E.D.I.T.H.

Timi Omotunde and Roberto Ramirez

Table of Contents

Abstract

The aim of this project is to design and implement a computer interface that you can interact with through your hand and voice using digital logic written in System Verilog HDL. In order to implement this interface it is necessary that the digital logic is written to follow the same logic of a regular mousepad. Thus, the basic scroll, left click, right click, double left click, and so forth.

The techniques that have been implemented include RGB values, FFT sectioning and serial UART communication. While these methods are sufficient for basic communication between the FPGA and a laptop we added additional complexity in each module in order to make the system robust. The purpose of implementing the aforementioned techniques is to significantly reduce the amount of edge cases that the user encounters, allowing the user to have a much smoother experience overall. The FPGA that will be used is the Nexys DDR development board. There were additional hardware components needed that we will delve into in the following sections.

Introduction

E.D.I.T.H is a modern and creative alternative to the computer mouse and laptop touchpad that combines visual and audio components to replicate all of the functionality of a normal mouse. The system consists of a black glove with LEDs attached, a Nexys4 FPGA Board, and a Teensy, along with an external circuit that sends audio information to the FPGA. There are three LEDs attached to the fingertips of the glove: a red one which is always on and is used to determine the direction and magnitude of each mouse movement, a green one that acts as the left button of the mouse (when the green LED is on, it's as if the left button of a mouse is pressed), and a blue one which acts as the right button of the mouse, similar to the green LED. The motivation behind the project comes from some of Tony Stark's technology and how he remotely interacts with computers via his hands and vocal commands.

Goals

Baseline

1. High and low tone detection - 2 tones
    a. High tone will correspond to a certain mouse action
    b. Low tone will correspond to another certain mouse action

2. LED light to track mouse coordinates onto the monitor

3. Wired connection between buttons on the glove and the FPGA

4. Filters on audio and visual portion to protect from noise

Expected

1. Increased functionality on audio segment
    a. Noise level/threshold adjustable

2. Programmable audio mode
    a. Frequency range is adjustable to use with human voices and sine generators

3. RGB LEDS to act as selection button on the glove for the mouse
    a. Red tracks to XY-position for the mouse
    b. Green corresponds to a left click
    c. Blue corresponds to a right click

4. The glove is fully wireless
    a. The glove uses a battery pack for the LED
    b. FPGA communicates to a laptop wirelessly via a Teensy
    c. All wiring hidden in the glove

Stretch

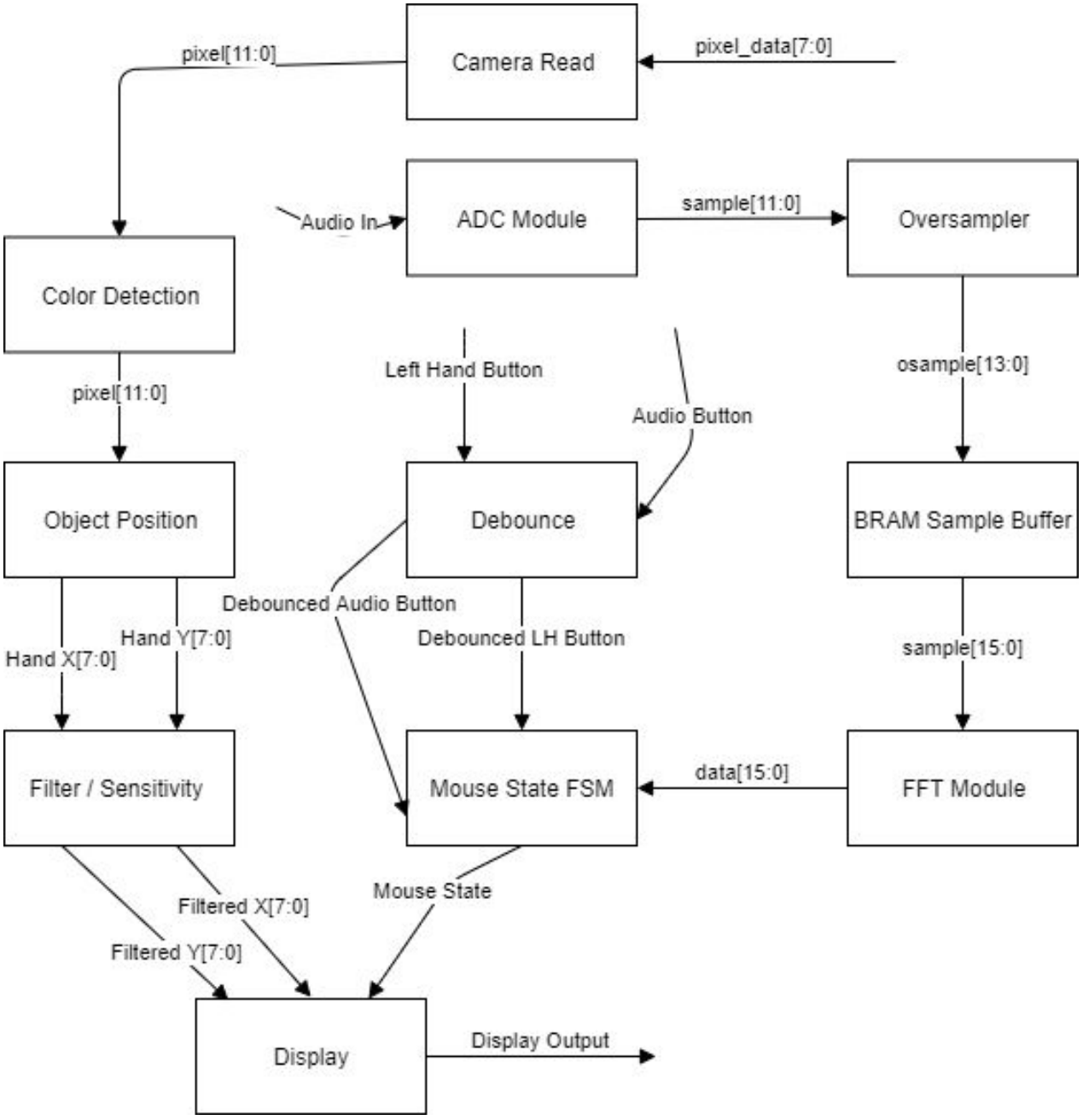
1. Increased functionality on audio button selection
    a. A continuous high tone corresponds to scrolling up
    b. A continuous low tone corresponds to scrolling down

2. Basic words recognition
    a. Train the FPGA to recognize certain words in order to give commands

3. The ability to change mouse sensitivities for faster and slower mouse movements

4. The ability to change LED colors without losing functionality in the glove - purely for preference

Block Diagram



Design and Overview

For our team, there were many initial difficult decisions that we had to make before we began. Which specific techniques that we wished to employ such as RGB against HSV and FFT against sound patterns. These two examples were two of the most important subset of things we needed to settle upon before we began. Additional aspects that we deliberated on included the connection between the Teensy and the FPGA as well as the serial commands for the Teensy in order for the Teensy to replicate a mouse.

The reason that we had many starting costs was because we realized that once we began the project there would not be much room for changing how we planned to implement a specific piece of our project. Thus, we needed to have a plan set in stone before we began writing even the first bitstream to go into the FPGA.

Subsystem Overview

<u>Visual</u>

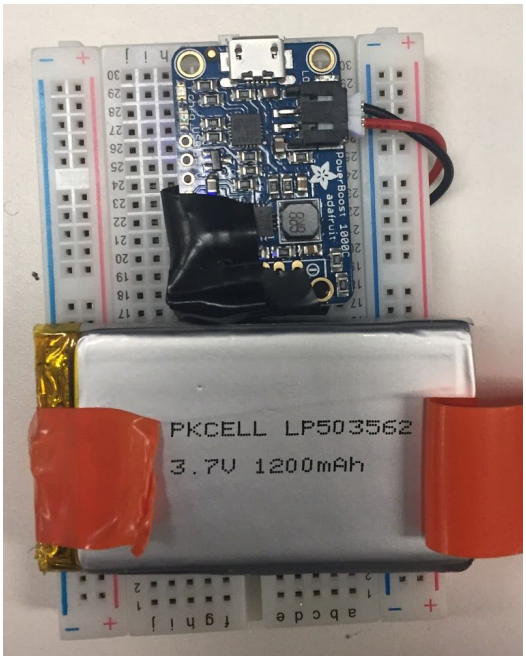
Glove

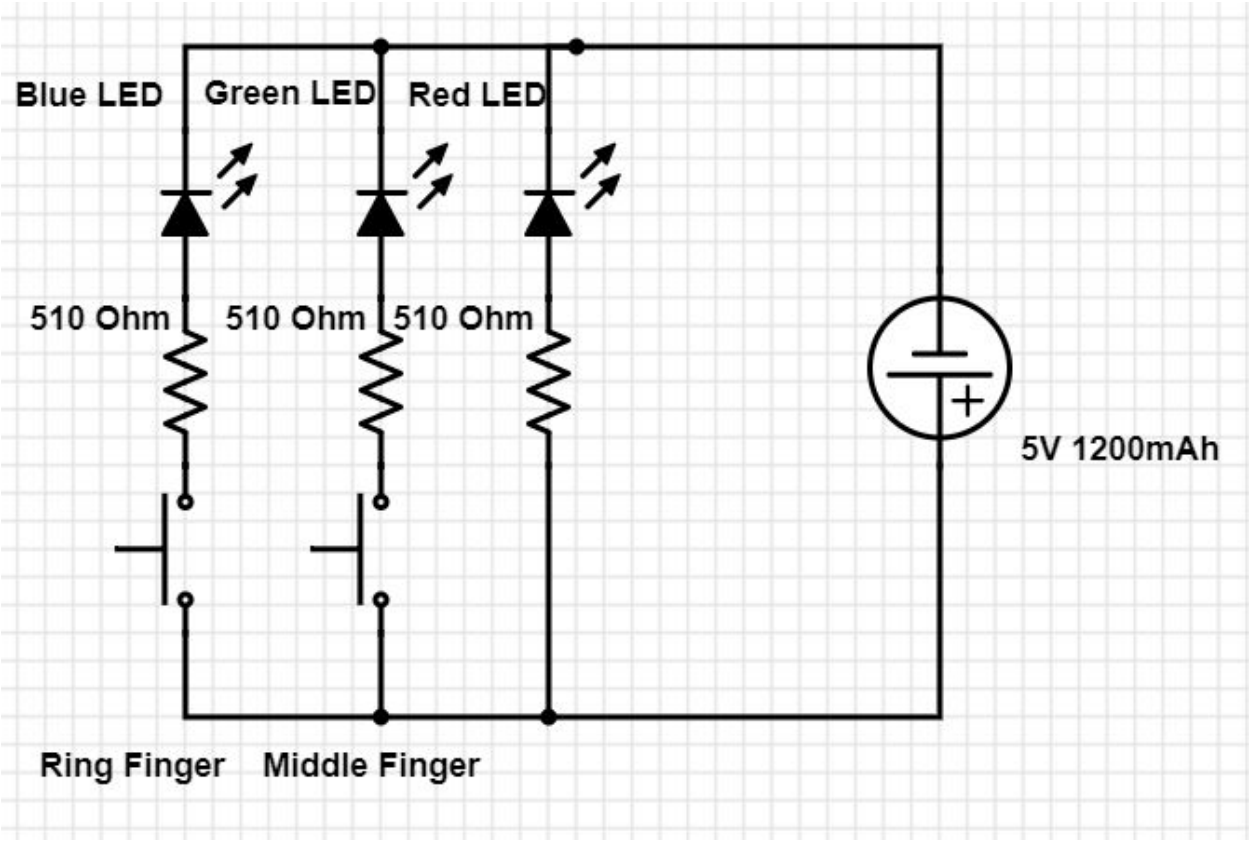
The purpose of the glove is to streamline the interaction of a user and a computer in order to improve upon the existing Human-Computer Interface device, the mouse. The inspiration behind the device came through Tony Stark and how he interacts with his computers using only his hands and none of the traditional tools such as the mouse and keyboard. And, there are many practical applications of what we made such as people who are bedridden and still want to use a computer comfortably and people that need to use a computer, but cannot physically be at that workstation.

