Touch Projector

Alan Cheng
Hope Harrison

13 December 2017

Contents

Chapter 1

Project Overview

We present an unique HMI device: the touch projector (Fig. 1.1. This projector allows users to interact with any projected image as if it were a touchscreen. Specifically, we present a hardware-focused approach to creating such device, which utilizes a FPGA to provide a fast and consistent operation. Our prototype consists of a projector, Kinect, and a FPGA. The Kinect provides depth-level sensing using a time of flight camera, which can be used to determine finger touchpoints. The FPGA provides fast image processing on the data from the Kinect, as well as acting as a HID-compliant mouse. This allows for our device to be used with any arbitrary computer, as the control is done at the OS level. Thus, users can simply touch the surface they are projecting onto, and the computer receives a click at the point they touched. Examples applications of this device are:

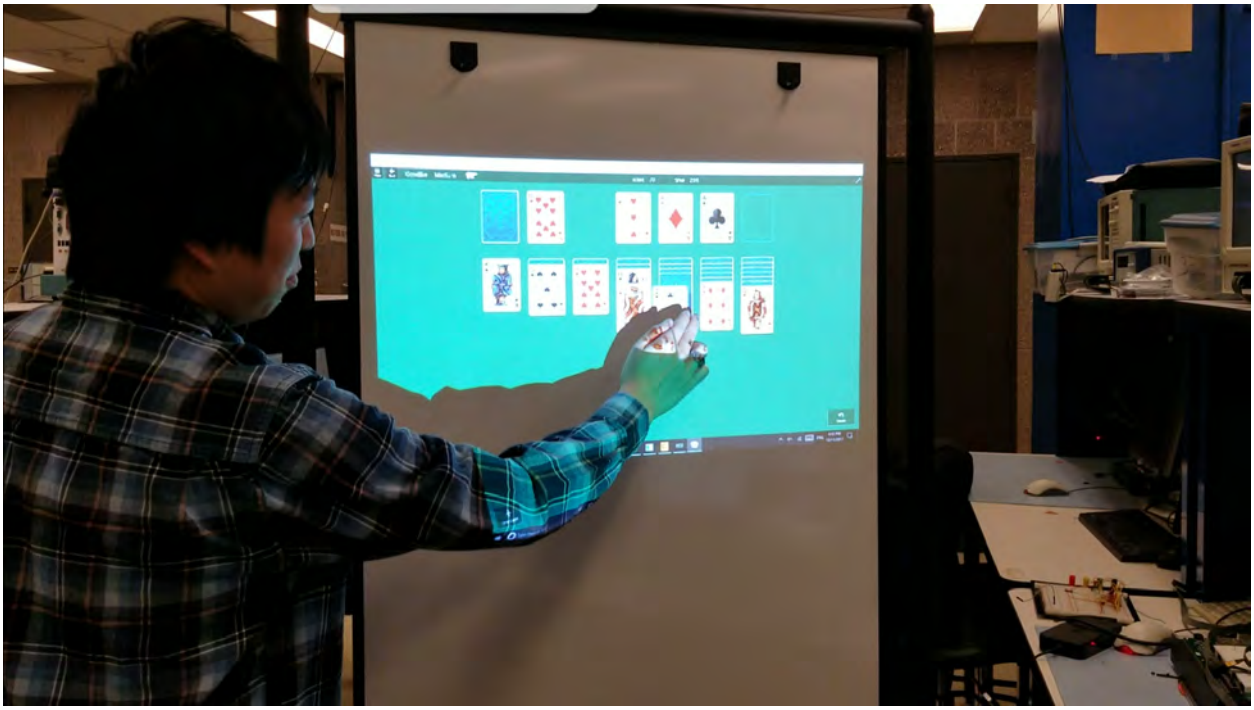


Figure 1.1: One of the authors, Alan Cheng, using the touch projector

- Education - allows instructors to give interactive presentations
- Business - creating a giant collaborative workspace akin to a smart whiteboard
- Entertainment - combinations with AR and/or VR tech to enable more personal interactive experiences

This project is divided into three technical main modules:

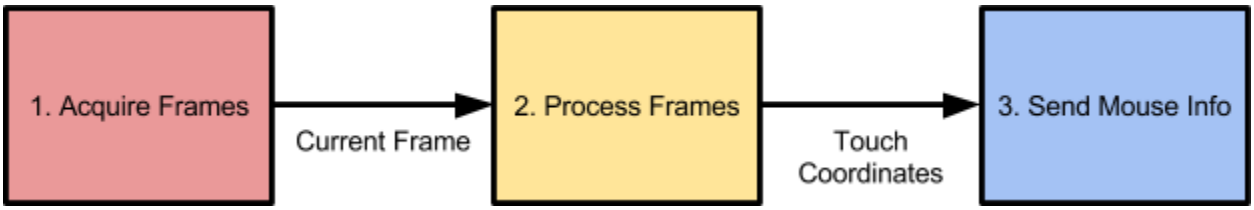


Figure 1.2: Process breakdown for project

1. Acquire Frames: Kinect acquires depth-image frames and sends them to the FPGA.
2. Process Frames: FPGA processes image frames to calculate the coordinates where the user is touching the surface. Thresholding, noise reduction, and edge detection is needed.
3. Send Mouse Info: Coordinates of the touch as well as whether it was just pressed or just released are transformed to the relative size of the monitor and send as a HID-compliant mouse signal.

Figure 1.3 gives a more descriptive overview of the system architecture, showing the subdivision of each process into hardware. The Acquire Frames module deals with the Kinect to PC connection, the Process Frames module deals with FPGA processing, and the Send Mouse Info module deals with FPGA to Teensy to PC connection. We aim to focus our project into FPGA processing, rather than convention software options, for faster runtimes. The follow chapters deal with a more descriptive overview of each process.

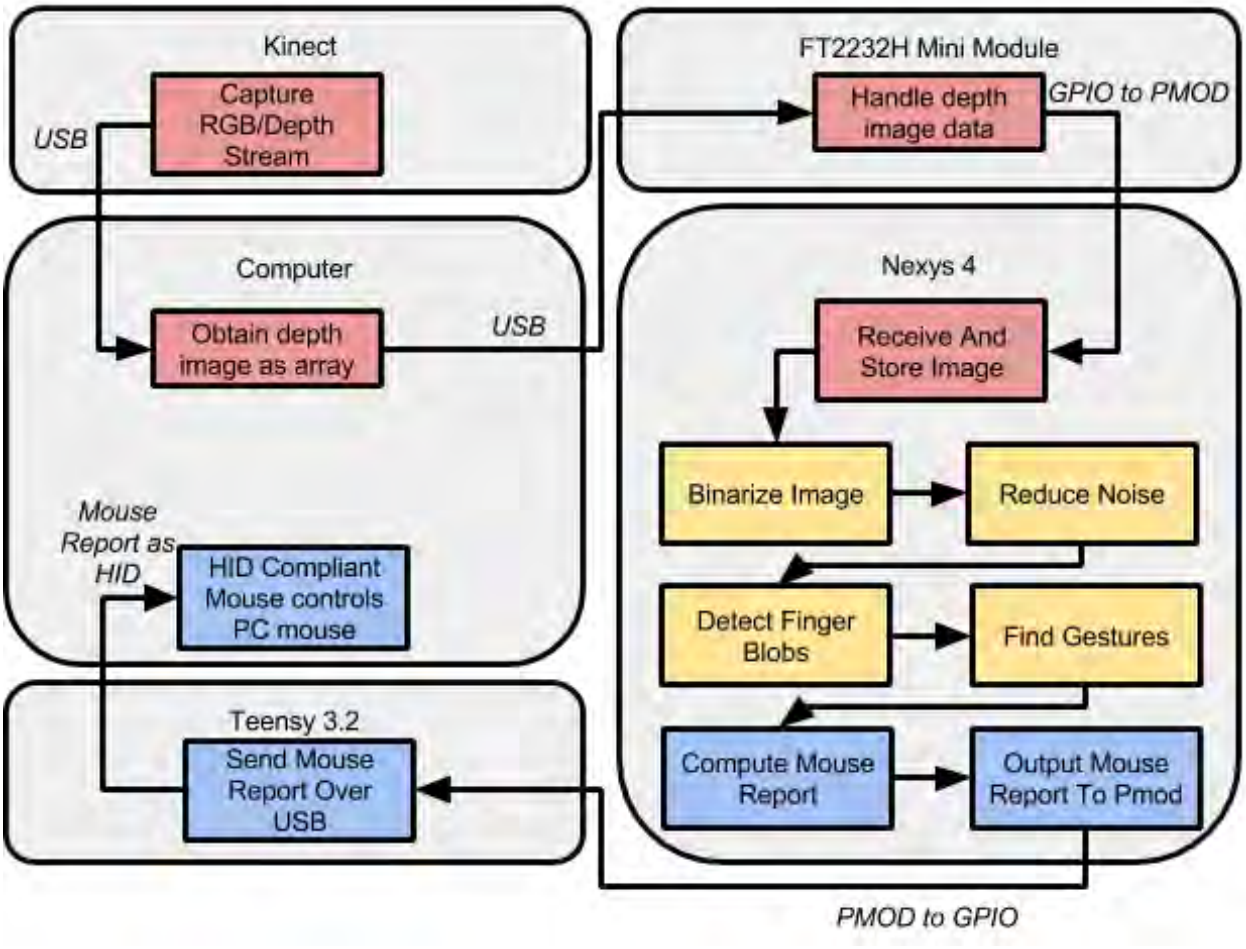


Figure 1.3: System architecture level design of project

Chapter 2

Process: Acquire Frames – Alan

In order to detect finger contact with a surface, the distance of a finger is needed. Using RGB data calibrated to detect fingers blobs of a certain size is a possible method, but can be plagued by a multitude of image properties such as lighting. Using a depth image from a depth-sensing camera reduces the impact of external image changes, and provides an easier interface. Thus, we use the Kinect 2.0, with its time of flight depth camera, to determine finger touchpoints.

There are no openly available drivers that allow for direct connection of the Kinect to the Nexys4 board (or any other FPGAs for that matter). To provide a bridge between Kinect and FPGA, we utilize a computer to handle the connection. Specifically, we use Kinect PC drivers coupled with OpenCV to acquire the depth image stream, and we send each frame at 30 frames per second(fps) to the Nexys via FTDI's FT2232H Hi-Speed Dual USB UART/FIFO IC.

2.1 Depth Frame Acquisition (Software)

Both Micosoft SDK 2.0 and libfreenect2 where used independently to acquire data stream. Microsoft's SDK is officially supported, and perhaps is the more polished library. However, the SDK is limited to use only on the Visual Studio platform. Moreover, the depth image acquired is not the raw depth image, as Microsoft applies both bilateral filtering and edge-aware filtering to the image before it can be read from.

In comparison libfreenect2 is a 3rd party open source cross-platform driver. It isn't as polished, nor is the documentation as available, but it does allow more control to the Kinect settings. Specifically for this project, the bilateral filtering and edge-aware filtering can be disable, so the raw depth frame can be used capture.

In this project, both drivers were used independently. Equivalent code for can be found in the Appendix: Microsoft's SDK 2.0, libfreenect (with OpenCL). Both work at an approximately equal frame rate.

2.2 Data Transmission

2.2.1 FT2232H Configuration

The Kinect 2.0 captures depth frames of resolution 512x424 at 30fps, with depth values in the range of 0 to 4500 mm having high precision. To maintain the depth data without loss, we must be able to supply a 104.2 Mbit/s data transmission rate. This is because we need to send a data bus of 512×424 pixels, each of size 2 bytes to cover the range of 0 to 4500. Options for direct connection between PC and the Nexys4 are either too slow (Serial Port) or too complex (Ethernet). We opted for using the FT2232H to provide fast data transmission (it supports up to 480 Mbit/s) yet reasonable difficulty in use. Using the FTDI D2XX driver as well as OpenCV, in a x86 VC++ application we can send the depth data (see Appendix for code).

The FT2232H supports several modes of operation. However, only the synchronous FIFO mode supports up to the 480 Mbit/s transmission rate. Using and outputting a 60 MHz clock, the FT2232H outputs 1 byte of data at the given clock rate. Using FTDI's configuration software FT_Prog (Fig 2.1), and the FTDI driver D2XX (Fig 2.2), the FT2232H can be configured to these settings.

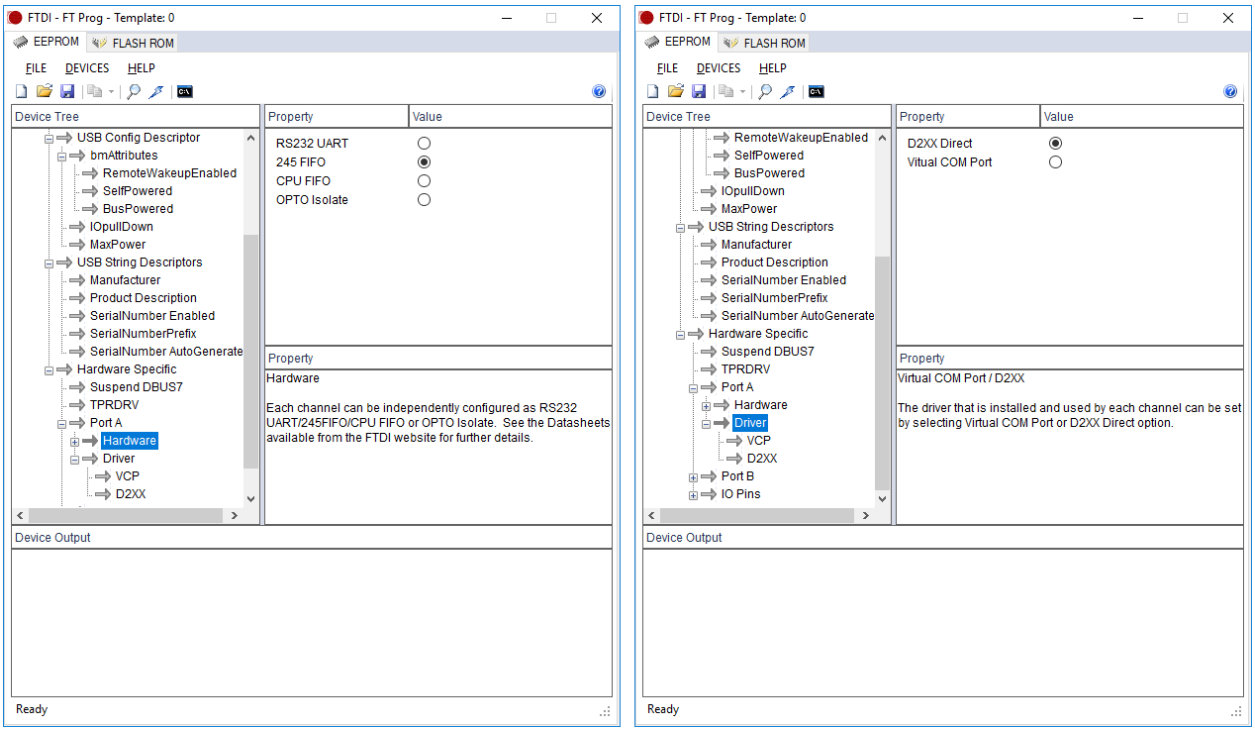


Figure 2.1: FT_Prog configuration settings for synchronous FIFO mode

FT_Prog Configuration

- PortA/B: Hardware = 245 FIFO
- PortA/B: Driver = D2XX

```
UCHAR mask = 0xff;
UCHAR mode;

mode = 0x00; // reset mode
ftstatus = FT_SetBitMode(fthandle, mask, mode);

Sleep(10);

mode = 0x40; // synchronous FIFO mode
ftstatus = FT_SetBitMode(fthandle, mask, mode);
```

Figure 2.2: D2XX C code to configure FT2232H to synchronous FIFO mode

D2XX Configuration

The wiring of the FT2232H is shown in figure A. We must connect the corresponding VCCIO pins to connect the board to USB power. Outputs labeled are then connected to the Nexys4 via 2 PMODs. PMOD JB corresponds to the data handling signals: CLK, RXF, WR, RD, and OE. PMOD JC corresponds to the sent byte of data AD[7:0]. **It should be noted that the resistors on the PMOD JC of the Nexys4 were removed to support the high rate of data transfer. Moreover, the ground signals of the PMODs should be connected to the same ground as the FT2232H**. Short physical connections should be done in order to reduce noise on these signals. For instance, with a long wire, the clock signal becomes noisy and unusable. A physical protoboard was used for the FT2232H (see Appendix A

- CLK – 60 MHz clock output
- RXF – Data received signal (active low: 1 = no data in FIFO, 0 = data in FIFO)
- WR – Write enable input (active low: 1 = disables write, 0 = write data to FIFO from data pins)
- RD – Read enable input (active low: 1 = disables read, 0 = read data from data pins)
- OE – Output enable input (active low: 1 = disables output, 0 = transfer data in FIFO to data pins)

RXF goes low when there is data sent over to the FIFO from the USB. In order to read the data, we must first set OE low to move data from the FIFO to the output pins AD[7:0]. Then, one clock cycle later, we set RD low to read from those pins. FTDI's documentation depicts the process in Fig. 2.4. The module "FT2232H_Control" handles this functionality, reading in signals CLK and RXF, while outputting signals WR, RD, and OE.

2.2.2 Image Storage and Thresholding

In order to detect finger touches on a surface, we first store a depth frame of the background in memory. Then, with each live incoming frame, we can compare the depth values between

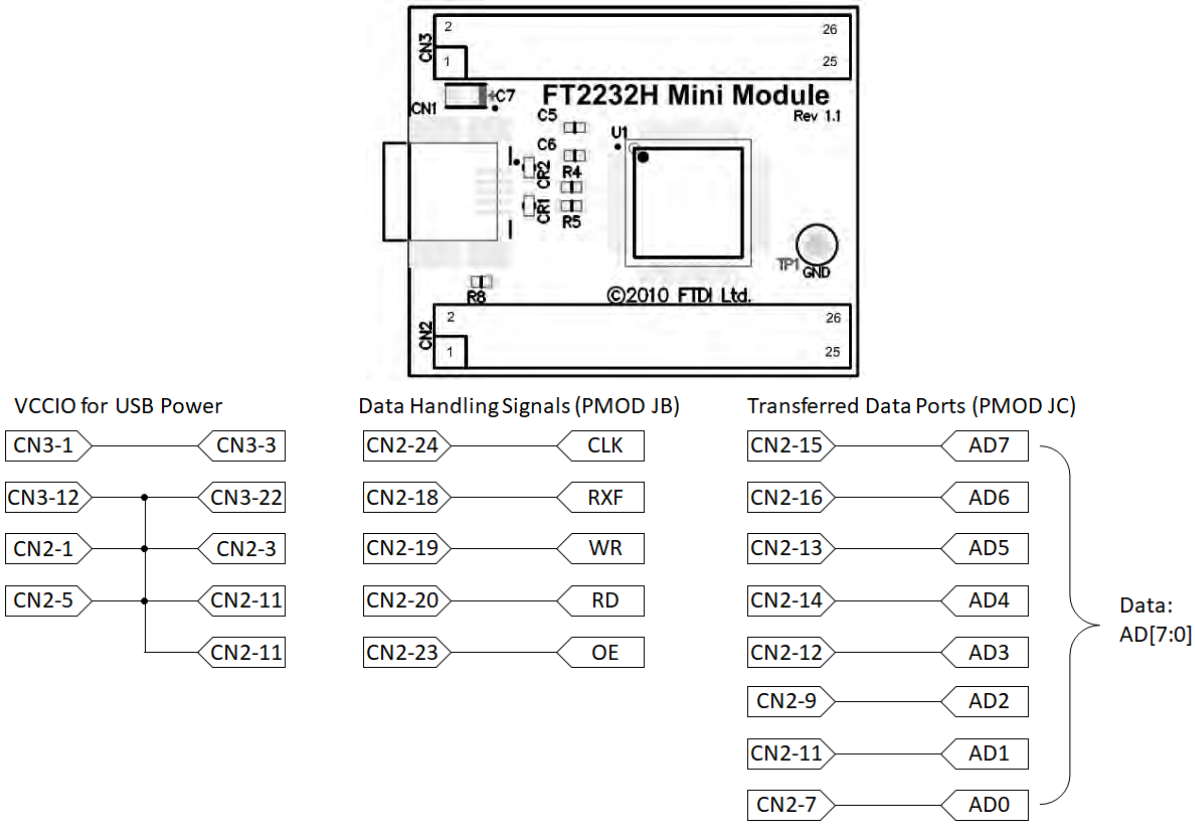


Figure 2.3: Wiring for FT2232H Mini Module

the two. Any object touching, or close to touching the surface will have a depth value around 0 to 20mm less than the stored background. In implementation, this means that we can receive a binarized image by subtracting the current frame depth values from the stored background, where pixels within a set difference range is set to white, and others are set to black. Fig. 2.5 shows the output of this technique.
There are several points to notice in the image:

1. The noise in an approximately rectangular frame is from the loss in accuracy of the background. In practice, this area is outside the project image and can be ignored.

