDTH  = 1024;      // display width
   parameter DISPLAY_HEIGHT = 768;       // number of lines

   parameter  H_FP = 24;                 // horizontal front porch
   parameter  H_SYNC_PULSE = 136;        // horizontal sync
   parameter  H_BP = 160;                // horizontal back porch

   parameter  V_FP = 3;                  // vertical front porch
   parameter  V_SYNC_PULSE = 6;          // vertical sync
   parameter  V_BP = 29;                 // vertical back porch

   // horizontal: 1344 pixels total
   // display 1024 pixels per line
   reg hblank,vblank;
   wire hsyncon,hsyncoff,hreset,hblankon;
   assign hblankon = (hcount_out == (DISPLAY_WIDTH -1));
   assign hsyncon = (hcount_out == (DISPLAY_WIDTH + H_FP - 1));  //1047
   assign hsyncoff = (hcount_out == (DISPLAY_WIDTH + H_FP + H_SYNC_PULSE - 1));  // 1183
   assign hreset = (hcount_out == (DISPLAY_WIDTH + H_FP + H_SYNC_PULSE + H_BP - 1));  //1343

   // vertical: 806 lines total
   // display 768 lines
   wire vsyncon,vsyncoff,vreset,vblankon;
   assign vblankon = hreset & (vcount_out == (DISPLAY_HEIGHT - 1));   // 767
   assign vsyncon = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP - 1));  // 771
   assign vsyncoff = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP + V_SYNC_PULSE - 1));  // 777
   assign vreset = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP + V_SYNC_PULSE + V_BP - 1)); // 805

   // sync and blanking
```

```
  wire next_hblank,next_vblank;
  assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
  assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
  always_ff @(posedge vclock_in) begin
    hcount_out <= hreset ? 0 : hcount_out + 1;
    hblank <= next_hblank;
    hsync_out <= hsyncon ? 0 : hsyncoff ? 1 : hsync_out;  // active low

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

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

endmodule


///////////////////////////////////////////////////////////////////////////////
// Engineer:   g.p.hom
//
// Create Date:    18:18:59 04/21/2013
// Module Name:    display_8hex
// Description:  Display 8 hex numbers on 7 segment display
//
///////////////////////////////////////////////////////////////////////////////

module display_8hex(
   input wire clk_in,                 // system clock
   input wire [31:0] data_in,         // 8 hex numbers, msb first
   output reg [6:0] seg_out,     // seven segment display output
   output reg [7:0] strobe_out   // digit strobe
   );

   localparam bits = 13;

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

   wire [6:0] segments[15:0]; // 16 7 bit memorys
   assign segments[0]  = 7'b100_0000;  // inverted logic
   assign segments[1]  = 7'b111_1001;  // gfedcba
   assign segments[2]  = 7'b010_0100;
   assign segments[3]  = 7'b011_0000;
   assign segments[4]  = 7'b001_1001;
   assign segments[5]  = 7'b001_0010;
   assign segments[6]  = 7'b000_0010;
   assign segments[7]  = 7'b111_1000;
   assign segments[8]  = 7'b000_0000;
   assign segments[9]  = 7'b001_1000;
   assign segments[10] = 7'b000_1000;
   assign segments[11] = 7'b000_0011;
   assign segments[12] = 7'b010_0111;
   assign segments[13] = 7'b010_0001;
   assign segments[14] = 7'b000_0110;
   assign segments[15] = 7'b000_1110;

   always_ff @(posedge clk_in) begin
     // Here I am using a counter and select 3 bits which provides
     // a reasonable refresh rate starting the left most digit
     // and moving left.
     counter <= counter + 1;
```

```
      case (counter[bits:bits-2])
          3'b000: begin  // use the MSB 4 bits
                  seg_out <= segments[data_in[31:28]];
                  strobe_out <= 8'b0111_1111 ;
                 end

          3'b001: begin
                  seg_out <= segments[data_in[27:24]];
                  strobe_out <= 8'b1011_1111 ;
                 end

          3'b010: begin
                   seg_out <= segments[data_in[23:20]];
                   strobe_out <= 8'b1101_1111 ;
                  end
          3'b011: begin
                  seg_out <= segments[data_in[19:16]];
                  strobe_out <= 8'b1110_1111;
                 end
          3'b100: begin
                  seg_out <= segments[data_in[15:12]];
                  strobe_out <= 8'b1111_0111;
                 end

          3'b101: begin
                  seg_out <= segments[data_in[11:8]];
                  strobe_out <= 8'b1111_1011;
                 end