The glove has three LEDS. A red LED on the index finger for the camera to track XY-coordinates for the monitor; a green LED on the middle finger to act as left click when turned on by touching the copper wire on the thumb to the copper wire on the base of the middle finger; and, a blue LED on the ring finger to act as

right click when turned on by touching the copper wire on the thumb to the copper wire on the base of the ring finger. The copper wire on the thumb is connected to a 5V source and acts as a switch when it is connected or unconnected to the copper tape wires on the middle and ring fingers. Each LED is connected to a 510 Ohm resistor so that the LED does not burn out as well as to decrease the bleed effect when the camera detects the LED. If you are recreating the circuit make sure that there is enough current flowing through the LED so depending on your LED you need to adjust the resistors accordingly. As seen below, we used a 3.7V 1200mAh battery that we fed through boost converter to get 5V.





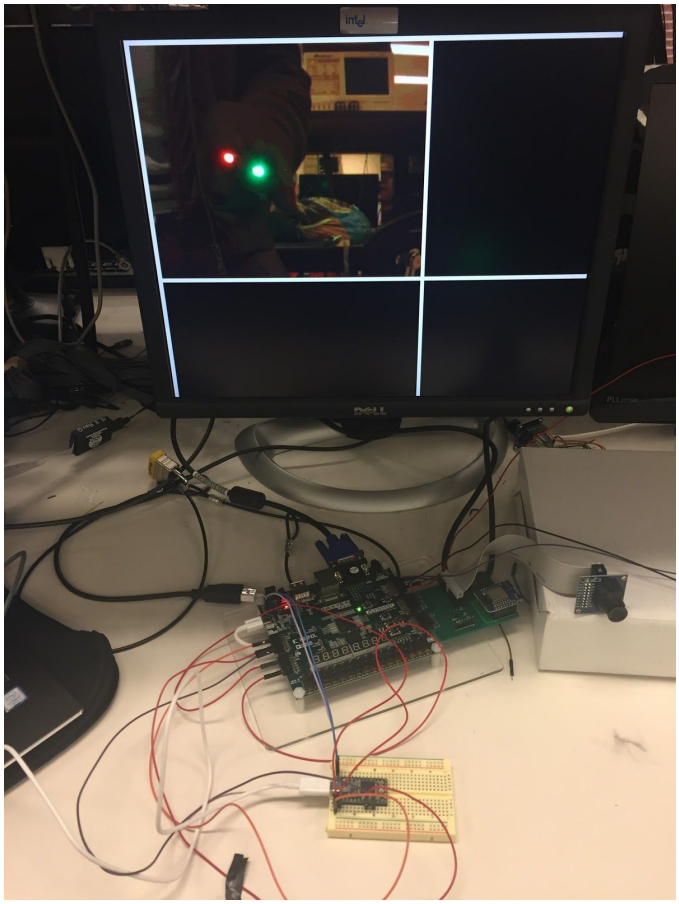


LED Tracking

Originally, the idea was to convert the RGB values for each pixel received from the camera to HSV values in order to determine whether or not the LEDs were in the output image and, if so, where the red LED was in the image. This is because HSV values are better for object detection since they separate color information (hue) from luminance/light, which RGB values fail to do. Out of curiosity, however, I decided to whether or not I could make RGB thresholds that would detect only the LEDs. Surprisingly, I was able to find thresholds that detected only the LEDs and nothing else (assuming there were no other similar LEDs in the background), so I decided to stick with RGB values throughout the rest of the project.

With these RGB thresholds in place, I check to see whether or not the LEDs were detected in the image. If the green LED or blue LED is detected in the image, I return a high value (1) to the FSM that will pass that information over to the Teensy, otherwise I return a low value (0). I perform this check every fifth of a second because checking too often doesn't provide the FPGA enough time to generate enough frames to see whether or not the LEDs have appeared and/or disappeared in the image. The same check is done for the red LED, but this time if the red LED is in the image, we also check to see where the LED is in relation to the center.

If the red LED is to the left of the center of the image, the mouse on the laptop will move left, if it is above the center, the mouse will move up, and the same rules apply for below or to the right of the center. The farther away the red LED is from the center of the image, the faster the mouse on the laptop screen will move, similar to a joystick for games. There are also two switches that can be flipped that allow for a total of four different mouse sensitivities. By increasing the sensitivity (done by increasing the value of the switches), changes in the position of the red LED result in larger changes in mouse speed.



Teensy/FSM

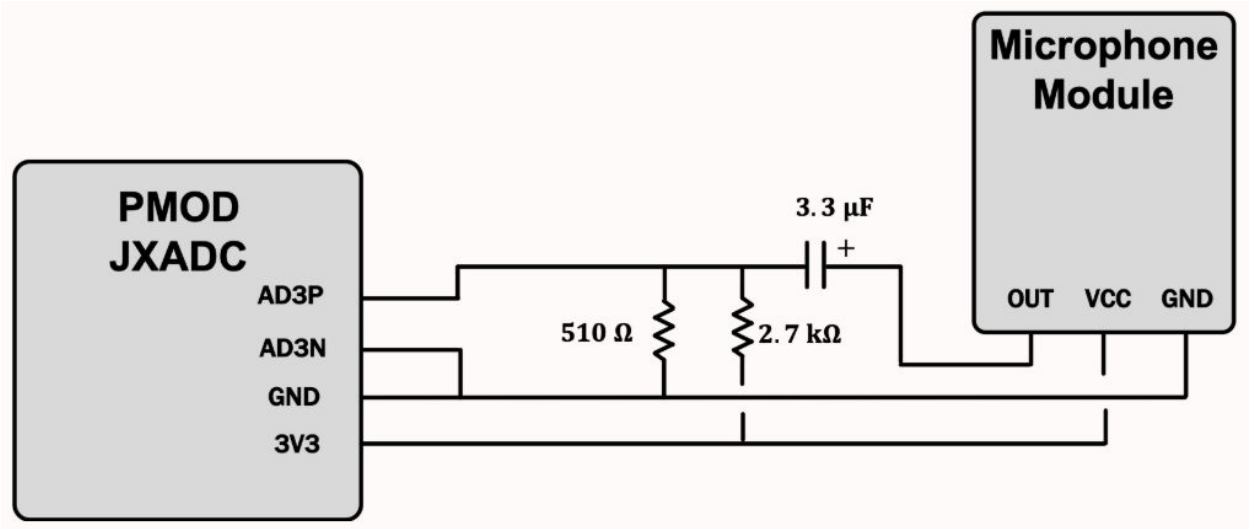
For everything to come together, we needed a finite state machine that would control what values got sent to the Teensy which would act as a mouse. The FSM would take in the LED values described above as well as audio values (described below in the next section, basically 1's are sent if a certain command is being performed), and determine what state the mouse should currently be in (what action should be performed). The actions are: pressing down the left button, right clicking, double left clicking, scrolling up, and scrolling down. The mouse is constantly being sent movement values (even when the mouse technically is not moving), so movement is not a state of the FSM. The FPGA is connected to the Teensy through three serial ports on the Teensy. The first port is used for communicating the mouse state, the second port for the mouse horizontal movement value, and the last port for the mouse vertical movement value. The Teensy has its own type of FSM compiled on to it to determine what action it should perform based on each incoming byte. In order to make the mouse

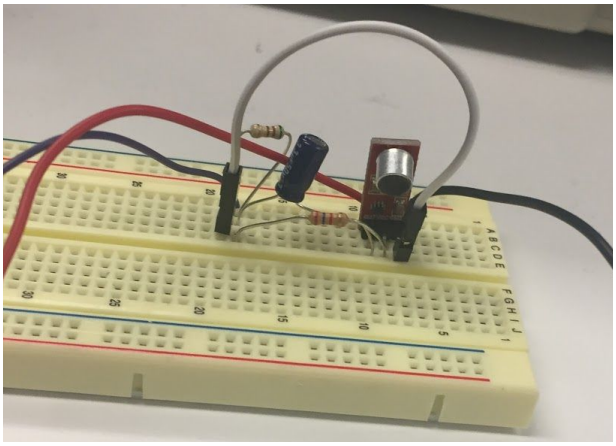
movement smooth, the movement values are sent every one twenty-fifth of a second.