2. The hand/arm outline is present. This outline effect is a result of the Kinect's time of flight depth camera, which reads a gradiant of depth values between the abrupt edge of the hand and air. Filtering is applied to remove the outlines, and will be discussed later.

In terms of FPGA memory, this means we have to have storage for both the background frame and the current received depth frame. The Nexys has 4860 kbits of fast BRAM storage, of which 3473 kbits are allocated to this background frame. We cannot store two of such images in BRAM, so for the current frame we only store its binarized image which takes 217 kbits. This is handled by the module "ram_control".

This module contains 3 modes of operation: IDLE, WRITING, and UPDATING.

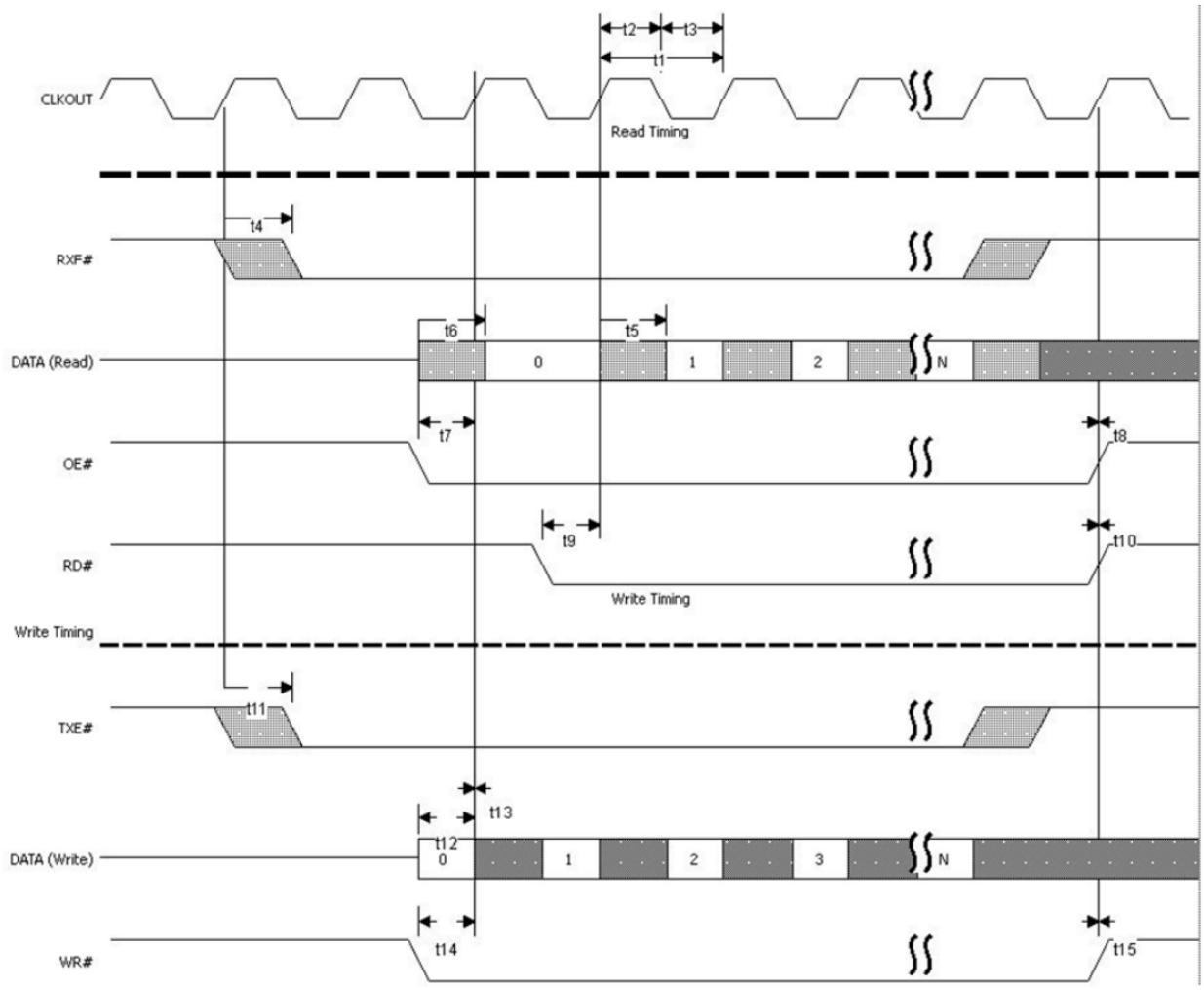


Figure 2.4: Wiring for FT2232H Mini Module

1. IDLE - There is no depth data sent by the PC to the FT2232H yet, so the module stays idle

2. WRITING - There is new depth data, and there is a signal to write, so the depth frame is stored as the background frame in BRAM

3. UPDATING - There is new depth data, but there is no signal to write, so the depth frame is read and thresholded with the existing background frame, and the resulting binarized frame is stored in BRAM.

A simple state machine for these modes of operations are described in Fig. 2.6. Three variables are used for state transition: write, new_frame, and ready. Write is connected to a pushbutton (BTND) that acts as the write signal. Thus, when a user presses the button, the module will stored the incoming frame as the background frame in BRAM. new_frame signals high where there exists a new depth frame to be read. Each new frame is first preceded by a specific code of DDCCBBAA. Module detect_start sets signal new_frame high when the

Figure 2.5: Thresholded frame of finger touching a surface

coding sequence is found. Finally, ready signifies when the incoming depth frame has fully been written to either the background or binary frame BRAM.

Operation in states WRITING and UPDATING are almost identical, as both needs to read a full depth frame from the FT2232H. These states are handled in the module frame_mem_control. Recall that each pixel is encoded by 2 bytes of data. Since the FT2232H can only output 1 byte of data per clock cycle, it takes 2 clock cycles to read a pixel value. Thus, in order to reconstruct the full depth value, we store the previous read value as well, combining both the current and previous value to form the depth value. Starting from new_frame signal, this module increments an index corresponding with the current pixel whose values are being read. After two clock cycles of read data, the index increments. Moreover, another register data_state keeps track of whether we are reading the first or second value of a depth value. If we are reading the second value (data_state == HI), then we can combine the current with the previous value to get our depth value. This depth either gets written to a dual port BRAM for the background frame (in state WRITING), or compared with the corresponding background frame value at the current index to return a single bit with 1 representing white and 0 representing black. This single bit value is written to a separately dual port BRAM for the current binarized frame. Finally, this module sets ready to high when the entire frame is ready, that is when the index exceeds the last pixel index of 512×424. Fig. 2.7 demonstrates the relation between index and read data.

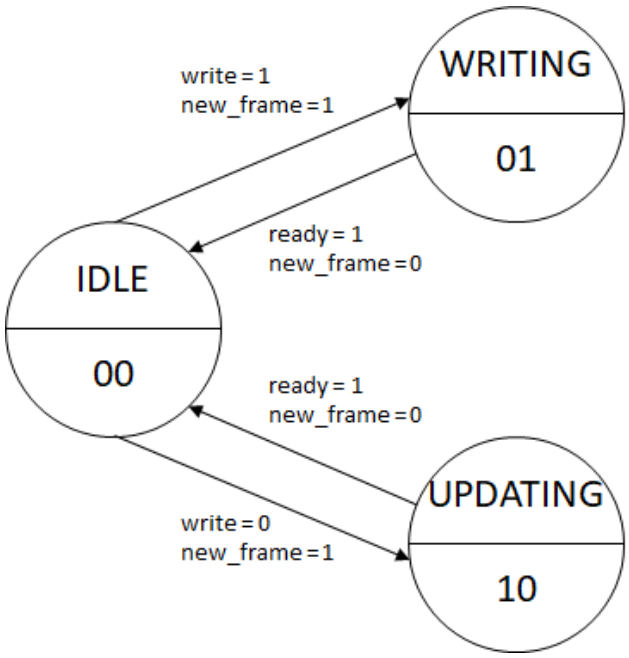


Figure 2.6: FSM describing the modes of operation of ram_control

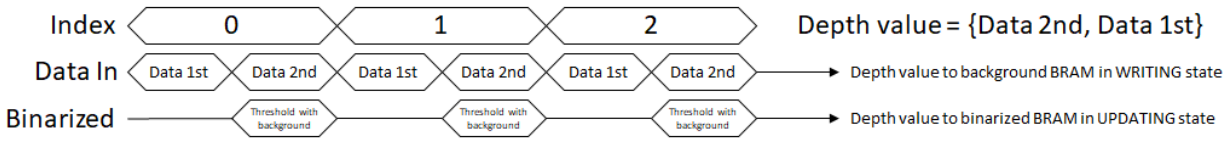


Figure 2.7: Signal propagation of index, data, and thresholded value

Chapter 3

Process Frames

3.1 Reduce Noise – Hope

After binarization, our image is represented as a series of pixels which are either 0 (black) or 1 (white). We then filter out any sequences of 1s which are too small to correspond to a finger touch, and generally smooth the image. The filtering is done by the opening morphological operator. first applying an erosion kernel where we set all the pixels to 0 unless every pixel directly surrounding them (in a 3x3 square) is 1. This will make any small blobs disappear. We then apply a dilation kernel where any pixel that has a direct neighbor that is 1 also becomes 1.

We originally planned on storing the entire image as a 2D 512×424 register and then applying the kernel directly for each pixel in the image. This worked well at first for small images and it was easy to implement because we could read any pixel value at any time and set each pixel value easily.

Unfortunately, this did not work on larger images, because of compile times. For synthesis and routing, dealing with 512×424 different wires would take a excruciatingly long time to compile. To make compilation times reasonable, we limit data storage (after reading from BRAM) to a 512×3 sized register. This sized register is enough to perform kernel operations on one row of image data. Thus, our current method is the following: We first populate a 512 x 3 register with the top 3 pixels of our image. Then while we apply the erosion or dilation kernel to these pixels, we populated a buffer of the next row down. After finishing the first three rows, the second row is moved up into the first slot, the third row into the second, and the next row into the third. This continues so that we always have a three row window to look at.

Opening worked very well for the images we received from the Kinect. After binarization, the image still shows a thin outline around the hand and arm, rather than just a finger. Figure 3.1 shows an example of this. It is very undesirable because it makes it difficult to differentiate just the fingers. But because the outlines are so thin, the dilation almost completely erases them and does leave us with only the denser fingertip, as seen in figure 3.2, which is the same image as before, but with the erosion and dilation applied.

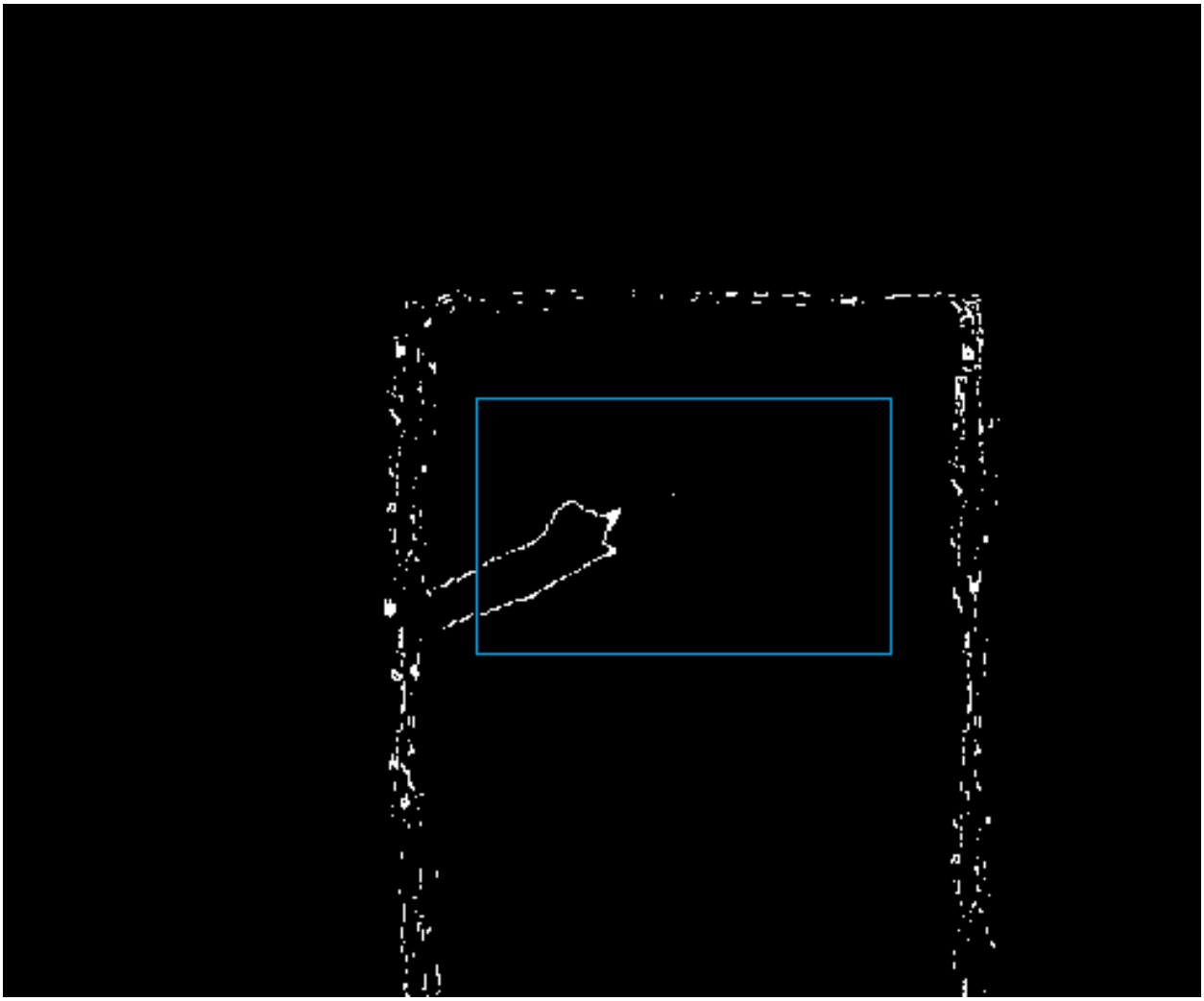


Figure 3.1: The fingertip and outline of hand after binarization, where the blue rectangle signifies the projection surface. *Note: this is an artistic replication of actual image data, as there wasn't an easy way to save live image frames from the FPGA*

3.2 Detect Finger Blobs – Hope

Because the noise reduction through erosion and dilation cleaned the image so well, the blob detection was not as difficult as it otherwise might be. The algorithm we use for blob finding is: once we find a pixel with a value of 1, set the current blob bounds to the coordinates of that pixel. For each other pixel in the image, check if it is within some set threshold of the current blob bounds, and if so, expand the blob bounds to include this pixel. Continue until we have the bounds of the entire blob of densely packed pixels. We then check if the bounds for this blob are the right size for a finger tip, and if so set the blob as valid.

We have the capacity in our algorithm to set the current blob to 0 if it is not valid (because it's either too big or too small) and then continue looking for another blob in an attempt to find the real fingertip, however in practice, this did not prove helpful or necessary. There are always a lot of small blobs around the edge of the screen – searching through these

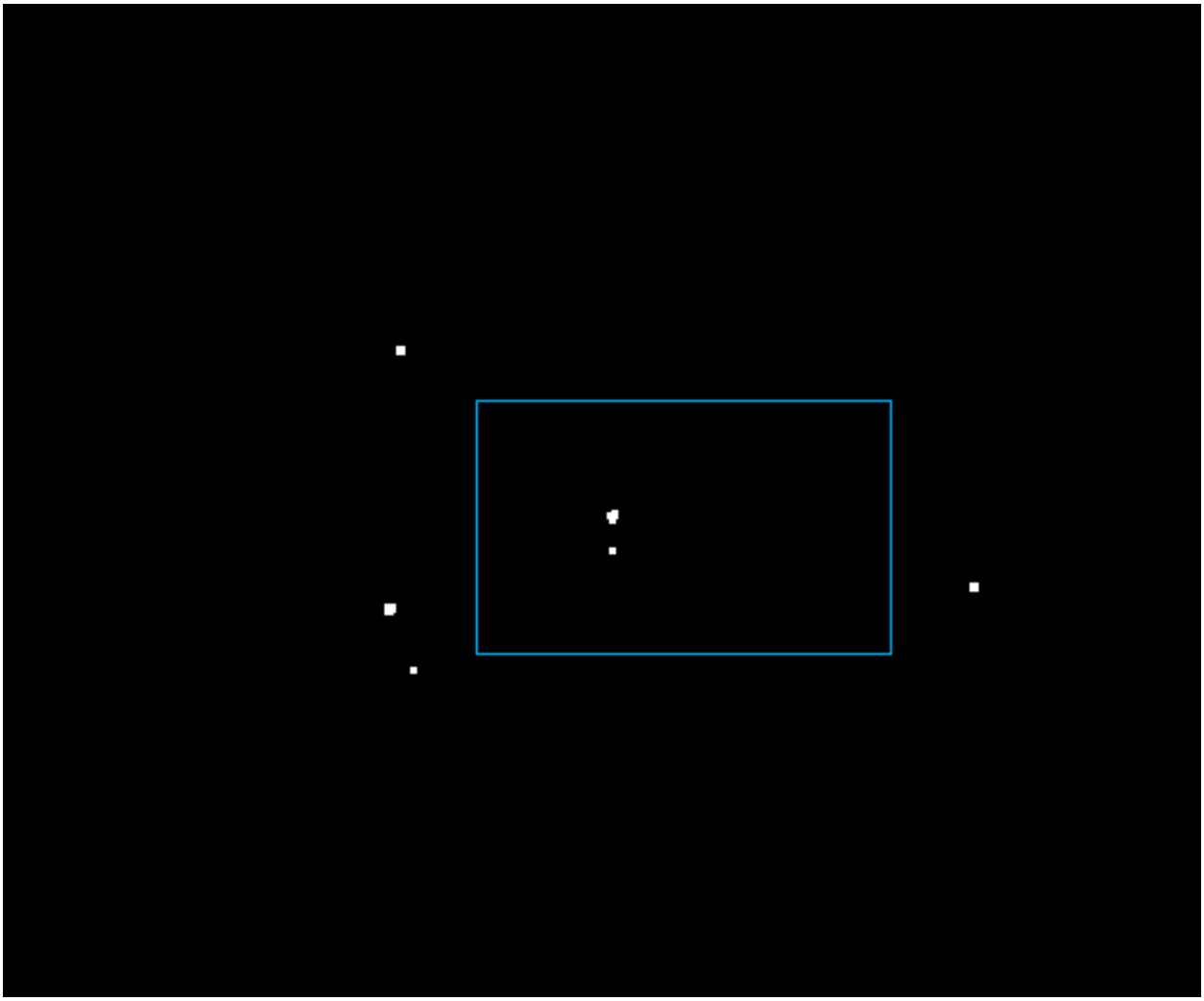


Figure 3.2: Resultant binary image after performing the opening morphological operation, where the blue rectangle significes the projection surface. *Note: this is the result of opening done in software on the previous image*

blobs would have just been a waste of time and would slow our system down. But in the center of the image where the screen is, there are rarely any extraneous blobs other than fingertips. So we instead confine our blob search to set coordinates within the middle of the frame, and this works well for us.