          3'b110: begin
                   seg_out <= segments[data_in[7:4]];
                   strobe_out <= 8'b1111_1101;
                  end
          3'b111: begin
                  seg_out <= segments[data_in[3:0]];
                  strobe_out <= 8'b1111_1110;
                 end

       endcase
      end

endmodule

module sync_delay (
   input wire vclock_in,        // 65MHz clock
   input wire reset_in,        // 1 to initialize module
   input wire hsync_in,         // XVGA horizontal sync signal (active low)
   input wire vsync_in,         // XVGA vertical sync signal (active low)
   input wire blank_in,         // XVGA blanking (1 means output black pixel)
   input wire ball_bounded_in,
   input wire target_bounded_in,
   input wire virtual_bounded_in,
   input wire [11:0] cam_in,

   output logic phsync_out,
   output logic pvsync_out,
   output logic pblank_out,
   output logic ball_bounded_out,
   output logic target_bounded_out,
   output logic virtual_bounded_out,
   output logic [11:0] cam_out
```

```
  );

   localparam DELAY = 13;  //16, 12
   logic [DELAY-1:0] hsync_buffer, vsync_buffer, blank_buffer;

   localparam DELAY_2 = 7; //10, 6
   logic [DELAY_2-1:0] ball_buffer, target_buffer, virtual_buffer;
   logic [DELAY_2-1:0][11:0] cam_buffer;


   genvar i;
   generate
      for (i = 1; i <= DELAY-1; i = i + 1) begin
         always @(posedge vclock_in) begin
            hsync_buffer[i] <= hsync_buffer[i-1];
            vsync_buffer[i] <= vsync_buffer[i-1];
            blank_buffer[i] <= blank_buffer[i-1];
            ball_buffer[i] <= ball_buffer[i-1];
            target_buffer[i] <= target_buffer[i-1];
            virtual_buffer[i] <= virtual_buffer[i-1];
            cam_buffer[i][11:0] <= cam_buffer[i-1][11:0];
         end
      end
   endgenerate

   always @(posedge vclock_in) begin
      hsync_buffer[0] <= hsync_in;
      vsync_buffer[0] <= vsync_in;
      blank_buffer[0] <= blank_in;
      ball_buffer[0] <= ball_bounded_in;
      target_buffer[0] <= target_bounded_in;
      virtual_buffer[0] <= virtual_bounded_in;
      cam_buffer[0][11:0] <= cam_in;

      phsync_out <= hsync_buffer[DELAY-1];
      pvsync_out <= vsync_buffer[DELAY-1];
      pblank_out <= blank_buffer[DELAY-1];
      ball_bounded_out <= ball_buffer[DELAY_2-1];
      target_bounded_out <= target_buffer[DELAY_2-1];
      virtual_bounded_out <= virtual_buffer[DELAY_2-1];
      cam_out <= cam_buffer[DELAY_2-1][11:0];
   end
endmodule

module within_bounding_box (
   input wire [10:0] hcount_in, // horizontal index of current pixel (0..1023)
   input wire [9:0]  vcount_in, // vertical index of current pixel (0..767)
   input wire [10:0] h_center_in,  // the horizontal center of the ball from the previous frame
   input wire [9:0] v_center_in,   // the vertical center of the ball from the previous frame
   input wire [10:0] radius_in,
   output logic bounded_out,
   output logic within_circle_out
);

   logic [10:0] h_diff;
   logic [9:0] v_diff;
   abs #(.WIDTH(11)) h_diff_mod (.a(hcount_in), .b(h_center_in), .c(h_diff));
   abs #(.WIDTH(10)) v_diff_mod (.a(vcount_in), .b(v_center_in), .c(v_diff));

   always_comb begin
      if ((h_diff + v_diff <= radius_in + 5) && (h_diff + v_diff >= radius_in)) begin
```

```
            bounded_out = 1;
        end else begin
            bounded_out = 0;
        end

        if ((2 * (h_diff * h_diff + v_diff * v_diff)) < (radius_in * radius_in)) begin // within the circle
            within_circle_out = 1;
        end else begin
            within_circle_out = 0;
        end
    end

endmodule


// computes the absolute value of (a-b) and puts the result in c
module abs #(parameter WIDTH = 11) (
    input wire [WIDTH-1:0] a,
    input wire [WIDTH-1:0] b,
    output logic [WIDTH-1:0] c
);

    always_comb begin

        if (a > b) begin
            c = a - b;
        end else begin
            c = b - a;
        end