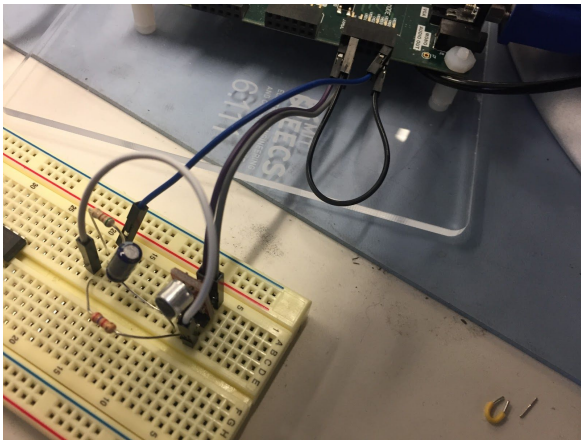
Audio

Fast Fourier Transform (FFT)

We are using an FFT system so that we can analyze the incoming sound from a microphone in terms of frequency. We are doing this so that we can control aspects of a mouse by the pitch that we pass into the microphone. Thus, specific frequencies will activitate specific corresponding outputs.







Testing and Debugging

In testing and debugging modules, some test benches were employed, but as the systems became more and more developed the best debugging tools became generating the bitstream and interacting with it as it either worked or it did not.

Each system was tested independently before the final integration. The final integration required the most rigorous testing as there existed numerous overlaps and dependencies that needed to be accounted.

For the visual portion of the project, there was a lot of trial and error involved in finding the correct RGB values to look for and use. In the Arduino code for the camera, there were three different gains: a blue gain, red gain, and green gain. The green gain was by default much lower than the red and blue gain, which made many of the pixels in the output image appear more red than they should have been. Fortunately, the effects of adjusting these gains could be seen immediately by compiling the code and viewing the output image, and eventually we found a combination of gains that worked pretty well.

Filtering out the correct RGB values that corresponded to the LED colors involved a similar approach to finding the correct gains for the camera settings. This time, however, some Verilog was written so that when a switch was flipped, the output image pixels would be red if the pixel output from the camera was red enough, blue if the pixel output was blue enough, and green if the output was green enough (and black otherwise). By doing this, we were able to see which pixels were being detected as red, green, and blue, and after many attempts we narrowed down the range for each pixel to correspond to the colors emitted by the LEDs so that only the LEDs would be detected.

In testing and debugging the audio portion, there were a few key phases: knowing what I could and could not remove, adding different tones, thresholding, and scrolling.

For testing what we could and could not remove, we had to read through the code and get a good understanding of what the FFT system is doing. In our first attempt we tried removing everything remotely related to the XVGA system however this had repercussions in respect to the system not working. Throughout this process it gave us a better intuition of what each part did as we repeated this process a few times. We found that there are only two things at most that we can remove. The XVGA initialization signal near the start and the XVGA call at the bottom. Everything else is closely tied to the histogram which gives meaning to the loudness of each frequency.

For adding different tones, we needed to experiment with what tones we could actually produce via a sine generator and human vocal chords. Additionally, we need to test what actual frequencies the FFT system would recognize. To do all of this we downloaded a sine generator on a phone for testing as well as an FFT on our phone to give meaning to specific frequency ranges in order to expedite the process so that we do not need to wait on every bitstream call. We discovered that the FFT we had on the FPGA could go up to 15kHz, but we did not need to 9kHz at the highest for humans - unless that person has incredible pitch control. We discovered that each bin contains approximately 15Hz worth of frequencies as the

sample rate is 62.5kHz and there are 4096 frames. Thus to target specific frequencies we would divide that frequency by 15 (Hz/bin) to find the bin that it would exist in.

Furthermore, to find an acceptable threshold, we had to play directly with the FPGA in order to find one that is high enough to disregard noise, but one that does not penalize intentional tones greatly. We did this by printing out bin magnitude values to the hex display and taking an average of the upper magnitude values.

Integrating the visual and audio components together involved doing a lot of testing on the FSM that merged the two together. This was very tricky, but fortunately the Teensy has a Serial Monitor that can print things to the screen, so we relied heavily on this monitor to test and debug the FSM. Every time a byte was transmitted to the Teensy, we would print it to the Serial Monitor to see what state was passed in, and by doing this we were able to see whether or not the state that was passed in was correct or not (and whether or not the byte was transmitted at the correct time).

Challenges and Improvements

One of the greatest challenges for this project was time, mainly referring to the amount of time it took to generate the bitstream in order to verify that our solution was actually usable. We could run all the test benches that we wanted on the system, but there were some aspects of the project that we had to verify by human testing. Namely, pitch control on the audio side and button selection on the Visual side. As we added more and more complexity layers onto the system, the bitstream generation time also increased. We should also add that this was before integration. Pre-integration, the bitstream generation times would average between 50 and 60 minutes. Post-integration bitstream generation times would average 60 minutes or more. Sufficient to say, but whenever we changed or loaded something onto the FPGA, we made certain that our change was efficient and effective for the problem on hand.

On the visual side, additional challenges included finding what RGB values correspond to each LED, and figuring out how often to send information to the Teensy to make mouse movements and actions appear smooth.

As mentioned in the testing and debugging section, finding the RGB values that corresponded to each LED involved a lot of trial and error through plugging in many different values until we found the ones that worked. Timing and sending information to the Teensy was definitely more difficult and required far more testing since the Teensy and the FPGA were running on different clocks. Also, when sending mouse movement information to the Teensy, we had to make sure to send the information at the perfect times, since sending it too fast would make the mouse zoom across the screen, and sending it too slow would make the mouse movement extremely choppy. Figuring out when to send the mouse actions to the Teensy was also difficult, since different actions needed to be sent at different times and at different intervals. The mouse right click, for example, only needed to be sent to the Teensy as soon as the blue LED is seen, but the green LED meant that the mouse left button was held down, so we had to keep track of when the green LED appears and when it disappears as well.

Some improvements that we would make to the visual aspect of the project would be allowing users to switch which LEDs correspond to which action. Currently the red LED is for movement, the green for left clicking, and the blue for right clicking, and all of these actions are fixed and cannot be changed. While we feel that the placement of each LED is very intuitive and easy to understand, some users might prefer clicking with different fingers or might not want to move the mouse with their index finger, so it makes sense to give users the option to select their preferences before using the glove. This wouldn't be too difficult to implement in Verilog, but modifying the glove's internal circuit to allow the LEDs to function differently (constantly on vs. controlled using switch) depending on user preferences would definitely be tricky.

On the audio side, additional challenges included microphone reliability, outside noise, and controlling the scroll. Movereover, we managed to handle each problem accordingly as they arose.

I can address microphone reliability and outside noise in the same paragraph because the implemented solution handled them as different sides of the same coin. We initially thought that getting a better microphone would solve our problems, but we realized not necessarily as even though the sound is refined it does not cancel the outside noise. Also a high quality microphone is not really necessary for 6.111 projects. It should be noted, we did consider switching microphones, but we realized that we would still be running into some of the same problems such as outside noise even though we have a more mobile, better microphone. Instead, we decided to filter out the noise and add a threshold to activate the system. In this way, only intentional sounds are processed. By definition an FFT is a filter, so we played around with the internal settings and settled on one that penalized ambient sounds, but still allowed generous spikes for intentional noises. Furthermore, we added a threshold not only for sine generators, but also for human voices. Thus, the threshold became critical for tones made by humans. This is important because you may not think about it, but when you speak, whistle, or make any type of noise with your vocal chords it is not purely in one frequency, but in numerous frequencies. Yet, not all these frequencies are in the same decibel range as there is almost always one dominant frequency in terms of decibels. So, in our project, we exploited this fact to hone in on the biggest spike and targeted that one and pretend as the other undertones are random noise that we wish to ignore.

For controlling the scroll, we wanted some way in order to generate a magnitude to be able to translate how much we want to scroll. It should be noted that we sandboxed this method to smooth and streamline the system. However the controls are still present in the system, so one can play around with it and also further activate it. We will still expound upon it as it took a considerable amount of time and contributed to the end product. Initially, we wanted to be able to be able to adjust/slide in a frequency range in order to gain a magnitude by subtracting the pointers as they hold different bin numbers. In this method, we would be

generating the beginning and ending pointers then subtracting the difference. The pointers would store the bin number that they were held in that range. The magnitude/difference would change every clock cycle. However, the main problem with this implementation is that we were trying to do everything in the same clock cycle which - even if it were successful which it would not be - would make the magnitudes small and not what we want. Thus, in the second implementation, the system only responds if you enter the range directly in the middle so that you can slide the frequency up or down with enough space on either side. The first pointer would be set because you started in the middle and would only change if you exit and start again. The second pointer is free to move up or down and record the current bin number.

Moving on, some improvements that we would make to the audio would be word recognition. One could already say there is some basic word recognition as when you say the word “double” the system will double click. However that comes about due to the word “double” matching the same frequency range that triggers a double click. Thus, we came up with multiple ways - after the project deadline - to generate some word recognition. The first way is to ignore frequency ranges and only focus on the aggregate amount of sound coming into the microphone. The microphone would listen for specific aspects of the word relating to peaks and troughs - local and maxima. This method would be less specific, but more general and likelier to pick up words. The second method is more complex and more powerful. You would still listen for the peaks and troughs of the aggregate sound, but you would add the frequency range as some words start high or low and pitch up or down accordingly. If you do not know what a transformers are in the context of word recognition and machine learning you would need to hard code each word to be recognized by the FPGA. In this way you could more confidently stretch and compress words, since everyone does not speak at the same speed. In the first method it would be harder because it could just be random noise, but it is still doable.

Conclusion

Overall, this project was a success. We were able to smoothly integrate our video and audio aspects so that a user can utilize their hands to control a computer monitor - doing everything that a regular computer mouse can plus more!

Additionally, there is not a high learning curve for our device, so literally anyone can pick it up and use it intuitively.

As it was said to me, I would like to emphasize the importance of starting as early as you can. Our project was substantial in size and complexity, and it barely made it over the finish line in time - luckily integration worked on the first attempt after we solved all the syntax issues. Starting early will save you time and stress, as integration usually does not work the first, second, or third time you try it. Another important thing to note is that you are not always guaranteed two weeks after Thanksgiving, so if you only have one week after the break life will get tough. So, please remember to sleep and do not put off your problems until the last moment for your own sake.

Moreover, through this project, we learned an incredible amount of project creation - seeing it through from inception to implementation. We learned what it means to come up with an original idea and how to test and debug your original concepts and creations. We also learned that innovation is not necessarily a straight path, but one with numerous twists and turns that one should be prepared to go down. Moreover, we would emphatically recommend 6.111 to any other students.