Currently, the coordinates for the blob search must be set manually using switches on the Nexys board. The user should set these coordinates to align with the bounds of the projected image. In the future, an autocalibrate option could be added by looking at color data rather than just depth data to find the edges of the screen, but we were not able to implement that at this time.

3.3 Find Gestures – Alan

We planned to be able to detect multiple unique gestures such as pinch to zoom. However, given time constraints, this was never achieved. However, it is important to determine finger press, finger move, and finger release gestures that equate to mouse clicks, which is handled in module gesture_detect.

Finger press is defined as the first instance where a finger touch is recognized by the blob detection as described in earlier sections. This translate to a mouse click on the PC. Finger moved is defined as the instance when a finger is moving around on the surface after the initial mouse click. Finger release is defined as the instance when a finger leaves the surfaces. This translates to the release of a mouse click on the PC. However, because of problems with noise and inconsistent framerates caused by the image processing modules, it is possible for some finger touch to be not recognized while the finger is still physically touching the surface. Thus, we implemented a buffer to allow frames with dropped finger recognition. Given a set maximum frame timeout value (10 was used for demonstration), the module will not signal mouse release until the number of frames counted sequentially without a finger presence reaches the timeout value.

Chapter 4

Send Mouse Info – Hope

In order to control the computer by touching the projection, we must have our device appear as an HID mouse to the computer. Since the Nexys 4 is only able to function as a USB host (but not slave), we are not able to output directly from the Nexys 4 to the computer. Therefore, we will use a Teensy as an intermediate.

4.1 Computing Mouse Report

Using the data from the gesture report of whether the user has pressed, released or moved, and the coordinates of their finger, we run a state machine which sets these parameters one after another to be sent over a serial connection to the Teensy. There are 15 bytes of data which must be sent:

- X coordinate of press (2 bytes)
- Y coordinate of press (2 bytes)
- X minimum bound (2 bytes)
- Y minimum bound (2 bytes)
- X maximum bound (2 bytes)
- Y maximum bound (2 bytes)
- Is pressed? (really 1 bit, but sent as 1 byte for simplicity)
- Is released? (really 1 bit, but sent as 1 byte for simplicity)
- Is moved? (really 1 bit, but sent as 1 byte for simplicity)

This module makes sure that these numbers are sent in this order (1 byte at a time) so that the Teensy can read them in the same order and know which is which. When the pmod output module indicates that it has finished sending a number, the Mouse Report sets the next number to be sent.

4.2 Output Mouse Report to Pmod

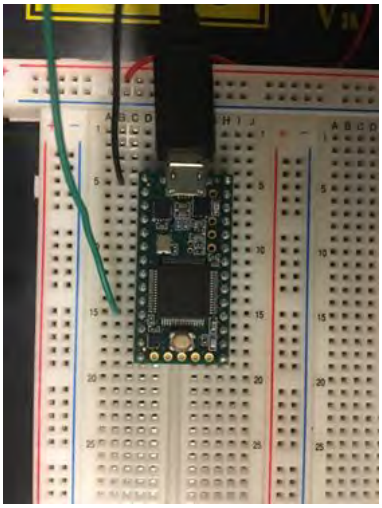


Figure 4.1: The Teensy with pin 9 wired to JA[0]

The pmod output module sends whatever parameter it is currently being fed to the Teensy. We use a 115200 baud rate, and send on JA[0] which is connected to pin 9 on the Teensy (fourth pin from the bottom on the left in figure 4.1). This module functions as a normal serial sender, except that it also gives a pulse to indicate when it has finished sending a 1 byte number so that the Mouse Report Module will know when to feed it a new number to send.

4.3 Send Mouse Commands Over USB

The Teensy uses a built in serial driver to receive the data from the digital output of the Nexys. As it receives the 15 bytes it uses the order to determine the values of all the parameters in the mouse report. It uses the minimum and maximum bounds as well as the absolute coordinates of the press and the resolution of the monitor of the computer it is connected to (this must be pre-set) to calculate the coordinates to move the mouse to. The reason the absolute coordinates and the bounds are sent to the Teensy rather than just the relative position within the bounds is to avoid doing division on the FPGA. The is_moved? is_pressed? and is_released? parameters control whether the Teensy actually sends a move command to those coordinates and whether that also includes a press and a release of the left mouse button.

Chapter 5

Conclusion and Future Work

Through this project, we show a working prototype of a touch project working primarily on FPGA hardware. We were able to stream high-speed depth data from the Kinect to the PC to the FPGA using the FT2232H at around 30fps for a 512 by 424 resolution image. We also shown image processing modules to remove noise in the resulting thresholding depth image, and were able to detect key blobs in the processed image. Finally, we were able to send commands for the Teensy to act as a HID compliant mouse for the computer. Because of this interface, we were able to run a variety of different applications, such as Fresh Paint, Solitare, Chess, and Osu!.

There are a plethora of problems with the prototype presented. Two key problems will be discussed on.

1. Performance - although we transfer data at 30 fps, the image processing module takes longer to complete. Therefore, the real framerate probably lies around 10-20 fps. We can improve accuracy and latency of the device by reducing the runtime of the image processing algorithm. One method would be to switch to a highly parallel system consisting of many modules that each represent a pixel. If a module knows the value of itself and it's neighbors, we could do 3x3 kernel operations in one clock cycle, significantly cutting down the runtime.

2. Accuracy - there is very low accuracy between finger touchpoint and the resulting mouse location. This is because the fingertip itself its really found, but rather the finger joint that protrudes out slightly farther from the surface background. Perhaps it would be useful to consider the skeletal composition of the hand. Furthermore, movement is rather jagged, and can be fixed with filters. Alan has previously worked on a hardware based implementation of the Kalman Filter using Faddeev's algorithm that may be helpful.

Chapter 6

Informal Remarks

Alan:

Driver work and data transmission is hard. Do not underestimate the time it takes to implement this, despite how boring or easy it looks to be. I anticipated completing this module in 1-2 weeks, but in actuality it ended up taking all the way up to the last 4 days before it was completely finished. And I guess what goes with that is to plan your timeline well, and leave a lot of buffer space of mistakes. Also, make sure that your measuring tools are plugged in properly. I have definitely wasted time trying to debug a scoping issue when it was just that the probe was plugged backwards.

In regards to the project as a whole, I am glad that we were able to get a working prototype. This touch projector idea has been a big part of my personal projects/interests throughout the years, and I am always glad to let other people experience it. As for the technical achievements, I think there is a lot to be desired. To be honest, I would have much prefer to work on interesting applications or utilities of the touch projector rather than driver work, but what must be done must be done.

Lastly, I am always look for people interested in working on this idea, or for people who would perhaps like to see this spun into a start-up type idea. Please contact me if you are interested!

Hope:

This project was challenging, yet also fun. It did however have a lot of frustrations. As advice to future students of 6.111, we would recommend that if you have any issues that you think might be a problem later on, try to bring them up and get ideas early on. For example, with the image processing, I (Hope) knew it took a long time to compile and could have realized earlier that we would not be able to hold the entire image in a register once it got bigger than the small images I used to test. If I'd addressed this earlier, the last week would not have been as stressful. Another good resource for debugging when you are at that stage is making test benches. At first I usually just debugged by trying my code and hoping it worked, or maybe outputting some signals to LEDs to look at them and try to see why it's not working. But when my partner Alan gave me the idea to make a testbench it made the debugging easier because I could probe all the signals and see everything that was going on. So it's really not just something to use when required in the labs, it's actually a tool that will help you a lot! In the end, though, the most important is to have fun while working hard and remember what you'll accomplish. We were really happy to see our project work

and get to try playing solitaire on just an ordinary whiteboard with a projector!

Chapter 7

Acknowledgements

We would like to thank our project mentor, Joe Steinmeyer, who provided us with the FT2232H, modified Nexys4 Board, and the multiple projectors. He went out of his way to help our struggling project throughout the most stressful times. More importantly, he provided us with invaluable advice that helped us make our way to the final prototype.

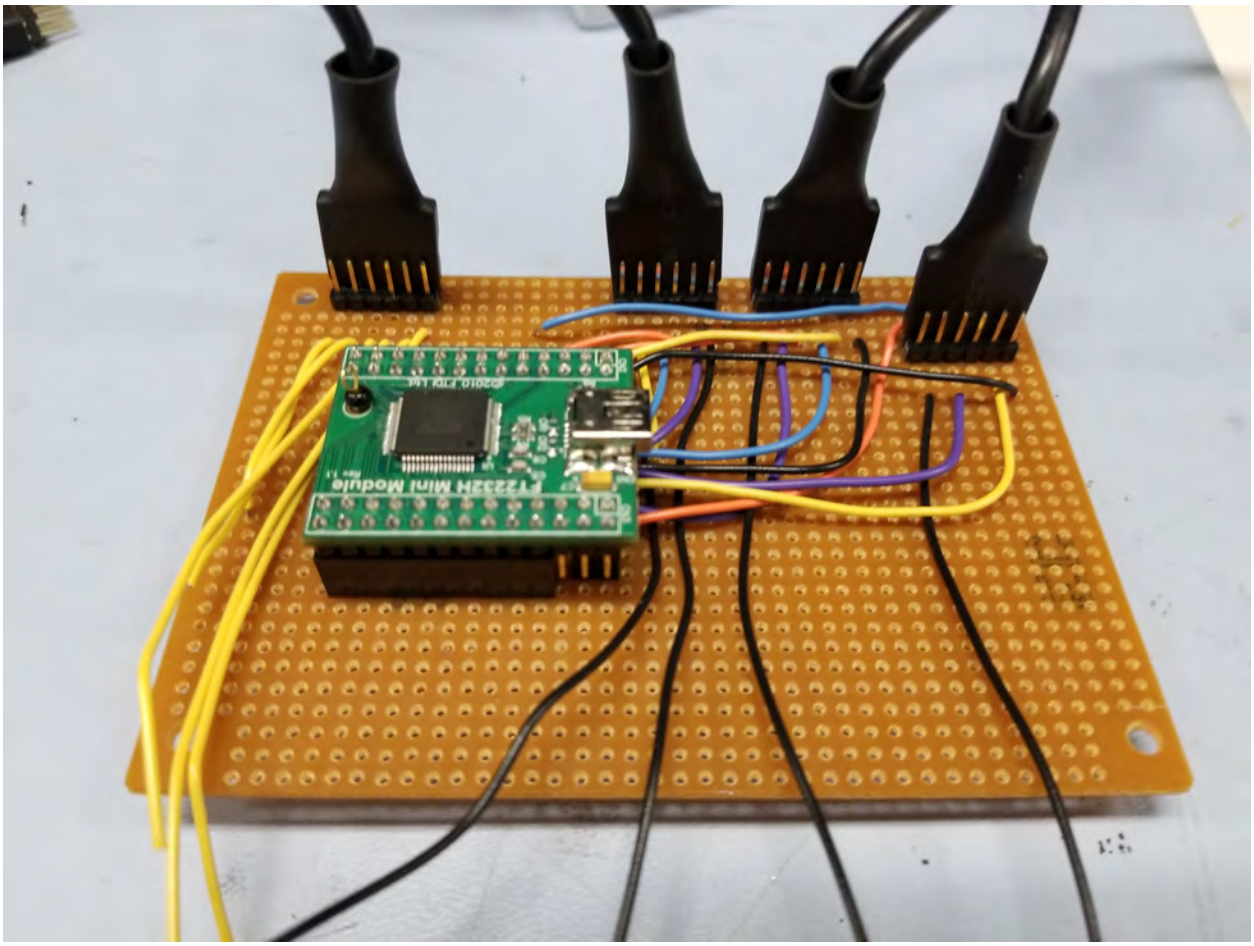
Alan: I would especially like to thank Joe for spending countless hours with me debugging the FT2232H module throughout the past couple weeks, and for dealing with the many lengthy emails that I sent. Without his help, this project would not be successful.

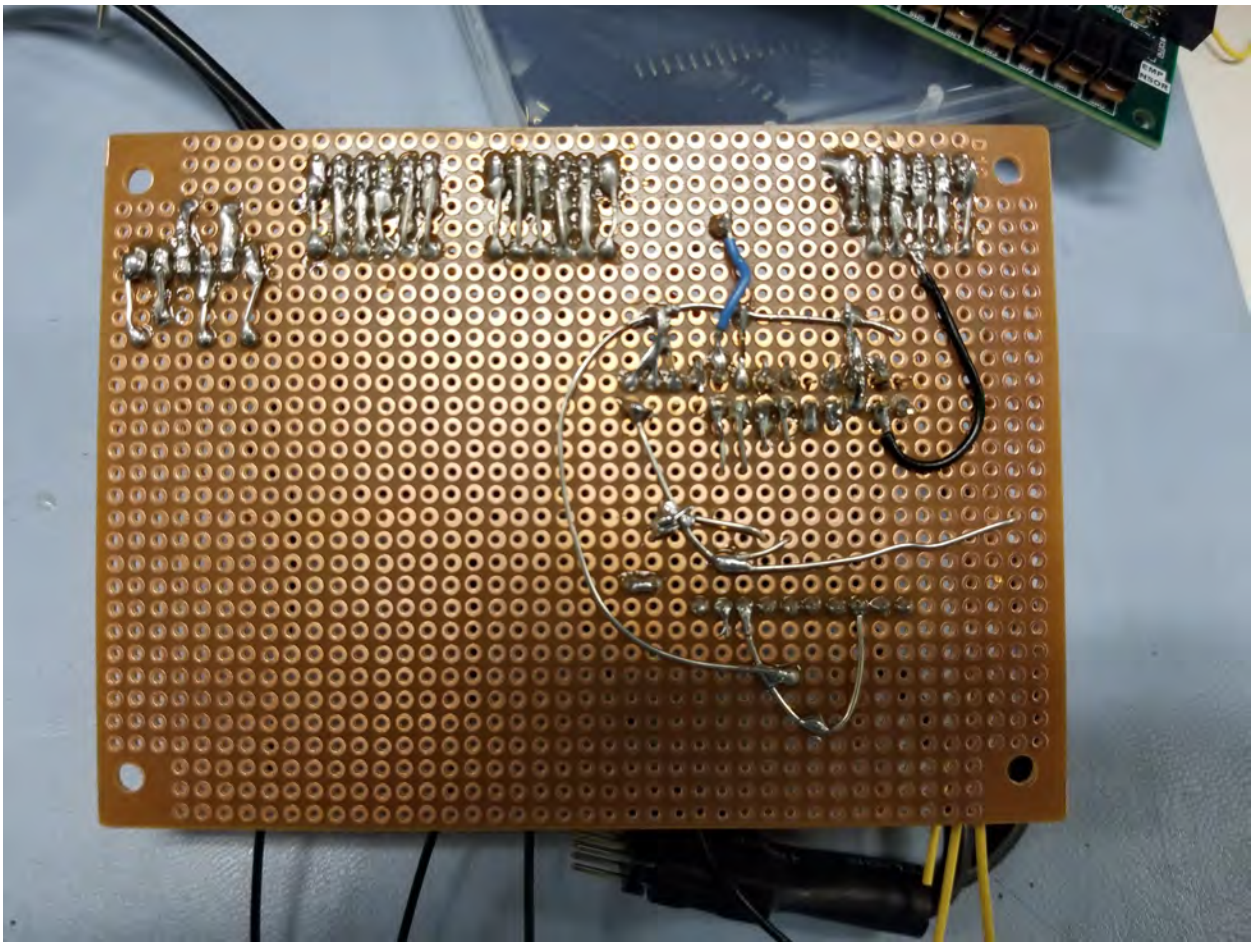
We would also like to thank Professor Gim Hom for making 6.111 a wonderful experience. He also helped with our image processing module, and suggested that we cut down on the register sizes in order to make the module compile in a relatively short amount of time. He truly is the most enthusiastic person for these projects, and 6.111 just wouldn't be the same without him.

Alan: Finally, I would also like to thank Professor Marek Perkowski of Portland State University. He provided initial guidance during the original project – a software implementation of the touch projector – back in 2013. Without his help during my highschool years, the original idea would have never happened, and I would have not been the engineer that I am now.