    end

endmodule
`default_nettype wire
```

hardware_helpers.sv

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

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


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

   localparam WAIT_FRAME_START = 0;
   localparam ROW_CAPTURE = 1;


   always_ff@(posedge p_clock_in)
   begin
   case(FSM_state)

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

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

endmodule

module ultrasound_controller (input wire reset_in, clock_in, up_in, down_in, start_in,
                             input wire [15:0] sw_in,
                             input wire [9:0] ball_y_in, target_y_in,
                             output logic [10:0] phase1_out, phase2_out);

   parameter HIGH_THRESHOLD = 10'd30; // How many pixels the ball can be off
   parameter MED_THRESHOLD = 10'd15; // How many pixels the ball can be off
   parameter LOW_THRESHOLD = 10'd2; // How many pixels the ball can be off
   parameter HIGH_DELTA = 10'd50;
   parameter MED_DELTA = 10'd25;
   parameter LOW_DELTA = 10'd10;

   always_ff @(posedge clock_in) begin
```

```
        if (reset_in) begin
            phase1_out <= 0;
            phase2_out <= 0;
        end else if (up_in == 1'b1 && sw_in[1] == 1'b1 && sw_in[2] == 1'b1 && sw_in[5] == 1'b1) begin
            if ((phase1_out + 125) <= 11'd1250) begin
                phase1_out <= phase1_out + 125;
            end else begin
                phase1_out <= 0;
            end
        end else if (down_in == 1'b1 && sw_in[1] == 1'b1 && sw_in[2] == 1'b1 && sw_in[5] == 1'b1) begin
            if ((phase2_out + 125) <= 11'd1250) begin
                phase2_out <= phase2_out + 125;
            end else begin
                phase2_out <= 0;
            end
        end else if (start_in == 1'b1 && sw_in[1] == 1'b1 && sw_in[2] == 1'b0 && sw_in[5] == 1'b1) begin
            if (ball_y_in > target_y_in + HIGH_THRESHOLD) begin
                if ((phase1_out + HIGH_DELTA) <= 11'd1250) begin
                    phase1_out <= phase1_out + HIGH_DELTA;
                end else begin
                    phase1_out <= 0;
                end
            end else if (ball_y_in > target_y_in + MED_THRESHOLD) begin
                if ((phase1_out + MED_DELTA) <= 11'd1250) begin
                    phase1_out <= phase1_out + MED_DELTA;
                end else begin
                    phase1_out <= 0;
                end
            end else if (ball_y_in > target_y_in + LOW_THRESHOLD) begin
                if ((phase1_out + LOW_DELTA) <= 11'd1250) begin
                    phase1_out <= phase1_out + LOW_DELTA;
                end else begin
                    phase1_out <= 0;
                end
            end else if (ball_y_in + HIGH_THRESHOLD < target_y_in) begin
                if ((phase2_out + HIGH_DELTA) <= 11'd1250) begin
                    phase2_out <= phase2_out + HIGH_DELTA;
                end else begin
                    phase2_out <= 0;
                end
            end else if (ball_y_in + MED_THRESHOLD < target_y_in) begin
                if ((phase2_out + MED_DELTA) <= 11'd1250) begin
                    phase2_out <= phase2_out + MED_DELTA;
                end else begin
                    phase2_out <= 0;
                end
            end else if (ball_y_in + LOW_THRESHOLD < target_y_in) begin
                if ((phase2_out + LOW_DELTA) <= 11'd1250) begin
                    phase2_out <= phase2_out + LOW_DELTA;
                end else begin
                    phase2_out <= 0;
                end
            end
        end
    end

endmodule

module ultrasound_out (input wire reset_in, clock_in,
                       input wire [10:0] phase1, phase2,
                       output logic output1, output2);
    parameter HZ_40k = 18'd1250;// 100mhz/40khz/2

    logic [18:0] counter;
    logic [18:0] counter1;
    logic [18:0] counter2;

    always_ff @(posedge clock_in) begin
        if (reset_in) begin
            counter <= 19'b0;
            counter1 <= phase1;
            counter2 <= phase2;
            output1 <= 0;
            output2 <= 0;
        end else begin
            // Reference counter that keeps the global 40hz waveform
            if (counter == HZ_40k) begin
                counter <= 0;
            end else begin
                counter <= counter + 1;
            end
            // Counter1 maintains top output
            if (counter1 == HZ_40k) begin
                if (counter + phase1 < 1250) begin
                    counter1 <= counter + phase1;
                end else begin
                    counter1 <= counter + phase1 - 1250;
                end
                output1 <= ~output1;
            end else begin
                counter1 <= counter1 + 1;
            end
            // Counter1 maintains bottom output
            if (counter2 == HZ_40k) begin
                if (counter + phase2 < 1250) begin
                    counter2 <= counter + phase2;
                end else begin
                    counter2 <= counter + phase2 - 1250;
                end
                output2 <= ~output2;
            end else begin
                counter2 <= counter2 + 1;
            end
        end
    end

endmodule


module rising_edge (input wire reset_in, clock_in, clean_in,
                    output logic rising_out);

    logic old;
    assign rising_out = !old & clean_in;

    always_ff @(posedge clock_in) begin
        if (reset_in) begin
            old <= 0;
        end else begin
            old <= clean_in;
        end
```

```
   end

endmodule

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

  logic [19:0] count;
  logic new_input;

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

endmodule

`default_nettype wire
```