Lastly, to any students attempting this or any similar project, we would recommend again that you start early and pick a fast computer that can generate the bitstream in under 30 minutes. These files are massive, so editing and running the bitstream on the older computers once took an hour and 20 minutes to finish the audio portion pre-integration alone! When we integrated both parts, the run time climbed to an hour and 40 minutes at its highest. However, the faster, newer

computers will generate the bitstream in around 40 minutes at their slowest. Additionally, try to reach all of our stretch goals as they are all doable if you are clever in your approach, we realized in hindsight. We realized the scroll and the multiple mouse sensitivities.

Acknowledgments

We found 6.111 to be a truly enjoyable class, and it would not have been possible without the amazing staff. Specifically, for our team, Gim, Joe, Sarah, Mike, and Diana who have been supportive and kept the lab open through late nights and early mornings.

Verilog Code:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Updated 11/10/2019 EDITH
// Updated 8/12/2018 V2.lab5c
// Create Date: 10/1/2015 V1.
// Module Name: top_level_visual
// Based on lab 3 code and code from Joe
//////////////////////////////////////////////////////////////////////////////////

module top_level(
  input clk_100mhz,
  input[15:0] sw,
  input btnc, btnu, btnl, btnr, btnd,
  input [7:0] ja,
  input [2:0] jb,
  input AD3N, AD3P,
  output logic [1:0] jc,
  output  jbclk,
  output logic [2:0] jd,
```

```
  output   jdclk,
  output[3:0] vga_r,
  output[3:0] vga_b,
  output[3:0] vga_g,
  output vga_hs,
  output vga_vs,
  output led16_b, led16_g, led16_r,
  output led17_b, led17_g, led17_r,
  output[15:0] led,
  output ca, cb, cc, cd, ce, cf, cg, dp,  // segments a-g, dp
  output[7:0] an    // Display location 0-7
  );

   parameter FIFTH = 13000000; // the number of cycles corresponding to one fifth
of a second
   parameter UPDATE = 2600000; // the number of cycles corresponding to one
twenty-fifth of a second
   wire clk_65mhz;
   wire clk_104mhz;
   // create 65mhz system clock, happens to match 1024 x 768 XVGA timing
   clk_wiz_0 clkdivider(.clk_in1(clk_100mhz), .reset(0), .clk_out1(clk_65mhz),
.clk_out2(clk_104mhz));

   wire [31:0] data;      //  instantiate 7-segment display; display (8) 4-bit hex
   wire [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 seg[6:0] = segments;
   assign  dp = 1'b1;  // turn off the period

   assign led16_b = audio_mode;
   assign data = {28'h0123456, sw[3:0]};   // display 0123456 + sw[3:0]
```

```
   assign led16_r = btnl;                // left button -> red led
   assign led16_g = btnc;                 // center button -> green led
   assign led17_r = btnl;
   assign led17_g = btnc;
   assign led17_b = btnr;

   wire [10:0] hcount;    // pixel on current line
   wire [9:0] vcount;     // line number
   wire hsync, vsync, blank;
   wire [11:0] pixel;
   reg [11:0] rgb;
   reg [23:0] hsv;
   logic audio_mode = 0; // audio commands can only be used when audio_mode is
1
   logic clean; // clean and old_clean used to determine audio_mode
   logic old_clean;
   logic left_click = 0; // 1 when left mouse button should be pressed down, 0
otherwise
   logic right_click = 0; // 1 when right mouse button should be pressed down, 0
otherwise
   logic double_click = 0;
   logic scroll_up = 0;
   logic scroll_down = 0;
   logic green_found = 0; // these "found" values are 1 if the corresponding values
were seen during the last fifth of a second, 0 otherwise
   logic red_found = 0;
   logic blue_found = 0;
   logic double_found = 0;
   logic scrollu_found = 0;
   logic scrolld_found = 0;
   logic double_out = led[14]; // 1 when audio command for double click is heard,
0 otherwise
```

```
   logic scrollu_out = led[7]; // 1 when audio command for scroll up is heard, 0 otherwise
   logic scrolld_out = led[0]; // 1 when audio command for scroll down is heard, 0 othwerise
   logic [25:0] left_counter = 0; // gets reset to 0 every fifth of a second
   logic [25:0] update_counter = 0; // gets reset to 0 when this counter reaches UPDATE
   logic [10:0] xcount = 320; // the current hcount pixel value of the center of the red LED (320 when LED is not found)
   logic [9:0] ycount = 240; // the current vcount pixel value of the center of the red LED (240 when LED is not found)
   logic signed [7:0] x_val; // the x value corresponding to how the mouse will move on the screen (sent to Teensy)
   logic signed [7:0] y_val; // the y value corresponding to how the mouse will move on the screen
   logic right;
   logic up;
   logic aud_pwm, aud_sd;
   xvga xvga1(.vclock_in(clk_65mhz),.hcount_out(hcount),.vcount_out(vcount),
        .hsync_out(hsync),.vsync_out(vsync),.blank_out(blank));

   nexys4_fft_demo fft(.CLK100MHZ(clk_100mhz),
                       .clk_104mhz(clk_104mhz),
                       .clk_65mhz(clk_65mhz),
                       .hcount(hcount),
                       .vcount(vcount),
                       .hsync(hsync),
                       .vsync(vsync),
                       .SW(sw),
                       .BTNU(btnu), .BTNR(btnr),
                       .LED(led),
                       .AD3P(AD3P),
                       .AD3N(AD3N),
```

```
                .AUD_PWM(aud_pwm), .AUD_SD(aud_sd));

    debounce audio(.clock_in(clk_65mhz),
                .reset_in(reset),
                .noisy_in(btnc),
                .clean_out(clean));

    debounce rbutton(.clock_in(clk_65mhz),
                .reset_in(reset),
                .noisy_in(btnr),
                .clean_out(right));

    debounce ubutton(.clock_in(clk_65mhz),
                .reset_in(reset),
                .noisy_in(btnu),
                .clean_out(up));

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

    top_serial left(.clk_100mhz(clk_100mhz),
                .left_click(left_click),
                .right_click(right_click),
                .double_click(double_click),
                .scroll_up(scroll_up),
                .scroll_down(scroll_down),
                .btnd(btnd),
                .jc(jc),
                .jd(jd),
                .x_val(x_val),
                .y_val(y_val));
```

```
    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 [3:0] red_diff;
    logic [3:0] green_diff;
    logic [3:0] blue_diff;
    logic valid_pixel;
    logic frame_done_out;

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

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

    assign red_diff =
(output_pixels[15:12]>old_output_pixels[15:12])?output_pixels[15:12]-old_output
_pixels[15:12]:old_output_pixels[15:12]-output_pixels[15:12];
```

```
   assign green_diff =
(output_pixels[10:7]>old_output_pixels[10:7])?output_pixels[10:7]-old_output_pi
xels[10:7]:old_output_pixels[10:7]-output_pixels[10:7];
   assign blue_diff =
(output_pixels[4:1]>old_output_pixels[4:1])?output_pixels[4:1]-old_output_pixels
[4:1]:old_output_pixels[4:1]-output_pixels[4:1];



   blk_mem_gen_0 jojos_bram(.addra(pixel_addr_in),
                            .clka(pclk_in),
                            .dina(processed_pixels),
                            .wea(valid_pixel),
                            .addrb(pixel_addr_out),
                            .enb(1),
                            .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_comb begin
      // sw[1:0] determines the mouse sensitivity (higher values mean higher
sensitivies)
      // when you increase sensitivity, smaller changes in x or y will result in larger
increases in mouse speed
      case (sw[1:0])
         2'b11: begin
```

```
if (xcount < 45) x_val = -28;
else if (xcount < 55) x_val = -27;
else if (xcount < 65) x_val = -26;
else if (xcount < 75) x_val = -25;
else if (xcount < 85) x_val = -24;
else if (xcount < 95) x_val = -23;
else if (xcount < 105) x_val = -22;
else if (xcount < 115) x_val = -21;
else if (xcount < 125) x_val = -20;
else if (xcount < 135) x_val = -19;
else if (xcount < 145) x_val = -18;
else if (xcount < 155) x_val = -17;
else if (xcount < 165) x_val = -16;
else if (xcount < 175) x_val = -15;
else if (xcount < 185) x_val = -14;
else if (xcount < 195) x_val = -13;
else if (xcount < 205) x_val = -12;
else if (xcount < 215) x_val = -11;
else if (xcount < 225) x_val = -10;
else if (xcount < 235) x_val = -9;
else if (xcount < 245) x_val = -8;
else if (xcount < 255) x_val = -7;
else if (xcount < 265) x_val = -6;
else if (xcount < 275) x_val = -5;
else if (xcount < 285) x_val = -4;
else if (xcount < 295) x_val = -3;
else if (xcount < 305) x_val = -2;
else if (xcount < 315) x_val = -1;
else if (xcount < 325) x_val = 0;
else if (xcount < 335) x_val = 1;
else if (xcount < 345) x_val = 2;
else if (xcount < 355) x_val = 3;
else if (xcount < 365) x_val = 4;
```

```
else if (xcount < 375) x_val = 5;
else if (xcount < 385) x_val = 6;
else if (xcount < 395) x_val = 7;
else if (xcount < 405) x_val = 8;
else if (xcount < 415) x_val = 9;
else if (xcount < 425) x_val = 10;
else if (xcount < 435) x_val = 11;
else if (xcount < 445) x_val = 12;
else if (xcount < 455) x_val = 13;
else if (xcount < 465) x_val = 14;
else if (xcount < 475) x_val = 15;
else if (xcount < 485) x_val = 16;
else if (xcount < 495) x_val = 17;
else if (xcount < 505) x_val = 18;
else if (xcount < 515) x_val = 19;
else if (xcount < 525) x_val = 20;
else if (xcount < 535) x_val = 21;
else if (xcount < 545) x_val = 22;
else if (xcount < 555) x_val = 23;
else if (xcount < 565) x_val = 24;
else if (xcount < 575) x_val = 25;
else if (xcount < 585) x_val = 26;
else if (xcount < 595) x_val = 27;
else x_val = 28;
if (ycount < 25) y_val = -22;
else if (ycount < 35) y_val = -21;
else if (ycount < 45) y_val = -20;
else if (ycount < 55) y_val = -19;
else if (ycount < 65) y_val = -18;
else if (ycount < 75) y_val = -17;
else if (ycount < 85) y_val = -16;
else if (ycount < 95) y_val = -15;
else if (ycount < 105) y_val = -14;
```