Appendix A

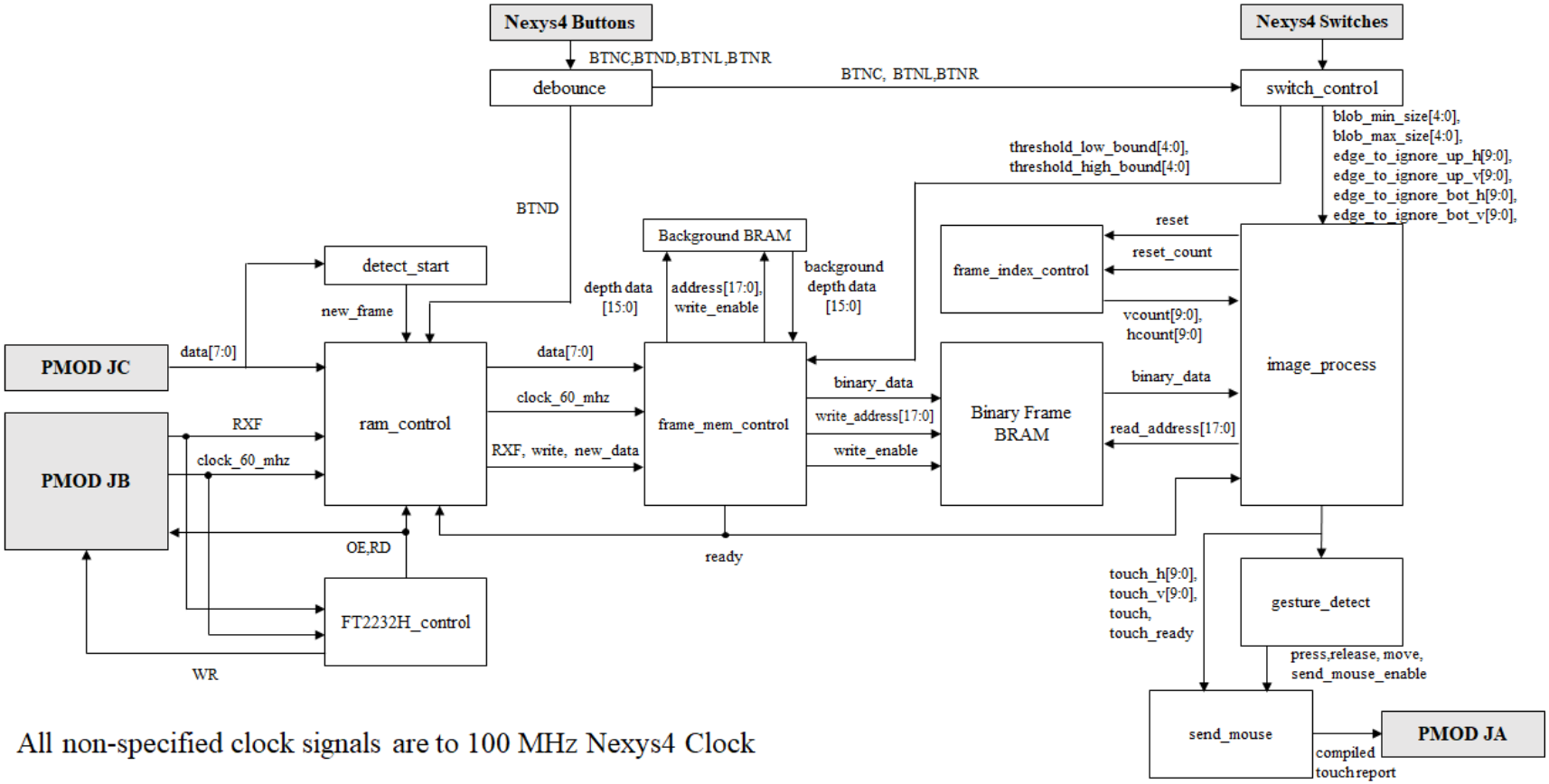
FT2232H Mini Module Setup





Appendix B

System Diagram



All non-specified clock signals are to 100 MHz Nexys4 Clock

Appendix C

Verilog Code

```
//////////////////////////////////////////////////////////////////////////////////
// Module Name: TP_Top
// Author: Alan Cheng
// Description: Top level module
//
// Dependencies: everything
//
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module TP_Top(
    input CLK100MHZ,
    input[15:0] SW,
    input BTNC, BTNU, BTNL, BTNR, BTND,
    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[7:0] SEG,     // segments A-G (0-6), DP (7)
    output[7:0] AN,      // Display 0-7
    output[7:0] JA,      // Output PMOD to Teensy
    input[7:0] JC,       // FT2232_H input Data
    input[1:0] JB_IN,    // FT2232_H {RXF,clock_60_mhz}
    output[2:0] JB_OUT   // FT2232_H {OE,RD,WE}
    );


     // create 25mhz system clock
     wire clock_25mhz;
     clock_quarter_divider clockgen(.clk100_mhz(CLK100MHZ), .clock_25mhz(clock_25mhz));

     //  instantiate 7-segment display;
     wire [31:0] data;
     wire [6:0] segments;
     assign data = (SW[15]) ? {22'd0, SW[9:0]} : {11'd0, SW[9:5], 11'd0, SW[4:0]};
     display_8hex display(.clk(clock_25mhz),.data(data), .seg(segments), .strobe(AN));
     assign SEG[6:0] = segments;
     assign SEG[7] = 1'b1;

     assign LED[15:2] = SW[15:2];
     assign LED16_R = BTNL;                  // left button -> red led
     assign LED16_G = BTNC;                  // center button -> green led
     assign LED16_B = BTNR;                  // right button -> blue led
     assign LED17_R = BTNL;
     assign LED17_G = BTNC;
     assign LED17_B = BTNR;

     wire BTNC_clean, BTNU_clean, BTND_clean, BTNL_clean, BTNR_clean;
     debounce debounce_btnc(.reset(global_reset), .clock(clock_25mhz), .noisy(BTNC), .clean(BTNC_clean));
     debounce debounce_btnu(.reset(global_reset), .clock(clock_25mhz), .noisy(BTNU), .clean(BTNU_clean));
     debounce debounce_btnd(.reset(global_reset), .clock(clock_25mhz), .noisy(BTND), .clean(BTND_clean));
     debounce debounce_btnl(.reset(global_reset), .clock(clock_25mhz), .noisy(BTNL), .clean(BTNL_clean));
     debounce debounce_btnr(.reset(global_reset), .clock(clock_25mhz), .noisy(BTNR), .clean(BTNR_clean));

     wire [4:0] threshold_high_bound, threshold_low_bound;
     wire [4:0] blob_min_size, blob_max_size;
     wire [9:0] edge_to_ignore_up_h;
     wire [9:0] edge_to_ignore_up_v;
     wire [9:0] edge_to_ignore_bot_h;
     wire [9:0] edge_to_ignore_bot_v;

     switch_control switch_parameters(.clk(clock_25mhz), .SW(SW), .BTNC_clean(BTNC_clean), .BTNL_clean(BTNL_clean), .
          ↪ BTNR_clean(BTNR_clean),
                                      .threshold_high_bound(threshold_high_bound),
                                      .threshold_low_bound(threshold_low_bound),
                                      .blob_min_size(blob_min_size),
                                      .blob_max_size(blob_max_size),
                                      .edge_to_ignore_up_h(edge_to_ignore_up_h),
                                      .edge_to_ignore_up_v(edge_to_ignore_up_v),
                                      .edge_to_ignore_bot_h(edge_to_ignore_bot_h),
                                      .edge_to_ignore_bot_v(edge_to_ignore_bot_v));

     parameter WIDTH = 512;                  // Width of image frame
     parameter HEIGHT = 424;                 // Depth of image frame
     parameter LOGSIZE = 18;                 // smallet LOGSIZE s.t. 2^LOGSIZE > WIDTH*HEIGHT

     wire WR, RD, OE;                        // Write/Read/Output_Enable from FT2232H
     wire img_ready;                         // Signals when binarized frame is ready in RAM

     assign JB_OUT[0] = WR;
     assign JB_OUT[1] = RD;
     assign JB_OUT[2] = OE;

     wire [7:0] ft_data;

     assign write_top_left = SW[12];
     assign write_bottom_right = SW[11];