```
else if (ycount < 115) y_val = -13;
else if (ycount < 125) y_val = -12;
else if (ycount < 135) y_val = -11;
else if (ycount < 145) y_val = -10;
else if (ycount < 155) y_val = -9;
else if (ycount < 165) y_val = -8;
else if (ycount < 175) y_val = -7;
else if (ycount < 185) y_val = -6;
else if (ycount < 195) y_val = -5;
else if (ycount < 205) y_val = -4;
else if (ycount < 215) y_val = -3;
else if (ycount < 225) y_val = -2;
else if (ycount < 235) y_val = -1;
else if (ycount < 245) y_val = 0;
else if (ycount < 255) y_val = 1;
else if (ycount < 265) y_val = 2;
else if (ycount < 275) y_val = 3;
else if (ycount < 285) y_val = 4;
else if (ycount < 295) y_val = 5;
else if (ycount < 305) y_val = 6;
else if (ycount < 315) y_val = 7;
else if (ycount < 325) y_val = 8;
else if (ycount < 335) y_val = 9;
else if (ycount < 345) y_val = 10;
else if (ycount < 355) y_val = 11;
else if (ycount < 365) y_val = 12;
else if (ycount < 375) y_val = 13;
else if (ycount < 385) y_val = 14;
else if (ycount < 395) y_val = 15;
else if (ycount < 405) y_val = 16;
else if (ycount < 415) y_val = 17;
else if (ycount < 425) y_val = 18;
else if (ycount < 435) y_val = 19;
```

```
            else if (ycount < 445) y_val = 20;
            else if (ycount < 455) y_val = 21;
            else y_val = 22;
        end
    2'b10: begin
            if (xcount < 50) x_val = -14;
            else if (xcount < 70) x_val = -13;
            else if (xcount < 90) x_val = -12;
            else if (xcount < 110) x_val = -11;
            else if (xcount < 130) x_val = -10;
            else if (xcount < 150) x_val = -9;
            else if (xcount < 170) x_val = -8;
            else if (xcount < 190) x_val = -7;
            else if (xcount < 210) x_val = -6;
            else if (xcount < 230) x_val = -5;
            else if (xcount < 250) x_val = -4;
            else if (xcount < 270) x_val = -3;
            else if (xcount < 290) x_val = -2;
            else if (xcount < 310) x_val = -1;
            else if (xcount < 330) x_val = 0;
            else if (xcount < 350) x_val = 1;
            else if (xcount < 370) x_val = 2;
            else if (xcount < 390) x_val = 3;
            else if (xcount < 410) x_val = 4;
            else if (xcount < 430) x_val = 5;
            else if (xcount < 450) x_val = 6;
            else if (xcount < 470) x_val = 7;
            else if (xcount < 490) x_val = 8;
            else if (xcount < 510) x_val = 9;
            else if (xcount < 530) x_val = 10;
            else if (xcount < 550) x_val = 11;
            else if (xcount < 570) x_val = 12;
            else if (xcount < 590) x_val = 13;
```

```
            else x_val = 14;
            if (ycount < 30) y_val = -11;
            else if (ycount < 50) y_val = -10;
            else if (ycount < 70) y_val = -9;
            else if (ycount < 90) y_val = -8;
            else if (ycount < 110) y_val = -7;
            else if (ycount < 130) y_val = -6;
            else if (ycount < 150) y_val = -5;
            else if (ycount < 170) y_val = -4;
            else if (ycount < 190) y_val = -3;
            else if (ycount < 210) y_val = -2;
            else if (ycount < 230) y_val = -1;
            else if (ycount < 250) y_val = 0;
            else if (ycount < 270) y_val = 1;
            else if (ycount < 290) y_val = 2;
            else if (ycount < 310) y_val = 3;
            else if (ycount < 330) y_val = 4;
            else if (ycount < 350) y_val = 5;
            else if (ycount < 370) y_val = 6;
            else if (ycount < 390) y_val = 7;
            else if (ycount < 410) y_val = 8;
            else if (ycount < 430) y_val = 9;
            else if (ycount < 450) y_val = 10;
            else y_val = 11;
        end
    2'b01: begin
            if (xcount < 60) x_val = -7;
            else if (xcount < 100) x_val = -6;
            else if (xcount < 140) x_val = -5;
            else if (xcount < 180) x_val = -4;
            else if (xcount < 220) x_val = -3;
            else if (xcount < 260) x_val = -2;
            else if (xcount < 300) x_val = -1;
```

```
            else if (xcount < 340) x_val = 0;
            else if (xcount < 380) x_val = 1;
            else if (xcount < 420) x_val = 2;
            else if (xcount < 460) x_val = 3;
            else if (xcount < 500) x_val = 4;
            else if (xcount < 540) x_val = 5;
            else if (xcount < 580) x_val = 6;
            else x_val = 7;
            if (ycount < 45) y_val = -7;
            else if (ycount < 75) y_val = -6;
            else if (ycount < 105) y_val = -5;
            else if (ycount < 135) y_val = -4;
            else if (ycount < 165) y_val = -3;
            else if (ycount < 195) y_val = -2;
            else if (ycount < 225) y_val = -1;
            else if (ycount < 255) y_val = 0;
            else if (ycount < 285) y_val = 1;
            else if (ycount < 315) y_val = 2;
            else if (ycount < 345) y_val = 3;
            else if (ycount < 375) y_val = 4;
            else if (ycount < 405) y_val = 5;
            else if (ycount < 435) y_val = 6;
            else y_val = 7;
        end
    default: begin
            if (xcount < 40) x_val = -4;
            else if (xcount < 120) x_val = -3;
            else if (xcount < 200) x_val = -2;
            else if (xcount < 280) x_val = -1;
            else if (xcount < 360) x_val = 0;
            else if (xcount < 440) x_val = 1;
            else if (xcount < 520) x_val = 2;
            else if (xcount < 600) x_val = 3;
```

```
                else x_val = 4;
                if (ycount < 30) y_val = -4;
                else if (ycount < 90) y_val = -3;
                else if (ycount < 150) y_val = -2;
                else if (ycount < 210) y_val = -1;
                else if (ycount < 270) y_val = 0;
                else if (ycount < 330) y_val = 1;
                else if (ycount < 390) y_val = 2;
                else if (ycount < 450) y_val = 3;
                else y_val = 4;
            end
        endcase
    end

    always_ff @(posedge clk_65mhz) begin
        old_clean <= clean;  //for rising edge detection
        if (clean&~old_clean) audio_mode <= ~audio_mode; // audio mode is turned
on when clean goes from 0 to 1
        if (left_counter == FIFTH) begin //every fifth of a second we check to see if
any changes were found to current state
            left_counter <= 0;
            green_found <= 0;
            blue_found <= 0;
            double_found <= 0;
            scrollu_found <= 0;
            scrolld_found <= 0;
            if (green_found) begin // left click takes priority over all other commands
                left_click <= 1;
            end else left_click <= 0;
            if (blue_found&~green_found) begin // only one command will be sent to
the Teensy at a time
                right_click <= 1;
            end else right_click <= 0;
```

```
            if (audio_mode&~blue_found&~green_found) begin
               if (double_found&~scrollu_found&~scrolld_found) begin
                  double_click <= 1;
               end else double_click <= 0;
               if (scrollu_found&~double_found&~scrolld_found) begin
                  scroll_up <= 1;
               end else scroll_up <= 0;
               if (scrolld_found&~double_found&~scrollu_found) begin
                  scroll_down <= 1;
               end else scroll_down <= 0;
            end else begin
               double_click <= 0;
               scroll_up <= 0;
               scroll_down <= 0;
            end
         end else left_counter <= left_counter + 1;
         if (update_counter == UPDATE) begin // every one twenty-fifth of a second a
new mouse movement value will be sent to the Teensy
            update_counter <= 0;
            red_found <= 0;
            if (!red_found) begin
               xcount <= 320;
               ycount <= 240;
            end
         end else begin
            update_counter <= update_counter + 1;
            if ((!red_found) && (rgb[11:8]>4'b1100) && (rgb[7:4]<4'b0100) &&
(rgb[3:0]<4'b0100) && (hcount<640) && (vcount<480)) begin
               xcount <= hcount + 10; // estimating the center of the red LED
               ycount <= vcount + 10;
               red_found <= 1;
            end else if (left_counter != FIFTH) begin
```

```
            if ((rgb[11:8]<4'b0100)&&(rgb[7:4]>4'b1000)&&(rgb[3:0]<4'b1000))
green_found <= 1;
            else if
((rgb[11:8]<4'b0100)&&(rgb[7:4]<4'b1000)&&(rgb[3:0]>4'b1100)) blue_found
<= 1;
            else if (double_out) double_found <= 1;
            else if (scrollu_out) scrollu_found <= 1;
            else if (scrolld_out) scrolld_found <= 1;
         end
      end
      pclk_buff <= jb[0];//WAS JB
      vsync_buff <= jb[1]; //WAS JB
      href_buff <= jb[2]; //WAS JB
      pixel_buff <= ja;
      pclk_in <= pclk_buff;
      vsync_in <= vsync_buff;
      href_in <= href_buff;
      pixel_in <= pixel_buff;
      old_output_pixels <= output_pixels;
      xclk_count <= xclk_count + 2'b01;
      if (sw[3])begin
         //processed_pixels <= {red_diff<<2, green_diff<<2, blue_diff<<2};
         // processed_pixels <= output_pixels - old_output_pixels;
         if
((output_pixels[15:12]>4'b1100)&&(output_pixels[10:7]<4'b0100)&&(output_pix
els[4:1]<4'b0100))begin
            processed_pixels <= 12'hF00;
         end else if
((output_pixels[15:12]<4'b0100)&&(output_pixels[10:7]>4'b1000)&&(output_pix
els[4:1]<4'b1000))begin
            processed_pixels <= 12'h0F0;
```

```
          end else if
((output_pixels[15:12]<4'b0100)&&(output_pixels[10:7]<4'b1000)&&(output_pix
els[4:1]>4'b1100))begin
             processed_pixels <= 12'h00F;
          end else begin
             processed_pixels <= 12'h000;
          end
       end else begin
          processed_pixels =
{output_pixels[15:12],output_pixels[10:7],output_pixels[4:1]};
       end

   end
   assign pixel_addr_out =
sw[2]?((hcount>>1)+(vcount>>1)*32'd320):hcount+vcount*32'd320;
   assign cam = sw[2]&&((hcount<640) &&
(vcount<480))?frame_buff_out:~sw[2]&&((hcount<320) &&
(vcount<240))?frame_buff_out:12'h000;

  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));
   // UP and DOWN buttons for pong paddle
   wire up,down;
   debounce
db2(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnu),.clean_out(up));
   debounce
db3(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnd),.clean_out(down));
```

```
    wire phsync,pvsync,pblank;
    pong_game pg(.vclock_in(clk_65mhz),.reset_in(reset),
                 .up_in(up),.down_in(down),.pspeed_in(sw[15:12]),
                 .hcount_in(hcount),.vcount_in(vcount),
                 .hsync_in(hsync),.vsync_in(vsync),.blank_in(blank),

.phsync_out(phsync),.pvsync_out(pvsync),.pblank_out(pblank),.pixel_out(pixel));

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

    reg b,hs,vs;
    always_ff @(posedge clk_65mhz) begin
        hs <= phsync;
        vs <= pvsync;
        b <= pblank;
        //rgb <= pixel;
        if
((hcount<10)||((hcount>630)&&(hcount<640))||(vcount<10)||((vcount>470)&&(vc
ount<480))||((hcount==320)&&(vcount==240))) rgb <= 12'b1111_1111_1111;
        else rgb <= cam;
    end

//    assign rgb = sw[0] ? {12{border}} : pixel ; //{{4{hcount[7]}}, {4{hcount[6]}},
{4{hcount[5]}}};

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

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

```
endmodule

//////////////////////////////////////////////////////////////////////////////
//
// pong_game: the game itself!
//
//////////////////////////////////////////////////////////////////////////////

module pong_game (
  input vclock_in,       // 65MHz clock
  input reset_in,        // 1 to initialize module
  input up_in,           // 1 when paddle should move up
  input down_in,         // 1 when paddle should move down
  input [3:0] pspeed_in,  // puck speed in pixels/tick
  input [10:0] hcount_in, // horizontal index of current pixel (0..1023)
  input [9:0]  vcount_in, // vertical index of current pixel (0..767)
  input hsync_in,        // XVGA horizontal sync signal (active low)
  input vsync_in,        // XVGA vertical sync signal (active low)
  input blank_in,        // XVGA blanking (1 means output black pixel)

  output phsync_out,       // pong game's horizontal sync
  output pvsync_out,       // pong game's vertical sync
  output pblank_out,       // pong game's blanking
  output [11:0] pixel_out  // pong game's pixel  // r=23:16, g=15:8, b=7:0
  );

  wire [2:0] checkerboard;

  // REPLACE ME! The code below just generates a color checkerboard
  // using 64 pixel by 64 pixel squares.

  assign phsync_out = hsync_in;
```

```
  assign pvsync_out = vsync_in;
  assign pblank_out = blank_in;
  assign checkerboard = hcount_in[8:6] + vcount_in[8:6];

  // here we use three bits from hcount and vcount to generate the
  // checkerboard

  assign pixel_out = {{4{checkerboard[2]}}, {4{checkerboard[1]}},
{4{checkerboard[0]}}} ;

endmodule

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

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

  reg [19:0] count;
  reg new_input;

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

  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

//////////////////////////////////////////////////////////////////////////////////
// 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 clk_in,                // system clock
   input [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

//////////////////////////////////////////////////////////////////////////////////
// Update: 8/8/2019 GH
// Create Date: 10/02/2015 02:05:19 AM
// Module Name: xvga
```

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

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

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

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

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

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

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

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

    vcount_out <= hreset ? (vreset ? 0 : vcount_out + 1) : vcount_out;
```

```
    vblank <= next_vblank;
    vsync_out <= vsyncon ? 0 : vsyncoff ? 1 : vsync_out;  // active low

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

Endmodule

module top_serial(  input           clk_100mhz,
                    input           left_click,
                    input           right_click,
                    input           double_click,
                    input           btnd,
                    input           scroll_up,
                    input           scroll_down,
                    input [7:0]       x_val,
                    input [7:0]       y_val,
                    output logic [1:0]  jc,
                    output logic [2:0] jd
   );

   parameter LEFT = 1; // left click
   parameter RIGHT = 0; // right click
   parameter SCROLL_UP = 2;
   parameter SCROLL_DOWN = 3;
   parameter DOUBLE = 4; // double left click
   parameter UNCLICK = 5; // release left click
   parameter CYCLES = 25000000; // number of cycles corresponding to a quarter
of a second
   parameter UPDATE = 4000000; // number of cycles corresponding to one
twenty-fifth of a second

   logic            clean1; // corresponds to left_click
```

```
    logic           old_clean1;
    logic           clean2; // corresponds to right_click
    logic           old_clean2;
    logic           clean3; // corresponds to scroll_up
    logic           old_clean3;
    logic           clean4; // corresponds to scroll_down
    logic           old_clean4;
    logic           clean5; // corresponds to double_click
    logic           old_clean5;
    logic           up_trigger; // tells the Teensy to scroll_up
    logic           down_trigger; // tells the Teensy to scroll_down
    logic           update_trigger; // tells the Teensy to move the mouse
    logic [24:0] up_counter = 0; // when up or down counter reaches CYCLES, the
Teensy will scroll the mouse by a set value
    logic [24:0] down_counter = 0;
    logic [24:0] update_counter = 0; // when this counter reaches UPDATE, a new
mouse movement value will be sent to the Teensy
    logic [7:0] state; // the current state corresponding to an action that will be sent
to the Teensy

    assign jc[1] = 0;
    assign jd[2] = 0;
    always_ff @(posedge clk_100mhz)begin
        if (update_counter == UPDATE) begin
            update_trigger <= 1;
            update_counter <= 0;
        end else begin
            update_trigger <= 0;
            update_counter <= update_counter + 1;
        end
        old_clean1 <= clean1;  //for rising edge detection
        old_clean2 <= clean2;
        old_clean3 <= clean3;
```

```
        old_clean4 <= clean4;
        old_clean5 <= clean5;
        if (up_counter == CYCLES) begin
            up_trigger <= 1;
            up_counter <= 0;
        end else if (down_counter == CYCLES) begin
            down_trigger <= 1;
            down_counter <= 0;
        end else begin
            up_trigger <= 0;
            down_trigger <= 0;
            if (clean3) begin
                up_counter <= up_counter + 1;
            end else up_counter <= 0;
            if (clean4) begin
                down_counter <= down_counter + 1;
            end else down_counter <= 0;
        end
    end

    always_comb begin
        if (clean1&~old_clean1) begin
            state = LEFT;
        end else if (clean2&~old_clean2) begin
            state = RIGHT;
        end else if ((clean3&~old_clean3)|up_trigger) begin
            state = SCROLL_UP;
        end else if ((clean4&~old_clean4)|down_trigger) begin
            state = SCROLL_DOWN;
        end  else if (clean5&~old_clean5) begin
            state = DOUBLE;
        end else if (old_clean1&~clean1) begin
            state = UNCLICK;
```

```
      end
   end

   debounce my_deb(.clock_in(clk_100mhz),
                   .reset_in(btnd),
                   .noisy_in(left_click),
                   .clean_out(clean1));

   debounce my_deb2(.clock_in(clk_100mhz),
                   .reset_in(btnd),
                   .noisy_in(right_click),
                   .clean_out(clean2));

   debounce my_deb3(.clock_in(clk_100mhz),
                   .reset_in(btnd),
                   .noisy_in(scroll_up),
                   .clean_out(clean3));

   debounce my_deb4(.clock_in(clk_100mhz),
                   .reset_in(btnd),
                   .noisy_in(scroll_down),
                   .clean_out(clean4));

   debounce my_deb5(.clock_in(clk_100mhz),
                   .reset_in(btnd),
                   .noisy_in(double_click),
                   .clean_out(clean5));

   serial_tx my_tx(.clk_in(clk_100mhz),
                   .rst_in(btnd),

.trigger_in((clean1&~old_clean1)|(clean2&~old_clean2)|(clean3&~old_clean3)|(cl
ean4&~old_clean4)|up_trigger|down_trigger|(clean5&~old_clean5)
```

```
                        |(old_clean1&~clean1)),
                .val_in(state),
                .data_out(jc[0]));

    serial_tx x_tx(.clk_in(clk_100mhz),
                .rst_in(btnd),
                .trigger_in(update_trigger),
                .val_in(x_val),
                .data_out(jd[0]));

    serial_tx y_tx(.clk_in(clk_100mhz),
                .rst_in(btnd),
                .trigger_in(update_trigger),
                .val_in(y_val),
                .data_out(jd[1]));
endmodule//top_level

module serial_tx(   input           clk_in,
                input           rst_in,
                input           trigger_in,
                input [7:0]     val_in,
                output logic    data_out);
    parameter   DIVISOR = 868; //treat this like a constant!!