```

```
92      FT2232_Data PC_Data(.clock_60_mhz(JB_IN[0]), .RXF(JB_IN[1]), .WR(WR), .RD(RD), .OE(OE));
93
94      wire pixel;                             // pixel value ([0,1]) at vga_index, or internal index while updating
95      wire [LOGSIZE-1:0] index_out;
96      wire [LOGSIZE-1:0] index_img_proc;
97
98      ram_control #(.WIDTH(WIDTH), .HEIGHT(HEIGHT), .LOGSIZE(LOGSIZE))
99                  stream_handle(.reset(BTNU_clean), .clk(CLOCK100MHZ), .clock_60_mhz(JB_IN[0]), .write_btn(BTND_clean),
100                               .RXF(~JB_IN[1]), .OE(~OE), .RD(~RD), .data(JC), .ready(img_ready), .read_index(vga_index)
101                               ,.wr_bin_mem(wr_bin_mem), .data_bin(wr_bin_val)
102                               ,.index_out(index_out),
103                                 .low_bound(threshold_high_bound), .high_bound(threshold_low_bound));
104
105     assign LED[0] = img_ready;
106     assign LED[1] = BTND_clean;
107
108     // Binary BRAM to be used for image_process
109     wire wr_bin_mem, wr_bin_val;
110     frame_bram #(.SIZE(WIDTH*HEIGHT), .LOGSIZE(LOGSIZE), .BITS(1'b1))
111                 binary_bram(.wr_addr(index_out),.rd_addr(index_img_proc),.wr_clk(JB_IN[0]),.rd_clk(CLK100MHZ),
112                             .we(wr_bin_mem),.rd(1'b1),.din(wr_bin_val),.dout(pixel));
113
114     // Binary BRAM to be used VGA display of thresholded image (without image process)
115     wire [17:0] index_img_input;
116     wire pixel_input;
117     assign index_img_input = (vcount_in << 9)+hcount_in;
118     frame_bram #(.SIZE(WIDTH*HEIGHT), .LOGSIZE(LOGSIZE), .BITS(1'b1))
119                 binary_input_bram(.wr_addr(index_out),.rd_addr(index_img_input),.wr_clk(JB_IN[0]),.rd_clk(clock_25mhz),
120                                   .we(wr_bin_mem),.rd(1'b1),.din(wr_bin_val),.dout(pixel_input));
121     wire[7:0] pixel_bin_input;
122     assign pixel_bin_input = (at_display_area_in && pixel_input)? 255 : 0;     // Map [1,0] to [255,0]
123     wire [9:0] hcount_in, vcount_in;
124     wire at_display_area_in;
125     vga #(.HSIZE(WIDTH), .VSIZE(HEIGHT)) vga_input_display(.vga_clock(clock_25mhz),.hcount(hcount_in),.vcount(vcount_in),
          ↪ .at_display_area(at_display_area_in));
126
127     // 2. Process Frames
128     wire [7:0] pixel_out;
129     wire [9:0] touch_h; // y coordinate of user touch
130     wire [9:0] touch_v; // x coordinate of user touch
131     wire touch; // whether there is a valid touch
132     wire [9:0] top_left_x, top_left_y, bottom_right_x, bottom_right_y;
133     wire touch_ready;
134     wire write_top_left, write_bottom_right;
135     wire in_blob_box, in_frame_box; // whether the current pixel is on the edge of the box around the blob
136     image_process #(.HSIZE(WIDTH),.VSIZE(HEIGHT)) image_process_1
137                   (.clock(CLK100MHZ), .frame_available(img_ready),
138                    .bin_index(index_img_proc),.pixel_out(pixel_out), .pixel_in(pixel),
139                    .blob_min_size(blob_min_size), .blob_max_size(blob_max_size),
140                    .edge_to_ignore_up_h(edge_to_ignore_up_h), .edge_to_ignore_up_v(edge_to_ignore_up_v),
141                    .edge_to_ignore_bot_h(edge_to_ignore_bot_h), .edge_to_ignore_bot_v(edge_to_ignore_bot_v),
142                    .hsync(hsync),.vsync(vsync),.in_blob_box(in_blob_box),.in_frame_box(in_frame_box),
143                    .touch_h(touch_h),.touch_v(touch_v),.touch(touch),.touch_ready(touch_ready),
144                    .top_left_h(top_left_x), .top_left_v(top_left_y), .bottom_right_h(bottom_right_x), .bottom_right_v(
                         ↪ bottom_right_y));
145
146     wire [7:0] pixel_vga;
147     assign pixel_vga = SW[10] ? pixel_bin_input : pixel_out;
148     assign VGA_R = in_blob_box? 255 : pixel_vga;
149     assign VGA_G = in_frame_box? 255 : in_blob_box? 0 : pixel_vga;
150     assign VGA_B = in_frame_box? 0 : pixel_vga;
151     assign VGA_HS = ~hsync;
152     assign VGA_VS = ~vsync;
153
154     wire is_press, is_release, is_move, is_scroll;
155     wire send_mouse_enable;
156
157     // 3. Send Mouse Info
158     // Sends the touch location to the Teensy over serial:
159     send_mouse send_mouse_module(.x(touch_h), .y(touch_v), .is_press(is_press), .is_release(is_release), .is_move(is_move)
          ↪ ,
160                                  .top_left_x(top_left_x), .top_left_y(top_left_y), .bottom_right_x(bottom_right_x), .
                                       ↪ bottom_right_y(bottom_right_y),
161                                  .reset(BTNU_clean), .clock(CLK100MHZ), .send_enable(send_mouse_enable),
162                                  .pmod(JA));
163
164     gesture_detect gesture_module(.clock(CLK100MHZ), .touch_h(touch_h), .touch_v(touch_v), .touch(touch), .touch_ready(
          ↪ touch_ready),
165                                   .is_press(is_press), .is_release(is_release), .is_move(is_move), .is_scroll(is_scroll),
166                                   .send_mouse_enable(send_mouse_enable));
167 endmodule
168
169 //////////////////////////////////////////////////////////////////////////////////
170 // The following are virtually untouched from the labkit. Only parameters for the
171 // correct resolution was added into VGA
172 module clock_quarter_divider(input clk100_mhz, output reg clock_25mhz = 0);
173     reg counter = 0;
174
175     // VERY BAD VERILOG
176     // VERY BAD VERILOG
177     // VERY BAD VERILOG
178     // But it's a quick and dirty way to create a 25Mhz clock
```

```
    // Please use the IP Clock Wizard under FPGA Features/Clocking
    //
    // For 1 Hz pulse, it's okay to use a counter to create the pulse as in
    // assign onehz = (counter == 100_000_000);
    // be sure to have the right number of bits.

    always @(posedge clk100_mhz) begin
        counter <= counter + 1;
        if (counter == 0) begin
            clock_25mhz <= ~clock_25mhz;
        end
    end
endmodule

module vga #(parameter HSIZE=512,
             parameter VSIZE=424)
            (input vga_clock,
            output reg [9:0] hcount = 0,    // pixel number on current line
            output reg [9:0] vcount = 0,    // line number
            output reg vsync, hsync,
            output at_display_area);

    // Comments applies to XVGA 1024x768, left in for reference
    // horizontal: 1344 pixels total
    // display 1024 pixels per line
    reg hblank,vblank;
    wire hsyncon,hsyncoff,hreset,hblankon;
    assign hblankon = (hcount == 639);    // active H  1023
    assign hsyncon = (hcount == 655);     // active H + FP 1047
    assign hsyncoff = (hcount == 751);    // active H + fp + sync  1183
    assign hreset = (hcount == 799);      // active H + fp + sync + bp 1343

    // vertical: 806 lines total
    // display 768 lines
    wire vsyncon,vsyncoff,vreset,vblankon;
    assign vblankon = hreset & (vcount == 479);    // active V   767
    assign vsyncon = hreset & (vcount ==490 );     // active V + fp   776
    assign vsyncoff = hreset & (vcount == 492);    // active V + fp + sync  783
    assign vreset = hreset & (vcount == 523);      // active V + fp + sync + bp 805

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

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

    end

    assign at_display_area = ((hcount >= 0) && (hcount < 512) && (vcount >= 0) && (vcount < 424));

endmodule

module switch_control(
    input clk,
    input [15:0] SW,
    input BTNC_clean,
    input BTNL_clean,
    input BTNR_clean,
    output reg [4:0] threshold_high_bound,
    output reg [4:0] threshold_low_bound,
    output reg [4:0] blob_min_size,
    output reg [4:0] blob_max_size,
    output reg [9:0] edge_to_ignore_up_h,
    output reg [9:0] edge_to_ignore_up_v,
    output reg [9:0] edge_to_ignore_bot_h,
    output reg [9:0] edge_to_ignore_bot_v);

    always @(posedge clk)  begin
        if (BTNC_clean) begin
            threshold_high_bound <= SW[4:0];
            threshold_low_bound <= SW[9:5];
        end
        if (BTNL_clean) begin
            blob_min_size <= SW[4:0];
            blob_max_size <= SW[9:5];
        end
        if (BTNR_clean) begin
            case(SW[14:13])
                2'b00: edge_to_ignore_up_h <= SW[9:0];
                2'b01: edge_to_ignore_up_v <= SW[9:0];
                2'b10: edge_to_ignore_bot_h <= SW[9:0];
                2'b11: edge_to_ignore_bot_v <= SW[9:0];
                default: edge_to_ignore_up_h <= SW[9:0];
            endcase
        end
```

```
271         end
272     endmodule
```

```
//////////////////////////////////////////////////////////////////////////////////
// Module Name: FT2232_Data
// Author: Alan Cheng
// Description: Controls RD and OE signals to read from FT2232H properly
//
// Dependencies:
//
// Additional Comments: See FT2232H FIFO Documentation for timing properties
//                      Reacts asynchronously with changes in RXF
//
//////////////////////////////////////////////////////////////////////////////////

module FT2232_Data(
    input clock_60_mhz,     // 60 MHz FT_2232H clock
    input RXF,              // RXF from FT_2232H: 0 if there is data to be read
    output WR,              // WR to FT_2232H: 0 if data needs to be written
    output RD,              // RD to FT_2232H: 0 to read from buffer
    output OE               // OE to FT_2232H: 0 to enable output for data
                            //      to be moved from buffer. Must be low before RD
    );

    reg RD_buf, OE_buf;
    reg tmp = 1;            // tmp buffer to delay signal

    always@(posedge clock_60_mhz) begin
        if(!RXF) begin
            RD_buf <= OE_buf;
            OE_buf <= tmp;
            tmp <= RXF;
        end
        else begin
            RD_buf <= 1'b1;
            OE_buf <= 1'b1;
            tmp <= 1'b1;
        end
    end

    assign WR = 1'b1;       // We never write to PC

    assign RD = RD_buf;
    assign OE = OE_buf;

endmodule

//////////////////////////////////////////////////////////////////////////////////
// Module Name: ram_control
// Author: Alan Cheng
// Description: handles incoming image data from FT2232H and:
//              1) stores full frame in BRAM with button press
//              2) thresholds for binary image using stored frame in BRAM
//
// Dependencies: bram, background_mem_control, detect_new_frame
//
// Additional Comments: read_index input allows to read from bram when possible
//                      this top module focuses on state control
//
//////////////////////////////////////////////////////////////////////////////////
module ram_control #(parameter WIDTH=512, HEIGHT=424, LOGSIZE=18,
                     DEPTH_SIZE=16, BIN_SIZE=1)
    (input reset,                       // global reset
    input clk,                          // 100 MHz FPGA system clock
    input clock_60_mhz,                 // 60 MHz FT_2232H clock
    input write_btn,                    // Write signal: 1 stores background frame in BRAM
    input RXF,                          // RXF from FT_2232H
    input OE,                           // Output_Enable from FT_2232H
    input RD,                           // RD from FT_2232H
    input [7:0] data,                   // Incoming byte of data from FT_2232H
    input [LOGSIZE-1:0] read_index,     // Input index of Binary BRAM frame to read
    input [4:0] low_bound,              // lower bound for threshold
    input [4:0] high_bound,             // upper bound for threshold
    output ready,                       // Ready signal: 1 if complete frame is stored; 0 if updating
    output bin_out,                     // Binary pixel value at read_index or index_bin if updating
    output wr_bin_mem,                  // Write to binary memory singal
    output data_bin,                    // binary value for pixel at index_bin
    output [LOGSIZE-1:0] index_out      // index of output binary pixel value
    );

    // States:
    parameter [1:0] UPDATING = 2'b10;   // Updating binary image
    parameter [1:0] WRITING = 2'b01;    // Storing frame to background frame BRAM
    parameter [1:0] IDLE = 2'b00;       // Idle; all processes done
    reg [1:0] state;

    wire [LOGSIZE-1:0] index_bin, index_bin_mem;    // coresponding index to data_bin, index input to binary BRAM

    wire wr_bin, new_frame;             // calculated write timing / write input for binary BRAM (wr_bin/wr_bin_mem)
                                                // new_frame flag

    detect_start detect_new_frame(.reset(reset), .clock_60_mhz(clock_60_mhz), .RD(RD), .data_in(data), .ready(new_frame));

    background_mem_control #(.WIDTH(WIDTH), .HEIGHT(HEIGHT), .LOGSIZE(LOGSIZE), .DEPTH_SIZE(DEPTH_SIZE))
            stream_control(.reset(reset || new_frame), .clk(clk), .clock_60_mhz(clock_60_mhz), .new_data(RD),
```

```
                            .RXF(RXF), .write(state[0]), .data_in(data), .data_bin_wr(data_bin), .WR_bin(wr_bin),
                            .index_bin(index_bin), .ready(ready), .low_bound(low_bound), .high_bound(high_bound));

    assign wr_bin_mem = (state == UPDATING) ? wr_bin : 0;               // prohibit writes to binary BRAM when not
        ↪ updating
    assign index_bin_mem = (state == IDLE) ? read_index : index_bin;    // when IDLE, take index from input; else use
        ↪ signal from mem_control

    // State Machine Model
    always@(posedge clock_60_mhz) begin
        // Global reset
        if(reset)
            state <= IDLE;
        else begin
            debug <= new_frame;
            if (state == IDLE) begin
                if (write_btn && new_frame)
                    state <= WRITING;
                else if (new_frame)
                    state <= UPDATING;
            end
            else if (state == UPDATING) begin
                if (ready && !new_frame)
                    state <= IDLE;
            end
            else if (state == WRITING) begin
                if (ready && !new_frame)
                    state <= IDLE;
            end
        end
    end

endmodule

//////////////////////////////////////////////////////////////////////////////////
// Module Name: background_mem_control
// Author: Alan Cheng
// Description: handles incoming image data from FT2232H and:
//              1) stores full frame in BRAM with button press
//              2) thresholds for binary image using stored frame in BRAM
//
// Dependencies: bram
//
// Additional Comments: adding some debug code for thresholding non-reliant
//                      on background frame BRAM
//////////////////////////////////////////////////////////////////////////////////
module background_mem_control #(parameter WIDTH=512,        // Depth of image resolution (512x424)
                                HEIGHT=424,                 // Width of image resolution
                                LOGSIZE=18,                 // Smallet LOGSIZE s.t. 2^LOGSIZE > WIDTH*HEIGHT
                                DEPTH_SIZE=16)              // Size of each pixel: 2 bytes = 16 bits
    (input reset,           // global reset
    input clk,              // 100 MHz FPGA system clock
    input clock_60_mhz,     // 60 MHz FT_2232H clock
    input new_data,         // 1 indicates there is new data to process
    input write,            // active-high write signal
    input RXF,              // RXF from FT_2232H
    input [7:0] data_in,    // data read from FT2232H
    input [4:0] low_bound,  // lower bound for threshold
    input [4:0] high_bound, // upper bound for threshold
    output reg data_bin_wr, // data for binary image
    output reg WR_bin,      // write enable for binary image ram
    output reg [LOGSIZE-1:0] index_bin,     // index corresponding to data_bin_wr
    output reg ready        // 1 when image frame is done processing
    );

    // Determines white area in binary image if within threshold margins
    wire [4:0] BACK_THRESH_MIN;     // mem - 30 = lower bound
    wire [4:0] BACK_THRESH_MAX;     // mem - 2 = upper bound

    assign BACK_THRESH_MIN = high_bound;
    assign BACK_THRESH_MAX = low_bound;

    // Fixed range limitations
    parameter BACK_DEPTH_MIN = 500;
    parameter BACK_DEPTH_MAX = 2000;
    parameter NUM_PIXELS = WIDTH*HEIGHT-1;      // Max number of pixels; used to flag ready

    // state of inc data: 1=2nd byte, 0=1st byte
    parameter HI = 1;
    parameter LO = 0;
    reg data_state;

    reg prev_RXF;               // previous RXF
    reg [7:0] prev_data;        // previous data

    reg [LOGSIZE-1:0] index;    // index of current pixel

    // 2 bytes make full distance values
    wire [15:0] data_back_wr;   // {2nd byte, 1st byte}\
    assign data_back_wr = {prev_data, data_in};

    wire [15:0] ram_data;       // 2 byte at index from background BRAM
```

```
183
184     // BRAM to store image
185     frame_bram #(.SIZE(WIDTH*HEIGHT), .LOGSIZE(LOGSIZE), .BITS(DEPTH_SIZE))
186             background_bram(.wr_addr(index),.rd_addr(index),.wr_clk(clock_60_mhz),.rd_clk(clock_60_mhz),
187                         .we(write&&new_data),.rd(1'b1),.din(data_back_wr),.dout(ram_data));
188
189     // Threshold based on set range parameters
190     wire bin;
191     assign bin = ((data_back_wr > (ram_data - BACK_THRESH_MIN)) &&
192                         (data_back_wr < (ram_data - BACK_THRESH_MAX)) &&
193                         (data_back_wr > BACK_DEPTH_MIN) &&
194                         (ram_data > BACK_DEPTH_MIN) &&
195                         (data_back_wr < BACK_DEPTH_MAX) &&
196                         (ram_data < BACK_DEPTH_MAX));
197
198
199     // DEBUG signals
200     parameter FLAT_THRES = 1500;    // 16'h05DC
201
202     always@(posedge clock_60_mhz) begin
203         // Global reset
204         if (reset) begin
205             WR_bin <= 0;
206             index <= 0;
207             index_bin <= 0;
208             data_bin_wr <= 0;
209             ready <= 0;
210             prev_data <= 0;
211             data_state <= LO;
212         end
213
214         // if we have new data to proces...
215         else if (new_data) begin
216             // we have both components of a distance value
217             if (data_state == HI) begin
218                 // If we are in UPDATE
219                 // ...create binary image from thresholds
220                 if(write == 0) begin
221                     data_bin_wr <= bin;
222                     index_bin <= index;
223                 end
224                 // Else we are in WRITING
225                 // ...and we prohibit wrting to binary BRAM
226                 else
227                     data_bin_wr <= 0;
228
229                 // Done throughout all cases:
230                 WR_bin <= 1;    // write to background BRAM
231
232                 // increment index
233                 if (index >= NUM_PIXELS)
234                     ready <= 1;
235                 else if (~(RXF && ~prev_RXF)) begin
236                     ready <= 0;
237                     index <= index + 1;
238                 end
239
240                 data_state <= LO;   // Toggle state of input
241             end
242
243             // We only have the first component of a distance value
244             else begin
245                 WR_bin <= 0;
246                 data_state <= HI;
247             end
248
249             // Store the previous value
250             prev_data <= data_in;
251             prev_RXF <= RXF;
252         end
253
254     end
255
256 endmodule
257
258 //////////////////////////////////////////////////////////////////////////////////
259 // Module Name: detect_start
260 // Author: Alan Cheng
261 // Description: Detect starting sequence for new frame {DD,CC,BB,AA}
262 //
263 // Dependencies:
264 //
265 // Additional Comments:
266 //
267 //////////////////////////////////////////////////////////////////////////////////
268 module detect_start (
269     input reset,            // global reset
270     input clock_60_mhz,     // 60 MHz FT_2232H clock
271     input RD,               // RD to FT_2232H: 0 to read from buffer
272     input [7:0] data_in,    // data read from FT2232H
273     output ready,            // flags when starting sequence is detected
274     output [7:0] prev_out
```

```
275       );
276
277       // storage containers for past states
278       reg [7:0] data_prev1, data_prev2, data_prev3;
279
280       // sequence to be detected
281       parameter [7:0] START0 = 8'hAA;
282       parameter [7:0] START1 = 8'hBB;
283       parameter [7:0] START2 = 8'hCC;
284       parameter [7:0] START3 = 8'hDD;
285
286       always@(posedge clock_60_mhz) begin
287           if(reset) begin
288               data_prev1 <= 0;
289               data_prev2 <= 0;
290               data_prev3 <= 0;
291           end
292           else if(RD) begin
293               data_prev1 <= data_in;
294               data_prev2 <= data_prev1;
295               data_prev3 <= data_prev2;
296           end
297       end
298
299       assign ready = ((data_in == START0) && (data_prev1 == START1)
300                       && (data_prev2 == START2) && (data_prev3 == START3));
301
302       assign prev_out = data_in;
303   endmodule
```

```
//////////////////////////////////////////////////////////////////////////////////
// Module Name: bram
// Author: Alan Cheng
// Description: Single port BRAM
//
// Dependencies:
//
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module bram #(parameter SIZE=512*424, LOGSIZE=18, BITS=1)
    (input wire [LOGSIZE-1:0] addr,
    input wire clk,
    input wire [BITS-1:0] din,
    output reg [BITS-1:0] dout,
    input wire we);
    // let the tools infer the right number of BRAMs
    (* ram_style = "block" *)
    reg [BITS-1:0] mem[SIZE-1:0];
    always @(posedge clk) begin
        if (we) mem[addr] <= din;
        dout <= mem[addr];
    end
endmodule

//////////////////////////////////////////////////////////////////////////////////
// Module Name: frame_bram
// Author: Alan Cheng
// Description: Dual port BRAM
//
// Dependencies:
//
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module frame_bram #(parameter SIZE=512*424, LOGSIZE=18, BITS=1)
    (input wire [LOGSIZE-1:0] wr_addr,
    input wire [LOGSIZE-1:0] rd_addr,
    input wire wr_clk,
    input wire rd_clk,
    input wire [BITS-1:0] din,
    output reg [BITS-1:0] dout,
    input wire we,
    input wire rd);
    // let the tools infer the right number of BRAMs
    (* ram_style = "block" *)
    reg [BITS-1:0] mem[SIZE-1:0];
    always @(posedge wr_clk)
        if (we) mem[wr_addr] <= din;
    always @(posedge rd_clk)
        if (rd) dout <= mem[rd_addr];
endmodule
```

```
//////////////////////////////////////////////////////////////////////////////////
// Module Name: frame_index_control
// Author: Hope Harrison, with conversion to 512x3 window by Alan Cheng
// Description: Processes an image in order to get the coordinates of a fingertip in the image.
//
// Dependencies: frame_index_control
//
// Additional Comments:
//////////////////////////////////////////////////////////////////////////////////

module image_process #(parameter HSIZE = 512, parameter VSIZE=424)
        (input clock,
        input reset,
        input frame_available,              // Indicates that new data is available in frame, should be a pulse
        input pixel_in,                     // the current pixel of the binarized image
        input [4:0] blob_min_size, blob_max_size, // parameters for which sized blobs will be detected as valid
        input [9:0] edge_to_ignore_up_h,    // sets the image area to look for blobs in
        input [9:0] edge_to_ignore_up_v,
        input [9:0] edge_to_ignore_bot_h,
        input [9:0] edge_to_ignore_bot_v,
        output [17:0] bin_index,            // the current index to look at in the binarized image (corresponds with
            ↪ pixel_in)
        output [7:0] pixel_out,             // the current pixel of the processed image
        output hsync, vsync,                // signals to send to VGA
        output in_blob_box, in_frame_box,   // whether the current pixel is on the edge of the box around the blob
        output [9:0] touch_h, touch_v,      // coordinates of user touch
        output reg touch, touch_ready       // whether there is a valid touch
        );


    // Create 25mhz clock - used for VGA display (the monitor complains if you use a faster clock)
    wire clock_25mhz;
    clock_quarter_divider clockgen(.clk100_mhz(clock), .clock_25mhz(clock_25mhz));

    // These are the possible states in image processing
    parameter IDLE = 0;
    parameter GET_IMAGE = 1;
    parameter ERODE = 2;
```

```
38      parameter DILATE = 3;
39      parameter BLOB = 4;
40      parameter PREP_DILATE = 5;
41      parameter WRITE_FINAL = 6;
42      reg [3:0] state;
43
44      // These parameters can be changed to control how the image processing works
45      parameter BLOB_DIST = 3;
46      parameter MAX_BLOB_COUNT = 4;               // the maximum number of blobs that we will look at in an attempt to find
            ↪ a finger touch (takes 1 cycle per pixel per blob)
47
48      wire [9:0] hcount, vcount;
49      wire [9:0] hcount_disp, vcount_disp;        // these are the hcount and vcount for display on the monitor, as opposed
            ↪ to for our processing which is faster
50      wire at_display_area;
51      wire pixel_on;                              // whether the current pixel should be on
52      reg [9:0] start_h, end_h, start_v, end_v;   // the bounds of the box around current blob
53      reg frame [HSIZE-1:0][2:0];
54      reg eroded [HSIZE-1:0] [2:0];               // the eroded version of the image
55      reg next_row [HSIZE-1:0];
56      reg begin_processing;                       // is triggered when a new frame comes in
57      reg started_blob;                           // whether we have found a blob yet
58      reg [5:0] blob_count;                       // the number of blobs perviously looked at (that didn't meet criteria)
59      reg last_frame_available;
60      reg new_frame;
61      wire [9:0] top_left_h, top_left_v, bottom_right_h, bottom_right_v;
62
63      // Make an hcount and vcount for displaying and also a faster one for cycling through pixels in image processing
64      vga #(.HSIZE(HSIZE), .VSIZE(VSIZE)) vga_display(.vga_clock(clock_25mhz),.hcount(hcount_disp),.vcount(vcount_disp),.
            ↪ hsync(hsync),.vsync(vsync),.at_display_area(at_display_area));
65
66      reg wr_erosion;
67      wire erosion_pixel;
68      wire erosion_out;
69      bram #(.SIZE(HSIZE*VSIZE), .LOGSIZE(18), .BITS(1))
70              erosion_bram(.addr(bin_index),.clk(clock),.we(wr_erosion),.din(erosion_pixel),.dout(erosion_out));
71
72      reg wr_dilation;
73      wire dilation_pixel;
74      wire process_out;
75
76      reg reset_count, hard_reset;
77
78      reg wr_blob;
79      reg [9:0] h_process, v_process;
80      wire process_pixel;
81      wire [17:0] process_bram_index;
82      wire [17:0] process_index;
83      wire [17:0] disp_index;
84
85      wire write_process;
86      assign write_process = wr_dilation | wr_erosion | wr_blob;
87
88      assign process_index = (v_process * HSIZE)+h_process;
89      assign process_bram_index = bin_index; //(wr_blob) ? process_index : bin_index;
90      assign process_pixel = (wr_blob) ? 0 : dilation_pixel;
91
92      frame_bram #(.SIZE(HSIZE*VSIZE), .LOGSIZE(18), .BITS(1'b1))
93                  process_bram(.wr_addr(process_bram_index),.rd_addr(bin_index),.wr_clk(clock),.rd_clk(clock),
94                              .we(write_process),.rd(1'b1),.din(process_pixel),.dout(process_out));
95
96      wire write_final;
97      assign write_final = (state == WRITE_FINAL);
98      frame_bram #(.SIZE(HSIZE*VSIZE), .LOGSIZE(18), .BITS(1'b1))
99                  final_bram(.wr_addr(bin_index),.rd_addr(disp_index),.wr_clk(clock),.rd_clk(clock_25mhz),
100                             .we(write_final),.rd(1'b1),.din(process_out),.dout(pixel_on));
101
102     frame_index_control #(.HSIZE(HSIZE),.VSIZE(VSIZE)) frame_division(.reset(reset || hard_reset), .reset_count(
            ↪ reset_count), .clk(clock), .vcount(vcount), .hcount(hcount));
103
104     assign bin_index = (vcount * HSIZE) + hcount;
105     assign disp_index = (vcount_disp * HSIZE) + hcount_disp;
106
107     // Erosion kernel
108     assign erosion_pixel =  frame[hcount-1][0] &
109                             frame[hcount-1][1] &
110                             frame[hcount-1][2] &
111                             frame[hcount]  [0] &
112                             frame[hcount]  [2] &
113                             frame[hcount+1][0] &
114                             frame[hcount+1][1] &
115                             frame[hcount+1][2];
116
117     // dilation kernel
118     assign dilation_pixel = eroded[hcount-1][0] |
119                             eroded[hcount-1][1] |
120                             eroded[hcount-1][2] |
121                             eroded[hcount]  [0] |
122                             eroded[hcount]  [2] |
123                             eroded[hcount+1][0] |
124                             eroded[hcount+1][1] |
125                             eroded[hcount+1][2];
```

```
126
127
128     wire look_for_blob;
129     assign look_for_blob = (hcount > edge_to_ignore_up_h
130                           & hcount < HSIZE - edge_to_ignore_bot_h
131                           & vcount > edge_to_ignore_up_v
132                           & vcount < VSIZE - edge_to_ignore_bot_v);
133
134     integer index;
135     assign top_left_h = edge_to_ignore_up_h;
136     assign top_left_v = edge_to_ignore_up_v;
137     assign bottom_right_h = (HSIZE - edge_to_ignore_bot_h);
138     assign bottom_right_v = (VSIZE - edge_to_ignore_bot_v);
139
140
141     always @(posedge clock) begin
142
143         last_frame_available <= frame_available;
144         new_frame <= (~last_frame_available & frame_available);
145
146         case (state)
147             IDLE: begin
148                 touch_ready <= 0;
149                 if (new_frame) begin
150                     state <= GET_IMAGE;
151                     started_blob <= 0;
152                     start_h <= 0;
153                     end_h <= 0;
154                     start_v <= 0;
155                     end_v <= 0;
156                     blob_count <= 0;
157                     touch <= 0;
158
159                     reset_count <= 1;
160                     wr_erosion <= 0;
161                     hard_reset <= 1;
162                 end
163                 else begin
164                     reset_count <= 0;
165                     wr_erosion <= 0;
166                     hard_reset <= 0;
167                 end
168             end
169             GET_IMAGE: begin
170                 hard_reset <= 0;
171                 if ((hcount == HSIZE-2) && (vcount == 2)) begin
172                     state <= ERODE;
173                     reset_count <= 1;
174                     wr_erosion <= 1;
175                 end
176                 else begin
177                     reset_count <= 0;
178                     wr_erosion <= 0;
179                 end
180                 frame[hcount][vcount] <= pixel_in;
181             end
182             ERODE: begin
183                 reset_count <= 0;
184
185                 if ((hcount == HSIZE-2) && (vcount == VSIZE-3)) begin
186                     state <= PREP_DILATE;
187                     hard_reset <= 1;
188                     wr_erosion <= 0;
189                 end
190                 else if (hcount == HSIZE-2) begin
191                     for (index=0; index<HSIZE-1; index=index+1) begin
192                         frame[index][0] <= frame[index][1];
193                         frame[index][1] <= frame[index][2];
194                         frame[index][2] <= next_row[index];
195                     end
196                 end
197                 next_row[hcount] <= pixel_in;
198             end
199             PREP_DILATE: begin
200                 hard_reset <= 0;
201                 if ((hcount == HSIZE-2) && (vcount == 2)) begin
202                     state <= DILATE;
203                     reset_count <= 1;
204                     wr_dilation <= 1;
205                 end
206                 eroded[hcount][vcount] <= erosion_out;
207             end
208             DILATE: begin
209                 reset_count <= 0;
210
211                 if ((hcount == HSIZE-2) && (vcount == VSIZE-3)) begin
212                     state <= WRITE_FINAL;
213                     reset_count <= 1;
214                     wr_dilation <= 0;
215                 end
216                 else if (hcount == HSIZE-2) begin
217                     for (index=0; index<HSIZE-1; index=index+1) begin
```

```
                        eroded[index][0] <= eroded[index][1];
                        eroded[index][1] <= eroded[index][2];
                        eroded[index][2] <= next_row[index];
                    end
                end
                next_row[hcount] <= erosion_out;

            end
            WRITE_FINAL: begin
                reset_count <= 0;
                if ((hcount == HSIZE-2) && (vcount == VSIZE-3)) state <= BLOB;
            end
            BLOB: begin
                reset_count <= 0;

                if (hcount >= HSIZE-2 & vcount >= VSIZE-3) begin
                    started_blob <= 0;
                    if (end_h - start_h >= blob_min_size & end_h - start_h <= blob_max_size
                      & end_v - start_v >= blob_min_size & end_v - start_v <= blob_max_size) begin
                      // blob is valid
                        state <= IDLE;
                        touch <= 1;
                        touch_ready <= 1;
                        wr_blob <= 0;
                    end
                    else begin
                    // blob is not valid or no blob found
                        touch <= 0;
                        if (blob_count < MAX_BLOB_COUNT) blob_count <= blob_count + 1;
                        else begin
                            touch <= 0;
                            touch_ready <= 1;
                            wr_blob <= 0;
                            state <= IDLE;
                            started_blob <= 0;
                        end
                    end
                end
                else if (look_for_blob & ~started_blob & process_out) begin
                    start_h <= hcount+1;
                    end_h <= hcount+1;
                    start_v <= vcount;
                    end_v <= vcount;
                    started_blob <= 1;
                    h_process <= hcount;
                    v_process <= vcount;
                    wr_blob <= 1;
                end
                else if (look_for_blob & started_blob & process_out & hcount+1 >= start_h - BLOB_DIST &
                                                            hcount+1 <= end_h + BLOB_DIST &
                                                            vcount >= start_v - BLOB_DIST &
                                                            vcount <= end_v + BLOB_DIST) begin
                    // pixel is part of blob
                    if (hcount+1 < start_h) start_h <= hcount+1;
                    if (hcount+1 > end_h) end_h <= hcount+1;
                    if (vcount < start_v) start_v <= vcount;
                    if (vcount > end_v) end_v <= vcount;
                    h_process <= hcount;
                    v_process <= vcount;
                    wr_blob <= 1;
                end
                else begin
                    wr_blob <= 0;
                end
            end
        endcase
    end

    assign in_blob_box = (hcount_disp > start_h) & (vcount_disp > start_v) & (hcount_disp < end_h) & (vcount_disp < end_v)
        ↪ ;

    assign in_frame_box = (hcount_disp == top_left_h & vcount_disp >= top_left_v & vcount_disp <= bottom_right_v) |
                                (hcount_disp == bottom_right_h & vcount_disp >= top_left_v & vcount_disp <=
                                    ↪ bottom_right_v) |
                                (vcount_disp == top_left_v & hcount_disp >= top_left_h & hcount_disp <= bottom_right_h)
                                    ↪ |
                                (vcount_disp == bottom_right_v & hcount_disp >= top_left_h & hcount_disp <=
                                    ↪ bottom_right_h);

    assign pixel_out = at_display_area? (pixel_on? 255 :0) : 0;
    assign touch_h = start_h + (end_h - start_h)/2;
    assign touch_v = start_v + (end_v - start_v)/2;

endmodule

//////////////////////////////////////////////////////////////////////////////////
// Module Name: frame_index_control
// Author: Alan Cheng
// Description: Controls horizontal and vertical index increments for image_process
//
// Dependencies:
//
```

```
306  // Additional Comments: reset = global reset or hard reset; sets it to (1,0)
307  //                      reset_count = soft_rest; sets it to (1,1)
308  //
309  //////////////////////////////////////////////////////////////////////////////////
310  module frame_index_control #(parameter HSIZE=512, VSIZE=424)
311      (input reset,                // hard reset
312      input reset_count,           // soft reset
313      input clk,                   // 100 MHz Nexys4 clock
314      input pause_v,               // halt counting
315      output reg [9:0] vcount,     // vertical index
316      output reg [9:0] hcount);    // horizontal index
317
318      always@(posedge clk) begin
319          if (reset) begin
320               hcount <= 1;
321               vcount <= 0;
322           end
323          else if (reset_count) begin
324              hcount <= 1;
325              vcount <= 1;
326          end
327          else begin
328              // If we have reached the very last pixel to process
329              if ((hcount >= HSIZE-2) && (vcount >= VSIZE-3)) begin
330                  hcount <= 1;
331                  vcount <= 1;
332              end
333              // If we have reached the end of a line
334              else if (hcount >= HSIZE-2) begin
335                  hcount <= 1;
336                  vcount <= vcount + 1;
337              end
338              else hcount <= hcount + 1;
339          end
340      end
341  endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
// Module Name: gesture_detect
// Author: Hope Harrison, with timeout buffer written by Alan Cheng
// Description: Calculates whether a user has pressed or released based on past inputs.
//
// Dependencies:
//
// Additional Comments: is_scroll is unused
//
////////////////////////////////////////////////////////////////////////////////

module gesture_detect(
        input clock,
        input [9:0] touch_h, touch_v, // coordinates of user's fingertip
        input touch,                  // whether the fingertip is touching
        input touch_ready,            // a pulse indicating calculations finished and the touch_h, touch_v, and touch readings are valid
        output reg is_press, is_release, is_move, is_scroll, // mouse actions
        output reg send_mouse_enable  // whether to send any command to the Teensy in order to control the mouse
        );

    reg touch_active;

    parameter TIMEOUT_TOUCH = 10;
    reg [5:0] miss_count;

    always @(posedge clock) begin
        if (touch_ready) begin
            if(touch & ~touch_active) begin
                is_press <= 1;
                is_release <= 0;
                is_move <= 1;
                is_scroll <= 0;
                send_mouse_enable <= 1;

                miss_count <= TIMEOUT_TOUCH;
                touch_active <= 1;
            end
            else if(touch) begin
                is_press <= 0;
                is_release <= 0;
                is_move <= 1;
                is_scroll <= 0;
                send_mouse_enable <= 1;

                miss_count <= TIMEOUT_TOUCH;
                touch_active <= 1;
            end
            else if(~touch & touch_active) begin
                if (miss_count == 0) begin
                    is_press <= 0;
                    is_release <= 1;
                    is_move <= 0;
                    is_scroll <= 0;
                    send_mouse_enable <= 1;

                    touch_active <= 0;
                end
                else begin
                    is_press <= 0;
                    is_release <= 0;
                    is_move <= 1;
                    is_scroll <= 0;
                    send_mouse_enable <= 1;

                    miss_count <= miss_count - 1;
                    touch_active <= 1;
                end
            end
            else if(~touch) begin
                is_press <= 0;
                is_release <= 0;
                is_move <= 0;
                is_scroll <= 0;
                send_mouse_enable <= 1;

                touch_active <= 0;
            end
        end

        else begin
            send_mouse_enable <= 0;
        end
    end

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
// Module Name: send_mouse
// Author: Hope Harrison
// Description: Assigns the byte to be sent over serial in the correct order for the mouse report.
//
// Dependencies:
//
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module send_mouse(
    input [15:0] x, y, // coordinates of mouse
    input [7:0] is_press, is_release, is_move, // type of mouse action
    input [15:0] top_left_x, top_left_y, bottom_right_x, bottom_right_y, // bounds of screen
    input send_enable, // whether or not to send to the mouse (should be true only after having processed new data)
    input reset, clock,
    output [7:0] pmod
    );

    reg tx_enable;
    reg [7:0] data_to_send;
    wire xmit_done;
    wire serial_stream;
    wire xmit_clk;

    reg state;
    parameter WAITING = 0;
    parameter SENDING = 1;
    parameter SPEED = 1;
    reg [3:0] parameter_to_send;
    parameter X1 = 0;
    parameter X2 = 1;
    parameter Y1 = 2;
    parameter Y2 = 3;
    parameter TOP_LEFT_X1 = 4;
    parameter TOP_LEFT_X2 = 5;
    parameter TOP_LEFT_Y1 = 6;
    parameter TOP_LEFT_Y2 = 7;
    parameter BOTTOM_RIGHT_X1 = 8;
    parameter BOTTOM_RIGHT_X2 = 9;
    parameter BOTTOM_RIGHT_Y1 = 10;
    parameter BOTTOM_RIGHT_Y2 = 11;
    parameter IS_PRESS = 12;
    parameter IS_RELEASE = 13;
    parameter IS_MOVE = 14;
    parameter NONE = 15;

    reg debug_tx_enable;
    reg debug_xmit_done;
    reg [2:0] count;
    reg [3:0] send_count;
    reg debug_xmit_clk;

    serial_send sender (.clk(clock), .data(data_to_send), .start_send(tx_enable), .xmit_data(serial_stream), .xmit_clk(
        ↪ xmit_clk), .xmit_done(xmit_done));

    assign pmod[7:1] = 7'b0;
    assign pmod[0] = serial_stream;

    always @(posedge clock) begin
        case (state)
            WAITING: begin
                if (send_enable) begin
                    state <= SENDING;
                    parameter_to_send <= X1;
                    data_to_send <= x[7:0];
                    tx_enable <= 1;
                end
                else begin
                    tx_enable <= 0;
                end
            end
            SENDING: begin
                if (xmit_done) begin
                    parameter_to_send <= parameter_to_send + 1;
                    case(parameter_to_send + 1)
                        X1: begin
                            data_to_send <= x[7:0];
                            tx_enable <= 1;
                        end
                        X2: begin
                            data_to_send <= x[15:8];
                            tx_enable <= 1;
                        end
                        Y1: begin
                            data_to_send <= y[7:0];
                            tx_enable <= 1;
                        end
                        Y2: begin
                            data_to_send <= y[15:8];
                            tx_enable <= 1;
```

```
92                          end
93                          TOP_LEFT_X1: begin
94                              data_to_send <= top_left_x[7:0];
95                              tx_enable <= 1;
96                          end
97                          TOP_LEFT_X2: begin
98                              data_to_send <= top_left_x[15:8];
99                              tx_enable <= 1;
100                         end
101                         TOP_LEFT_Y1: begin
102                             data_to_send <= top_left_y[7:0];
103                             tx_enable <= 1;
104                         end
105                         TOP_LEFT_Y2: begin
106                             data_to_send <= top_left_y[15:8];
107                             tx_enable <= 1;
108                         end
109                         BOTTOM_RIGHT_X1: begin
110                             data_to_send <= bottom_right_x[7:0];
111                             tx_enable <= 1;
112                         end
113                         BOTTOM_RIGHT_X2: begin
114                             data_to_send <= bottom_right_x[15:8];
115                             tx_enable <= 1;
116                         end
117                         BOTTOM_RIGHT_Y1: begin
118                             data_to_send <= bottom_right_y[7:0];
119                             tx_enable <= 1;
120                         end
121                         BOTTOM_RIGHT_Y2: begin
122                             data_to_send <= bottom_right_y[15:8];
123                             tx_enable <= 1;
124                         end
125                         IS_PRESS: begin
126                             data_to_send <= is_press;
127                             tx_enable <= 1;
128                         end
129                         IS_RELEASE: begin
130                             data_to_send <= is_release;
131                             tx_enable <= 1;
132                         end
133                         IS_MOVE: begin
134                             data_to_send <= is_move;
135                             tx_enable <= 1;
136                         end
137                         NONE: begin
138                             tx_enable <= 0;
139                             state <= WAITING;
140                         end
141                     endcase
142                 end
143                 else begin
144                     tx_enable <= 0;
145                 end
146             end
147         endcase
148     end
149
150 endmodule
```

```
//////////////////////////////////////////////////////////////////////////////////
// Module Name: serial_send
// Author: Hope Harrison
// Description: Sends the data input over serial on xmit_data.
//
// Dependencies:
//
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module serial_send(
     input clk,
     input [7:0] data,       // 8 bit ascii data
     input start_send,       // one clock pulse width, enter pressed, data available
     output reg xmit_data,   // serial data sent to RS232 output driver
     output reg xmit_clk,     // baud rate; sent to logic analyzer for debugging
     output reg xmit_done    // indicates that transmission of 1 byte is finished
     );

     // this section sets up the clk;
    parameter DIVISOR =  868; // create 115,200 baud rate clock, not exact, but should work.
    parameter MARK = 1'b1;
    parameter STOP_BIT = 1'b1;
    parameter START_BIT =1'b0;

    parameter WAIT = 0;
    parameter SEND = 1;

    reg [31:0] count;
    reg [9:0] data_buffer;
    reg state;
    reg [3:0] sent_bits;

    always @(posedge clk)
    begin
        count <= xmit_clk ? 0 : count+1;
        xmit_clk <= count == DIVISOR-1;

        case (state)
            WAIT: begin
                xmit_done <= 0;
                if (start_send) begin
                    data_buffer <= {STOP_BIT, data, START_BIT};
                    sent_bits <= 0;
                    state <= SEND;
                end
            end
            SEND: begin
                if (sent_bits >= 10) begin
                    state <= WAIT;
                    xmit_done <= 1;
                end
                else if (xmit_clk) begin
                    sent_bits <= sent_bits + 1;
                    data_buffer <= {STOP_BIT, data_buffer[9:1]};
                    xmit_data <= data_buffer[0];
                    xmit_done <= 0;
                end

            end
        endcase

    end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:   g.p.hom
// Engineer:
//
// Create Date:    18:18:59 04/21/2013
// Module Name:    display_8hex

// Description:  Display 8 hex numbers on 7 segment display
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module display_8hex(
    input clk,                  // system clock
    input [31:0] data,          // 8 hex numbers, msb first
    output reg [6:0] seg,       // seven segment display output
    output reg [7:0] strobe     // digit strobe
    );

    localparam bits = 13;