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

    // assign data_out = shift_buffer[0];

    always @(posedge clk_in)begin
        if (rst_in) begin
            count <= 32'b0;
            shift_buffer <= 10'b11_1111_1111;
```

```
            data_out <= shift_buffer[0];
        end else if (trigger_in) begin
            shift_buffer <= {1'b1, val_in, 1'b0};
            data_out <= shift_buffer[0];
        end else if (count == DIVISOR - 1) begin
            count <= 32'b0;
            shift_buffer <= {1'b1, shift_buffer[9:1]};
            data_out <= shift_buffer[0];
        end else begin
            count <= count + 1;
        end
    end
Endmodule

//`default_nettype none
////////////////////////////////////////////////////////////////////////////////
// Audio Portion
// Based on code from Mitchell Gu
// Project Name: Nexys4 FFT Demo
////////////////////////////////////////////////////////////////////////////////

module nexys4_fft_demo (
    input wire CLK100MHZ,
    input clk_104mhz,
    input clk_65mhz,
    input [10:0] hcount,
    input [9:0] vcount,
    input hsync, vsync, blank,
    input wire [15:0] SW,
    input wire BTNU, BTNR,
    input wire AD3P, AD3N,  // The top pair of ports on JXADC on Nexys 4
    output wire AUD_PWM, AUD_SD,
```

```
   output logic [15:0] LED // LEDs above switches                    //Can cut back on
this
   );

//Need this
// ********** BEGIN BASIC IO SETUP
**********//

   // INSTANTIATE SEVEN SEGMENT DISPLAY
   logic [18:0] filler;

// ********** END BASIC IO SETUP
**********//
   //Need this
   wire [15:0] sample_reg;
   wire eoc, xadc_reset;
   // INSTANTIATE XADC IP
   xadc_demo xadc_demo (
      .dclk_in(clk_104mhz),  // Master clock for DRP and XADC.
      .di_in(0),             // DRP input info (0 becuase we don't need to write)
      .daddr_in(6'h13),      // The DRP register address for the third analog input
register
      .den_in(1),            // DRP enable line high (we want to read)
      .dwe_in(0),            // DRP write enable low (never write)
      .drdy_out(),           // DRP ready signal (unused)
      .do_out(sample_reg),   // DRP output from register (the ADC data)
      .reset_in(xadc_reset), // reset line
      .vp_in(0),             // dedicated/built in analog channel on bank 0
      .vn_in(0),             // can't use this analog channel b/c of nexys 4 setup
      .vauxp3(AD3P),         // The third analog auxiliary input channel
      .vauxn3(AD3N),         // Choose this one b/c it's on JXADC header 1
      .channel_out(),        // Not useful in sngle channel mode
      .eoc_out(eoc),         // Pulses high on end of ADC conversion
```

```
        .alarm_out(),          // Not useful
        .eos_out(),           // End of sequence pulse, not useful
        .busy_out()           // High when conversion is in progress. unused.
    );
    assign xadc_reset = BTNR;

//    //Low Pass filter
//    parameter SAMPLE_COUNT = 2082;//gets approximately (will generate
audio at approx 48 kHz sample rate.
//    logic [15:0] sample_counter;
//    always @(posedge CLK100MHZ) begin
//        if (sample_counter == SAMPLE_COUNT)begin
//            sample_counter <= 16'b0;
//        end else begin
//            sample_counter <= sample_counter + 16'b1;
//        end
//    end
//    wire [17:0] filtered_data;
//    fir31 lowpass(.clk_in(CLK100MHZ), .rst_in(BTND_clean), .ready_in(),
.x_in(sample_reg[11:4]), .y_out(filtered_data));

    //Need this
    // INSTANTIATE 16x OVERSAMPLING
    // This outputs 14-bit samples at a 62.5kHz sample rate
    // (2 more bits, 1/16 the sample rate)
    wire [13:0] osample16;
    wire done_osample16;
    oversample16 osamp16_1 (
        .clk(clk_104mhz),
        .sample(sample_reg[15:4]),  //originally .sample(sample_reg[15:4]),
potentially change too .sample(filtered_data[17:6]) if want filter
        .eoc(eoc),
        .oversample(osample16),
```

```
        .done(done_osample16));

    //Need this
    // INSTANTIATE SAMPLE FRAME BLOCK RAM
    // This 16x4096 bram stores the frame of samples
    // The write port is written by osample16.
    // The read port is read by the bram_to_fft module and sent to the fft.
    wire fwe;
    reg [11:0] fhead = 0; // Frame head - a pointer to the write point, works as
circular buffer
    wire [15:0] fsample;  // The sample data from the XADC, oversampled 15x
    wire [11:0] faddr;    // Frame address - The read address, controlled by
bram_to_fft
    wire [15:0] fdata;    // Frame data - The read data, input into bram_to_fft
    bram_frame bram1 (
        .clka(clk_104mhz),
        .wea(fwe),
        .addra(fhead),
        .dina(fsample),
        .clkb(clk_104mhz),
        .addrb(faddr),
        .doutb(fdata));

    //Need this
    // SAMPLE FRAME BRAM WRITE PORT SETUP
    always @(posedge clk_104mhz) if (done_osample16) fhead <= fhead + 1; //
Move the pointer every oversample
    assign fsample = {osample16, 2'b0}; // Pad the oversample with zeros to pretend
it's 16 bits
    assign fwe = done_osample16; // Write only when we finish an oversample
(every 104*16 clock cycles)

    // SAMPLE FRAME BRAM READ PORT SETUP
```

```
    // For this demo, we just need to display the FFT on 60Hz video, so let's only send the frame of samples
    // once every 60Hz. If you want to though, you can send frames much faster, one right after each other.
    // For this 4096pt fully pipelined FFT, the limit is 104Mhz/4096cycles_per_frame = 25kHz (approx)
    // The next two modules just synchronize the 60Hz vsync to the 104Mhz domain and convert it to a 1 cycle pulse.
    wire vsync_104mhz, vsync_104mhz_pulse;
    synchronize vsync_synchronize(
        .clk(clk_104mhz),
        .in(vsync),
        .out(vsync_104mhz));

    level_to_pulse vsync_ltp(
        .clk(clk_104mhz),
        .level(~vsync_104mhz),
        .pulse(vsync_104mhz_pulse));

    //Need this
    // INSTANTIATE BRAM TO FFT MODULE
    // This module handles the magic of reading sample frames from the BRAM whenever start is asserted,
    // and sending it to the FFT block design over the AXI-stream interface.
    wire last_missing; // All these are control lines to the FFT block design
    wire [31:0] frame_tdata;
    wire frame_tlast, frame_tready, frame_tvalid;
    bram_to_fft bram_to_fft_0(
        .clk(clk_104mhz),
        .head(fhead),
        .addr(faddr),
        .data(fdata),
        .start(vsync_104mhz_pulse),
```

```
        .last_missing(last_missing),
        .frame_tdata(frame_tdata),
        .frame_tlast(frame_tlast),
        .frame_tready(frame_tready),
        .frame_tvalid(frame_tvalid)
    );

    //Need this
    // This is the FFT module, implemented as a block design with a 4096pt, 16bit FFT
    // that outputs in magnitude by doing sqrt(Re^2 + Im^2) on the FFT result.
    // It's fully pipelined, so it streams 4096-wide frames of frequency data as fast as
    // you stream in 4096-wide frames of time-domain samples.
    wire [23:0] magnitude_tdata; // This output bus has the FFT magnitude for the current index
    wire [11:0] magnitude_tuser; // This represents the current index being output, from 0 to 4096
    wire [11:0] scale_factor; // This input adjusts the scaling of the FFT, which can be tuned to the input magnitude.
    wire magnitude_tlast, magnitude_tvalid;
    fft_mag fft_mag_i(
        .clk(clk_104mhz),
        .event_tlast_missing(last_missing),
        .frame_tdata(frame_tdata),
        .frame_tlast(frame_tlast),
        .frame_tready(frame_tready),
        .frame_tvalid(frame_tvalid),
        .scaling(12'b110000011101),                    //SCALING IS GIVEN THIS NUMBER FROM EXPERIENCE TO CUT DOWN ON NOISE - otherwise .scaling(SW_clean[15:4])
        .magnitude_tdata(magnitude_tdata),
        .magnitude_tlast(magnitude_tlast),
        .magnitude_tuser(magnitude_tuser),
```

```
        .magnitude_tvalid(magnitude_tvalid));

    // Let's only care about the range from index 0 to 1023, which represents
frequencies 0 to omega/2
    // where omega is the nyquist frequency (sample rate / 2)
    wire in_range = ~|magnitude_tuser[11:10]; // When 11 and 10 are 0, we're on
indexes 0 to 1023

        //The two different thresholds - just for fun in case you want to be change it
    logic [15:0] test_thresh = 16'h32c8;
    logic [15:0] vocal_thresh = 16'h1b58;
        logic [15:0] thresh;

    //Here are the frequency ranges for all ranges
        //The bit sizes can go up to the 12k range - if you want to above increase the
sizes
        //Range for the voices
        logic [9:0] low_bottom_vocal = 10;
        logic [9:0] low_top_vocal = 23;
        logic [9:0] high_bottom_vocal = 25;
        logic [9:0] high_top_vocal = 30;
        logic [9:0] slide_max_lower_vocal = 300;
        logic [9:0] slide_max_upper_vocal = 525;
        logic [9:0] slide_bound_bottom_vocal = 360;
        logic [9:0] slide_bound_upper_vocal = 475;
        logic [9:0] slide_termination_lower_vocal = 400;
        logic [9:0] slide_termination_upper_vocal = 425;
        logic [9:0] scroll_lower_vocal = 333;
        logic [9:0] scroll_upper_vocal = 340;

        //Range for testing - sine generator
        logic [9:0] low_bottom_test = 200;
        logic [9:0] low_top_test = 230;
```

```
logic [9:0] high_bottom_test = 255;
logic [9:0] high_top_test = 280;
logic [9:0] slide_max_lower_test = 300;
logic [9:0] slide_max_upper_test = 525;
logic [9:0] slide_bound_bottom_test = 360;
logic [9:0] slide_bound_upper_test = 475;
logic [9:0] slide_termination_lower_test = 400;
logic [9:0] slide_termination_upper_test = 425;
logic [9:0] scroll_lower_test = 333;
logic [9:0] scroll_upper_test = 340;

//Range variables that I will apply in the actual execution
logic [9:0] low_bottom_true = 200;
logic [9:0] low_top_true = 230;
logic [9:0] high_bottom_true = 255;
logic [9:0] high_top_true = 280;
logic [9:0] slide_max_lower_true = 300;
logic [9:0] slide_max_upper_true = 525;
logic [9:0] slide_bound_bottom_true = 360;
logic [9:0] slide_bound_upper_true = 475;
logic [9:0] slide_termination_lower_true = 400;
logic [9:0] slide_termination_upper_true = 425;
logic [9:0] scroll_lower_true = 333;
logic [9:0] scroll_upper_true = 340;

//Switch variables - change threshold and frequencies
//Will not use - there would be too much interference ie stuff going on
logic [9:0] thresh_low_change;
logic [9:0] thresh_high_change;
logic [9:0] freq_low_change;
logic [9:0] freq_high_change;
```

```
        //Variables for testing - I am leaving them because perfection fake and you
can always improve
    logic [11:0] bin_peak;
    logic [19:0] max_peak;
    logic [19:0] thresh_display;
    logic [11:0] thresh_bin;

        //Variables for audio scrolling
    logic [11:0] scroll_bin = 0;
    logic [11:0] scroll_freeze_bin;
    logic [19:0] scroll_peak;
    logic scroll_flag = 0;