```

```
25      reg [bits:0] counter = 0;  // clear on power up
26
27      wire [6:0] segments[15:0]; // 16 7 bit memorys
28      assign segments[0]  = 7'b100_0000;
29      assign segments[1]  = 7'b111_1001;
30      assign segments[2]  = 7'b010_0100;
31      assign segments[3]  = 7'b011_0000;
32      assign segments[4]  = 7'b001_1001;
33      assign segments[5]  = 7'b001_0010;
34      assign segments[6]  = 7'b000_0010;
35      assign segments[7]  = 7'b111_1000;
36      assign segments[8]  = 7'b000_0000;
37      assign segments[9]  = 7'b001_1000;
38      assign segments[10] = 7'b000_1000;
39      assign segments[11] = 7'b000_0011;
40      assign segments[12] = 7'b010_0111;
41      assign segments[13] = 7'b010_0001;
42      assign segments[14] = 7'b000_0110;
43      assign segments[15] = 7'b000_1110;
44
45      always @(posedge clk) begin
46        counter <= counter + 1;
47        case (counter[bits:bits-2])
48            3'b000: begin
49                    seg <= segments[data[31:28]];
50                    strobe <= 8'b0111_1111 ;
51                   end
52
53            3'b001: begin
54                    seg <= segments[data[27:24]];
55                    strobe <= 8'b1011_1111 ;
56                   end
57
58            3'b010: begin
59                     seg <= segments[data[23:20]];
60                     strobe <= 8'b1101_1111 ;
61                    end
62            3'b011: begin
63                    seg <= segments[data[19:16]];
64                    strobe <= 8'b1110_1111;
65                   end
66            3'b100: begin
67                    seg <= segments[data[15:12]];
68                    strobe <= 8'b1111_0111;
69                   end
70
71            3'b101: begin
72                    seg <= segments[data[11:8]];
73                    strobe <= 8'b1111_1011;
74                   end
75
76            3'b110: begin
77                     seg <= segments[data[7:4]];
78                     strobe <= 8'b1111_1101;
79                    end
80            3'b111: begin
81                    seg <= segments[data[3:0]];
82                    strobe <= 8'b1111_1110;
83                   end
84
85
86
87
88         endcase
89        end
90
91  endmodule
```

```
1  // Switch Debounce Module
2  // use your system clock for the clock input
3  // to produce a synchronous, debounced output
4  module debounce #(parameter DELAY=1000000)   // .01 sec with a 100Mhz clock
5              (input reset, clock, noisy,
6               output reg clean);
7
8     reg [19:0] count;
9     reg new;
10
11    always @(posedge clock)
12      if (reset)
13        begin
14       count <= 0;
15       new <= noisy;
16       clean <= noisy;
17        end
18      else if (noisy != new)
19        begin
20       new <= noisy;
21       count <= 0;
22        end
23      else if (count == DELAY)
24        clean <= new;
25      else
26        count <= count+1;
27
28 endmodule
```

Appendix D

Nexys4 Constraint File (xdc)

```
## This file is a general .xdc for the Nexys4 DDR Rev. C
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];


##Switches

set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { SW[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { SW[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { SW[13] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { SW[14] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14
    ↪ ]
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { SW[15] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]


## LEDs

set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { LED[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { LED[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { LED[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { LED[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { LED[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { LED[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { LED[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { LED[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { LED[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { LED[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { LED[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { LED[11] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=
    ↪ led[11]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { LED[12] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports { LED[13] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { LED[14] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { LED[15] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[
    ↪ 15]

set_property -dict { PACKAGE_PIN R12   IOSTANDARD LVCMOS33 } [get_ports { LED16_B }]; #IO_L5P_T0_D06_14 Sch=led16_b
set_property -dict { PACKAGE_PIN M16   IOSTANDARD LVCMOS33 } [get_ports { LED16_G }]; #IO_L10P_T1_D14_14 Sch=led16_g
set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { LED16_R }]; #IO_L11P_T1_SRCC_14 Sch=led16_r
set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { LED17_B }]; #IO_L15N_T2_DQS_ADV_B_15 Sch=led17_b
set_property -dict { PACKAGE_PIN R11   IOSTANDARD LVCMOS33 } [get_ports { LED17_G }]; #IO_0_14 Sch=led17_g
set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports { LED17_R }]; #IO_L11N_T1_SRCC_14 Sch=led17_r


##7 segment display

set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { SEG[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { SEG[1] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { SEG[2] }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { SEG[3] }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { SEG[4] }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { SEG[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { SEG[6] }]; #IO_L4P_T0_D04_14 Sch=cg

set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { SEG[7] }]; #IO_L19N_T3_A21_VREF_15 Sch=dp

set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { AN[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { AN[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { AN[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { AN[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { AN[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { AN[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { AN[6] }]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { AN[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]


##Buttons

#set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 } [get_ports { CPU_RESETN }]; #IO_L3P_T0_DQS_AD1P_15 Sch=
    ↪ cpu_resetn

set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { BTNC }]; #IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { BTNU }]; #IO_L4N_T0_D05_14 Sch=btnu
set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { BTNL }]; #IO_L12P_T1_MRCC_14 Sch=btnl
set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { BTNR }]; #IO_L10N_T1_D15_14 Sch=btnr
set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { BTND }]; #IO_L9N_T1_DQS_D13_14 Sch=btnd
```

```
89
90
91  ##Pmod Headers
92
93
94  ##Pmod Header JA
95
96  set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33 } [get_ports { JA[0] }]; #IO_L20N_T3_A19_15 Sch=ja[1]
97  set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { JA[1] }]; #IO_L21N_T3_DQS_A18_15 Sch=ja[2]
98  set_property -dict { PACKAGE_PIN E18   IOSTANDARD LVCMOS33 } [get_ports { JA[2] }]; #IO_L21P_T3_DQS_15 Sch=ja[3]
99  set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { JA[3] }]; #IO_L18N_T2_A23_15 Sch=ja[4]
100 set_property -dict { PACKAGE_PIN D17   IOSTANDARD LVCMOS33 } [get_ports { JA[4] }]; #IO_L16N_T2_A27_15 Sch=ja[7]
101 set_property -dict { PACKAGE_PIN E17   IOSTANDARD LVCMOS33 } [get_ports { JA[5] }]; #IO_L16P_T2_A28_15 Sch=ja[8]
102 set_property -dict { PACKAGE_PIN F18   IOSTANDARD LVCMOS33 } [get_ports { JA[6] }]; #IO_L22N_T3_A16_15 Sch=ja[9]
103 set_property -dict { PACKAGE_PIN G18   IOSTANDARD LVCMOS33 } [get_ports { JA[7] }]; #IO_L22P_T3_A17_15 Sch=ja[10]
104
105
106 ##Pmod Header JB
107
108 #set_property -dict { PACKAGE_PIN D14   IOSTANDARD LVCMOS33 } [get_ports { JB[0] }]; #IO_L1P_T0_AD0P_15 Sch=jb[1]
109 #set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { JB[1] }]; #IO_L14N_T2_SRCC_15 Sch=jb[2]
110 #set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 } [get_ports { JB[2] }]; #IO_L13N_T2_MRCC_15 Sch=jb[3]
111 #set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [get_ports { JB[3] }]; #IO_L15P_T2_DQS_15 Sch=jb[4]
112 #set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33 } [get_ports { JB[4] }]; #IO_L11N_T1_SRCC_15 Sch=jb[7]
113 #set_property -dict { PACKAGE_PIN F13   IOSTANDARD LVCMOS33 } [get_ports { JB[5] }]; #IO_L5P_T0_AD9P_15 Sch=jb[8]
114 #set_property -dict { PACKAGE_PIN G13   IOSTANDARD LVCMOS33 } [get_ports { JB[6] }]; #IO_0_15 Sch=jb[9]
115 #set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { JB[7] }]; #IO_L13P_T2_MRCC_15 Sch=jb[10]
116
117 set_property -dict { PACKAGE_PIN F16   IOSTANDARD LVCMOS33 } [get_ports { JB_IN[1] }]; #IO_L14N_T2_SRCC_15 Sch=jb[2]
118 set_property -dict { PACKAGE_PIN G16   IOSTANDARD LVCMOS33 } [get_ports { JB_OUT[0] }]; #IO_L13N_T2_MRCC_15 Sch=jb[3]
119 set_property -dict { PACKAGE_PIN H14   IOSTANDARD LVCMOS33 } [get_ports { JB_OUT[1] }]; #IO_L15P_T2_DQS_15 Sch=jb[4]
120 set_property -dict { PACKAGE_PIN E16   IOSTANDARD LVCMOS33 } [get_ports { JB_OUT[2] }]; #IO_L11N_T1_SRCC_15 Sch=jb[7]
121 set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { JB_IN[0] }]; #IO_L13P_T2_MRCC_15 Sch=jb[10]
122
123 set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {JB_IN_IBUF[0]}]
124 ##Pmod Header JC
125
126 set_property -dict { PACKAGE_PIN K1    IOSTANDARD LVCMOS33 } [get_ports { JC[0] }]; #IO_L23N_T3_35 Sch=jc[1]
127 set_property -dict { PACKAGE_PIN F6    IOSTANDARD LVCMOS33 } [get_ports { JC[1] }]; #IO_L19N_T3_VREF_35 Sch=jc[2]
128 set_property -dict { PACKAGE_PIN J2    IOSTANDARD LVCMOS33 } [get_ports { JC[2] }]; #IO_L22N_T3_35 Sch=jc[3]
129 set_property -dict { PACKAGE_PIN G6    IOSTANDARD LVCMOS33 } [get_ports { JC[3] }]; #IO_L19P_T3_35 Sch=jc[4]
130 set_property -dict { PACKAGE_PIN E7    IOSTANDARD LVCMOS33 } [get_ports { JC[4] }]; #IO_L6P_T0_35 Sch=jc[7]
131 set_property -dict { PACKAGE_PIN J3    IOSTANDARD LVCMOS33 } [get_ports { JC[5] }]; #IO_L22P_T3_35 Sch=jc[8]
132 set_property -dict { PACKAGE_PIN J4    IOSTANDARD LVCMOS33 } [get_ports { JC[6] }]; #IO_L21P_T3_DQS_35 Sch=jc[9]
133 set_property -dict { PACKAGE_PIN E6    IOSTANDARD LVCMOS33 } [get_ports { JC[7] }]; #IO_L5P_T0_AD13P_35 Sch=jc[10]
134
135
136 ##Pmod Header JD
137
138 set_property -dict { PACKAGE_PIN H4    IOSTANDARD LVCMOS33 } [get_ports { JD[0] }]; #IO_L21N_T3_DQS_35 Sch=jd[1]
139 set_property -dict { PACKAGE_PIN H1    IOSTANDARD LVCMOS33 } [get_ports { JD[1] }]; #IO_L17P_T2_35 Sch=jd[2]
140 set_property -dict { PACKAGE_PIN G1    IOSTANDARD LVCMOS33 } [get_ports { JD[2] }]; #IO_L17N_T2_35 Sch=jd[3]
141 set_property -dict { PACKAGE_PIN G3    IOSTANDARD LVCMOS33 } [get_ports { JD[3] }]; #IO_L20N_T3_35 Sch=jd[4]
142 set_property -dict { PACKAGE_PIN H2    IOSTANDARD LVCMOS33 } [get_ports { JD[4] }]; #IO_L15P_T2_DQS_35 Sch=jd[7]
143 set_property -dict { PACKAGE_PIN G4    IOSTANDARD LVCMOS33 } [get_ports { JD[5] }]; #IO_L20P_T3_35 Sch=jd[8]
144 set_property -dict { PACKAGE_PIN G2    IOSTANDARD LVCMOS33 } [get_ports { JD[6] }]; #IO_L15N_T2_DQS_35 Sch=jd[9]
145 set_property -dict { PACKAGE_PIN F3    IOSTANDARD LVCMOS33 } [get_ports { JD[7] }]; #IO_L13N_T2_MRCC_35 Sch=jd[10]
146
147 #set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {JD_IBUF[7]}]
148
149 #set_property -dict { PACKAGE_PIN F3    IOSTANDARD LVCMOS33 } [get_ports { JD_IN[0] }]; #IO_L13N_T2_MRCC_35 Sch=jd[10]
150 #set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {JD_IN_IBUF[0]}]
151 ##set_property -dict { PACKAGE_PIN H4    IOSTANDARD LVCMOS33 } [get_ports { JD_IN[0] }]; #IO_L21N_T3_DQS_35 Sch=jd[1]
152 #set_property -dict { PACKAGE_PIN H1    IOSTANDARD LVCMOS33 } [get_ports { JD_IN[1] }]; #IO_L17P_T2_35 Sch=jd[2]
153
154 #set_property -dict { PACKAGE_PIN G1    IOSTANDARD LVCMOS33 } [get_ports { JD_OUT[0] }]; #IO_L17N_T2_35 Sch=jd[3]
155
156 #set_property -dict { PACKAGE_PIN G3    IOSTANDARD LVCMOS33 } [get_ports { JD_OUT[1] }]; #IO_L20N_T3_35 Sch=jd[4]
157 #set_property -dict { PACKAGE_PIN H2    IOSTANDARD LVCMOS33 } [get_ports { JD_OUT[2] }]; #IO_L15P_T2_DQS_35 Sch=jd[7]
158 ##set_property -dict { PACKAGE_PIN G4    IOSTANDARD LVCMOS33 } [get_ports { JD_OUT[3] }]; #IO_L20P_T3_35 Sch=jd[8]
159 ##set_property -dict { PACKAGE_PIN G2    IOSTANDARD LVCMOS33 } [get_ports { JD_OUT[4] }]; #IO_L15N_T2_DQS_35 Sch=jd[9]
160 ##set_property -dict { PACKAGE_PIN F3    IOSTANDARD LVCMOS33 } [get_ports { JD_OUT[5] }]; #IO_L13N_T2_MRCC_35 Sch=jd[10]
161
162
163 ##Pmod Header JXADC
164
165 #set_property -dict { PACKAGE_PIN A14   IOSTANDARD LVDS     } [get_ports { XA_N[1] }]; #IO_L9N_T1_DQS_AD3N_15 Sch=xa_n[1]
166 #set_property -dict { PACKAGE_PIN A13   IOSTANDARD LVDS     } [get_ports { XA_P[1] }]; #IO_L9P_T1_DQS_AD3P_15 Sch=xa_p[1]
167 #set_property -dict { PACKAGE_PIN A16   IOSTANDARD LVDS     } [get_ports { XA_N[2] }]; #IO_L8N_T1_AD10N_15 Sch=xa_n[2]
168 #set_property -dict { PACKAGE_PIN A15   IOSTANDARD LVDS     } [get_ports { XA_P[2] }]; #IO_L8P_T1_AD10P_15 Sch=xa_p[2]
169 #set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVDS     } [get_ports { XA_N[3] }]; #IO_L7N_T1_AD2N_15 Sch=xa_n[3]
170 #set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVDS     } [get_ports { XA_P[3] }]; #IO_L7P_T1_AD2P_15 Sch=xa_p[3]
171 #set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVDS     } [get_ports { XA_N[4] }]; #IO_L10N_T1_AD11N_15 Sch=xa_n[4]
172 #set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVDS     } [get_ports { XA_P[4] }]; #IO_L10P_T1_AD11P_15 Sch=xa_p[4]
173
174
175 ##VGA Connector
176
177 set_property -dict { PACKAGE_PIN A3    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[0] }]; #IO_L8N_T1_AD14N_35 Sch=vga_r[0]
178 set_property -dict { PACKAGE_PIN B4    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[1] }]; #IO_L7N_T1_AD6N_35 Sch=vga_r[1]
179 set_property -dict { PACKAGE_PIN C5    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[2] }]; #IO_L1N_T0_AD4N_35 Sch=vga_r[2]
180 set_property -dict { PACKAGE_PIN A4    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[3] }]; #IO_L8P_T1_AD14P_35 Sch=vga_r[3]
```

```
181
182 set_property -dict { PACKAGE_PIN C6    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[0] }]; #IO_L1P_T0_AD4P_35 Sch=vga_g[0]
183 set_property -dict { PACKAGE_PIN A5    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[1] }]; #IO_L3N_T0_DQS_AD5N_35 Sch=vga_g[1]
184 set_property -dict { PACKAGE_PIN B6    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[2] }]; #IO_L2N_T0_AD12N_35 Sch=vga_g[2]
185 set_property -dict { PACKAGE_PIN A6    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[3] }]; #IO_L3P_T0_DQS_AD5P_35 Sch=vga_g[3]
186
187 set_property -dict { PACKAGE_PIN B7    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[0] }]; #IO_L2P_T0_AD12P_35 Sch=vga_b[0]
188 set_property -dict { PACKAGE_PIN C7    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[1] }]; #IO_L4N_T0_35 Sch=vga_b[1]
189 set_property -dict { PACKAGE_PIN D7    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[2] }]; #IO_L6N_T0_VREF_35 Sch=vga_b[2]
190 set_property -dict { PACKAGE_PIN D8    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[3] }]; #IO_L4P_T0_35 Sch=vga_b[3]
191
192 set_property -dict { PACKAGE_PIN B11   IOSTANDARD LVCMOS33 } [get_ports { VGA_HS }]; #IO_L4P_T0_15 Sch=vga_hs
193 set_property -dict { PACKAGE_PIN B12   IOSTANDARD LVCMOS33 } [get_ports { VGA_VS }]; #IO_L3N_T0_DQS_AD1N_15 Sch=vga_vs
194
195 ##Micro SD Connector
196
197 #set_property -dict { PACKAGE_PIN E2    IOSTANDARD LVCMOS33 } [get_ports { SD_RESET }]; #IO_L14P_T2_SRCC_35 Sch=sd_reset
198 #set_property -dict { PACKAGE_PIN A1    IOSTANDARD LVCMOS33 } [get_ports { SD_CD }]; #IO_L9N_T1_DQS_AD7N_35 Sch=sd_cd
199 #set_property -dict { PACKAGE_PIN B1    IOSTANDARD LVCMOS33 } [get_ports { SD_SCK }]; #IO_L9P_T1_DQS_AD7P_35 Sch=sd_sck
200 #set_property -dict { PACKAGE_PIN C1    IOSTANDARD LVCMOS33 } [get_ports { SD_CMD }]; #IO_L16N_T2_35 Sch=sd_cmd
201 #set_property -dict { PACKAGE_PIN C2    IOSTANDARD LVCMOS33 } [get_ports { SD_DAT[0] }]; #IO_L16P_T2_35 Sch=sd_dat[0]
202 #set_property -dict { PACKAGE_PIN E1    IOSTANDARD LVCMOS33 } [get_ports { SD_DAT[1] }]; #IO_L18N_T2_35 Sch=sd_dat[1]
203 #set_property -dict { PACKAGE_PIN F1    IOSTANDARD LVCMOS33 } [get_ports { SD_DAT[2] }]; #IO_L18P_T2_35 Sch=sd_dat[2]
204 #set_property -dict { PACKAGE_PIN D2    IOSTANDARD LVCMOS33 } [get_ports { SD_DAT[3] }]; #IO_L14N_T2_SRCC_35 Sch=sd_dat[3]
205
206
207 ##Accelerometer
208
209 #set_property -dict { PACKAGE_PIN E15   IOSTANDARD LVCMOS33 } [get_ports { ACL_MISO }]; #IO_L11P_T1_SRCC_15 Sch=acl_miso
210 #set_property -dict { PACKAGE_PIN F14   IOSTANDARD LVCMOS33 } [get_ports { ACL_MOSI }]; #IO_L5N_T0_AD9N_15 Sch=acl_mosi
211 #set_property -dict { PACKAGE_PIN F15   IOSTANDARD LVCMOS33 } [get_ports { ACL_SCLK }]; #IO_L14P_T2_SRCC_15 Sch=acl_sclk
212 #set_property -dict { PACKAGE_PIN D15   IOSTANDARD LVCMOS33 } [get_ports { ACL_CSN }]; #IO_L12P_T1_MRCC_15 Sch=acl_csn
213 #set_property -dict { PACKAGE_PIN B13   IOSTANDARD LVCMOS33 } [get_ports { ACL_INT[1] }]; #IO_L2P_T0_AD8P_15 Sch=acl_int[1
      ↪ ]
214 #set_property -dict { PACKAGE_PIN C16   IOSTANDARD LVCMOS33 } [get_ports { ACL_INT[2] }]; #IO_L20P_T3_A20_15 Sch=acl_int[2
      ↪ ]
215
216
217 ##Temperature Sensor
218
219 #set_property -dict { PACKAGE_PIN C14   IOSTANDARD LVCMOS33 } [get_ports { TMP_SCL }]; #IO_L1N_T0_AD0N_15 Sch=tmp_scl
220 #set_property -dict { PACKAGE_PIN C15   IOSTANDARD LVCMOS33 } [get_ports { TMP_SDA }]; #IO_L12N_T1_MRCC_15 Sch=tmp_sda
221 #set_property -dict { PACKAGE_PIN D13   IOSTANDARD LVCMOS33 } [get_ports { TMP_INT }]; #IO_L6N_T0_VREF_15 Sch=tmp_int
222 #set_property -dict { PACKAGE_PIN B14   IOSTANDARD LVCMOS33 } [get_ports { TMP_CT }]; #IO_L2N_T0_AD8N_15 Sch=tmp_ct
223
224 ##Omnidirectional Microphone
225
226 #set_property -dict { PACKAGE_PIN J5    IOSTANDARD LVCMOS33 } [get_ports { M_CLK }]; #IO_25_35 Sch=m_clk
227 #set_property -dict { PACKAGE_PIN H5    IOSTANDARD LVCMOS33 } [get_ports { M_DATA }]; #IO_L24N_T3_35 Sch=m_data
228 #set_property -dict { PACKAGE_PIN F5    IOSTANDARD LVCMOS33 } [get_ports { M_LRSEL }]; #IO_0_35 Sch=m_lrsel
229
230
231 ##PWM Audio Amplifier
232
233 #set_property -dict { PACKAGE_PIN A11   IOSTANDARD LVCMOS33 } [get_ports { AUD_PWM }]; #IO_L4N_T0_15 Sch=aud_pwm
234 #set_property -dict { PACKAGE_PIN D12   IOSTANDARD LVCMOS33 } [get_ports { AUD_SD }]; #IO_L6P_T0_15 Sch=aud_sd
235
236
237 ##USB-RS232 Interface
238
239 #set_property -dict { PACKAGE_PIN C4    IOSTANDARD LVCMOS33 } [get_ports { UART_TXD_IN }]; #IO_L7P_T1_AD6P_35 Sch=
      ↪ uart_txd_in
240 #set_property -dict { PACKAGE_PIN D4    IOSTANDARD LVCMOS33 } [get_ports { UART_RXD_OUT }]; #IO_L11N_T1_SRCC_35 Sch=
      ↪ uart_rxd_out
241 #set_property -dict { PACKAGE_PIN D3    IOSTANDARD LVCMOS33 } [get_ports { UART_CTS }]; #IO_L12N_T1_MRCC_35 Sch=uart_cts
242 #set_property -dict { PACKAGE_PIN E5    IOSTANDARD LVCMOS33 } [get_ports { UART_RTS }]; #IO_L5N_T0_AD13N_35 Sch=uart_rts
243
244 ##USB HID (PS/2)
245
246 #set_property -dict { PACKAGE_PIN F4    IOSTANDARD LVCMOS33 } [get_ports { PS2_CLK }]; #IO_L13P_T2_MRCC_35 Sch=ps2_clk
247 #set_property -dict { PACKAGE_PIN B2    IOSTANDARD LVCMOS33 } [get_ports { PS2_DATA }]; #IO_L10N_T1_AD15N_35 Sch=ps2_data
248
249
250 ##SMSC Ethernet PHY
251
252 #set_property -dict { PACKAGE_PIN C9    IOSTANDARD LVCMOS33 } [get_ports { ETH_MDC }]; #IO_L11P_T1_SRCC_16 Sch=eth_mdc
253 #set_property -dict { PACKAGE_PIN A9    IOSTANDARD LVCMOS33 } [get_ports { ETH_MDIO }]; #IO_L14N_T2_SRCC_16 Sch=eth_mdio
254 #set_property -dict { PACKAGE_PIN B3    IOSTANDARD LVCMOS33 } [get_ports { ETH_RSTN }]; #IO_L10P_T1_AD15P_35 Sch=eth_rstn
255 #set_property -dict { PACKAGE_PIN D9    IOSTANDARD LVCMOS33 } [get_ports { ETH_CRSDV }]; #IO_L6N_T0_VREF_16 Sch=eth_crsdv
256 #set_property -dict { PACKAGE_PIN C10   IOSTANDARD LVCMOS33 } [get_ports { ETH_RXERR }]; #IO_L13N_T2_MRCC_16 Sch=eth_rxerr
257 #set_property -dict { PACKAGE_PIN C11   IOSTANDARD LVCMOS33 } [get_ports { ETH_RXD[0] }]; #IO_L13P_T2_MRCC_16 Sch=eth_rxd[
      ↪ 0]
258 #set_property -dict { PACKAGE_PIN D10   IOSTANDARD LVCMOS33 } [get_ports { ETH_RXD[1] }]; #IO_L19N_T3_VREF_16 Sch=eth_rxd[
      ↪ 1]
259 #set_property -dict { PACKAGE_PIN B9    IOSTANDARD LVCMOS33 } [get_ports { ETH_TXEN }]; #IO_L11N_T1_SRCC_16 Sch=eth_txen
260 #set_property -dict { PACKAGE_PIN A10   IOSTANDARD LVCMOS33 } [get_ports { ETH_TXD[0] }]; #IO_L14P_T2_SRCC_16 Sch=eth_txd[
      ↪ 0]
261 #set_property -dict { PACKAGE_PIN A8    IOSTANDARD LVCMOS33 } [get_ports { ETH_TXD[1] }]; #IO_L12N_T1_MRCC_16 Sch=eth_txd[
      ↪ 1]
262 #set_property -dict { PACKAGE_PIN D5    IOSTANDARD LVCMOS33 } [get_ports { ETH_REFCLK }]; #IO_L11P_T1_SRCC_35 Sch=
      ↪ eth_refclk
263 #set_property -dict { PACKAGE_PIN B8    IOSTANDARD LVCMOS33 } [get_ports { ETH_INTN }]; #IO_L12P_T1_MRCC_16 Sch=eth_intn
```

```
264
265
266  ##Quad SPI Flash
267
268  #set_property -dict { PACKAGE_PIN K17   IOSTANDARD LVCMOS33 } [get_ports { QSPI_DQ[0] }]; #IO_L1P_T0_D00_MOSI_14 Sch=
         ↪ qspi_dq[0]
269  #set_property -dict { PACKAGE_PIN K18   IOSTANDARD LVCMOS33 } [get_ports { QSPI_DQ[1] }]; #IO_L1N_T0_D01_DIN_14 Sch=
         ↪ qspi_dq[1]
270  #set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS33 } [get_ports { QSPI_DQ[2] }]; #IO_L2P_T0_D02_14 Sch=qspi_dq[2]
271  #set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { QSPI_DQ[3] }]; #IO_L2N_T0_D03_14 Sch=qspi_dq[3]
272  #set_property -dict { PACKAGE_PIN L13   IOSTANDARD LVCMOS33 } [get_ports { QSPI_CSN }]; #IO_L6P_T0_FCS_B_14 Sch=qspi_csn
```