        //Change the threshold value and frequency ranges
        //switches will be used for testing
        //If you change based on frequency use the upper threshold
        always @(posedge CLK100MHZ) begin
                if (SW[14]) thresh = vocal_thresh;                              //If
sw[14] change threshold to use it for your voice
                else thresh = test_thresh;
//Else change to the testing mode

                //A LOT of conditional statements
                if (SW[15])
        //If sw[15] change frequency range for the vocal range
                        begin
                                low_bottom_true = low_bottom_vocal;
                                low_top_true = low_top_vocal;
                                high_bottom_true = high_bottom_vocal;
                                high_top_true = high_top_vocal;
                                slide_max_lower_true = slide_max_lower_vocal;
                                slide_max_upper_true = slide_max_upper_vocal;
                                slide_bound_bottom_true = slide_bound_bottom_vocal;
```

```
                        slide_bound_upper_true = slide_bound_upper_vocal;
                        slide_termination_lower_true =
slide_termination_lower_vocal;
                        slide_termination_upper_true =
slide_termination_upper_vocal;
                        scroll_lower_true = scroll_lower_vocal;
                        scroll_upper_true = scroll_upper_vocal;
                    end
                else
        //Else use the testing mode frequencies
                    begin
                        low_bottom_true = low_bottom_test;
                        low_top_true = low_top_test;
                        high_bottom_true = high_bottom_test;
                        high_top_true = high_top_test;
                        slide_max_lower_true = slide_max_lower_test;
                        slide_max_upper_true = slide_max_upper_test;
                        slide_bound_bottom_true = slide_bound_bottom_test;
                        slide_bound_upper_true = slide_bound_upper_test;
                        slide_termination_lower_true =
slide_termination_lower_test;
                        slide_termination_upper_true =
slide_termination_upper_test;
                        scroll_lower_true = scroll_lower_test;
                        scroll_upper_true = scroll_upper_test;
                    end
        end
        //Finds the absolute peak for every iteration of the FFT
   always @(posedge CLK100MHZ) begin
      if (magnitude_tuser == 0)
         begin
            scroll_peak <= 0;
            scroll_bin <= 0;
```

```
            end
        else if (magnitude_tdata > scroll_peak && magnitude_tdata > thresh)
            begin
                scroll_peak <= magnitude_tdata;
                scroll_bin <= magnitude_tuser;
            end
    end


    always @(posedge CLK100MHZ) begin
        //Isolate and identify frequencies
        if (BTNU)
            begin
            max_peak <= 0;
            bin_peak <= 0;
            end

        else
            begin
            if (magnitude_tuser >= low_bottom_true && magnitude_tuser <=
low_top_true)                                   //Lower frequency
                begin
                thresh_display = magnitude_tdata;
                                                //Use this to see all
magnitude_tdata - will be too fast
                if (magnitude_tdata >= thresh)
                                                //Only consider bins above this
threshold
                    begin

                    if (magnitude_tdata > max_peak)
                                                //If this bin magnitude is larger
than our last greatest bin magnitude proceed
```

```
                begin
                max_peak <= magnitude_tdata;
                                                    //Record bin magnitude
                bin_peak <= magnitude_tuser;
                                                    //Record bin number
                end

            thresh_bin <= magnitude_tuser;
                                                    //Record bin magnitude in a
seperate variable
            LED[0] <= 1;
                                                            //For humans to
confirm this process is happening
            LED[1] <= 1;
            LED[15:2] <= 0;
            end
          else
            begin
            LED[0] <= 0;

            LED[1] <= 0;
            end
          end
        else if (magnitude_tuser >= high_bottom_true && magnitude_tuser <=
high_top_true)                          //Higher frequecy
          begin
                                                                    //Similar to
what is happeing above - consult above if confused
            if (magnitude_tdata >= thresh)
              begin
              LED[15] <= 1;
              LED[14] <= 1;   // double click
              LED[13:0] <= 0;
```

```
                end
            else
                begin
                LED[15] <= 0;
                LED[14] <= 0;
                end
        end
    else if (magnitude_tuser >= scroll_lower_true && magnitude_tuser <=
scroll_upper_true)
        begin
            if (magnitude_tdata >= thresh)
                begin
                    LED[5] <= 1;
                    LED[4:0] <= 0;
                    LED[15:6] <= 0;
                end
            else LED[5] <= 0;

        end
    else if (magnitude_tuser >= slide_max_lower_true && magnitude_tuser <=
slide_max_upper_true)    //Sliding frequency
                        begin
                            if (magnitude_tdata >= thresh)
                                begin
                                    if (magnitude_tuser <
slide_bound_bottom_true || magnitude_tuser > slide_bound_upper_true) //If you
reach the ends of this frequency range cut everything off before we leave this range
                                        begin
                                            scroll_flag <= 0;
                                            LED[4] <= 0;
                                            LED[7] <= 0;
                                            LED[11] <= 0;
                                        end
```

```
                                        else if (magnitude_tuser >=
slide_termination_lower_true && magnitude_tuser <=
slide_termination_upper_true)  //Must turn this frequency range on before you can
scroll
                                            begin
                                                scroll_flag = 1;

                                    //1 - scrolling allowed; 0 - scrolling not
allowed
                                                scroll_freeze_bin = 415;
            //magnitude_tuser
                    //Record the magnitude for the scroll - if necessary
                                                LED[7] <= 1;
                                                LED[4] <= 0;
                                                LED[11] <= 0;
                                            end
                                        else if (scroll_flag == 1 &&
(magnitude_tuser <= slide_termination_lower_true && magnitude_tuser >=
slide_termination_upper_true))

//Scrolling allowed
                                            begin
                                                if (scroll_peak >
scroll_freeze_bin)
                                                    begin
                                                        LED[4] <=
1;                                  // scroll up
        //If you are on the upper side of the bound scroll up
                                                        //LED[11]
<= 0;
//No interference wanted
                                                    end
                                                else LED[4] <= 0;
```

```
                                                    if (scroll_peak <
scroll_freeze_bin)
                                                        begin
                                                            LED[11] <=
1;                                                  // scroll down
                //If you are on the lower side of the bound scroll down
                                                            //LED[4] <=
0;
//No interference wanted
                                                        end
                                                    else LED[11] <= 0;
                                                end
                                            end
                                        end

      end
   end

   // INSTANTIATE HISTOGRAM BLOCK RAM
   // This 16x1024 bram stores the histogram data.
   // The write port is written by process_fft.
   // The read port is read by the video outputter or the SD care saver
   // Assign histogram bram read address to histogram module unless saving
   wire [9:0] haddr; // The read port address
   wire [15:0] hdata; // The read port data
   bram_fft bram2 (
      .clka(clk_104mhz),
      .wea(in_range & magnitude_tvalid),  // Only save FFT output if in range and
output is valid
      .addra(magnitude_tuser[9:0]),       // The FFT output index, 0 to 1023
      .dina(magnitude_tdata[15:0]),       // The actual FFT magnitude
      .clkb(clk_104mhz),  // input wire clkb used to be clk_65mhz
```

```
        .addrb(haddr),     // input wire [9 : 0] addrb
        .doutb(hdata)      // output wire [15 : 0] doutb
    );

    // INSTANTIATE HISTOGRAM VIDEO
    // A simple module that outputs a VGA histogram based on
    // hcount, vcount, and the BRAM read values
    wire [2:0] hist_pixel;
    wire [1:0] hist_range;
    histogram fft_histogram(
        .clk(clk_65mhz),
        .hcount(hcount),
        .vcount(vcount),
        .blank(blank),
        .range(2'b00), // How much to zoom on the first part of the spectrum -
OTHERWISE SW_clean[1:0]!!!!
        .vaddr(haddr),
        .vdata(hdata),
        .pixel(hist_pixel));

    // INSTANTIATE PWM AUDIO OUT MODULE
    // 11 bit PWM audio out is reasonable because otherwise, the PWM frequency
would
    // drop close to the audible and unfiltered range. 11bits -> 104Mhz/2^11=51Khz
    wire [10:0] pwm_sample;
    pwm11 pwm_out(
        .clk(clk_104mhz),
        .PWM_in(osample16[13:3]),
        .PWM_out(AUD_PWM),
        .PWM_sd(AUD_SD));

Endmodule
```

Teensy Code:

```
// set this to the hardware serial port you wish to use
#define HWSERIAL Serial1
#define HWSERIAL2 Serial2
#define HWSERIAL3 Serial3

void setup() {
  // put your setup code here, to run once:
  HWSERIAL.begin(115200); // will receive the current action for the mouse to
perform
  HWSERIAL2.begin(115200); // receives the x value for mouse movement
  HWSERIAL3.begin(115200); // receives the y value for mouse movement
}

void loop() {
  // put your main code here, to run repeatedly:
  int8_t incomingByte; // the current state of the system (aka action to be
performed)
  int8_t incomingByte2;
  int8_t incomingByte3;
  if (HWSERIAL.available() > 0) {
    incomingByte = HWSERIAL.read();
     // for testing purposes
     Serial.print("UART received: ");
     Serial.println(incomingByte, DEC);
    //HWSERIAL.print("UART received:");
    // HWSERIAL.println(incomingByte, DEC);
    if (incomingByte == 0) { // corresponds to right click
      Mouse.set_buttons(0, 0, 1);
      Mouse.set_buttons(0, 0, 0);
    }
    else if (incomingByte == 1) { // pressing down left button
```

```
    Mouse.set_buttons(1, 0, 0);
   }
   else if (incomingByte == 2) { // scrolling up
    Mouse.scroll(1);
   }
   else if (incomingByte == 3) { // scrolling down
    Mouse.scroll(-1);
   }
   else if (incomingByte == 4) { // double left click
    Mouse.click();
    Mouse.click();
   }
   else if (incomingByte == 5) { // releasing the left button (and also the right
button)
    Mouse.set_buttons(0, 0, 0);
   }
  }
  if (HWSERIAL2.available() > 0 && HWSERIAL3.available() > 0) { // will only
move the mouse when both x and y values are received
    incomingByte2 = HWSERIAL2.read();
    incomingByte3 = HWSERIAL3.read();
    Mouse.move(incomingByte2, incomingByte3);
    // for testing purposes
    // Serial.print("UART X received: ");
    // Serial.println(incomingByte2, DEC);
   // Serial.print("UART Y received: ");
    // Serial.println(incomingByte3, DEC);
  }
}
```