Appendix E

Kinect Software Code – Kinect 2.0 SDK

```
1  ////////////////////////////////////////////////////////////////////////////
2  // Program Name: kinectDepthStream
3  // Author: Alan Cheng
4  // Description: Streams depth frame to FT2232H using Kinect SDK 2.0 drivers
5  //
6  // Additional Comments: very unpolished; basic off example D2XX/Kinect code
7  //
8  ////////////////////////////////////////////////////////////////////////////
9  #include "stdafx.h"
10 #include <opencv2/core/core.hpp>
11 #include <opencv2/highgui/highgui.hpp>
12 #include <opencv2/imgproc/imgproc.hpp>
13 #include <math.h>
14 #include <Windows.h>
15 #include <Kinect.h>
16 #include <iostream>
17 #include <fstream>
18 #include <string>
19 #include "ftd2xx.h"
20
21 using namespace cv;
22 using namespace std;
23
24 IKinectSensor* pSensor;
25 HRESULT hResult;
26 IColorFrameSource* pColorSource;
27 IColorFrameReader* pColorReader;
28 IFrameDescription* pColorDescription;
29 IDepthFrameSource* pDepthSource;
30 IDepthFrameReader* pDepthReader;
31 IFrameDescription* pDescription;
32
33 template<class Interface>
34 inline void SafeRelease(Interface *& pInterfaceToRelease)
35 {
36     if (pInterfaceToRelease != NULL) {
37         pInterfaceToRelease->Release();
38         pInterfaceToRelease = NULL;
39     }
40 }
41
42 inline int KinectInit()
43 {
44     cv::setUseOptimized(true);
45
46     // Sensor
47     hResult = S_OK;
48     hResult = GetDefaultKinectSensor(&pSensor);
49     if (FAILED(hResult)) {
50         std::cerr << "Error␣:␣GetDefaultKinectSensor" << std::endl;
51         return -1;
52     }
53
54     hResult = pSensor->Open();
55     if (FAILED(hResult)) {
56         std::cerr << "Error␣:␣IKinectSensor::Open()" << std::endl;
57         return -1;
58     }
59
60     //--------------------[Depth]---------------------------
61     // Source
62     hResult = pSensor->get_DepthFrameSource(&pDepthSource);
63     if (FAILED(hResult)) {
64         std::cerr << "Error␣:␣IKinectSensor::get_DepthFrameSource()" << std::endl;
65         return -1;
66     }
67
68     // Reader
69     hResult = pDepthSource->OpenReader(&pDepthReader);
70     if (FAILED(hResult)) {
71         std::cerr << "Error␣:␣IDepthFrameSource::OpenReader()" << std::endl;
72         return -1;
73     }
74
75     // Description
76     hResult = pDepthSource->get_FrameDescription(&pDescription);
77     if (FAILED(hResult)) {
78         std::cerr << "Error␣:␣IDepthFrameSource::get_FrameDescription()" << std::endl;
79         return -1;
80     }
81
82 }
83
84 int main()
85 {
86     KinectInit();
87
88     //-------------------------[Depth Initialization]-------------------------
89     int dWidth = 0;
90     int dHeight = 0;
91     pDescription->get_Width(&dWidth); // 512
92     pDescription->get_Height(&dHeight); // 424
```

```
93      unsigned int bufferSize = dWidth * dHeight * sizeof(unsigned short);
94
95      // Range ( Range of Depth is 500-8000[mm], Range of Detection is 500-4500[mm] )
96      unsigned short min = 0;
97      unsigned short max = 0;
98      pDepthSource->get_DepthMinReliableDistance(&min); // 500
99      pDepthSource->get_DepthMaxReliableDistance(&max); // 4500
100     std::cout << "Range : " << min << " - " << max << std::endl;
101
102     //------------------------------------------------------------------------
103
104     cv::Mat backgroundMat(dHeight, dWidth, CV_16UC1);
105     cv::Mat bufferMat(dHeight, dWidth, CV_16UC1);
106     cv::Mat binaryMat(dHeight, dWidth, CV_8UC1);
107     cv::Mat depthMat(dHeight, dWidth, CV_8UC1);
108
109     //------------------------------------------------------------------------
110
111     //cv::namedWindow("Color");
112     cv::namedWindow("Depth");
113
114     //IColorFrame* pColorFrame = nullptr;
115     IDepthFrame* pDepthFrame = nullptr;
116
117     UINT16* backBuffer = nullptr;
118     cv::Mat backFrame(dHeight, dWidth, CV_16UC1);
119
120     //------------------------------------------------------------------------
121
122     FT_HANDLE fthandle;
123     FT_STATUS ftstatus;
124
125     UCHAR LatencyTimer = 2; //our default setting is 16
126
127     /*****************************************************/
128     // Configure the FT2232HL to Synchronous FIFO
129     /*****************************************************/
130     ftstatus = FT_Open(0, &fthandle);
131
132     if (ftstatus != FT_OK) {
133         printf("opening failed! with error %d\n", ftstatus);
134         return 1;
135     }
136
137     DWORD InTransferSize = 64000;
138     ftstatus = FT_SetUSBParameters(fthandle, InTransferSize, 0);
139     if (ftstatus != FT_OK) {
140         printf("changing buffer size failed! with error %d\n", ftstatus);
141         return 1;
142     }
143
144     UCHAR mask = 0xff;
145     UCHAR mode;
146
147     mode = 0x00; // reset mode
148     ftstatus = FT_SetBitMode(fthandle, mask, mode);
149
150     Sleep(10);
151
152     mode = 0x40; // synchronous FIFO mode
153     ftstatus = FT_SetBitMode(fthandle, mask, mode);
154     FT_Purge(fthandle, FT_PURGE_RX);
155
156     if (ftstatus != FT_OK) {
157         printf("configuration failed! with error %d\n", ftstatus);
158         return 1;
159     }
160     else {
161         ftstatus = FT_SetLatencyTimer(fthandle, LatencyTimer);
162         ftstatus = FT_SetUSBParameters(fthandle, 0x10000, 0x10000);
163         ftstatus = FT_SetFlowControl(fthandle, FT_FLOW_RTS_CTS, 0x0, 0x0);
164         ftstatus = FT_SetTimeouts(fthandle, 5000, 5000);
165     }
166     printf("Synchronous FIFO mode set!\n");
167
168     DWORD EventDWord;
169     DWORD RxBytes;
170     DWORD TxBytes;
171     DWORD BytesWritten;
172
173     UCHAR data_buf_init[8] = { 0x00, 0x00, 0x00, 0x00, 0xDD, 0xCC, 0xBB, 0xAA};
174
175     //-------------------------------[Get Background Loop]----------------------------------
176     while (1) {
177 #define sendBufferSize 640000
178         unsigned char ImgBuffer[640000];
179         pDepthFrame = nullptr;
180
181         hResult = pDepthReader->AcquireLatestFrame(&pDepthFrame);
182         if (SUCCEEDED(hResult))
183         {
184             hResult = pDepthFrame->AccessUnderlyingBuffer(&bufferSize, reinterpret_cast<UINT16**>(&bufferMat.data));
```

```
            if (SUCCEEDED(hResult))
            {
                pDepthFrame->AccessUnderlyingBuffer(&bufferSize, &backBuffer);
                for (int i = 0; i < 15; i++)
                    ImgBuffer[i] = data_buf_init[i];
                int index = 16;
                for (int i = 0; i < dHeight; i++)
                {
                    for (int j = 0; j < dWidth; j++)
                    {
                        UINT16 value = backBuffer[i * dWidth + j];
                        char lo = value & 0xFF;
                        char hi = value >> 8;

                        ImgBuffer[index] = hi;
                        ImgBuffer[index + 1] = lo;

                        index += 2;
                    }
                }
                bufferMat.convertTo(depthMat, CV_8U, -255.0f / 8000.0f, 255.0f);
            }
        }

        /*********************************************************/
        // Write to FT2232HL
        /*********************************************************/
        ftstatus = FT_GetStatus(fthandle, &RxBytes, &TxBytes, &EventDWord);

        if (ftstatus == FT_OK) {
            ftstatus = FT_Write(fthandle, ImgBuffer, sizeof(ImgBuffer), &BytesWritten);
        }

        SafeRelease(pDepthFrame);

        cv::imshow("Depth", depthMat);

        if (cv::waitKey(30) == VK_ESCAPE) {
            break;
        }
    }

    cv::destroyAllWindows();

    SafeRelease(pColorSource);
    SafeRelease(pColorReader);
    SafeRelease(pColorDescription);

    SafeRelease(pDepthSource);
    SafeRelease(pDepthReader);
    SafeRelease(pDescription);
    if (pSensor) {
        pSensor->Close();
    }
    SafeRelease(pSensor);
    cv::destroyAllWindows();

    return 0;
}
```

Appendix F

Kinect Software Code – libfreenect2

```
////////////////////////////////////////////////////////////////////////////
// Program Name: libfreenect2Stream
// Author: Alan Cheng
// Description: Streams depth frame to FT2232H using libfreenect2 drivers
//
// Additional Comments: very unpolished; basic off example libfreenect2 code
//
////////////////////////////////////////////////////////////////////////////////

//! [headers]
#include <iostream>
#include <stdio.h>
#include <iomanip>
#include <time.h>
#include <signal.h>

//! [OpenCV library]
#include <opencv2/core/core.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>

#include <math.h>
#include <Windows.h>

//! [libfreenect2 library]
#include <libfreenect2/libfreenect2.hpp>
#include <libfreenect2/frame_listener_impl.h>
#include <libfreenect2/registration.h>
#include <libfreenect2/packet_pipeline.h>
#include <libfreenect2/logger.h>

#include "ftd2xx.h"
//! [headers]

using namespace std;
using namespace cv;

bool protonect_shutdown = false; // Whether the running application should shut down.

void sigint_handler(int s)
{
    protonect_shutdown = true;
}

int main()
{
    std::cout << "Streaming␣from␣Kinect␣One␣sensor!" << std::endl;

    //! [context]
    libfreenect2::Freenect2 freenect2;
    libfreenect2::Freenect2Device *dev = 0;
    libfreenect2::PacketPipeline *pipeline = 0;
    //! [context]

    //! [discovery]
    if (freenect2.enumerateDevices() == 0)
    {
        std::cout << "no␣device␣connected!" << std::endl;
        return -1;
    }

    string serial = freenect2.getDefaultDeviceSerialNumber();

    std::cout << "SERIAL:␣" << serial << std::endl;

    pipeline = new libfreenect2::OpenCLPacketPipeline();

    if (pipeline)
    {
        //! [open]
        dev = freenect2.openDevice(serial, pipeline);
        //! [open]
    }
    else {
        dev = freenect2.openDevice(serial);
    }

    if (dev == 0)
    {
        std::cout << "failure␣opening␣device!" << std::endl;
        return -1;
    }

    signal(SIGINT, sigint_handler);
    protonect_shutdown = false;

    libfreenect2::Freenect2Device::Config config;
    config.EnableBilateralFilter = true;
    config.EnableEdgeAwareFilter = true;
    dev->setConfiguration(config);

    //! [listeners]
```

```
93      libfreenect2::SyncMultiFrameListener listener(libfreenect2::Frame::Depth);
94      libfreenect2::FrameMap frames;
95
96      //dev->setColorFrameListener(&listener);
97      dev->setIrAndDepthFrameListener(&listener);
98      //! [listeners]
99
100     //! [start]
101     dev->start();
102
103     std::cout << "device␣serial:␣" << dev->getSerialNumber() << std::endl;
104     std::cout << "device␣firmware:␣" << dev->getFirmwareVersion() << std::endl;
105     //! [start]
106
107     //! [registration setup]
108     libfreenect2::Registration* registration = new libfreenect2::Registration(dev->getIrCameraParams(), dev->
            ↪ getColorCameraParams());
109
110     Mat depthmat;
111
112     //! [ft2232h setup]
113     FT_HANDLE fthandle;
114     FT_STATUS ftstatus;
115
116     UCHAR LatencyTimer = 2; //our default setting is 16
117
118     // Configure the FT2232HL to Synchronous FIFO
119     ftstatus = FT_Open(0, &fthandle);
120
121     if (ftstatus != FT_OK) {
122         printf("opening␣failed!␣with␣error␣%d\n", ftstatus);
123         return 1;
124     }
125
126     DWORD InTransferSize = 64000;
127     ftstatus = FT_SetUSBParameters(fthandle, InTransferSize, 0);
128     if (ftstatus != FT_OK) {
129         printf("changing␣buffer␣size␣failed!␣with␣error␣%d\n", ftstatus);
130         return 1;
131     }
132
133     UCHAR mask = 0xff;
134     UCHAR mode;
135
136     mode = 0x00; // reset mode
137     ftstatus = FT_SetBitMode(fthandle, mask, mode);
138
139     Sleep(10);
140
141     mode = 0x40; // synchronous FIFO mode
142     ftstatus = FT_SetBitMode(fthandle, mask, mode);
143     FT_Purge(fthandle, FT_PURGE_RX);
144
145     if (ftstatus != FT_OK) {
146         printf("configuration␣failed!␣with␣error␣%d\n", ftstatus);
147         return 1;
148     }
149     else {
150         ftstatus = FT_SetLatencyTimer(fthandle, LatencyTimer);
151         ftstatus = FT_SetUSBParameters(fthandle, 0x10000, 0x10000);
152         ftstatus = FT_SetFlowControl(fthandle, FT_FLOW_RTS_CTS, 0x0, 0x0);
153         ftstatus = FT_SetTimeouts(fthandle, 5000, 5000);
154     }
155     printf("Synchronous␣FIFO␣mode␣set!\n");
156
157     DWORD EventDWord;
158     DWORD RxBytes;
159     DWORD TxBytes;
160     DWORD BytesWritten;
161
162     int dWidth = 512;
163     int dHeight = 424;
164
165     // New frame starting sequence: DDCCBBAA
166     UCHAR data_buf_init[8] = { 0x00, 0x00, 0x00, 0x00, 0xDD, 0xCC, 0xBB, 0xAA };
167
168     //! [Depth image stream loop]
169     while (!protonect_shutdown)
170     {
171         unsigned char ImgBuffer[640000];    // Buffer to store data to send to FT2232H
172
173         // Capture new depth frame
174         listener.waitForNewFrame(frames);
175         libfreenect2::Frame *depth = frames[libfreenect2::Frame::Depth];
176
177         cv::Mat(depth->height, depth->width, CV_32FC1, depth->data).copyTo(depthmat);
178
179         // Fill beginning of buffer with starting sequence
180         for (int n = 0; n < 15; n++)
181             ImgBuffer[n] = data_buf_init[n];
182         int index = 16;
183
```

```
184         // Convert 2D OpenCV mat to buffer data that can be sent
185         for (int i = 0; i < dHeight; i++) {
186             for (int j = 0; j < dWidth; j++) {
187                 UINT16 value = depthmat.at<float>(i, j);    // Read value at (i, j)
188                 char lo = value & 0xFF;                     // Split value into two bytes
189                 char hi = value >> 8;
190 
191                 ImgBuffer[index] = hi;                      // Store the 2 bytes into buffer
192                 ImgBuffer[index + 1] = lo;
193 
194                 index += 2;
195             }
196         }
197 
198         ftstatus = FT_GetStatus(fthandle, &RxBytes, &TxBytes, &EventDWord);     // Get status of FT2232H
199 
200         // Only send data when ready
201         if (ftstatus == FT_OK) {
202             // Function to send data to FT2232_H
203             ftstatus = FT_Write(fthandle, ImgBuffer, sizeof(ImgBuffer), &BytesWritten);
204         }
205 
206         int key = cv::waitKey(1);
207         protonect_shutdown = protonect_shutdown || (key > 0 && ((key & 0xFF) == 27)); // shutdown on escape
208 
209                                                                                       //! [loop end]
210         listener.release(frames);
211     }
212     //! [loop end]
213 
214     //! [stop]
215     dev->stop();
216     dev->close();
217     //! [stop]
218 
219     delete registration;
220 
221     std::cout << "Streaming␣Ends!" << std::endl;
222     return 0;
223 }
```

Appendix G

Arduino HID Mouse Code

```
// Author: Hope Harrison
// Description: Teensy as HID mouse from FPGA PMOD data

// set this to the hardware serial port you wish to use
#define HWSERIAL Serial2

int x_1 = 0;
int x_2 = 0;
int y_1 = 0;
int y_2 = 0;
int x = 0;
int y = 0;
int top_left_x_1 = 0;
int top_left_x_2 = 0;
int top_left_x = 0;
int top_left_y_1 = 0;
int top_left_y_2 = 0;
int top_left_y = 0;
int bottom_right_x_1 = 0;
int bottom_right_x_2 = 0;
int bottom_right_x = 0;
int bottom_right_y_1 = 0;
int bottom_right_y_2 = 0;
int bottom_right_y = 0;
int relative_x = 0;
int relative_y = 0;
int is_press = 0;
int is_release = 0;
int is_move = 0;

const int MONITOR_X = 3840;
const int MONITOR_Y = 2160;

const int OFFSET_X = 200;
const int OFFSET_Y = -200;

void setup() {
  Serial.begin(9600);
  HWSERIAL.begin(115200);
  Mouse.screenSize(MONITOR_X, MONITOR_Y);
}

void loop() {
        int incomingByte;

  if (HWSERIAL.available() >= 15) {
    x_1 = HWSERIAL.read();
    x_2 = HWSERIAL.read();
    y_1 = HWSERIAL.read();
    y_2 = HWSERIAL.read();
    top_left_x_1 = HWSERIAL.read();
    top_left_x_2 = HWSERIAL.read();
    top_left_y_1 = HWSERIAL.read();
    top_left_y_2 = HWSERIAL.read();
    bottom_right_x_1 = HWSERIAL.read();
    bottom_right_x_2 = HWSERIAL.read();
    bottom_right_y_1 = HWSERIAL.read();
    bottom_right_y_2 = HWSERIAL.read();
    is_press = HWSERIAL.read();
    is_release = HWSERIAL.read();
    is_move = HWSERIAL.read();

    x = x_2*256 + x_1;
    y = y_2*256 + y_1;
    top_left_x = top_left_x_2*256 + top_left_x_1;
    top_left_y = top_left_y_2*256 + top_left_y_1;
    bottom_right_x = bottom_right_x_2*256 + bottom_right_x_1;
    bottom_right_y = bottom_right_y_2*256 + bottom_right_y_1;

    relative_x = MONITOR_X - float(x - top_left_x) / float(bottom_right_x - top_left_x) * MONITOR_X + OFFSET_X;
    relative_y = float(y - top_left_y) / float(bottom_right_y - top_left_y) * MONITOR_Y + OFFSET_Y;

    if (is_move) {
      if((0 < x) && (x < MONITOR_X) && (0 < y) && (y < MONITOR_Y))
        Mouse.moveTo(relative_x, relative_y);
    }
    if (is_press) {
      Mouse.press(MOUSE_LEFT);
    }
    else if (is_release) {
      Mouse.release(MOUSE_LEFT);
    }

    //Use this to scroll:
//    Mouse.scroll(scroll_up - scroll_down, 0);

// For debugging
    Serial.print("Absolute touch point (x,y) = (");
    Serial.print(x, DEC);
    Serial.print(", ");
    Serial.print(y, DEC);
    Serial.println(")");
```

```
93
94      Serial.print("Top␣left␣(x,y)␣=␣(");
95      Serial.print(top_left_x, DEC);
96      Serial.print(",␣");
97      Serial.print(top_left_y, DEC);
98      Serial.println(")");
99
100     Serial.print("Bottom␣right␣(x,y)␣=␣(");
101     Serial.print(bottom_right_x, DEC);
102     Serial.print(",␣");
103     Serial.print(bottom_right_y, DEC);
104     Serial.println(")");
105
106     Serial.print("Move␣mouse␣to␣(x,y)␣=␣(");
107     Serial.print(relative_x, DEC);
108     Serial.print(",␣");
109     Serial.print(relative_y, DEC);
110     Serial.println(")");
111
112     Serial.print("Press?␣");
113     Serial.println(is_press, BIN);
114     Serial.print("Release?␣");
115     Serial.println(is_release, BIN);
116     Serial.print("Move?␣");
117     Serial.println(is_move, BIN);
118
119 //    Serial.print("Scroll = ");
120 //    Serial.println(scroll_up - scroll_down);
121   }
122 }
```